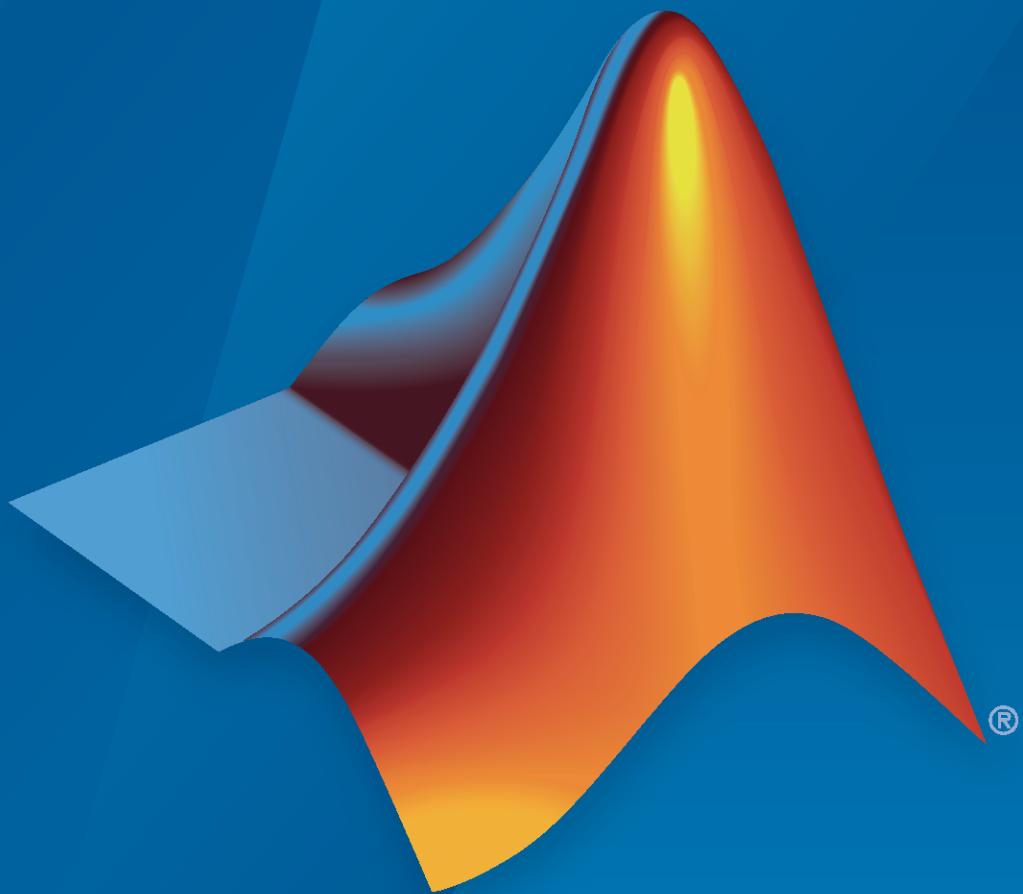


Image Processing Toolbox™

用户指南



MATLAB®

R2021a

 MathWorks®

如何联系 MathWorks



最新动态: www.mathworks.com
销售和服务: www.mathworks.com/sales_and_services
用户社区: www.mathworks.com/matlabcentral
技术支持: www.mathworks.com/support/contact_us



电话: 010-59827000



迈斯沃克软件(北京)有限公司
北京市朝阳区望京东园四区6号楼
北望金辉大厦16层1604

Image Processing Toolbox™ 用户指南

© COPYRIGHT 1993–2021 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

商标

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

专利

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

修订历史记录

1993 年 8 月	第一次印刷	版本 1
1997 年 5 月	第二次印刷	版本 2
2001 年 4 月	第三次印刷	版本 3.0 中的修订内容
2001 年 6 月	仅限在线版本	版本 3.1 (版本 12.1) 中的修订内容
2002 年 7 月	仅限在线版本	版本 3.2 (版本 13) 中的修订内容
2003 年 5 月	第四次印刷	版本 4.0 (版本 13.0.1) 中的修订内容
2003 年 9 月	仅限在线版本	版本 4.1 (版本 13.SP1) 中的修订内容
2004 年 6 月	仅限在线版本	版本 4.2 (版本 14) 中的修订内容
2004 年 8 月	仅限在线版本	版本 5.0 (版本 14+) 中的修订内容
2004 年 10 月	第五次印刷	版本 5.0.1 (版本 14SP1) 中的修订内容
2005 年 3 月	仅限在线版本	版本 5.0.2 (版本 14SP2) 中的修订内容
2005 年 9 月	仅限在线版本	版本 5.1 (版本 14SP3) 中的修订内容
2006 年 3 月	仅限在线版本	版本 5.2 (版本 2006a) 中的修订内容
2006 年 9 月	仅限在线版本	版本 5.3 (版本 2006b) 中的修订内容
2007 年 3 月	仅限在线版本	版本 5.4 (版本 2007a) 中的修订内容
2007 年 9 月	仅限在线版本	版本 6.0 (版本 2007b) 中的修订内容
2008 年 3 月	仅限在线版本	版本 6.1 (版本 2008a) 中的修订内容
2008 年 10 月	仅限在线版本	版本 6.2 (版本 2008b) 中的修订内容
2009 年 3 月	仅限在线版本	版本 6.3 (版本 2009a) 中的修订内容
2009 年 9 月	仅限在线版本	版本 6.4 (版本 2009b) 中的修订内容
2010 年 3 月	仅限在线版本	版本 7.0 (版本 2010a) 中的修订内容
2010 年 9 月	仅限在线版本	版本 7.1 (版本 2010b) 中的修订内容
2011 年 4 月	仅限在线版本	版本 7.2 (版本 2011a) 中的修订内容
2011 年 9 月	仅限在线版本	版本 7.3 (版本 2011b) 中的修订内容
2012 年 3 月	仅限在线版本	版本 8.0 (版本 2012a) 中的修订内容
2012 年 9 月	仅限在线版本	版本 8.1 (版本 2012b) 中的修订内容
2013 年 3 月	仅限在线版本	版本 8.2 (版本 2013a) 中的修订内容
2013 年 9 月	仅限在线版本	版本 8.3 (版本 2013b) 中的修订内容
2014 年 3 月	仅限在线版本	版本 9.0 (版本 2014a) 中的修订内容
2014 年 10 月	仅限在线版本	版本 9.1 (版本 2014b) 中的修订内容
2015 年 3 月	仅限在线版本	版本 9.2 (版本 2015a) 中的修订内容
2015 年 9 月	仅限在线版本	版本 9.3 (版本 2015b) 中的修订内容
2016 年 3 月	仅限在线版本	版本 9.4 (版本 2016a) 中的修订内容
2016 年 9 月	仅限在线版本	版本 9.5 (版本 2016b) 中的修订内容
2017 年 3 月	仅限在线版本	版本 10.0 (版本 2017a) 中的修订内容
2017 年 9 月	仅限在线版本	版本 10.1 (版本 2017b) 中的修订内容
2018 年 3 月	仅限在线版本	版本 10.2 (版本 2018a) 中的修订内容
2018 年 9 月	仅限在线版本	版本 10.3 (版本 2018b) 中的修订内容
2019 年 3 月	仅限在线版本	版本 10.4 (版本 2019a) 中的修订内容
2019 年 9 月	仅限在线版本	版本 11.0 (版本 2019b) 中的修订内容
2020 年 3 月	仅限在线版本	版本 11.1 (版本 2020a) 中的修订内容
2020 年 9 月	仅限在线版本	11.2 版 (版本 2020b) 中的修订内容
2021 年 3 月	仅限在线版本	版本 11.3 (版本 2021a) 中的修订内容

目录

快速入门

1

基本图像导入、处理和导出	1-2
校正亮度不均匀问题并分析前景对象	1-7

简介

2

MATLAB 中的图像	2-2
图像坐标系	2-3
像素索引	2-3
空间坐标	2-3
平移所显示图像的 X 和 Y 坐标范围	2-5
检测交通视频中的汽车	2-8

读取和写入图像数据

3

以图形格式将图像数据写入文件	3-2
从 DICOM 文件中读取图像数据	3-3
查看 DICOM 图像	3-3

显示和浏览图像

4

在图窗窗口中显示图像	4-2
概述	4-2
指定初始图像放大倍率	4-4
控制图窗的外观	4-4

显示多个图像	4-6
在单独的图窗窗口中显示多个图像	4-6
以蒙太奇方式显示多个图像	4-6
在同一图窗中显示多个图像	4-7
比较一对图像	4-8

用模块化工具构建 GUI

5

几何变换

6

使用 imresize 函数调整图像大小	6-2
裁剪图像	6-6
几何变换的矩阵表示	6-7
二维仿射变换	6-7
二维投影变换	6-8
创建复合二维仿射变换	6-8
三维仿射变换	6-10
探查三维 MRI 数据集的切片	6-12

图像配准

7

为基于强度的图像配准创建优化器和指标	7-2
控制点选择过程	7-3

图像数据线性滤波器的设计与实现

8

什么是空间域中的图像滤波?	8-2
卷积	8-2
相关性	8-3
imfilter 边界填充选项	8-5
什么是引导图像滤波?	8-8

9

傅里叶变换	9-2
傅里叶变换的定义	9-2
离散傅里叶变换	9-5
傅里叶变换的应用	9-8
离散余弦变换	9-12
DCT 定义	9-12
DCT 变换矩阵	9-13
使用离散余弦变换的图像压缩	9-13
Hough 变换	9-16
使用 Hough 变换检测图像中的线条	9-16

形态学运算

10

形态学运算的类型	10-2
形态学膨胀和腐蚀	10-2
基于膨胀和腐蚀的运算	10-4
像素连通性	10-9
定义图像中的连通性	10-9
选择连通性	10-10
指定自定义连通性	10-10
泛洪填充运算	10-11
指定连通性	10-11
指定起点	10-11
填充孔	10-12
使用边缘检测和形态学检测细胞	10-13
二值图像的距离变换	10-18
标注和度量二值图像中的连通分量	10-20
检测连通分量	10-20
标注连通分量	10-21
选择二值图像中的对象	10-22
测量连通分量的属性	10-22

分析和增强图像

11

像素值	11-2
确定图像中单个像素的值	11-2

图像均值、标准差和相关系数	11-3
边缘检测	11-4
检测图像中的边缘	11-4
检测和测量图像中的圆形目标	11-6
标识圆形目标	11-18
使用灰度共生矩阵 (GLCM) 的纹理分析	11-26
创建灰度共生矩阵	11-27
对比度增强方法	11-28
直方图均衡化	11-32
使用直方图均衡化调整强度值	11-32
绘制直方图均衡化的变换曲线	11-33
将高斯平滑滤波器应用于图像	11-35
去除噪声	11-41
通过线性滤波去除噪声	11-41
使用平均值滤波器和中位数滤波器去除噪声	11-41
通过自适应滤波去除噪声	11-44
使用附加功能资源管理器安装示例数据	11-47

基于关注区域的处理

12

13

图像分割

使用纹理滤波器的纹理分割	13-2
使用 L*a*b* 颜色空间实现基于颜色的分割	13-10
使用 K 均值聚类实现基于颜色的分割	13-16
标记控制的分水岭分割	13-21

14

使用 Wiener 滤波器对图像进行去模糊	14-2
使用盲反卷积算法对图像进行去模糊	14-9

15**16****17****18**

使用预训练的神经网络去除彩色图像中的噪声	18-2
使用深度学习的神经样式迁移	18-8

19**20**

图像处理的代码生成	20-2
使用共享库进行代码生成	20-2
为目标检测生成代码	20-4

GPU 上的图像处理	21-2
----------------------	------

快速入门

本主题提供两个示例，帮助您开始使用 MATLAB® 和 Image Processing Toolbox 软件进行图像处理。这些示例包含对文档中其他章节的交叉引用，这些章节对示例中提及的概念进行了深入讨论。

- “基本图像导入、处理和导出”（第 1-2 页）
- “校正亮度不均匀问题并分析前景对象”（第 1-7 页）

基本图像导入、处理和导出

此示例说明如何将图像读入工作区、调整图像的对比度，然后将调整后的图像写入文件。

步骤 1：读取并显示图像

使用 `imread` 命令将图像读入工作区。该示例读取工具箱附带的一个示例图像（名为 `pout.tif` 的文件中一个小女孩的图像），并将其存储在名为 `I` 的数组中。`imread` 根据文件推断图形文件格式为标记图像文件格式 (TIFF)。

```
I = imread('pout.tif');
```

使用 `imshow` 函数显示该图像。您也可以在图像查看器中查看图像。`imtool` 函数打开图像查看器，该 App 提供显示图像和执行一些常见图像处理任务的集成环境。图像查看器提供 `imshow` 的所有图像显示功能，还支持访问其他几个图像导航和浏览工具，如滚动条、像素区域工具、图像信息工具和对比度调节工具。

```
imshow(I)
```



步骤 2：检查图像在工作区中的显示方式

使用 `whos` 命令，检查 `imread` 函数如何在工作区中存储图像数据。您也可以在工作区浏览器中检查变量。`imread` 函数返回变量 `I` 中的图像数据，这是由 `uint8` 数据组成的 291×240 元素数组。

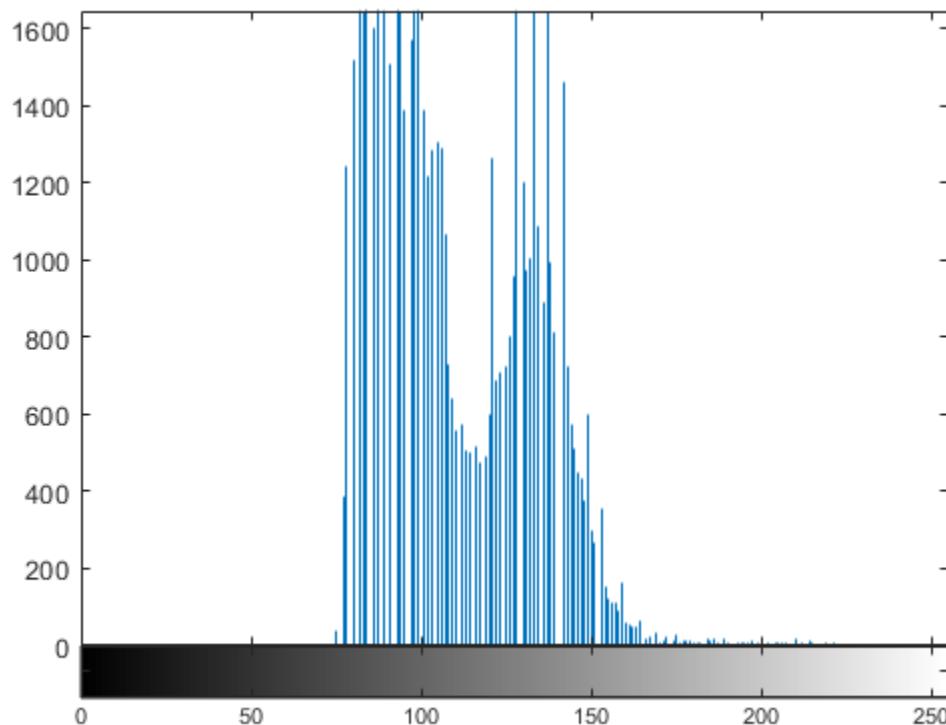
```
whos I
```

Name	Size	Bytes	Class	Attributes
I	291x240	69840	uint8	

步骤 3：提高图像对比度

查看图像像素强度的分布。图像 **pout.tif** 是一个对比度较低的图像。要查看图像中的强度分布，请通过调用 **imhist** 函数创建直方图。（请在调用 **imhist** 之前使用 **figure** 命令，这样直方图就不会覆盖当前图窗窗口中显示的图像 **I**。）请注意直方图表明图像的强度范围相当窄。该范围未能涵盖 [0, 255] 之间尽可能多的范围，并缺少能产生良好对比度的高值和低值。

```
figure  
imhist(I)
```



使用 **histeq** 函数提高图像的对比度。直方图均衡使强度值扩展分布到了图像的完整范围内。显示图像。

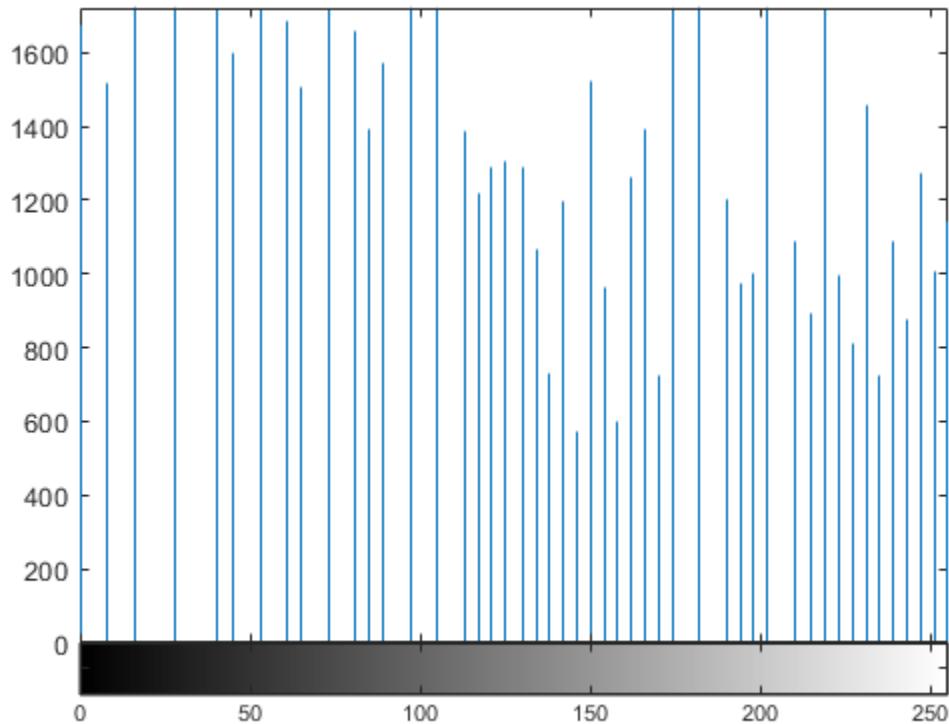
（工具箱包含用于执行对比度调整的其他几个函数，包括 **imadjust** 和 **adaphisteq**，以及图像查看器中提供的交互式工具，如 **Adjust Contrast** 工具。）

```
I2 = histeq(I);  
figure  
imshow(I2)
```



再次调用 `imhist` 函数，创建经过均衡处理的图像 `I2` 的直方图。如果比较这两个直方图，会看到 `I2` 的直方图比 `I` 的直方图在整个强度范围内分布得更广。

```
figure  
imhist(I2)
```



步骤 4：将调整后的图像写入磁盘文件

使用 `imwrite` 函数，将刚刚经过调整的图像 `I2` 写入磁盘文件。此示例在文件名中包含文件扩展名 '`.png`'，因此 `imwrite` 函数将图像以可移植网络图形 (PNG) 格式写入文件，但您可以指定其他格式。

```
imwrite(I2, 'pout2.png');
```

步骤 5：检查新写入文件的内容

使用 `imfinfo` 函数查看 `imwrite` 写入磁盘文件的内容。`imfinfo` 函数返回文件中图像的相关信息，例如图像的格式、大小、宽度和高度。

```
imfinfo('pout2.png')

ans = struct with fields:
    Filename: '/tmp/Bdoc21a_1606923_218278/tp29affe54/images-ex89505080/pout2.png'
    FileModDate: '23-Feb-2021 13:04:06'
    FileSize: 36938
    Format: 'png'
    FormatVersion: []
    Width: 240
    Height: 291
    BitDepth: 8
    ColorType: 'grayscale'
    FormatSignature: [137 80 78 71 13 10 26 10]
    Colormap: []
    Histogram: []
    InterlaceType: 'none'
```

```
Transparency: 'none'  
SimpleTransparencyData: []  
BackgroundColor: []  
RenderingIntent: []  
Chromaticities: []  
    Gamma: []  
    XResolution: []  
    YResolution: []  
ResolutionUnit: []  
    XOffset: []  
    YOffset: []  
    OffsetUnit: []  
SignificantBits: []  
ImageModTime: '23 Feb 2021 18:04:06 +0000'  
    Title: []  
    Author: []  
    Description: []  
    Copyright: []  
CreationTime: []  
    Software: []  
    Disclaimer: []  
    Warning: []  
    Source: []  
    Comment: []  
OtherText: []
```

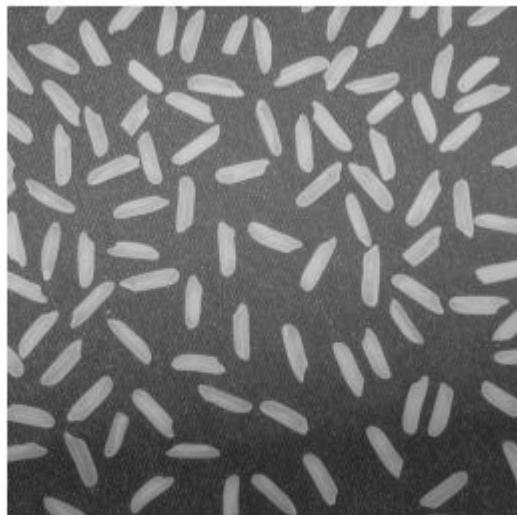
校正亮度不均匀问题并分析前景对象

此示例说明如何在分析前的预处理步骤增强图像。在此示例中，您可以校正背景亮度不均匀问题，并将图像转换为二值图像，以便于识别前景对象（单个米粒）。然后，您可以分析对象，例如计算每粒大米的面积，并可以计算图像中所有对象的统计量。

预处理图像

将图像读入工作区。

```
I = imread('rice.png');  
imshow(I)
```



图像中心的背景亮度比底部亮度高。预处理图像，使背景亮度更加均匀。

第一步，使用形态学开运算删除所有前景（米粒）。开运算会删除无法完全包含结构元素的小对象。定义半径为 15 的盘形结构元素，它完全可放入一粒米内。

```
se = strel('disk',15)  
  
se =  
strel is a disk shaped structuring element with properties:  
  
    Neighborhood: [29x29 logical]  
    Dimensionality: 2
```

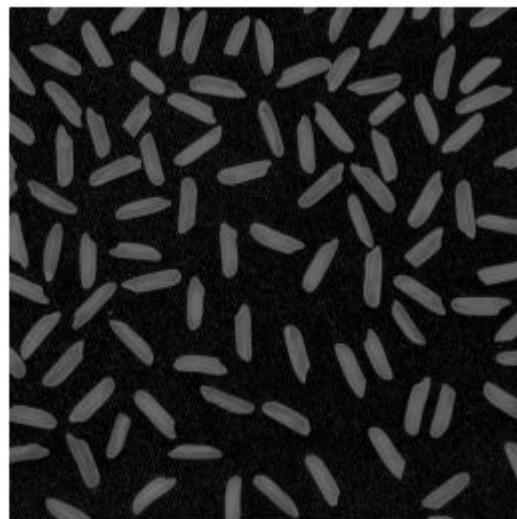
要执行形态学开运算，请使用具有结构元素的 **imopen**。

```
background = imopen(I,se);  
imshow(background)
```



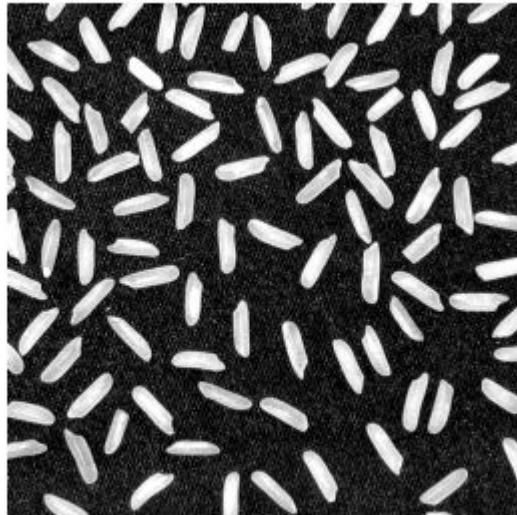
从原始图像 `I` 中减去背景逼近图像 `background`, 然后查看生成的图像。从原始图像中减去调整后的背景图像后, 生成的图像具有均匀的背景, 但现在对于分析来说有点暗。

```
I2 = I - background;  
imshow(I2)
```



使用 **imadjust**, 通过在低强度和高强度下都对 1% 的数据进行饱和处理，并通过拉伸强度值以填充 **uint8** 动态范围，来提高处理后的图像 **I2** 的对比度。

```
I3 = imadjust(I2);
imshow(I3)
```

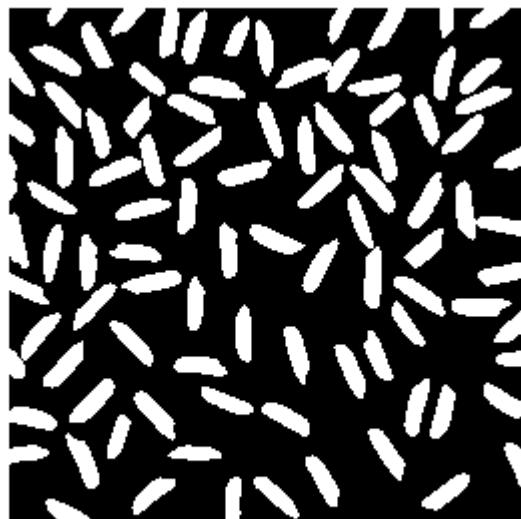


请注意，前面的两个步骤可以由使用 **imtophat** 的一个步骤来代替完成，后者先计算形态学开运算，然后从原始图像中减去它。

```
I2 = imtophat(I,strel('disk',15));
```

创建处理后的图像的二值版本，以便使用工具箱函数进行分析。使用 **imbinarize** 函数将灰度图像转换为二值图像。使用 **bwareaopen** 函数去除图像中的背景噪声。

```
bw = imbinarize(I3);
bw = bwareaopen(bw,50);
imshow(bw)
```



识别图像中的对象

现在您已创建原始图像的二值版本，您可以对图像中的对象执行分析。

在二值图像中查找所有连通分量（对象）。结果的准确度取决于对象的大小、连通性参数（4、8 或任意值），以及是否有相互接触的对象（在这种情况下，它们可能被标记为一个对象）。二值图像 **bw** 中的一些米粒相互接触。

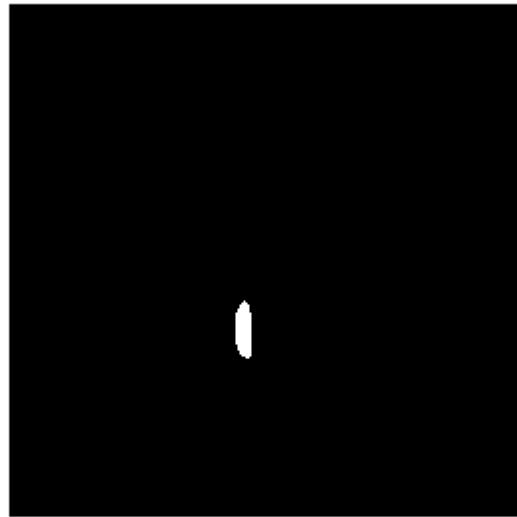
```
cc = bwconncomp(bw,4)  
  
cc = struct with fields:  
    Connectivity: 4  
    ImageSize: [256 256]  
    NumObjects: 95  
    PixelIdxList: {1x95 cell}
```

```
cc.NumObjects
```

```
ans = 95
```

查看图像中标记为 50 的米粒。

```
grain = false(size(bw));  
grain(cc.PixelIdxList{50}) = true;  
imshow(grain)
```



通过创建标签矩阵，然后将其显示为伪彩色索引图像，可视化图像中的所有连通分量。

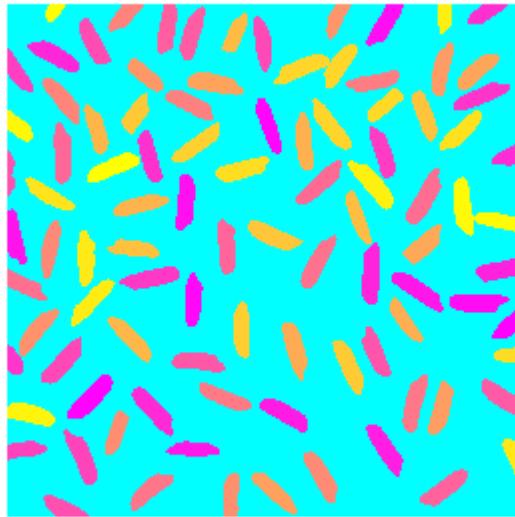
使用 **labelmatrix** 根据 **bwconncomp** 的输出创建标签矩阵。请注意，**labelmatrix** 将标签矩阵存储在依对象数量得出的最小数值类中。

```
labeled = labelmatrix(cc);
whos labeled
```

Name	Size	Bytes	Class	Attributes
labeled	256x256	65536	uint8	

使用 **label2rgb** 选择颜色图、背景颜色以及标签矩阵中的对象如何映射到颜色图中的颜色。在伪彩色图像中，用于标识标签矩阵中每个对象的标签映射到相关联的颜色图矩阵中的不同颜色。

```
RGB_label = label2rgb(labeled,'spring','c','shuffle');
imshow(RGB_label)
```



计算基于面积的统计量

使用 **regionprops** 计算图像中每个对象的面积。每个米粒均为 cc 结构体中的一个连通分量。

```
graindata = regionprops(cc,'basic')
```

```
graindata=95×1 struct array with fields:  
    Area  
    Centroid  
    BoundingBox
```

创建新向量 **grain_areas**, 它保存每个米粒的面积测量值。

```
grain_areas = [graindata.Area];
```

计算第 50 个分量的面积。

```
grain_areas(50)
```

```
ans = 194
```

找到并显示面积最小的米粒。

```
[min_area, idx] = min(grain_areas)
```

```
min_area = 61
```

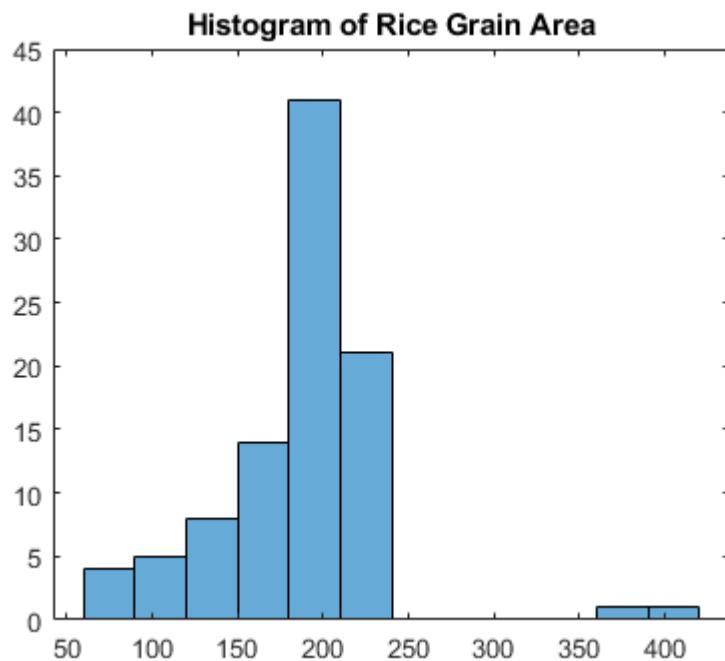
```
idx = 16
```

```
grain = false(size(bw));  
grain(cc.PixelIdxList{idx}) = true;  
imshow(grain)
```



使用 `histogram` 命令创建米粒面积的直方图。

```
histogram(grain_areas)  
title('Histogram of Rice Grain Area')
```



另请参阅

[imopen](#) | [bwareaopen](#) | [bwconncomp](#) | [regionprops](#) | [imadjust](#) | [imbinarize](#) | [label2rgb](#) |
[labelmatrix](#) | [imread](#) | [imshow](#)

简介

本章介绍使用 MATLAB 和 Image Processing Toolbox 软件进行图像处理的基础知识。

- “MATLAB 中的图像” (第 2-2 页)
- “图像坐标系” (第 2-3 页)
- “平移所显示图像的 X 和 Y 坐标范围” (第 2-5 页)
- “检测交通视频中的汽车” (第 2-8 页)

MATLAB 中的图像

MATLAB 中的基本数据结构体是数组，即实数或复数元素的有序集合。此对象天然适合表示图像，即颜色或强度数据的实数值有序集合。

MATLAB 将大多数图像存储为二维矩阵，其中矩阵的每个元素对应于所显示图像中的单个离散像素。（像素派生自图片元素，通常表示计算机显示上的一个点。）例如，由 200 行和 300 列不同颜色的点组成的图像将作为 200×300 矩阵存储在 MATLAB 中。

有些图像，如真彩色图像，使用三维数组表示图像。在真彩色图像中，第三个维度中的第一个平面表示红色像素强度，第二个平面表示绿色通道强度，第三个平面表示蓝色像素强度。此约定使用户可以像处理任何其他类型的数值数据一样处理 MATLAB 中的图像，并充分利用 MATLAB 的强大功能来进行图像处理应用。

有关 Image Processing Toolbox 如何分配像素索引以及如何将像素索引与连续空间坐标相关联的详细信息，请参阅“[图像坐标系](#)”（第 2-3 页）。

另请参阅

[imread](#) | [imshow](#)

相关示例

- “[基本图像导入、处理和导出](#)”（第 1-2 页）

详细信息

- “[Image Types in the Toolbox](#)”

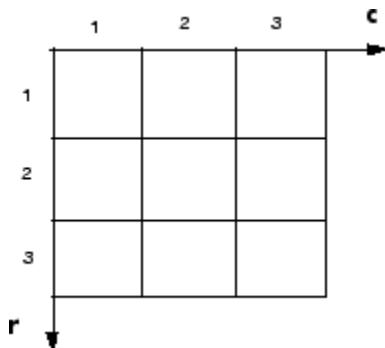
图像坐标系

您可以使用几种不同图像坐标系来访问图像中的位置。您可以使用离散像素索引指定位置，因为图像存储为数组。您也可以使用连续空间坐标来指定位置，因为图像表示连续空间中的真实世界场景。

像素索引

如“MATLAB 中的图像”（第 2-2 页）中所述，MATLAB 将大多数图像存储为数组。数组的每个（行、列）索引对应于所显示图像中的单个像素。

前两个矩阵维度的像素索引和下标之间有一对一的对应关系。与 MATLAB 中的数组索引一样，像素索引是整数值，范围是从 1 到行或列的长度。索引的顺序是从上到下、从左到右。



例如，第五行第二列中的像素的数据存储在矩阵元素 (5,2) 中。您可以使用普通的 MATLAB 矩阵下标来访问单个像素的值。例如，MATLAB 代码

`I(2,15)`

返回单通道图像 `I` 的第 2 行第 15 列的像素值。同样，MATLAB 代码

`RGB(2,15,:)`

返回多通道图像 `RGB` 的第 2 行第 15 列的像素的颜色值。

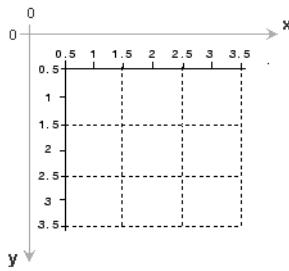
空间坐标

在空间坐标系中，图像中的位置是连续平面上的位置。位置用笛卡尔 x 和 y 坐标（而不是如像素索引系统中的行和列索引）来描述。从笛卡尔坐标角度来看， (x,y) 位置（如 (3.2,5.3)）是有意义的，并且不同于坐标 (5,3)。

Image Processing Toolbox 根据参考框架定义两种类型的空间坐标系。内部坐标指定相对于图像参考框架的位置。世界坐标指定相对于外部世界观察者的位置。

内部坐标

默认情况下，工具箱使用内部坐标系定义空间图像坐标。此空间坐标系对应于图像的像素索引。任何像素的中心点的内部坐标 (x,y) 与该像素的列和行索引相同。例如，第 5 行第 3 列中像素的中心点的空间坐标为 $x = 3.0$ 、 $y = 5.0$ 。但请注意，内部坐标 (3.0,5.0) 的顺序与像素索引 (5,3) 相反。



每个像素中心的内部坐标都是整数值。左上角像素中心的内部坐标为 $(1.0, 1.0)$ 。右下角像素中心的内部坐标为 $(\text{numCols}, \text{ numRows})$ ，其中 numCols 和 numRows 是图像中的行数和列数。通常，具有像素索引 (m, n) 的像素的中心位于内部坐标系中的点 $x = n$ 、 $y = m$ 上。

由于内部坐标系中每个像素的大小是一个单位，因此图像的边界具有小数坐标。图像的左上角位于 $(0.5, 0.5)$ ，而不是 $(0, 0)$ 。同样，图像的右下角位于 $(\text{numCols} + 0.5, \text{ numRows} + 0.5)$ 。

有几个函数主要使用空间坐标而不是像素索引，但是，只要使用默认的空间坐标系（内部坐标），就可以根据其列（ x ）和行（ y ）来指定位置。

世界坐标

在某些情况下，您可能希望使用世界坐标系（也称为非默认空间坐标系）。可能需要使用世界坐标系的一些情况包括：

- 当您对图像执行几何运算（如平移）并希望保留新位置与原始位置的关系信息时。
- 当像素布局不是正方形时。例如，在磁共振成像（MRI）中，您可以收集数据，使像素在一个方向上的采样率高于正交方向。
- 当您知道像素的范围如何与现实世界中的位置对应时。例如，在航空照片中，每个像素可能覆盖地面上一个特定的 5×5 米地块。
- 当您要反转 x 轴或 y 轴的方向时。这是一种用于地理空间数据的常见方法。

定义世界坐标系有几种方法。您可以使用空间参照对象，这些对象对图像在世界坐标系中的位置、图像分辨率以及图像范围与内部坐标和世界坐标的关系进行编码。您也可以指定每个维度中的最大和最小坐标。有关详细信息，请参阅“Define World Coordinate System of Image”。

另请参阅

相关示例

- “平移所显示图像的 X 和 Y 坐标范围”（第 2-5 页）

详细信息

- “MATLAB 中的图像”（第 2-2 页）
- “Define World Coordinate System of Image”

平移所显示图像的 X 和 Y 坐标范围

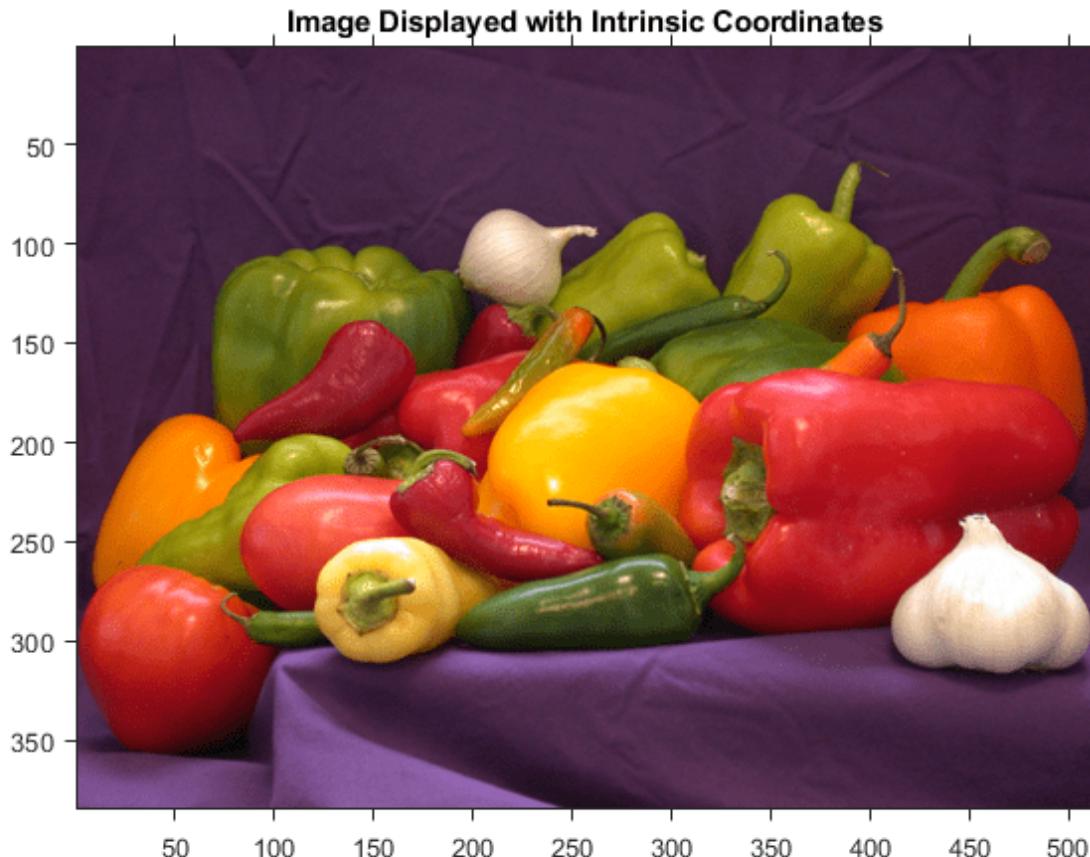
此示例说明如何通过更改所显示图像的 `XData` 和 `YData` 属性来指定非默认世界坐标系。

读取一个图像。

```
I = imread("peppers.png");
```

使用内部坐标系显示该图像，在 `ax` 中返回图像的属性。打开轴以显示坐标系。

```
figure  
ax = imshow(I);  
title('Image Displayed with Intrinsic Coordinates')  
axis on
```



检查存储在 `ax` 的 `XData` 和 `YData` 属性中的 `x` 和 `y` 坐标的范围。范围与图像的维度相匹配。

```
xrange = ax.XData
```

```
xrange = 1×2
```

```
1 512
```

```
yrange = ax.YData
```

```
yrange = 1×2
```

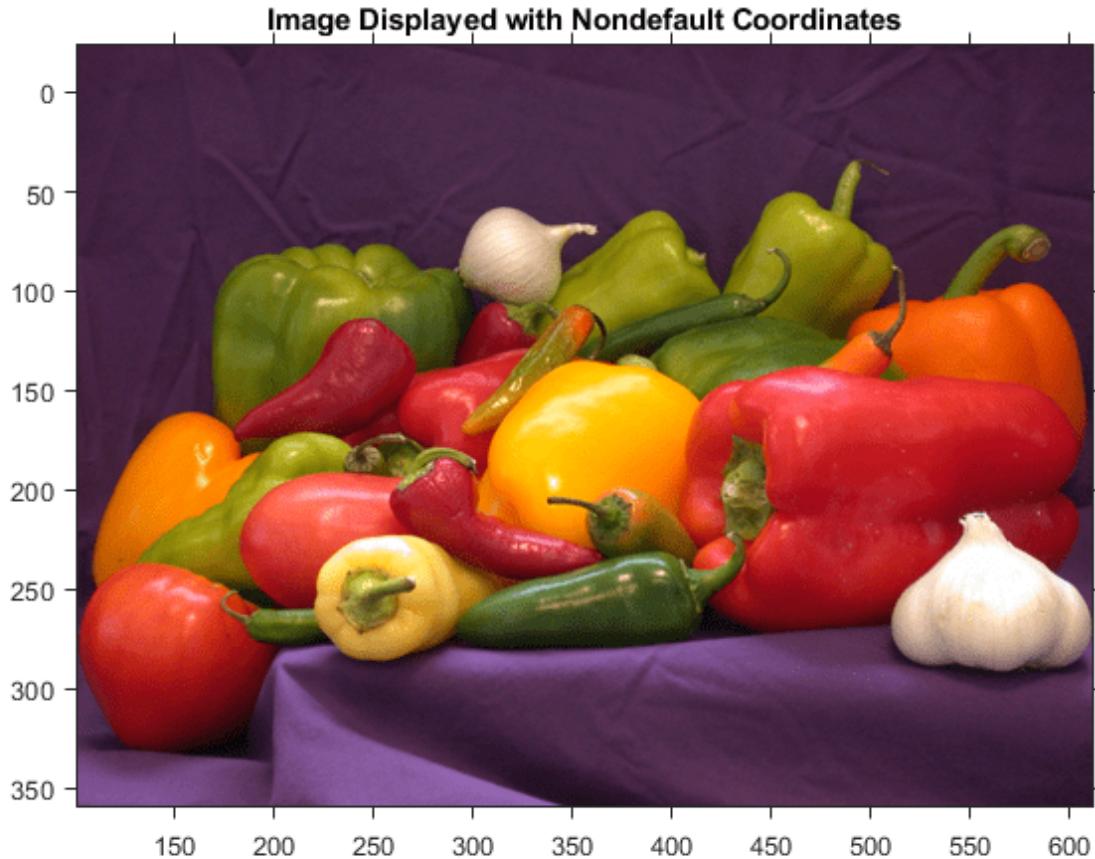
```
1 384
```

更改 x 和 y 坐标的范围。此示例通过对 x 坐标加上 100 来右移图像，并通过从 y 坐标中减去 25 来上移图像。

```
xrangeNew = xrange + 100;  
yrangeNew = yrange - 25;
```

显示图像，指定平移后的空间坐标。

```
figure  
axNew = imshow(I,'XData',xrangeNew,'YData',yrangeNew);  
title('Image Displayed with Nondefault Coordinates');  
axis on
```



确认新图像的 x 和 y 坐标范围与 xrangeNew 和 yrangeNew 指定的平移后的范围相匹配。

```
axNew.XData
```

```
ans = 1×2
```

```
101 612
```

```
axNew.YData
```

```
ans = 1×2
```

```
-24 359
```

另请参阅

详细信息

- “[图像坐标系](#)” (第 2-3 页)
- “[Define World Coordinate System of Image](#)”

检测交通视频中的汽车

此示例说明如何使用 Image Processing Toolbox™ 可视化和分析视频或图像序列。此示例使用 **VideoReader** (MATLAB®)、**implay** 和其他 Image Processing 工具箱函数来检测交通视频中的浅色汽车。请注意，**VideoReader** 的有些功能特定于平台，可能无法在某些平台上读取提供的 Motion JPEG2000 视频。

步骤 1：使用 VideoReader 访问视频

VideoReader 函数构造一个多媒体读取器对象，可以从多媒体文件中读取视频数据。有关您的平台支持哪些格式的信息，请参阅 **VideoReader**。

使用 **VideoReader** 访问视频并获取相关基本信息。

```
trafficVid = VideoReader('traffic.mj2')
```

```
trafficVid =
```

VideoReader with properties:

General Properties:

Name: 'traffic.mj2'

Path: '/mathworks/devel/bat/Bdoc21a/build/matlab/toolbox/images/imdata'

Duration: 8

CurrentTime: 0

NumFrames: 120

Video Properties:

Width: 160

Height: 120

FrameRate: 15

BitsPerPixel: 24

VideoFormat: 'RGB24'

get 方法提供有关视频的详细信息，例如视频的持续时间（以秒为单位）。

```
get(trafficVid)
```

```
obj =
```

VideoReader with properties:

General Properties:

Name: 'traffic.mj2'

Path: '/mathworks/devel/bat/Bdoc21a/build/matlab/toolbox/images/imdata'

Duration: 8

CurrentTime: 0

NumFrames: 120

Video Properties:

Width: 160

Height: 120

FrameRate: 15

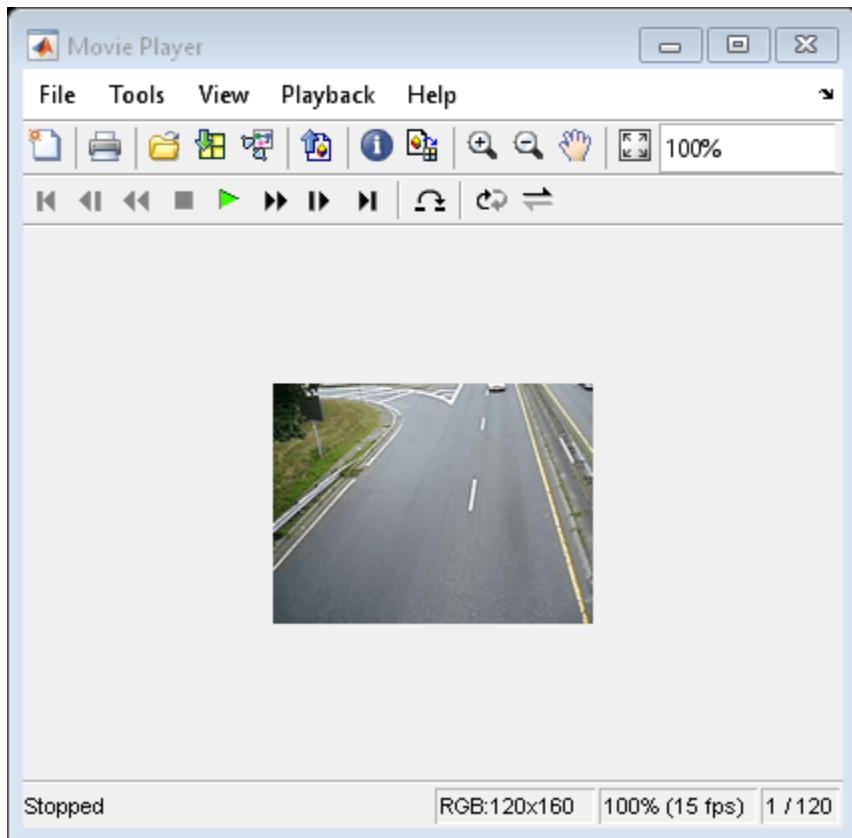
BitsPerPixel: 24

VideoFormat: 'RGB24'

步骤 2：使用 IMPLAY 浏览视频

使用 `implay` 浏览视频。

```
implay('traffic.mj2');
```



步骤 3：开发您的算法

处理视频数据时，可以从视频中选择一个有代表性的帧并基于该帧开发算法，这样会很有帮助。然后，再将该算法应用于视频中所有帧的处理。

对于此汽车标记应用，请找出一个同时包含浅色汽车和深色汽车的帧。当一个图像具有许多结构时，例如交通视频帧，在尝试检测关注的对象之前尽可能简化图像很有必要。对汽车标记应用来说，实现此目的的一种方法是隐藏图像中不是浅色汽车的所有对象（深色汽车、车道、草地等）。通常，需要结合使用多种方法才能去除这些无关的对象。

从视频帧中去除深色汽车的一种方法是使用 `imextendedmax` 函数。此函数返回一个二值图像，该图像识别强度值高于指定阈值（称为区域最大值）的区域。图像中像素值低于此阈值的所有其他对象都将成为背景。要消除深色汽车，请确定图像中这些对象的平均像素值。（使用 `rgb2gray` 将原始视频从 RGB 转换为灰度。）您可以使用 `implay` 中的像素区域工具查看像素值。当您调用 `imextendedmax` 时，请指定平均像素值（或再稍高一些的值）作为阈值。对于此示例，将该值设置为 50。

```
darkCarValue = 50;
darkCar = rgb2gray(read(trafficVid,71));
```

```
noDarkCar = imextendedmax(darkCar, darkCarValue);
imshow(darkCar)
figure, imshow(noDarkCar)
```



观察处理后的图像，会发现大部分的深色汽车对象已经去除，但许多其他无关对象仍然存在，特别是车道标志。区域最大值处理不会去除车道标记，因为其像素值高于阈值。要去除这些对象，可以使用形态学函数 `imopen`。该函数使用形态学处理从二值图像中删除小对象，同时保留大对象。使用形态学处理时，必须决定运算中使用的结构元素的大小和形状。由于车道标志是细长的对象，因此使用半径与车道标志宽度对应的盘形结构元素。您可以在 `implay` 中使用像素区域工具来估计这些对象的宽度。对于此示例，将该值设置为 2。

```
sedisk = strel('disk',2);
noSmallStructures = imopen(noDarkCar, sedisk);
imshow(noSmallStructures)
```



要完成该算法，请使用 `regionprops` 找到 `noSmallStructures` 中对象（应只是浅色汽车）的质心。使用此信息将标记定位在原始视频中的浅色汽车上。

步骤 4：将算法应用于视频

汽车标记应用在循环中以一次一帧的方式处理视频。（由于典型视频一般都包含大量帧，因此一次读取和处理所有帧需要大量内存。）

小视频（如本例中的视频）可以一次性处理，并且有许多函数可以提供这种功能。有关详细信息，请参阅“Process Image Sequences”。

为了加快处理速度，请预分配用于存储已处理视频的内存。

```
nframes = trafficVid.NumberOfFrames;
I = read(trafficVid, 1);
taggedCars = zeros([size(I,1) size(I,2) 3 nframes], class(I));

for k = 1 : nframes
    singleFrame = read(trafficVid, k);

    % Convert to grayscale to do morphological processing.
    I = rgb2gray(singleFrame);

    % Remove dark cars.
    noDarkCars = imextendedmax(I, darkCarValue);

    % Remove lane markings and other non-disk shaped structures.
    noSmallStructures = imopen(noDarkCars, sedisk);

    % Remove small structures.
    noSmallStructures = bwareaopen(noSmallStructures, 150);

    % Get the area and centroid of each remaining object in the frame. The
    % object with the largest area is the light-colored car. Create a copy
    % of the original frame and tag the car by changing the centroid pixel
    % value to red.
    taggedCars(:,:,:,k) = singleFrame;

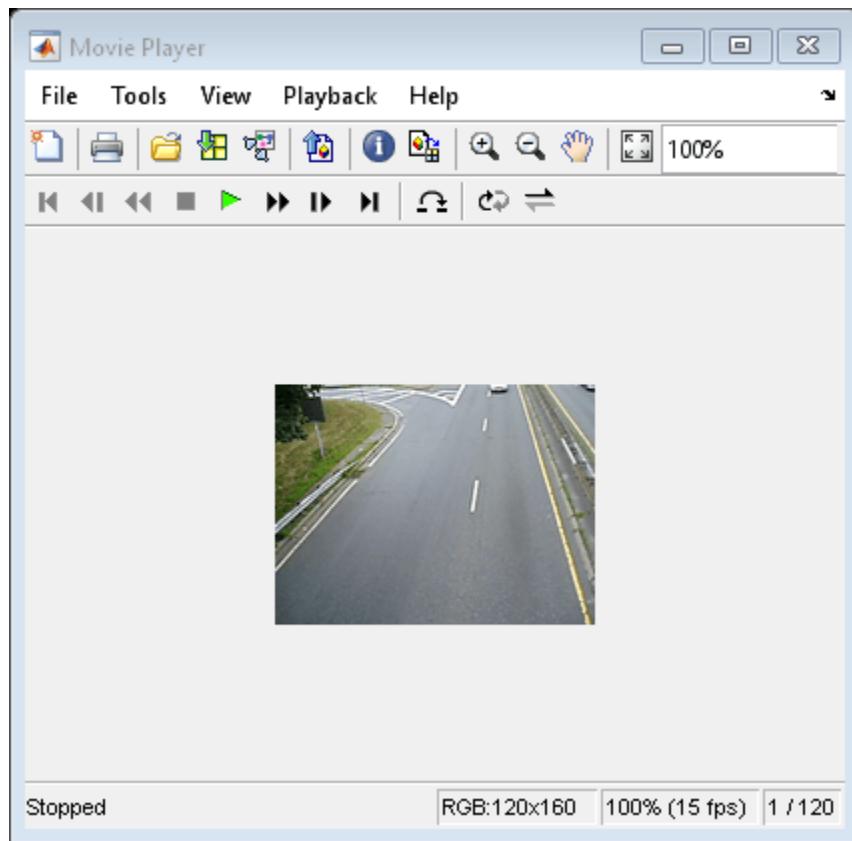
    stats = regionprops(noSmallStructures, {'Centroid','Area'});
    if ~isempty([stats.Area])
        areaArray = [stats.Area];
        [junk,idx] = max(areaArray);
```

```
c = stats(idx).Centroid;
c = floor(fliplr(c));
width = 2;
row = c(1)-width:c(1)+width;
col = c(2)-width:c(2)+width;
taggedCars(row,col,1,k) = 255;
taggedCars(row,col,2,k) = 0;
taggedCars(row,col,3,k) = 0;
end
end
```

步骤 5：可视化结果

获取原始视频的帧速率，并使用它查看 `taggedCars implay`。

```
frameRate = trafficVid.FrameRate;
implay(taggedCars,frameRate);
```



另请参阅

[VideoReader](#) | [implay](#) | [rgb2gray](#) | [imextendedmax](#) | [imopen](#) | [regionprops](#) | [bwareaopen](#)

详细信息

- “Work with Image Sequences as Multidimensional Arrays”
- “Perform an Operation on a Sequence of Images”

读取和写入图像数据

本章介绍如何使用标准图形和医疗文件格式获取图形文件内容的信息、从文件中读取图像数据以及将图像数据写入文件。

- “以图形格式将图像数据写入文件” (第 3-2 页)
- “从 DICOM 文件中读取图像数据” (第 3-3 页)

以图形格式将图像数据写入文件

此示例说明如何使用 `imwrite` 函数将图像数据从工作区写入支持的图形文件格式之一的文件。

将图像数据加载到工作区。此示例从 MAT 文件 `trees.mat` 和关联的颜色图 `map` 加载索引图像 `X`。

```
load trees
whos
```

Name	Size	Bytes Class	Attributes
X	258x350	722400 double	
caption	1x66	132 char	
map	128x3	3072 double	

使用 `imwrite` 将图像数据导出为位图文件，指定变量的名称和要创建的输出文件的名称。如果您在文件名中包含扩展名，`imwrite` 会尝试从中推断所需的文件格式。例如，文件扩展名 `.bmp` 指定 Microsoft Windows 位图格式。您也可以将格式显式指定为 `imwrite` 的参数。

```
imwrite(X,map,'trees.bmp')
```

将格式特定参数与 `imwrite` 结合使用可控制导出过程的各个方面。例如，对于 PNG 文件，您可以指定位深。为了演示，此处以 TIFF 格式将图像读入工作区，并说明其位深。

```
I = imread('cameraman.tif');
s = imfinfo('cameraman.tif');
s.BitDepth
```

```
ans = 8
```

将该图像写入 PNG 格式的图形文件，指定位深为 4。

```
imwrite(I,'cameraman.png','Bitdepth',4)
```

检查新创建文件的位深。

```
newfile = imfinfo('cameraman.png');
newfile.BitDepth
```

```
ans = 4
```

另请参阅

`imwrite`

详细信息

- “Image Types in the Toolbox”
- “Read Image Data into the Workspace”

从 DICOM 文件中读取图像数据

要从 DICOM 文件中读取图像数据，请使用 `dicomread` 函数。`dicomread` 函数读取符合 DICOM 规范的文件，但也可以读取某些常见的不符合规范的文件。

使用 `dicomread` 时，可以将文件名指定为参数，如以下示例所示。该示例读取工具箱附带的 DICOM 示例文件。

```
I = dicomread('CT-MONO2-16-ankle.dcm');
```

您也可以使用 `dicominfo` 返回的元数据结构体来指定要读取的文件，如以下示例所示。

```
info = dicominfo('CT-MONO2-16-ankle.dcm');
I = dicomread(info);
```

查看 DICOM 图像

要查看从 DICOM 文件导入的图像数据，请使用工具箱图像显示函数 `imshow` 或 `imtool`。但请注意，由于此 DICOM 文件中的图像数据是有符号 16 位数据，您必须对任一显示函数使用自动缩放语法才能使图像可见。

```
imshow(I,'DisplayRange',[ ]);
```



另请参阅

App
DICOM 浏览器

函数
`dicomread` | `dicominfo` | `dicomreadVolume`

详细信息

- “Read Metadata from DICOM Files”
- “Create New DICOM Series”
- “Remove Confidential Information from DICOM File”

- “Write Image Data to DICOM Files”

显示和浏览图像

本节介绍 Image Processing Toolbox 软件提供的图像显示和浏览工具。

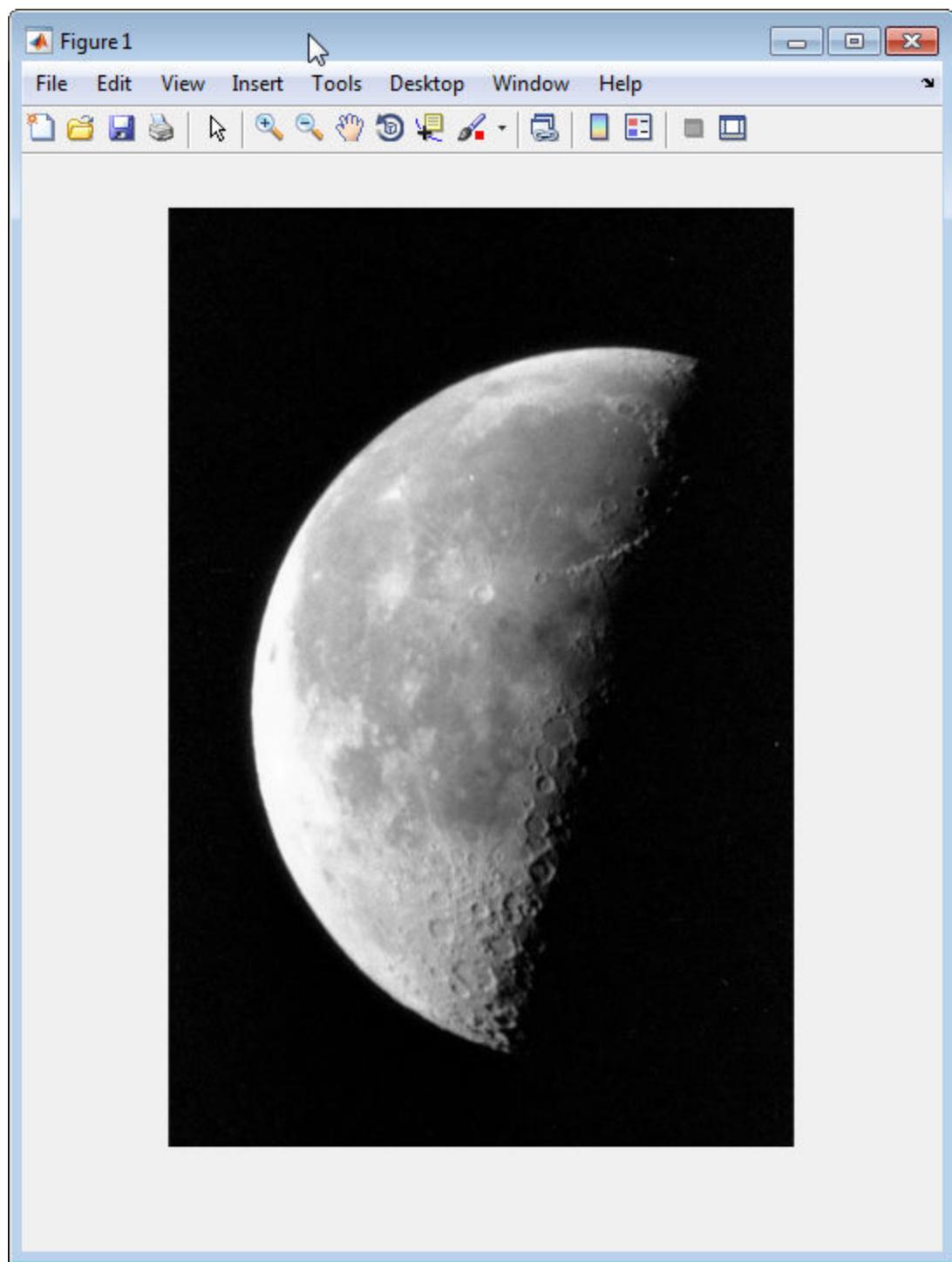
- “在图窗窗口中显示图像” (第 4-2 页)
- “显示多个图像” (第 4-6 页)

在图窗窗口中显示图像

概述

要显示图像数据，请使用 `imshow` 函数。以下示例将一个图像读入工作区中，然后使用 `imshow` 函数在图窗窗口中显示该图像。

```
moon = imread('moon.tif');
imshow(moon);
```



您还可以将包含图像的文件名传递给 **imshow**。

```
imshow('moon.tif');
```

此语法对于扫描图像非常有用。但是，请注意，当您使用此语法时，**imread** 不会将图像数据存储在工作区中。如果要将图像放入工作区，必须使用 **getimage** 函数，该函数从当前图像对象中检索图像数据。如果显示 **moon.tif** 图像数据的图窗窗口当前处于活动状态，则此示例会将其图像数据赋给变量 **moon**。

```
moon = getimage;
```

有关使用 `imshow` 显示工具箱支持的各种图像类型的详细信息，请参阅“Display Different Image Types”。

指定初始图像放大倍率

默认情况下，`imshow` 尝试以 100% 的放大倍率显示整个图像（每个图像像素对应一个屏幕像素）。但是，如果图像太大，无法以 100% 的放大倍率显示在屏幕上的图窗窗口中，`imshow` 会缩放图像以适应屏幕并发出警告消息。

若要覆盖对 `imshow` 的特定调用的默认初始放大行为，请指定 `InitialMagnification` 参数。例如，要以 150% 的放大倍率查看图像，请使用以下代码。

```
pout = imread('pout.tif');
imshow(pout, 'InitialMagnification', 150)
```

`imshow` 尝试遵从您指定的放大倍率。但是，如果图像无法以指定的放大倍率显示在屏幕上，`imshow` 会将图像缩放至合适的大小。您还可以将 '`fit`' 指定为初始放大倍率值。在这种情况下，`imshow` 会缩放图像以适应图窗窗口的当前大小。

要更改 `imshow` 的默认初始放大行为，请设置 `ImshowInitialMagnification` 工具箱预设项。要设置预设项，请通过调用 `iptprefs` 打开 Image Processing Toolbox “预设项” 对话框，或在 MATLAB 主页选项卡的环境部分中，点击  预设。

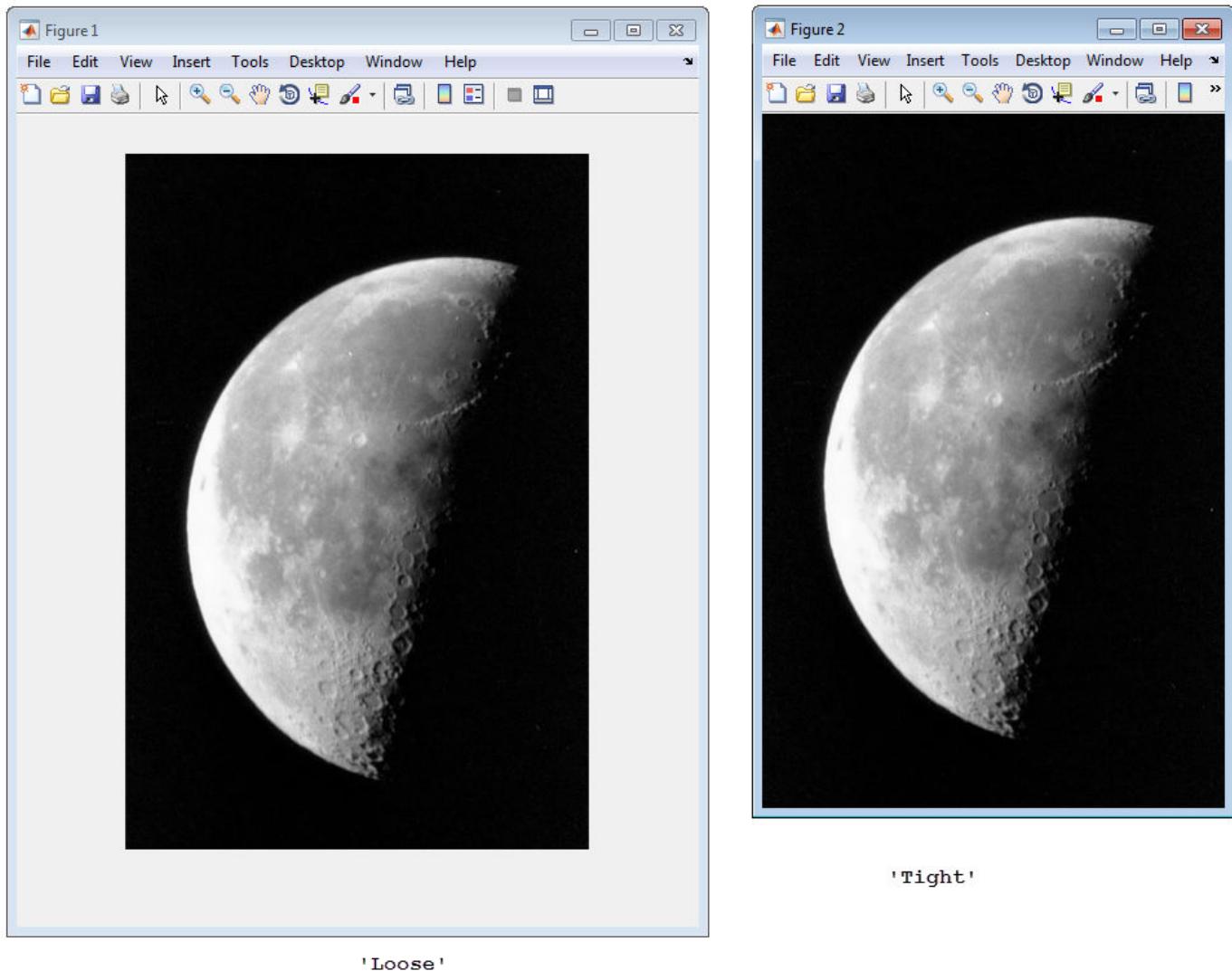
当 `imshow` 缩放图像时，它使用插值来确定与图像矩阵中的元素未直接对应的屏幕像素值。有关指定插值方法的详细信息，请参阅“使用 `imresize` 函数调整图像大小”（第 6-2 页）。

控制图窗的外观

默认情况下，当 `imshow` 在图窗中显示图像时，图像周围会有灰色边框。您可以更改此默认值，并使用 '`border`' 参数隐藏边框，如以下示例所示。

```
imshow('moon.tif','Border','tight')
```

下面的图窗显示同一个图像，一个带边框，一个不带边框。



'border' 参数仅影响调用 `imshow` 时显示的图像。如果您希望用 `imshow` 显示的所有图像都不带灰色边框，请将 Image Processing Toolbox 的 'ImshowBorder' 预设项设置为 'tight'。您也可以使用预设项设置在图窗中包含可见坐标区。有关预设项的详细信息，请参阅 `iptprefs`。

另请参阅

详细信息

- “显示多个图像”（第 4-6 页）

显示多个图像

本节说明同时查看多个图像的各种方法。

在单独的图窗窗口中显示多个图像

同时显示多个图像的最简单方法是在单独的图窗窗口中显示它们。MATLAB 对可同时显示的图像数量没有任何限制。

`imshow` 始终在当前图窗中显示一个图像。如果连续显示两个图像，则第二个图像将替换第一个图像。要使用 `imshow` 查看多个图窗，请使用 `figure` 命令显式创建新的空图窗，然后为下一个图像调用 `imshow`。以下示例查看灰度图像数组 `I` 中的前三帧。

```
imshow(I(:,:,1))
figure, imshow(I(:,:,2))
figure, imshow(I(:,:,3))
```

以蒙太奇方式显示多个图像

您可以使用 `montage` 函数在一个图窗窗口中将多个图像作为单个图像对象进行查看。默认情况下，`montage` 根据图像数量和屏幕大小缩放图像，并将它们排列成一个方形。`montage` 会保留原始图像的纵横比。您可以使用 `ThumbnailSize` 参数指定缩略图的大小。

以蒙太奇方式显示的图像可以有不同类型和大小。`montage` 使用文件中的颜色图将索引图像转换为 RGB。

默认情况下，`montage` 函数在以蒙太奇方式显示图像时不会在图像之间留任何空白。您可以使用 `BorderSize` 参数指定图像之间的空白空间量。您也可以使用 `BackgroundColor` 参数指定图像之间所留空间的颜色。

以下示例说明如何以蒙太奇方式查看图像序列。

以蒙太奇方式查看图像序列

此示例说明如何使用 `montage` 函数一次查看多帧数组中的多个帧。`montage` 显示所有图像帧，并将它们排列成一个矩形网格。图像的蒙太奇被视为单个图像对象。图像帧可以是灰度图像、索引图像或真彩色图像。如果指定索引图像，它们必须都使用相同的颜色图。

创建一个真彩色图像数组。

```
onion = imread('onion.png');
onionArray = repmat(onion, [1 1 4]);
```

以蒙太奇方式一次显示所有图像。默认情况下，`montage` 函数将这些图像显示在一个网格中。第一个图像帧在第一行的第一个位置，下一个帧在第一行的第二个位置，依此类推。

```
montage(onionArray);
```



要指定不同的行数和列数，请使用 'size' 参数。例如，要在一个水平行中显示图像，请将 'size' 参数的值指定为 [1 NaN]。您还可以使用其他 `montage` 参数来指定要显示哪些图像，以及调整所显示图像的对比度。

```
montage(onionArray,'size',[1 NaN]);
```



在同一图窗中显示多个图像

您可以使用 `imshow` 函数和 MATLAB `subplot` 函数在单个图窗窗口中显示多个图像。有关其他选项，请参阅“Work with Image Sequences as Multidimensional Arrays”。

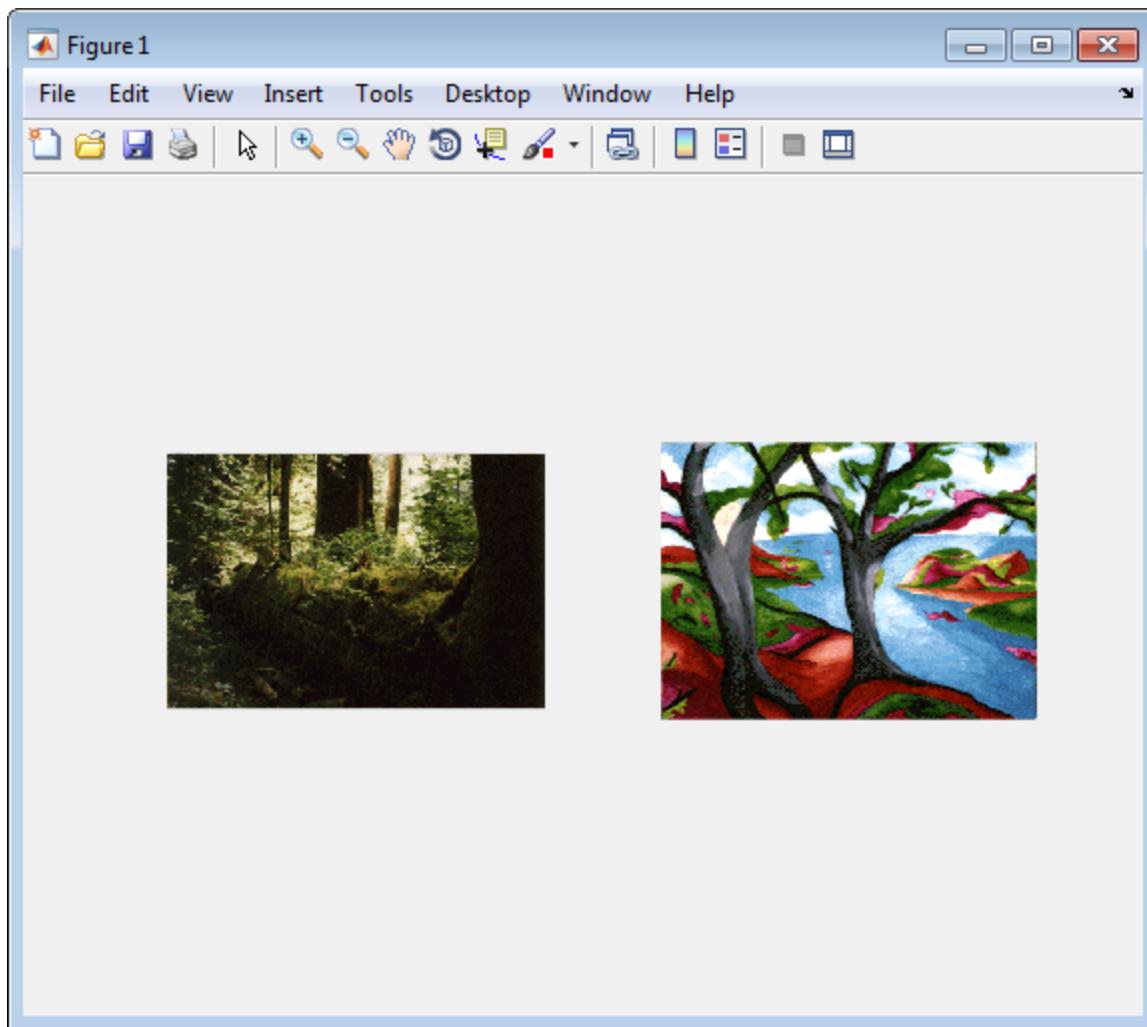
注意 图像查看器 (imtool) 不支持此功能。

将一个图窗窗口分成多个显示区域

subplot 将一个图窗分成多个显示区域。使用语法 **subplot(m,n,p)**，您可以定义一个由多个显示区域组成的 $m \times n$ 矩阵，并指定处于活动状态的区域（即 **p**）。

例如，您可以使用此语法并排显示两个图像。

```
[X1,map1]=imread('forest.tif');
[X2,map2]=imread('trees.tif');
subplot(1,2,1), imshow(X1,map1)
subplot(1,2,2), imshow(X2,map2)
```



比较一对图像

imshowpair 函数在同一图窗窗口中显示一对图像。当比较图像时，这种显示方式很有帮助。**imshowpair** 支持许多可视化方法，包括：

- 伪彩色，即叠加两个基于不同色带的图像。灰色区域表示图像具有相同强度的位置，彩色区域表示图像强度值不同的位置。RGB 图像在以伪彩色显示之前先转换为灰度图像。
- alpha 混合，图像的显示强度是两个输入图像的均值。alpha 混合支持灰度和真彩色图像。
- 棋盘，输出图像由来自两个输入图像的交替矩形区域组成。
- 两个图像的差异。RGB 图像转换为灰度图像。
- 蒙太奇，两个图像并排显示。这种可视化模式类似于使用 **montage** 函数的显示。

imshowpair 使用可选的空间参照信息来显示图像对。

另请参阅

imshow | imshowpair | montage

详细信息

- “在图窗窗口中显示图像”（第 4-2 页）
- “Display Different Image Types”

用模块化工具构建 GUI

本章介绍如何使用交互式模块化工具和创建自定义图像处理应用程序。

几何变换

几何变换（也称为空间变换）会修改图像中像素之间的空间关系，将运动图像中的像素位置映射到输出图像中的新位置。工具箱包含执行特定几何变换的函数，如调整图像大小和旋转图像。此外，工具箱还包含一些其他函数，您可以使用这些函数来执行多种类型的二维和 N 维几何变换，包括自定义变换。

- “使用 `imresize` 函数调整图像大小” （第 6-2 页）
- “裁剪图像” （第 6-6 页）
- “几何变换的矩阵表示” （第 6-7 页）
- “探查三维 MRI 数据集的切片” （第 6-12 页）

使用 imresize 函数调整图像大小

此示例说明如何使用 `imresize` 函数调整图像的大小。

指定放大倍率值

将图像读入工作区。

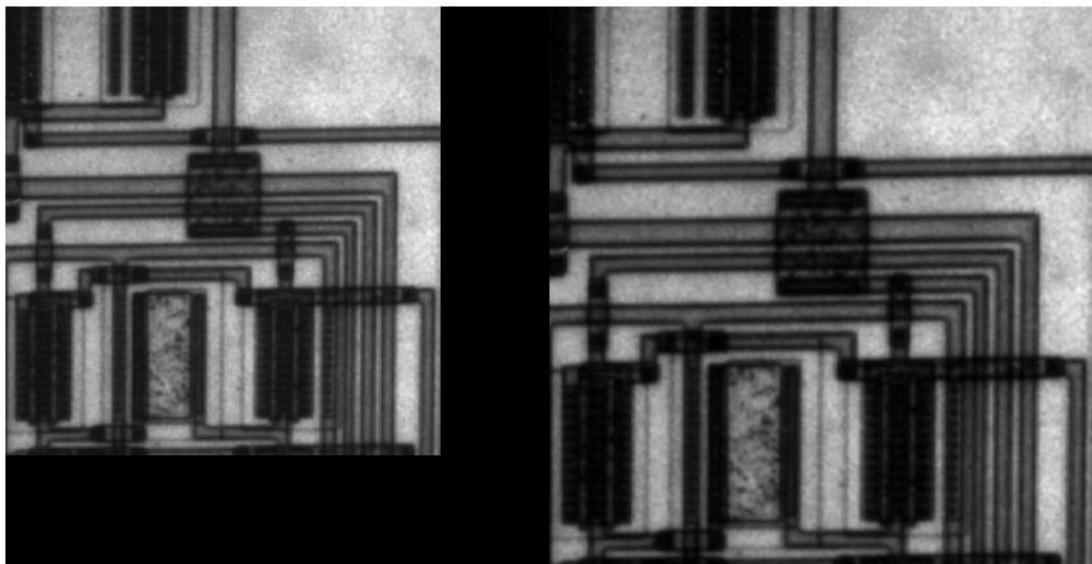
```
I = imread('circuit.tif');
```

使用 `imresize` 函数调整该图像的大小。在此示例中，请指定放大倍率系数。要放大图像，请指定大于 1 的放大倍率系数。

```
J = imresize(I,1.25);
```

在放大版本的图像旁边显示原始图像。

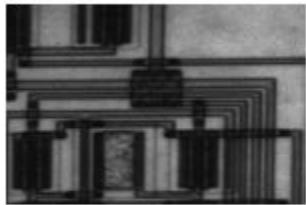
```
figure  
imshowpair(I,J,'montage')  
axis off
```



指定输出图像的大小

再次调整图像大小，这次指定输出图像的所需大小，而不是放大倍率值。向 `imresize` 传递一个包含输出图像中行数和列数的向量。如果指定的大小不能产生与输入图像相同的纵横比，输出图像将会失真。如果将向量中的元素之一指定为 `NaN`，`imresize` 会计算该维度的值以保持图像的纵横比。要执行多分辨率处理所需的大小调整，请使用 `impyramid`。

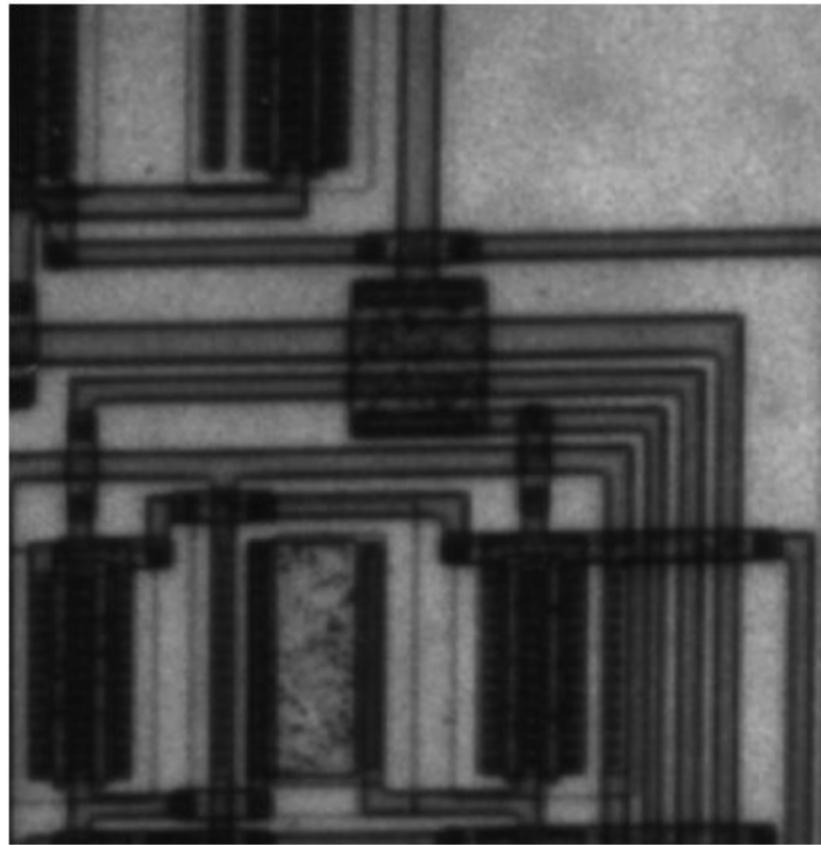
```
K = imresize(I,[100 150]);  
figure, imshow(K)
```



指定插值方法

再次调整图像大小，这次指定插值方法。放大图像时，输出图像包含的像素比原始图像多。`imresize` 使用插值来确定这些像素的值，计算像素位置附近的一些像素的加权平均值。`imresize` 根据每个像素离该点的距离来确定权重。默认情况下，`imresize` 使用双三次插值，但您可以指定其他插值方法或插值核。有关完整列表，请参阅 `imresize` 参考页。您也可以指定自己的自定义插值核。此示例使用双线性插值。

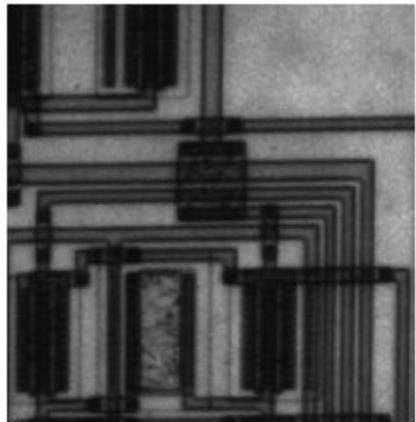
```
L = imresize(I,1.5,'bilinear');  
figure, imshow(L)
```



缩小图像时防止锯齿

再次调整图像大小，这次缩小图像。缩小图像时，会丢失一些原始像素，因为输出图像中的像素要少一些。这可能会引入伪影，例如锯齿。由于大小减小而导致的锯齿在输出图像中通常显示为阶梯图案（尤其是在高对比度图像中）或波纹（波纹效应）图案。默认情况下，对于除最近邻点法以外的所有插值方法，`imresize` 都会使用抗锯齿功能来限制输出图像的锯齿影响。要关闭抗锯齿功能，请指定 'Antialiasing' 参数并将值设置为 `false`。即使打开了抗锯齿功能，调整大小也会引入伪影，因为缩小图像大小时始终会丢失信息。

```
M = imresize(I,.75,'Antialiasing',false);
figure, imshow(M)
```



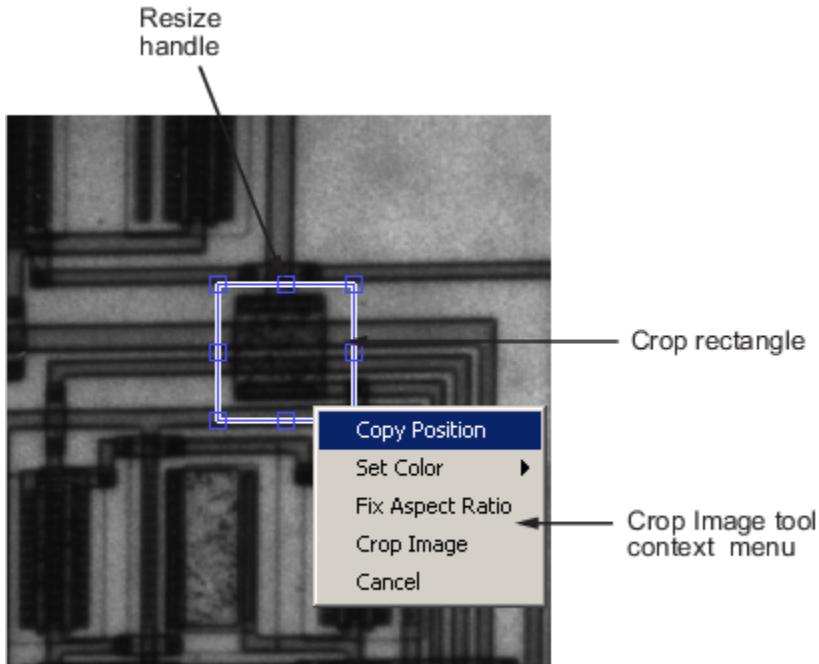
裁剪图像

注意 您也可以使用图像工具以交互方式裁剪图像，请参阅“Crop Image Using Image Viewer App”。

要从图像中提取一个矩形部分，请使用 **imcrop** 函数。使用 **imcrop**，您可以使用鼠标以交互方式指定裁剪区域，或以编程方式通过指定裁剪区域的大小和位置指定裁剪区域。

以下示例介绍了一种交互式语法。该示例将图像读入 MATLAB 工作区并调用 **imcrop**，将图像指定为参数。**imcrop** 在图窗窗口中显示该图像，并等待您在图像上拖出裁剪矩形。当您将指针移至图像上时，指针的形状会变为十字准线 。点击并拖动指针以指定裁剪矩形的大小和位置。您可以使用鼠标移动和调整裁剪矩形的大小。当您对裁剪矩形满意时，双击以执行裁剪操作，或在裁剪矩形内点击右键并从上下文菜单中选择裁剪图像。**imcrop** 在 J 中返回裁剪图像。

```
I = imread('circuit.tif');
J = imcrop(I);
```



也可以在调用 **imcrop** 时将裁剪矩形的大小和位置指定为参数。将裁剪矩形指定为四元素位置向量 [**xmin** **ymin** **width** **height**]。

此示例调用 **imcrop** 并指定要裁剪的图像 I 和裁剪矩形。**imcrop** 在 J 中返回裁剪图像。

```
I = imread('circuit.tif');
J = imcrop(I,[60 40 100 90]);
```

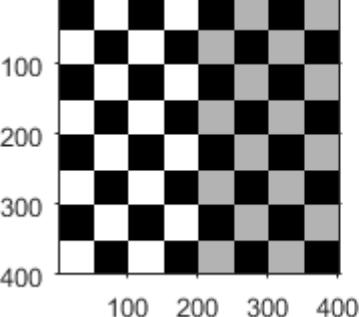
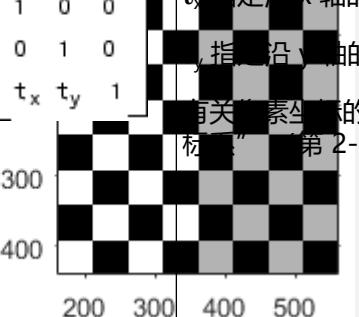
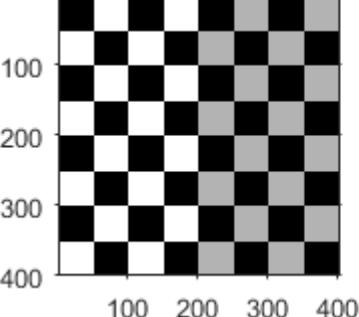
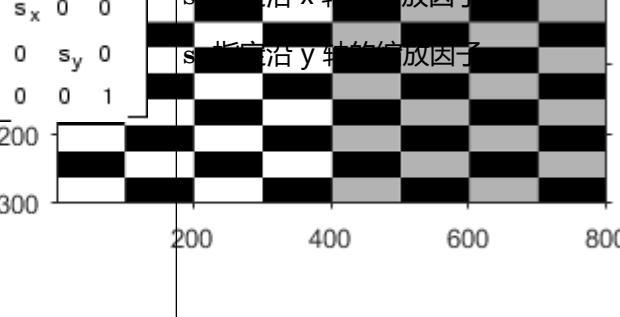
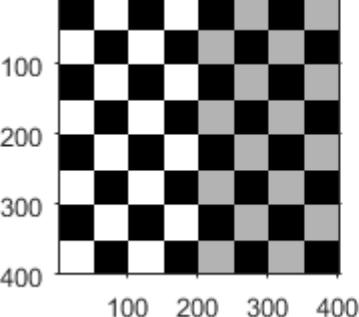
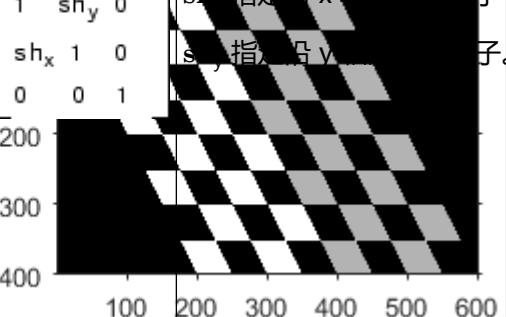
几何变换的矩阵表示

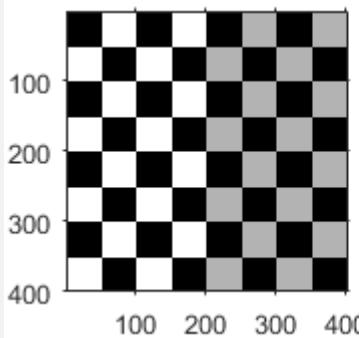
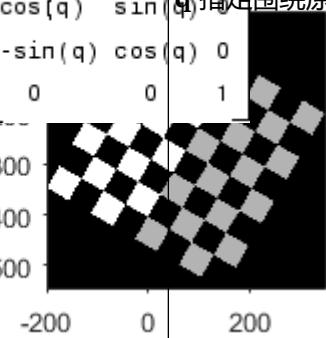
您可以使用几何变换矩阵来执行图像的全局变换。首先，定义变换矩阵，并使用它创建几何变换对象。然后，通过使用几何变换对象调用 `imwarp` 对图像应用全局变换。有关示例，请参阅“Perform Simple 2-D Translation Transformation”。

二维仿射变换

下表列出了二维仿射变换以及用于定义它们的变换矩阵。对于二维仿射变换，最后一列必须包含 $[0 \ 0 \ 1]$ 同构坐标。

使用二维变换矩阵的任意组合来创建一个 `affine2d` 几何变换对象。使用二维平移和旋转矩阵的组合来创建一个 `rigid2d` 几何变换对象。

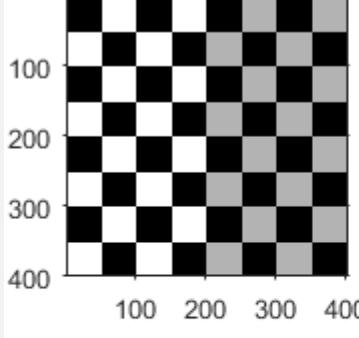
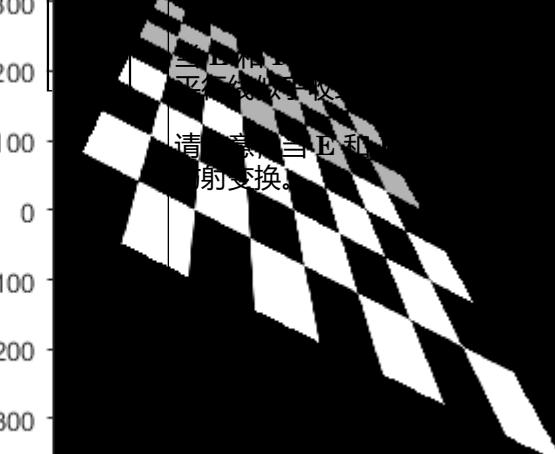
二维仿射变换	示例（原始图像和变换后的图像）	变换矩阵
平移		$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$ <p>t_x 指定沿 x 轴的位移。 t_y 指定沿 y 轴的位移。 有关像素坐标的详细信息，请参阅“图像坐标系”（第 2-3 页）。</p> 
缩放		$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ <p>s_x 指定沿 x 轴的缩放因子。 s_y 指定沿 y 轴的缩放因子。</p> 
剪切		$\begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ <p>sh_x 指定沿 x 轴的剪切因子。 sh_y 指定沿 y 轴的剪切因子。</p> 

二维仿射变换	示例 (原始图像和变换后的图像)	变换矩阵
旋转		$\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$ <p>θ 指定围绕原点的旋转角度。</p> 

二维投影变换

投影变换使图像平面倾斜。平行线可以向一个消失点收敛，形成深度的外观。

变换矩阵是一个 3×3 矩阵。与仿射变换不同，变换矩阵的最后一列没有限制。

二维投影变换	示例	变换矩阵
倾斜		$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ <p>原点，因此 平行线收敛于 消失点。</p>  <p>请参阅当 E 和 F 不为零时的仿射变换。</p>

投影变换经常用于配准未对齐的图像。如果您要对齐两个图像，请先使用 `cpselect` 选择控制点对。然后，使用 `fitgeotrans` 并将 `transformationType` 设置为 '`projective`' 来对成对控制点进行投影变换矩阵拟合。这将自动创建 `projective2d` 几何变换对象。变换矩阵作为属性存储在 `projective2d` 对象中。然后，可以使用 `imwarp` 将变换应用于其他图像。

创建复合二维仿射变换

您可以使用矩阵乘法将多个变换合并成一个矩阵。矩阵乘法的顺序很重要。

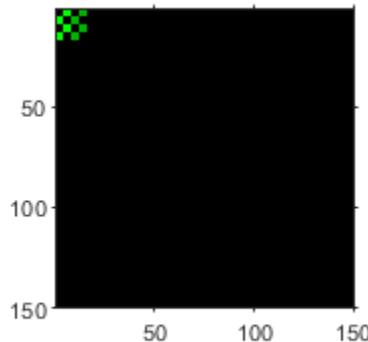
此示例说明如何创建二维平移和旋转变换组合。

创建一个将发生变换的棋盘图像。同时为图像创建一个空间参照对象。

```
cb = checkerboard(4,2);
cb_ref = imref2d(size(cb));
```

要展示图像的空间位置，请创建一个平面背景图像。将棋盘叠加到背景上，使用绿色突出显示棋盘位置。

```
background = zeros(150);
imshowpair(cb,cb_ref,background,imref2d(size(background)))
```



创建一个变换矩阵，并将其存储为 `affine2d` 几何变换对象。此平移将使图像水平移动 100 个像素。

```
T = [1 0 0;0 1 0;100 0 1];
tform_t = affine2d(T);
```

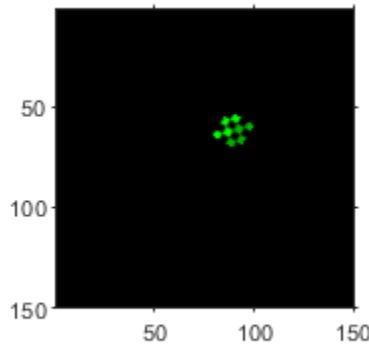
创建一个旋转矩阵，并将其存储为 `affine2d` 几何变换对象。此旋转将使图关于原点顺时针旋转 30 度。

```
R = [cosd(30) sind(30) 0;-sind(30) cosd(30) 0;0 0 1];
tform_r = affine2d(R);
```

平移后旋转

首先执行平移，然后执行旋转。在变换矩阵的乘法运算中，平移矩阵 `T` 位于左侧，旋转矩阵 `R` 位于右侧。

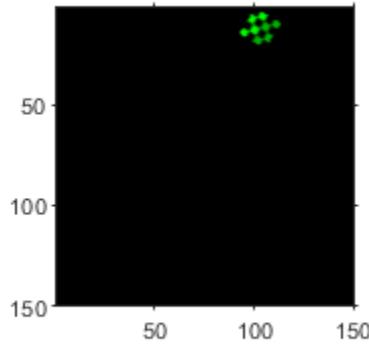
```
TR = T*R;
tform_tr = affine2d(TR);
[out,out_ref] = imwarp(cb,cb_ref,tform_tr);
imshowpair(out,out_ref,background,imref2d(size(background)))
```



旋转后平移

颠倒变换的顺序：先执行旋转，再执行平移。在变换矩阵的乘法运算中，旋转矩阵 R 位于左侧，平移矩阵 T 位于右侧。

```
RT = R*T;
tform_rt = affine2d(RT);
[out,out_ref] = imwarp(cb,cb_ref,tform_rt);
imshowpair(out,out_ref,background,imref2d(size(background)))
```



请注意变换后的图像的空间位置与先平移后旋转情形的不同之处。

三维仿射变换

下表列出了三维仿射变换以及用于定义它们的变换矩阵。请注意，在三维情况下具有多个矩阵，具体取决于您要如何旋转或剪切图像。最后一列必须包含 $[0 \ 0 \ 0 \ 1]$ 。

使用三维变换矩阵的任意组合来创建一个 `affine3d` 几何变换对象。使用三维平移和旋转矩阵的组合来创建一个 `rigid3d` 几何变换对象。

三维仿射变换	变换矩阵		
平移	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$		
缩放	$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		
剪切	x,y 剪切: $\begin{aligned} x' &= x + az \\ y' &= y + bz \\ z' &= z \end{aligned}$ $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	x,z 剪切: $\begin{aligned} x' &= x + ay \\ y' &= y \\ z' &= z + cy \end{aligned}$ $\begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & c & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	y, z 剪切: $\begin{aligned} x' &= x \\ y' &= y + bx \\ z' &= z + cx \end{aligned}$ $\begin{bmatrix} 1 & b & c & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
旋转	关于 x 轴: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & \sin(a) & 0 \\ 0 & -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	关于 y 轴: $\begin{bmatrix} \cos(a) & 0 & -\sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	关于 z 轴: $\begin{bmatrix} \cos(a) & \sin(a) & 0 & 0 \\ -\sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

对于 N 维仿射变换，最后一列必须包含 [zeros(N,1); 1]。imwarp 不支持三维以上的变换。

另请参阅

imwarp | fitgeotrans | affine2d | affine3d | rigid2d | rigid3d | projective2d

相关示例

- “Perform Simple 2-D Translation Transformation”

详细信息

- “2-D and 3-D Geometric Transformation Process Overview”

探查三维 MRI 数据集的切片

此示例说明如何使用 `imtransform` 和 `tformarray` 函数从三维 MRI 数据集提取切片来探查数据体。

步骤 1：加载和查看水平 MRI

此示例使用 MATLAB® 附带的用在 `montage` 和 `immovie` 的帮助示例中的 MRI 数据集。加载 `mri.mat` 会向工作区添加两个变量：`D` ($128 \times 128 \times 1 \times 27$, `uint8` 类) 和灰度颜色图 `map` (89×3 , `double` 类)。

`D` 包含人体头盖骨 MRI 扫描数据的 27 个 128×128 水平切片。`D` 中的值的范围是从 0 到 88，因此需要颜色图来生成实用的可视图形。`D` 的维度与 `montage` 兼容。前两个维度是空间维度。第三个维度是颜色维度，大小为 1，因为它是颜色图的索引。对于 RGB 图像序列，`size(D,3)` 为 3。第四个维度是时序维度（和任何图像序列一样），但在本例中，它也是空间维度。因此，`D` 中有三个空间维度，我们可以使用 `imtransform` 或 `tformarray` 将水平切片转换为矢状切片（显示头部侧面的视图）或冠状（正面）切片（显示头部正面或背面的视图）。

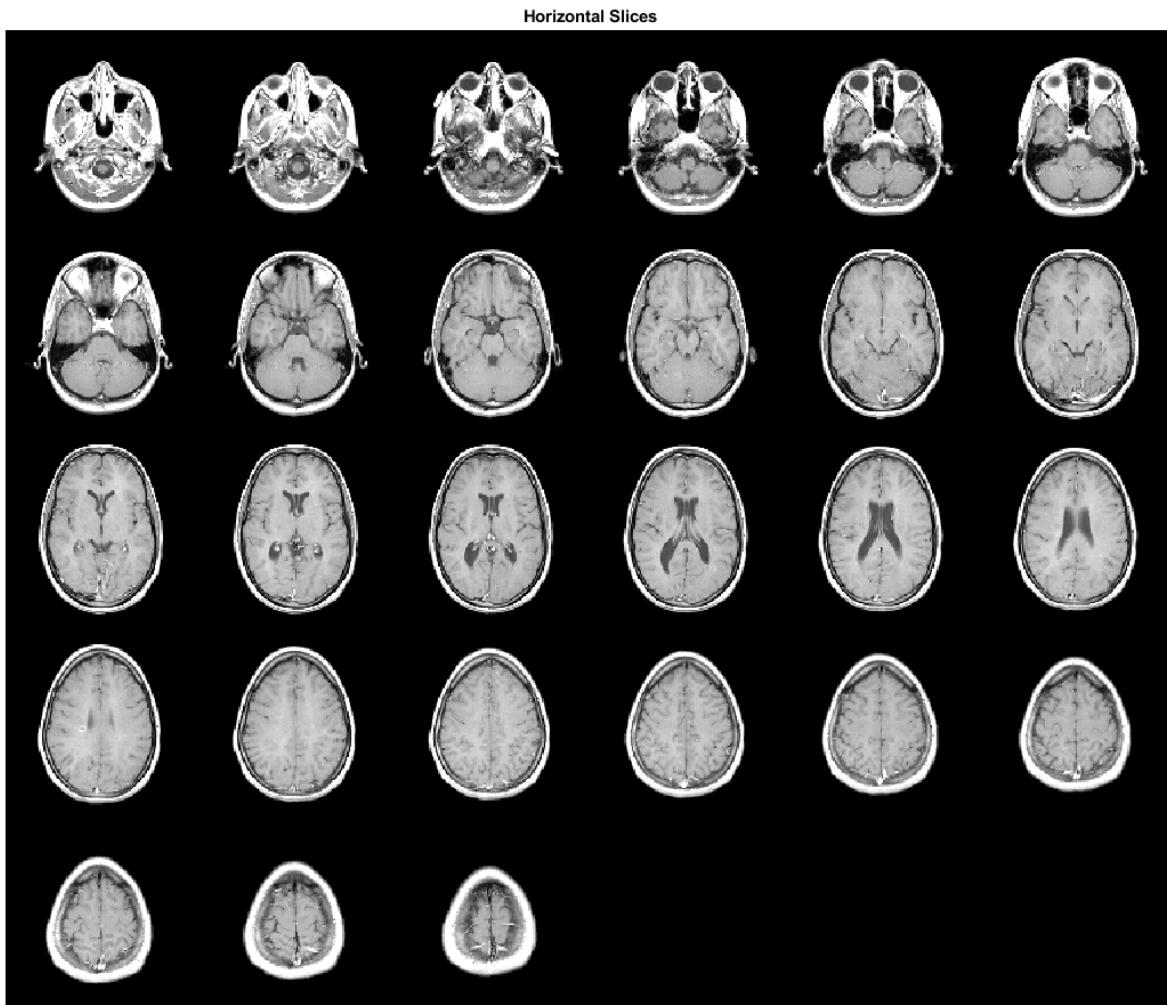
`D` 的空间维度排序如下：

- 维度 1：头部从前到后（从吻突/前侧到尾椎/后侧）
- 维度 2：头部从左到右
- 维度 4：头部从底到顶（从下到上）。

一个重要因素是三个维度上的采样间隔不同：沿垂直维度 (4) 的采样间隔比沿水平维度的采样间隔宽 2.5 倍。

加载 MRI 数据集，将 27 个水平切片以蒙太奇形式进行查看。

```
load mri;
montage(D,map)
title('Horizontal Slices');
```



步骤 2：使用 IMTRANSFORM 从水平切片中提取矢状切片

我们可以通过提取 **D** 的子集，并对其进行变换以处理 **D** 维度的不同采样间隔和空间方向，从而构建 MRI 数据的正中矢状切片。

以下语句提取正中矢状切片所需的所有数据。

```
M1 = D(:,64,:,:); size(M1)
```

```
ans = 1×4
```

```
128    1    1    27
```

但是，我们无法以图像形式查看 **M1**，因为它是 $128 \times 1 \times 1 \times 27$ 。**reshape**（或 **squeeze**）可以将 **M1** 转换为可使用 **imshow** 查看的 128×27 图像。

```
M2 = reshape(M1,[128 27]); size(M2)
```

```
ans = 1×2
```

```
128 27
```

```
figure, imshow(M2,map);
title('Sagittal - Raw Data');
```

Sagittal - Raw Data



M2 中的维度的排序如下：

- 维度 1：头部从前到后（从吻突到尾椎）
- 维度 2：头部从底到顶（从下到上）。

我们可以通过变换 M2，更改其方向并将垂直（下-上）维度的采样增加为 2.5 倍，使采样间隔在所有三个空间维度上相等，从而获得更令人满意的视图。我们可以从转置开始分多个步骤执行此过程，但以下仿射变换可以实现单步变换并且更节省内存。

```
T0 = maketform('affine',[0 -2.5; 1 0; 0 0]);
```

传递给 maketform 的矩阵的上 2×2 块 ($[0 \text{ } -2.5; 1 \text{ } 0]$) 结合使用了旋转和缩放。变换后可得到以下维度：

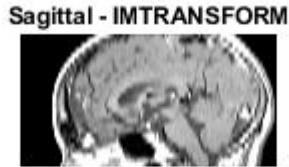
- 维度 1：头部从顶到底（从上到下）。
- 维度 2：头部从前到后（从吻突到尾椎）

以下调用

```
imtransform(M2,T0,'cubic')
```

可将 T 应用于 M2，并足以在沿从顶到底方向插值时提供良好的分辨率。不过，从前到后方向无需执行三次插值，因为沿（输出）维度 2 不会进行重采样。因此，我们在此维度中指定最近邻点重采样，可获得更高的效率和相同的结果。

```
R2 = makeresampler({'cubic','nearest'},'fill');
M3 = imtransform(M2,T0,R2);
figure, imshow(M3,map);
title('Sagittal - IMTRANSFORM')
```



步骤 3：使用 TFORMARRAY 从水平切片中提取矢状切片

在此步骤中，我们获得的结果与步骤 2 相同，只是使用了 `tformarray` 通过一次运算将三个空间维度转换为二个空间维度。步骤 2 确实也是从开始的具有三个空间维度的数组转换为最后的具有两个空间维度的数组，但中间生成了二维图像（`M1` 和 `M2`），以便于调用 `imtransform` 来创建 `M3`。如果我们使用 `tformarray` 代替 `imtransform`，则这些中间图像是不必要的。`imtransform` 对于二维到二维的变换非常方便，但 `tformarray` 支持从 `N` 维到 `M` 维的变换，其中 `M` 不必等于 `N`。

通过其 `TDIMS_A` 参数，`tformarray` 允许我们定义输入数组的置换。由于我们要创建具有以下维度的图像：

- 维度 1：从上到下（颠倒的原始维度 4）
- 维度 2：尾椎到吻突（原始维度 1）

并通过原始维度 2 仅提取单个矢状平面，可指定 `tdims_a = [4 1 2]`。我们通过变换组合来创建 `tform`，首先进行二维仿射变换 `T1`，将维度 1（新）缩放 -2.5 倍，并平移 68.5 以使数组坐标为正值。组合的第二部分是自定义变换 `T2`，通过很简单的 `INVERSE_FCN` 提取第 64 个矢状平面。

```
T1 = maketform('affine',[-2.5 0; 0 1; 68.5 0]);
inverseFcn = @(X,t) [X repmat(t.tdata,[size(X,1) 1])];
T2 = maketform('custom',3,2,[],inverseFcn,64);
Tc = maketform('composite',T1,T2);
```

请注意，`T2` 和 `Tc` 将三维输入转换为二维输入。

我们使用与以前相同的方法进行重采样，但包括第三个维度。

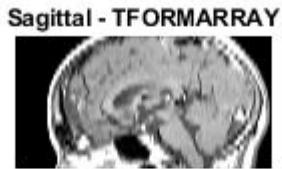
```
R3 = makeresampler({'cubic','nearest','nearest'},'fill');
```

`tformarray` 通过一个步骤将 `D` 的三个空间维度转换为二维输出。我们的输出图像是 66×128 ，在垂直（下-上）方向上最初的 27 个平面扩展到 66 个。

```
M4 = tformarray(D,Tc,R3,[4 1 2],[1 2],[66 128],[],0);
```

结果与先前的 `imtransform` 的输出相同。

```
figure, imshow(M4,map);
title('Sagittal - TFORMARRAY');
```



步骤 4：创建和显示矢状切片

我们创建一个四维数组（第三个维度是颜色维度），它可用于生成一个图像序列，该图像序列从第 30 个平面开始，从左到右每隔一个平面取一帧，总共有 35 个帧。变换后的数组具有以下维度：

- 维度 1：从顶到底（从上到下）
- 维度 2：从前到后（从吻部到尾椎）
- 维度 4：从左到右。

与上一个步骤一样，我们使用 `TDIMS_A = [4 1 2]` 置换输入数组，然后再翻转和重新缩放/重采样垂直维度。我们的仿射变换与上面的 `T1` 相同，不同之处在于添加了第三个维度，其中 (3,3) 元素为 0.5、(4,3) 元素为 -14，以将 30、32、...98 映射到 1、2、...、35。这可将 35 个帧居中置于正中矢状切片上。

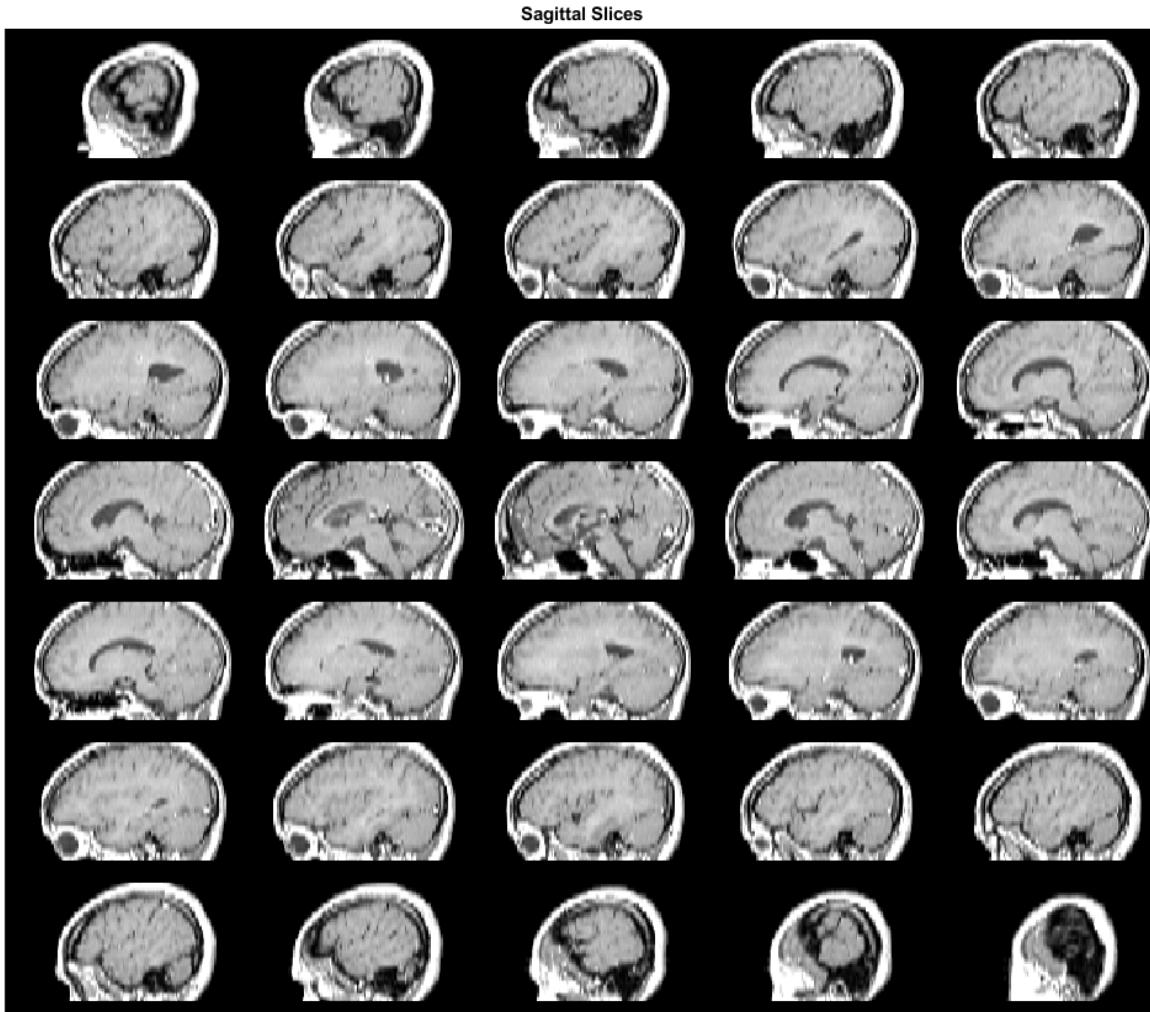
```
T3 = maketform('affine',[-2.5 0 0; 0 1 0; 0 0 0.5; 68.5 0 -14]);
```

在我们调用 `tformarray` 时，`TSIZE_B = [66 128 35]` 现在将 35 个帧包含在从左到右的第 4 个维度（即第三个变换维度）中。重采样器保持不变。

```
S = tformarray(D,T3,R3,[4 1 2],[1 2 4],[66 128 35],[],0);
```

以蒙太奇方式查看矢状切片（稍微填充数组以分隔蒙太奇的元素）。

```
S2 = padarray(S,[6 0 0 0],0,'both');  
figure, montage(S2,map)  
title('Sagittal Slices');
```



步骤 5：创建和显示冠状切片

构造冠状切片与构造矢状切片几乎完全相同。我们将 TDIMS_A 从 [4 1 2] 更改为 [4 2 1]。创建一个包含 45 个帧的系列，从第 8 个平面开始，从后向前移动，每两帧跳过一帧。输出数组的维度排序如下：

- 维度 1：从顶到底（从上到下）
- 维度 2：从左到右
- 维度 4：从后到前（从尾椎到吻突）。

```
T4 = maketform('affine',[ -2.5 0 0; 0 1 0; 0 0 -0.5; 68.5 0 61]);
```

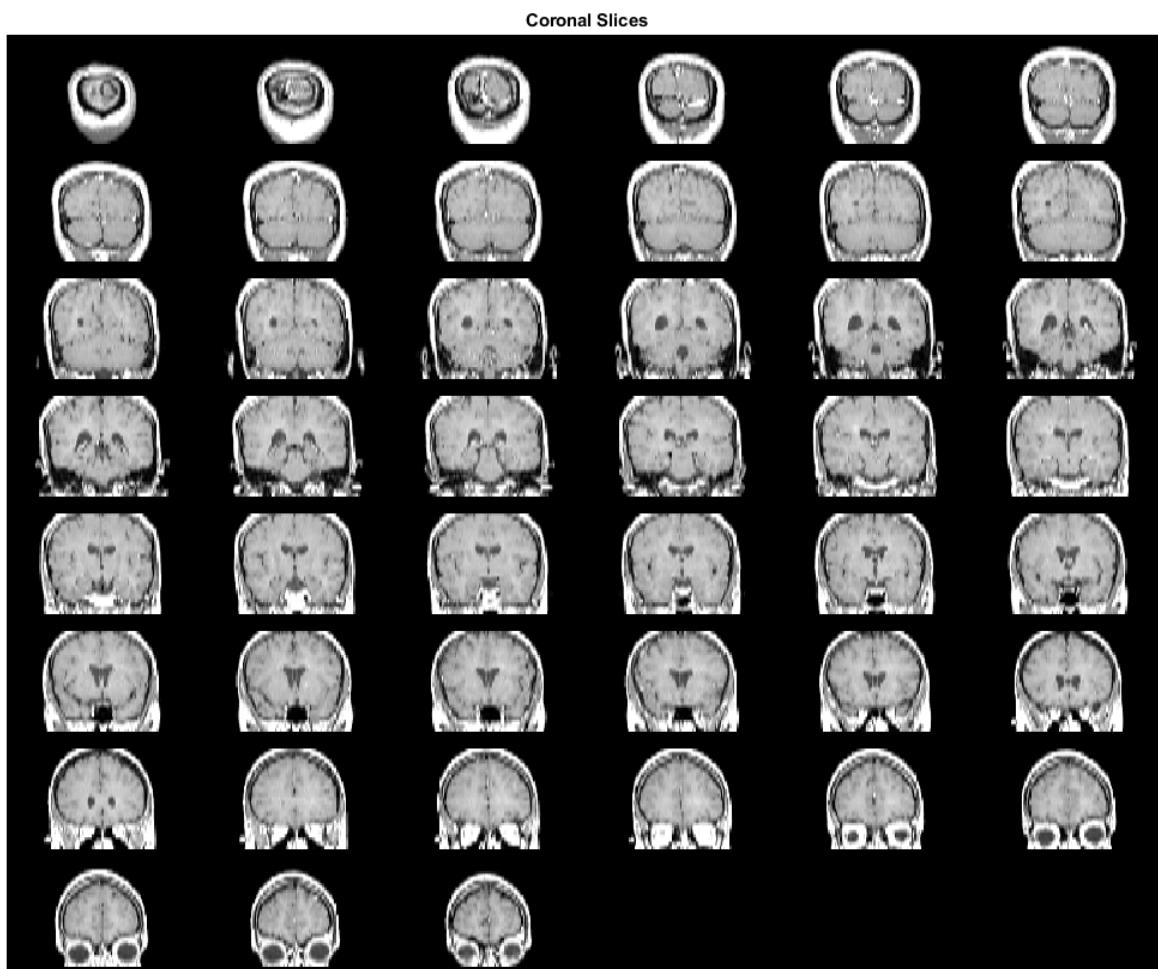
在 tformarray 调用中，TSIZE_B = [66 128 48] 分别指定垂直、从一侧到另一侧和从前到后的维度。重采样器保持不变。

```
C = tformarray(D,T4,R3,[4 2 1],[1 2 4],[66 128 45],[],0);
```

请注意，步骤 3、4 和 5 中的所有数组置换和翻转均作为 tformarray 运算的一部分来处理。

以蒙太奇方式查看冠状切片（稍微填充数组以分离蒙太奇的元素）。

```
C2 = padarray(C,[6 0 0 0],0,'both');  
figure, montage(C2,map)  
title('Coronal Slices');
```



图像配准

本章介绍 Image Processing Toolbox 软件的图像配准功能。图像配准是对齐同一场景的两个或多个图像的过程。图像配准通常用作其他图像处理应用中的初步步骤。

- “基于强度的图像配准创建优化器和指标” (第 7-2 页)
- “控制点选择过程” (第 7-3 页)

为基于强度的图像配准创建优化器和指标

您可以将图像相似性指标和优化器方法传递给 `imregister`。图像相似性指标采用两个图像，并返回说明图像相似程度的标量值。您传递给 `imregister` 的优化器定义最小化或最大化相似性指标的方法。

`imregister` 支持两个相似性指标：

- Mattes 互信息
- 均方误差

此外，`imregister` 支持两种优化图像指标的方法：

- 1+1 演化
- 规则时间步梯度下降

您可以将指标和优化器的任意组合传递给 `imregister`，但有些对组更适合某些图像类。请参考下表来帮助您选择合适的起点。

捕获方案	指标	优化器
单模态	MeanSquares	RegularStepGradientDescent
多模态	MattesMutualInformation	OnePlusOneEvolutionary

使用 `imregconfig` 一步创建用于捕获方案的默认指标和优化器。例如，以下命令返回适合配准单模态图像的优化器和指标对象。

```
[optimizer,metric] = imregconfig('monomodal');
```

您也可以单独创建对象。这使您能够创建替代组合来解决特定的配准问题。以下代码可创建相同的单模态优化器和指标组合。

```
optimizer = registration.optimizer.RegularStepGradientDescent();
metric = registration.metric.MeanSquares();
```

从基于优化的图像配准中获得良好的结果可能需要修改优化器或指标设置。有关如何使用 `imregister` 修 改和使用指标和优化器的示例，请参阅“Register Multimodal MRI Images”。

另请参阅

`imregister` | `imregconfig`

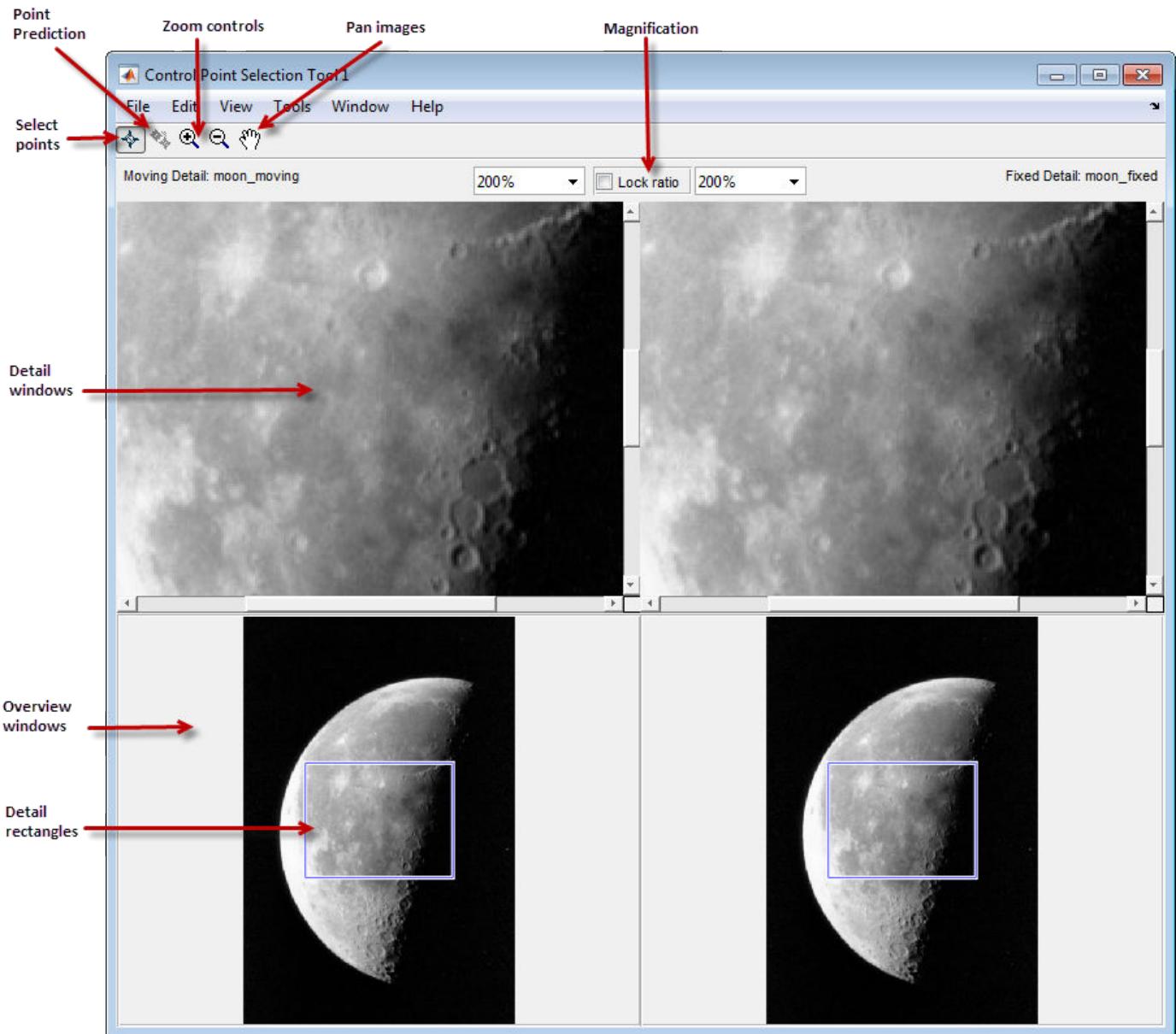
控制点选择过程

要以交互方式指定一对图像中的控制点，请使用控制点选择工具 `cpselect`。该工具在参考图像（称为固定图像）旁边显示要配准的图像（称为运动图像）。

指定控制点过程包含四个步骤：

- 1 启动工具，指定运动图像和固定图像。
- 2 使用导航辅助功能浏览图像，寻找在两个图像中都能找到的视觉元素。`cpselect` 提供许多在图像中进行导航的方法。您可以通过平移和缩放来更详细地查看图像区域。
- 3 在运动图像和固定图像中指定匹配的控制点对组。
- 4 将控制点保存到工作区。

下图显示了首次启动该工具时的默认外观。



另请参阅
`cpselect`

详细信息

- “Start the Control Point Selection Tool”
- “Control Point Registration”

图像数据线性滤波器的设计与实现

Image Processing Toolbox 软件提供了许多函数用来设计和实现图像数据的二维线性滤波器。本章说明这些函数以及如何有效地使用它们。

- “什么是空间域中的图像滤波？”（第 8-2 页）
- “imfilter 边界填充选项”（第 8-5 页）
- “什么是引导图像滤波？”（第 8-8 页）

什么是空间域中的图像滤波？

滤波是一种用于修正或增强图像的技术。例如，您可以对图像进行滤波以强调某些特征或删除其他特征。用滤波实现的图像处理运算包括平滑、锐化和边缘增强。

滤波是一种邻域运算，它通过对对应输入像素邻域中的像素值应用某种算法来确定输出图像中任何给定像素的值。像素的领域是一组像素，由它们相对于该像素的位置来确定。（有关邻域运算的一般性讨论，请参阅“Neighborhood or Block Processing: An Overview”。）线性滤波是一种滤波，其中输出像素的值是输入像素邻域中像素值的线性组合。

卷积

图像的线性滤波是通过称为卷积的运算完成的。卷积是一种邻域运算，其中每个输出像素是相邻输入像素的加权和。权重矩阵称为卷积核，也称为滤波器。卷积核是旋转了 180 度的相关性核。

例如，假设图像是

$$A = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

相关性核是

$$h = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

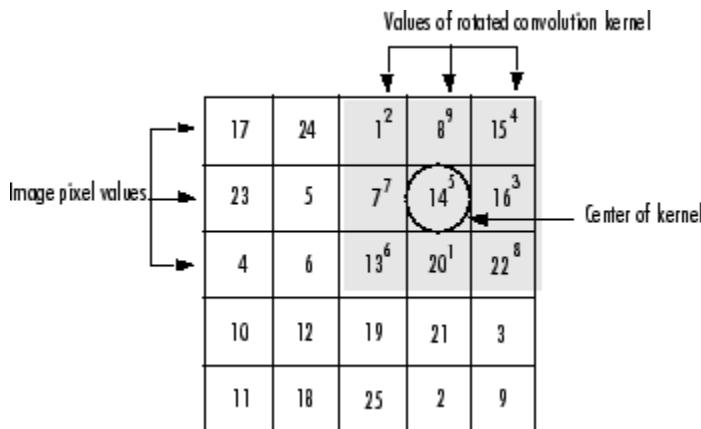
您将使用以下步骤计算位置 (2,4) 处的输出像素：

- 1 将相关性核关于其中心元素旋转 180 度以创建卷积核。
- 2 滑动卷积核的中心元素，将其叠加在 A 的 (2,4) 元素之上。
- 3 将旋转后的卷积核中的每个权重乘以下方的 A 的像素。
- 4 对步骤 3 中得到的各个乘积求和。

因此，(2,4) 输出像素是

$$1 \cdot 2 + 8 \cdot 9 + 15 \cdot 4 + 7 \cdot 7 + 14 \cdot 5 + 16 \cdot 3 + 13 \cdot 6 + 20 \cdot 1 + 22 \cdot 8 = 575$$

如下图所示。



计算卷积的 (2,4) 输出

相关性

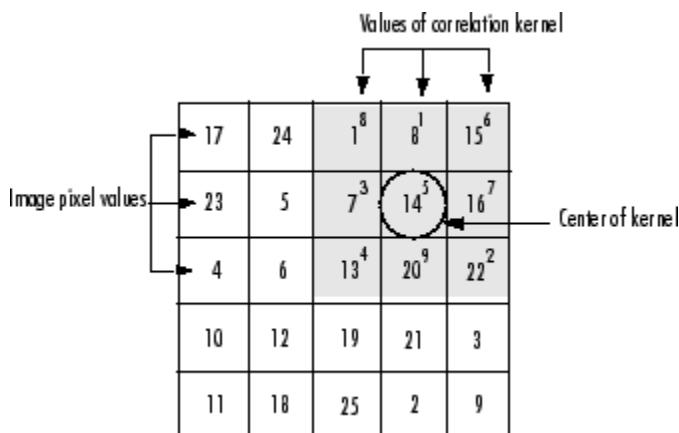
称为相关性的运算与卷积密切相关。在相关性中，输出像素的值也计算为相邻像素的加权和。不同之处在于权重矩阵（在本例中称为相关性核）在计算过程中不旋转。Image Processing Toolbox 滤波器设计函数返回相关性核。

下图显示如何计算 A 的相关性的 (2,4) 输出像素，假设 h 是相关性核而不是卷积核，使用以下步骤：

- 1 滑动相关性核的中心元素，将其叠加到 A 的 (2,4) 元素的上方。
- 2 将相关性核中的每个权重乘以下方 A 的像素。
- 3 对各乘积求和。

相关性的 (2,4) 输出像素是

$$1 \cdot 8 + 8 \cdot 1 + 15 \cdot 6 + 7 \cdot 3 + 14 \cdot 5 + 16 \cdot 7 + 13 \cdot 4 + 20 \cdot 9 + 22 \cdot 2 = 585$$



计算相关性的 (2,4) 输出

另请参阅

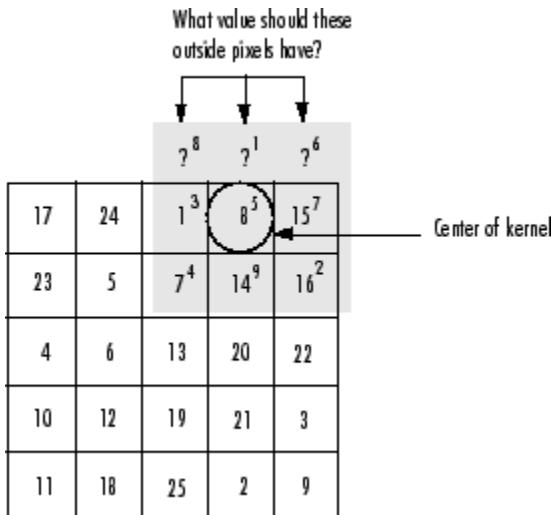
[imfilter](#) | [conv2](#) | [convn](#)

相关示例

- “Design Linear Filters in the Frequency Domain”
- “去除噪声” (第 11-41 页)

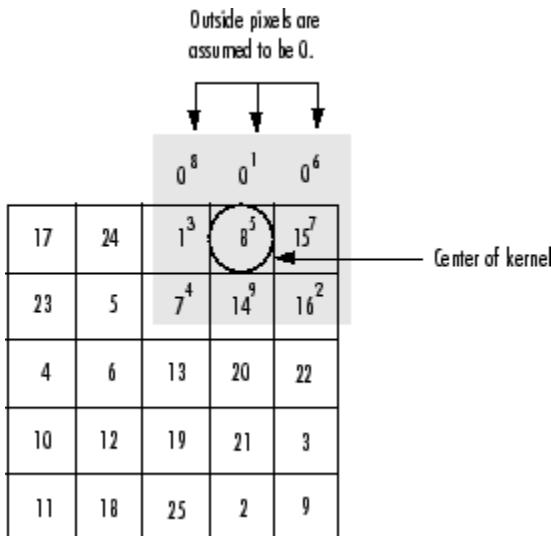
imfilter 边界填充选项

计算图像边界的输出像素时，卷积或相关性核的一部分通常不在图像边缘上，如下图所示。



当核的值落在图像之外时

imfilter 函数通常通过假设图像边缘之外的像素为 0 来填充这些像素。这称为零填充，如下图所示。



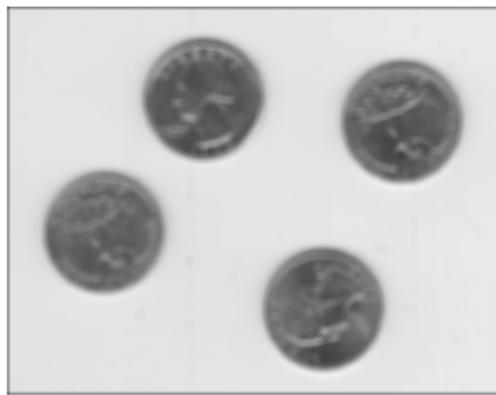
外部像素的零填充

对图像进行滤波时，零填充会导致图像边缘周围出现深色条带，如以下示例所示。

```
I = imread('eight.tif');
h = ones(5,5) / 25;
I2 = imfilter(I,h);
imshow(I), title('Original Image');
figure, imshow(I2), title('Filtered Image with Black Border')
```

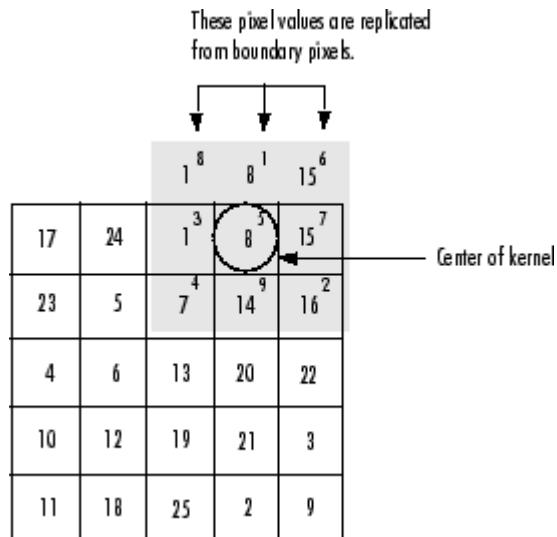


Original Image



Filtered Image with Black Border

为了消除图像边缘周围零填充所带来的伪影，`imfilter` 提供一种称为边界复制的替代边界填充方法。在边界复制中，图像外部任何像素的值都是通过复制最近边界像素的值来确定的。如下图所示。



复制的边界像素

要使用边界复制进行滤波，请将附加可选参数 '`'replicate'`' 传递给 `imfilter`。

```
I3 = imfilter(I,h,'replicate');
figure, imshow(I3);
title('Filtered Image with Border Replication')
```



Filtered Image with Border Replication

imfilter 函数支持 'circular' 和 'symmetric' 等其他边界填充选项。有关详细信息，请参阅 imfilter 的参考页。

另请参阅

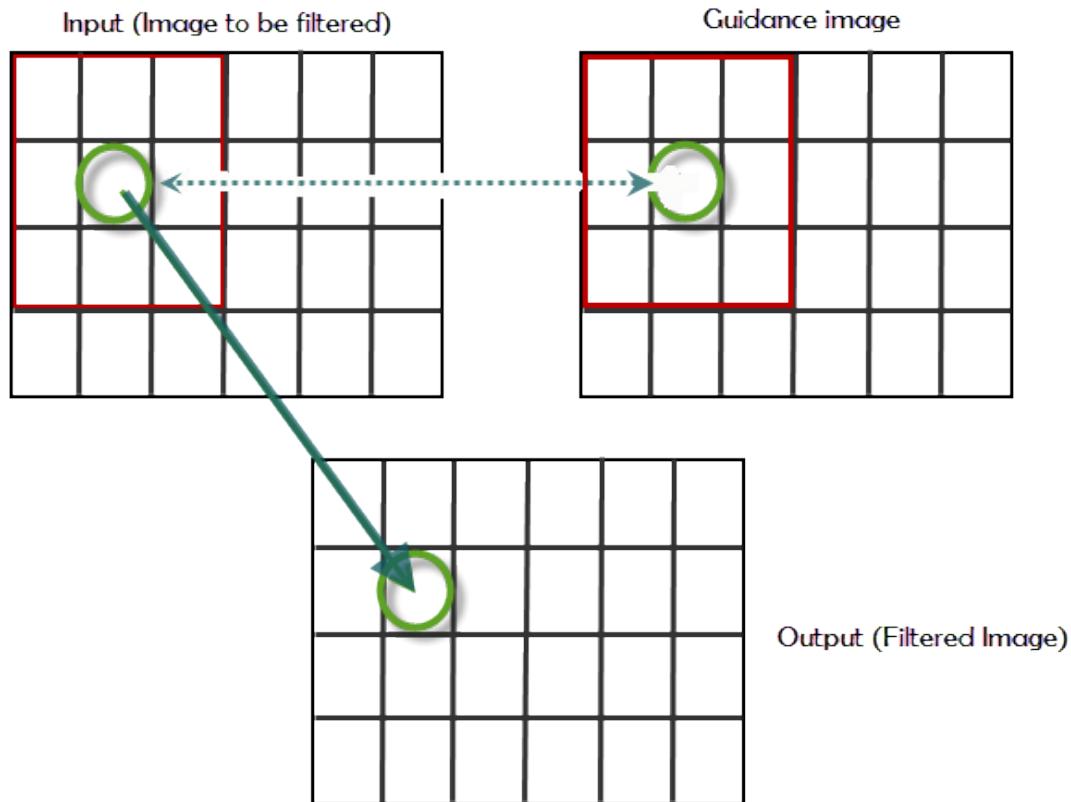
imfilter

相关示例

- “Design Linear Filters in the Frequency Domain”

什么是引导图像滤波？

`imguidedfilter` 函数对图像执行边缘保持平滑处理，使用第二个图像（称为引导图像）的内容来影响滤波。引导图像可以是图像本身、图像的不同版本或完全不同的图像。与其他滤波运算一样，引导图像滤波是一种邻域运算，但它在计算输出像素的值时，会考虑引导图像中对应空间邻域中某个区域的统计量。



如果引导图像与要滤波的图像相同，则结构也相同 - 原始图像中的边缘与引导图像中的边缘相同。如果引导图像不同，则引导图像中的结构将影响滤波后的图像，实际上是将这些结构压印在原始图像上。这种效应称为结构转移。

另请参阅
`imguidedfilter`

相关示例

- “Perform Flash/No-flash Denoising with Guided Filter”

变换

通常，图像的数学表示是两个空间变量的函数： $f(x,y)$ 。特定位置 (x,y) 上的函数值表示图像在该点的强度。这称为空间域。变换一词是图像的另一种数学表示。例如，傅里叶变换是将图像表示为不同幅值、频率和相位的复指数之和。这称为频域。变换有广泛的用途，包括卷积、增强、特征检测和压缩。

本章定义几种重要的变换，并展示它们在图像处理中的应用示例。

- “傅里叶变换”（第 9-2 页）
- “离散余弦变换”（第 9-12 页）
- “Hough 变换”（第 9-16 页）

傅里叶变换

本节内容

“傅里叶变换的定义” (第 9-2 页)

“离散傅里叶变换” (第 9-5 页)

“傅里叶变换的应用” (第 9-8 页)

傅里叶变换的定义

傅里叶变换是将图像表示为不同幅值、频率和相位的复指数之和。傅里叶变换在广泛的图像处理应用中起着至关重要的作用，包括增强、分析、还原和压缩。

如果 $f(m,n)$ 是两个离散空间变量 m 和 n 的函数，则 $f(m,n)$ 的二维傅里叶变换由如下关系定义：

$$F(\omega_1, \omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n) e^{-j\omega_1 m} e^{-j\omega_2 n}.$$

变量 ω_1 和 ω_2 是频率变量；其单位是弧度/采样。 $F(\omega_1, \omega_2)$ 通常称为 $f(m, n)$ 的频域表示。 $F(\omega_1, \omega_2)$ 是复数值函数，在 ω_1 和 ω_2 中均呈现周期性，期间为 2π 。由于具有周期性，通常只显示范围 $-\pi \leq \omega_1, \omega_2 \leq \pi$ 。请注意， $F(0,0)$ 是 $f(m, n)$ 的所有值的总和。因此， $F(0,0)$ 通常称为傅里叶变换的常量分量或 DC 分量。（DC 表示直流电；这是一个电气工程术语，指恒压电源，不同于电压呈正弦变化的电源。）

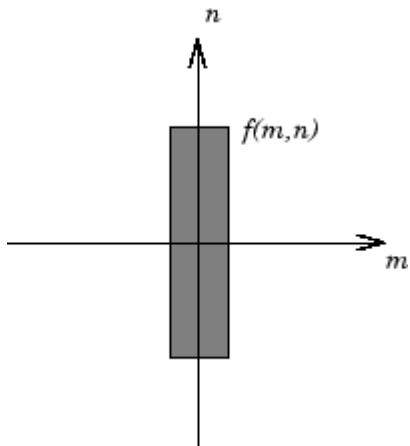
变换的逆运算是可应用于变换后的图像以生成原始图像的运算。二维傅里叶逆变换由下式给出：

$$f(m, n) = \frac{1}{4\pi^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} F(\omega_1, \omega_2) e^{j\omega_1 m} e^{j\omega_2 n} d\omega_1 d\omega_2.$$

粗略地说，此方程意味着 $f(m, n)$ 可以表示为无限个不同频率的复指数（正弦曲线）的总和。 $F(\omega_1, \omega_2)$ 给出了 (ω_1, ω_2) 频率的贡献的幅值和相位。

傅里叶变换的可视化

为说明傅里叶变换，我们以下面的函数 $f(m, n)$ 为例，它在矩形区域内等于 1，在其他位置等于 0。为了简化图， $f(m, n)$ 显示为连续函数，即使变量 m 和 n 是离散的。

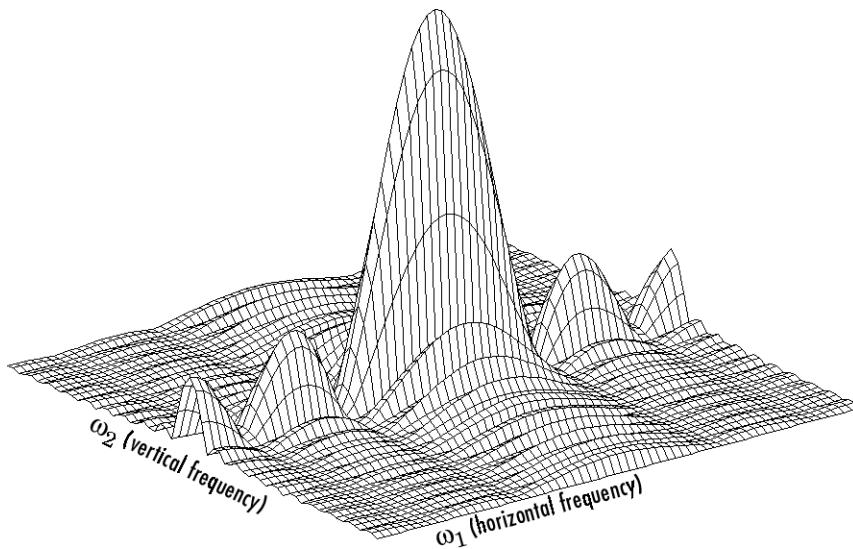


矩形函数

下图以网格图形式显示上图所示的矩形函数的傅里叶变换的幅值，即

$$|F(\omega_1, \omega_2)|,$$

。幅值的网格图是可视化傅里叶变换的常用方法。



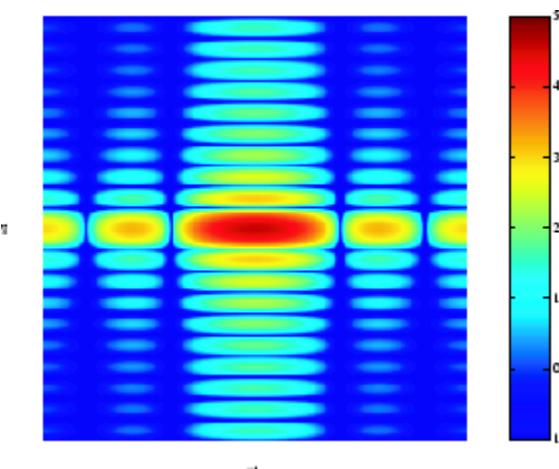
矩形函数的幅值图像

图中心的峰值是 $F(0,0)$ ，这是 $f(m,n)$ 中所有值的总和。该图还显示， $F(\omega_1, \omega_2)$ 在高水平频率下比在高垂直频率下具有更多能量。这反映出 $f(m,n)$ 的水平横截面是窄脉冲，而垂直横截面是宽脉冲。窄脉冲比宽脉冲具有更多高频成分。

另一种可视化傅里叶变换的常见方法是将

$$\log|F(\omega_1, \omega_2)|$$

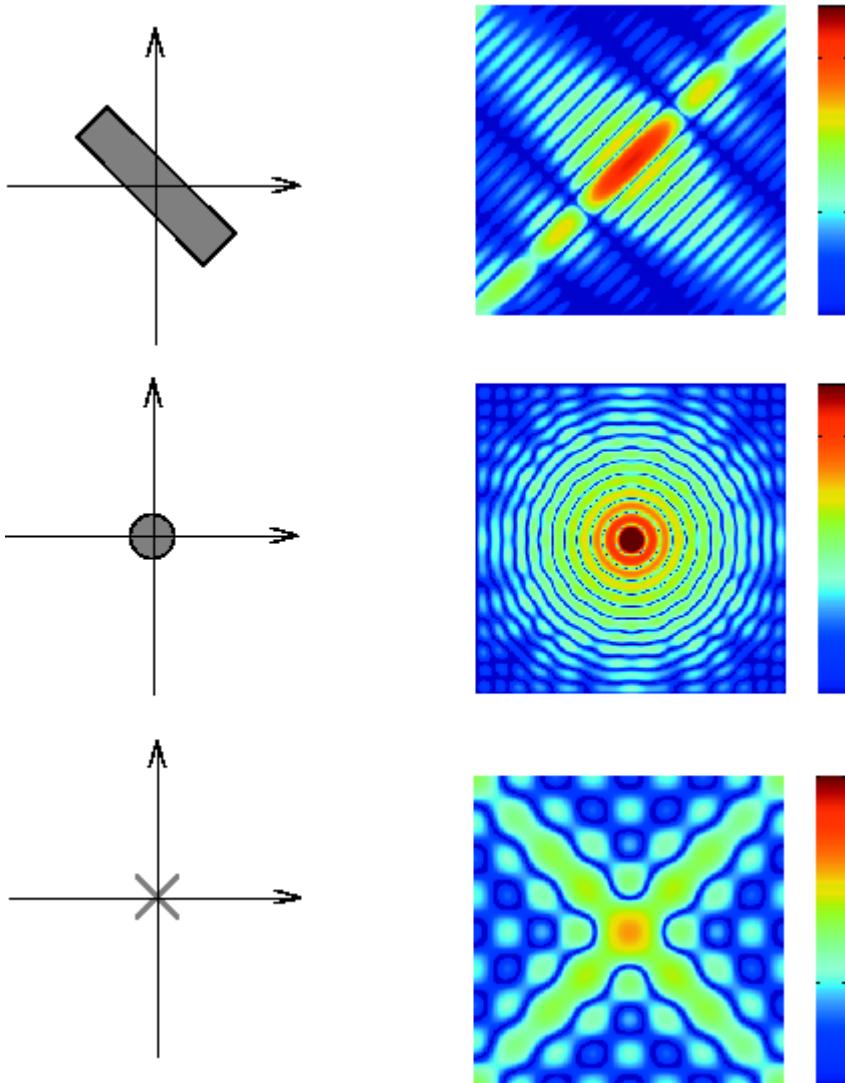
显示为图像，如图所示。



矩形函数的傅里叶变换的对数

在 $F(\omega_1, \omega_2)$ 非常接近 0 的区域，使用对数有助于显示傅里叶变换的细节。

其他简单形状的傅里叶变换示例如下所示。



一些简单形状的傅里叶变换

离散傅里叶变换

在计算机上使用傅里叶变换通常涉及一种称为离散傅里叶变换 (DFT) 的变换形式。离散变换是一种其输入和输出值均为离散样本的变换，便于计算机操作。使用这种形式的变换有两个主要原因：

- DFT 的输入和输出均为离散值，便于计算机操作。
- 有一种计算 DFT 的快速算法，称为快速傅里叶变换 (FFT)。

DFT 通常针对仅在有限区域 $0 \leq m \leq M - 1$ 和 $0 \leq n \leq N - 1$ 上为非零值的离散函数 $f(m,n)$ 定义。二维 $M \times N$ DFT 和逆 $M \times N$ DFT 的关系由下式给出：

$$F(p, q) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) e^{-j2\pi pm/M} e^{-j2\pi qn/N} \quad p = 0, 1, \dots, M-1 \\ q = 0, 1, \dots, N-1$$

和

$$f(m, n) = \frac{1}{MN} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} F(p, q) e^{j2\pi pm/M} e^{j2\pi qn/N} \quad m = 0, 1, \dots, M-1 \\ n = 0, 1, \dots, N-1$$

值 $F(p,q)$ 是 $f(m,n)$ 的 DFT 系数。零频率系数 $F(0,0)$ 通常称为“DC 分量”。DC 是一个电气工程术语，表示直流电。（请注意，MATLAB 中的矩阵索引始终从 1 开始，而不是从 0 开始；因此，矩阵元素 $f(1,1)$ 和 $F(1,1)$ 分别对应于数学量 $f(0,0)$ 和 $F(0,0)$ 。）

MATLAB 函数 `fft`、`fft2` 和 `fftn` 分别实现用于计算一维 DFT、二维 DFT 和 N 维 DFT 的快速傅里叶变换算法。函数 `ifft`、`ifft2` 和 `ifftn` 计算逆 DFT。

与傅里叶变换的关系

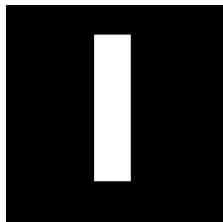
DFT 系数 $F(p,q)$ 是傅里叶变换 $F(\omega_1, \omega_2)$ 的采样。

$$F(p, q) = F(\omega_1, \omega_2) \Big|_{\begin{subarray}{l} \omega_1 = 2\pi p/M \\ \omega_2 = 2\pi q/N \end{subarray}} \quad p = 0, 1, \dots, M-1 \\ q = 0, 1, \dots, N-1$$

离散傅里叶变换的可视化

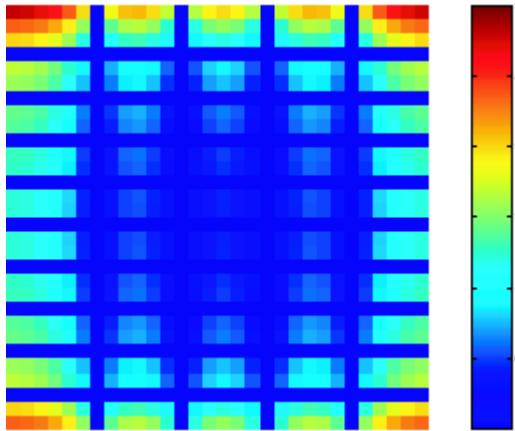
- 构造矩阵 f ，它类似于“傅里叶变换的定义”（第 9-2 页）的示例中的函数 $f(m,n)$ 。前面提到过， $f(m,n)$ 在矩形区域内等于 1，在其他位置等于 0。使用二值图像来表示 $f(m,n)$ 。

```
f = zeros(30,30);
f(5:24,13:17) = 1;
imshow(f,'InitialMagnification','fit')
```



- 使用以下命令计算并可视化 f 的 30×30 DFT。

```
F = fft2(f);
F2 = log(abs(F));
imshow(F2,[-1 5],'InitialMagnification','fit');
colormap(jet); colorbar
```



在无填充情况下计算的离散傅里叶变换

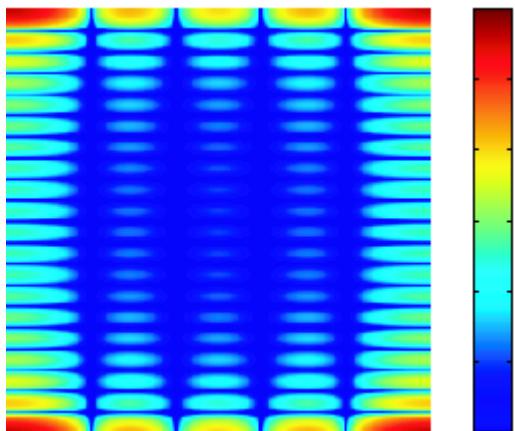
该图不同于“傅里叶变换的可视化”（第 9-2 页）中显示的傅里叶变换。首先，傅里叶变换的采样要粗略得多。其次，零频系数显示在左上角，而不是传统的中心位置。

- 3 为了获得傅里叶变换的更精细采样，在计算 f 的 DFT 时可在其中填零。使用以下命令，可一步执行填零和 DFT 计算。

```
F = fft2(f,256,256);
```

该命令在计算 DFT 之前用零将 f 填充为 256×256 。

```
imshow(log(abs(F)),[-1 5]); colormap(jet); colorbar
```



在填充情况下计算的离散傅里叶变换

- 4 然而，零频系数仍显示在左上角，而不是中心位置。您可以使用函数 `fftshift` 解决此问题，该函数交换 F 的象限，使零频系数位于中心位置。

```
F = fft2(f,256,256);F2 = fftshift(F);
imshow(log(abs(F2)),[-1 5]); colormap(jet); colorbar
```

生成的图与“傅里叶变换的可视化”（第 9-2 页）中所示的图相同。

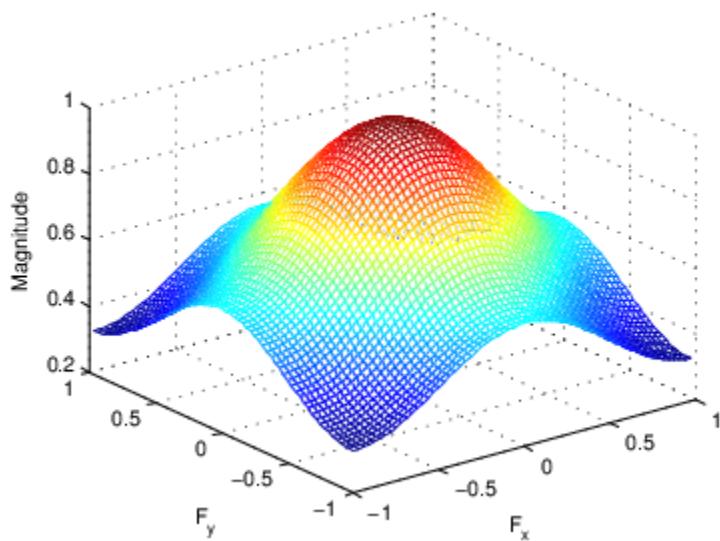
傅里叶变换的应用

本节介绍傅里叶变换的许多图像处理相关应用中的一些应用。

线性滤波器的频率响应

线性滤波器的脉冲响应的傅里叶变换可得到滤波器的频率响应。函数 `freqz2` 计算并显示滤波器的频率响应。高斯卷积核的频率响应表明，此滤波器允许低频通过，但会使高频衰减。

```
h = fspecial('gaussian');
freqz2(h)
```



高斯滤波器的频率响应

有关线性滤波、滤波器设计和频率响应的详细信息，请参阅“Design Linear Filters in the Frequency Domain”。

使用傅里叶变换执行快速卷积

此示例说明如何使用傅里叶变换对两个矩阵执行快速卷积。傅里叶变换的一个关键特性是两个傅里叶变换相乘对应于相关联的空间函数的卷积。此特性与快速傅里叶变换一起构成了快速卷积算法的基础。

注意：基于 FFT 的卷积方法最常用于大型输入。对于小型输入，使用 `imfilter` 函数通常更快。

创建两个简单的矩阵 **A** 和 **B**。**A** 是 $M \times N$ 矩阵，**B** 是 $P \times Q$ 矩阵。

```
A = magic(3);
B = ones(3);
```

对 **A** 和 **B** 执行填零，使它们至少为 $(M+P-1) \times (N+Q-1)$ 。（通常填零后的 **A** 和 **B** 的大小要为 2 的幂，这时 `fft2` 的执行速度最快。）该示例将矩阵填充为 8×8 。

```
A(8,8) = 0;
B(8,8) = 0;
```

使用 `fft2` 函数计算 **A** 和 **B** 的二维 DFT。将两个 DFT 相乘，并使用 `ifft2` 函数计算结果的逆二维 DFT。

```
C = ifft2(fft2(A).*fft2(B));
```

提取结果的非零部分，并删除舍入误差导致的虚部。

```
C = C(1:5,1:5);
C = real(C)
```

C = 5×5

8.0000	9.0000	15.0000	7.0000	6.0000
11.0000	17.0000	30.0000	19.0000	13.0000
15.0000	30.0000	45.0000	30.0000	15.0000
7.0000	21.0000	30.0000	23.0000	9.0000
4.0000	13.0000	15.0000	11.0000	2.0000

执行基于 FFT 的相关性分析来定位图像特征

此示例说明如何使用傅里叶变换来执行相关性分析，这与卷积密切相关。相关性可用于定位图像中的特征。在这种情况下，相关性通常称为模板匹配。

将示例图像读入工作区。

```
bw = imread('text.png');
```

通过从图像中提取字母“a”来创建匹配模板。请注意，您也可以使用 **imcrop** 函数的交互式语法来创建模板。

```
a = bw(32:45,88:98);
```

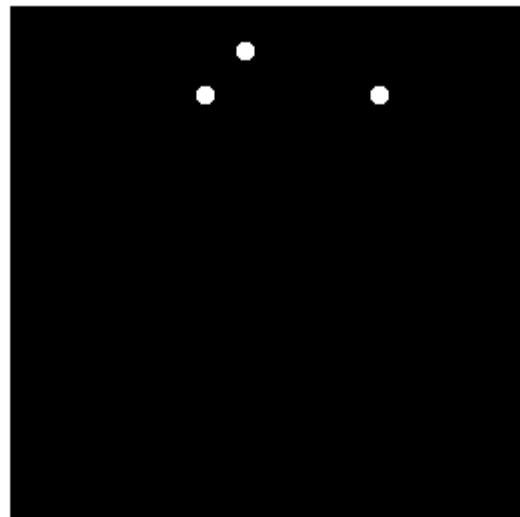
通过将模板图像旋转 180 度，然后使用基于 FFT 的卷积技术，计算模板图像与原始图像的相关性。（如果将卷积核旋转 180 度，卷积等效于相关性。）要将模板与图像匹配，请使用 **fft2** 和 **ifft2** 函数。在生成的图像中，明亮的峰值对应于字母的出现次数。

```
C = real(ifft2(fft2(bw) .* fft2(rot90(a,2),256,256)));
figure
imshow(C,[]) % Scale image to appropriate display range.
```



要查看模板在图像中的位置，请找到最大像素值，然后定义一个小于该最大值的阈值。阈值化图像将这些峰值的位置显示为阈值化相关性图像中的白色斑点。（为了使位置在此图中更容易看到，此示例扩大了阈值化图像以放大点的大小。）

```
max(C(:))  
ans = 68  
  
thresh = 60; % Use a threshold that's a little less than max.  
D = C > thresh;  
se = strel('disk',5);  
E = imdilate(D,se);  
figure  
imshow(E) % Display pixels with values over the threshold.
```



离散余弦变换

本节内容

- “DCT 定义” (第 9-12 页)
- “DCT 变换矩阵” (第 9-13 页)
- “使用离散余弦变换的图像压缩” (第 9-13 页)

DCT 定义

离散余弦变换 (DCT) 将图像表示为不同幅值和频率的正弦的总和。`dct2` 函数计算图像的二维离散余弦变换 (DCT)。对于典型图像，DCT 具有这样的属性，即关于图像的大部分视觉上显著的信息只集中在 DCT 的几个系数中。因此，DCT 经常用于图像压缩应用。例如，DCT 是国际标准有损图像压缩算法 JPEG 的核心。此名称来自制定标准的工作组：联合图像专家组，即 Joint Photographic Experts Group。

$M \times N$ 矩阵 A 的二维 DCT 定义如下。

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad 0 \leq p \leq M-1, 0 \leq q \leq N-1$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p = 0 \\ \sqrt{2/M}, & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} 1/\sqrt{N}, & q = 0 \\ \sqrt{2/N}, & 1 \leq q \leq N-1 \end{cases}$$

值 B_{pq} 称为 A 的 DCT 系数。(请注意，MATLAB 中的矩阵索引始终从 1 开始，而不是从 0 开始；因此，MATLAB 矩阵元素 $A(1,1)$ 和 $B(1,1)$ 分别对应于数学量 A_{00} 和 B_{00} 。)

DCT 是一种可逆变换，其逆变换由下式给出

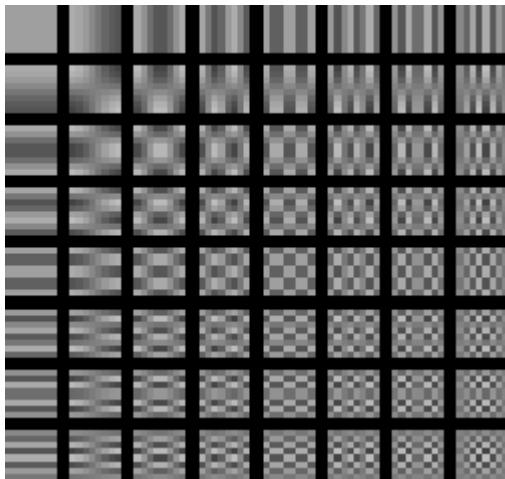
$$A_{mn} = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha_p \alpha_q B_{pq} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad 0 \leq m \leq M-1, 0 \leq n \leq N-1$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p = 0 \\ \sqrt{2/M}, & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} 1/\sqrt{N}, & q = 0 \\ \sqrt{2/N}, & 1 \leq q \leq N-1 \end{cases}$$

逆 DCT 方程可以理解为任何 $M \times N$ 矩阵 A 可写为以下形式的 MN 函数之和：

$$\alpha_p \alpha_q \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad 0 \leq p \leq M-1, 0 \leq q \leq N-1$$

这些函数称为 DCT 的基函数。然后，DCT 系数 B_{pq} 可视为应用于每个基函数的权重。对于 8×8 矩阵，该图可表示为 64 个基函数。



构成 8×8 矩阵的 64 个基函数

水平频率从左到右递增，垂直频率从上到下递增。左上角的常量值基函数通常称为 DC 基函数，对应的 DCT 系数 B_{00} 通常称为 DC 系数。

DCT 变换矩阵

使用 Image Processing Toolbox 软件有两种计算 DCT 的方法。第一种方法是使用 `dct2` 函数。`dct2` 使用基于 FFT 的算法对大型输入实现快速计算。第二种方法是使用 DCT 变换矩阵，该矩阵由函数 `dctmtx` 返回，对于小型方阵输入（如 8×8 或 16×16 ）可能更高效。 $M \times M$ 变换矩阵 T 由下式给出

$$T_{pq} = \begin{cases} \frac{1}{\sqrt{M}} & p = 0, \quad 0 \leq q \leq M - 1 \\ \sqrt{\frac{2}{M}} \cos \frac{\pi(2q+1)p}{2M} & 1 \leq p \leq M - 1, \quad 0 \leq q \leq M - 1 \end{cases}$$

对于 $M \times M$ 矩阵 A ， T^*A 是 $M \times M$ 矩阵，其列包含由 A 的列组成的一维 DCT。 A 的二维 DCT 可以计算为 $B = T^*A*T$ 。由于 T 是实数正交矩阵，它的逆矩阵与它的转置矩阵相同。因此， B 的逆二维 DCT 由 T^*B*T 给出。

使用离散余弦变换的图像压缩

此示例说明如何使用离散余弦变换 (DCT) 压缩图像。该示例计算输入图像中 8×8 个数据块的二维 DCT，丢弃（设置为零）每个数据块中 64 个 DCT 系数中除 10 个以外的所有系数，然后使用每个数据块的二维逆 DCT 重构图像。该示例使用变换矩阵计算方法。

在 JPEG 图像压缩算法中使用 DCT。将输入图像分成 8×8 或 16×16 个数据块，并对每个数据块计算二维 DCT。然后对 DCT 系数进行量化、编码和传输。JPEG 接收器（或 JPEG 文件读取器）对量化的 DCT 系数进行解码，计算每个数据块的逆二维 DCT，然后将这些数据块一起放回单个图像中。对于典型图像，许多 DCT 系数的值接近于零。可以丢弃这些系数，而不会严重影响重构图像的质量。

将图像读入工作区，并将其转换为 `double` 类。

```
I = imread('cameraman.tif');
I = im2double(I);
```

计算图像中 8×8 个数据块的二维 DCT。函数 `dctmtx` 返回 $N \times N$ DCT 变换矩阵。

```
T = detmtx(8);
det = @(block_struct) T * block_struct.data * T';
B = blockproc(I,[8 8],dct);
```

丢弃每个数据块中 64 个 DCT 系数的大部分系数，仅保留 10 个。

```
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];
B2 = blockproc(B,[8 8],@(block_struct) mask .* block_struct.data);
```

使用每个数据块的二维逆 DCT 重构图像。

```
invdct = @(block_struct) T' * block_struct.data * T;
I2 = blockproc(B2,[8 8],invdct);
```

并排显示原始图像和重构图像。尽管几乎 85% 的 DCT 系数被丢弃，导致重构图像的质量有所下降，但它仍是清晰可辨的。

```
imshow(I)
```



```
figure
imshow(I2)
```



Hough 变换

Image Processing Toolbox 支持使您能够使用 Hough 变换检测图像中线条的函数。

hough 函数实现标准 Hough 变换 (SHT)。Hough 变换设计为使用线条的参数化表示来检测线条：

$$\rho = x \cos(\theta) + y \sin(\theta)$$

变量 **rho** 是沿垂直于线条的向量从原点到线条的距离。**theta** 是 x 轴与此向量之间的角度。**hough** 函数生成一个参数空间矩阵，其行和列分别对应于这些 **rho** 和 **theta** 值。

在计算 Hough 变换后，您可以使用 **houghpeaks** 函数来求得参数空间中的峰值。这些峰值表示输入图像中可能存在的线条。

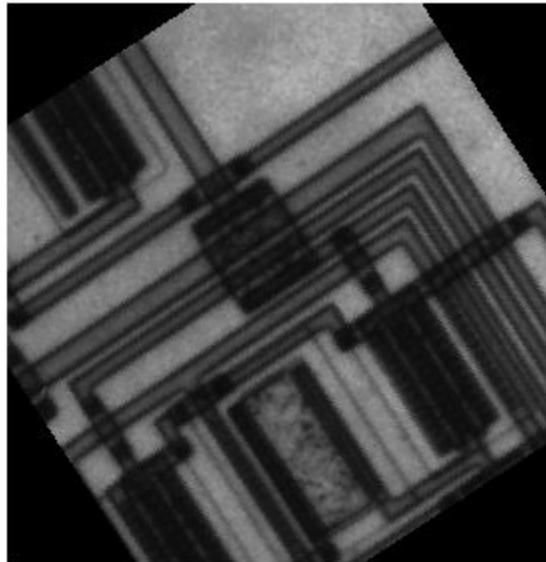
在标识 Hough 变换中的峰值后，可以使用 **houghlines** 函数求得 Hough 变换中对应于峰值的线段的端点。此函数自动填充线段中的小间隙。

使用 Hough 变换检测图像中的线条

此示例说明如何使用 Hough 变换检测图像中的线条。

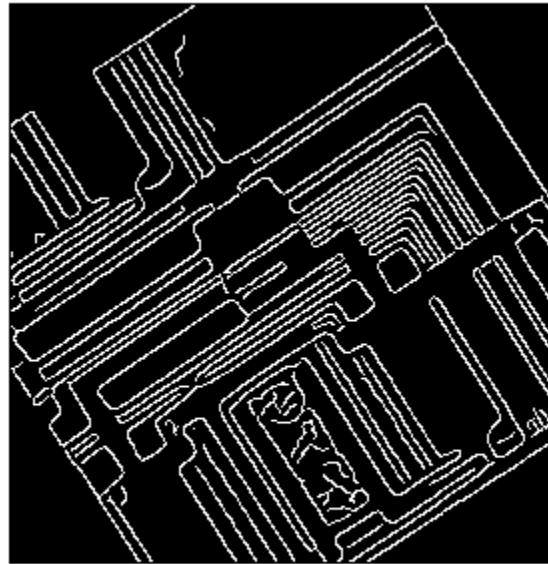
将一个图像读入工作区中，为了使此示例更具说明性，请旋转该图像。显示图像。

```
I = imread('circuit.tif');
rotI = imrotate(I,33,'crop');
imshow(rotI)
```



使用 **edge** 函数找到图像中的边缘。

```
BW = edge(rotI,'canny');
imshow(BW);
```

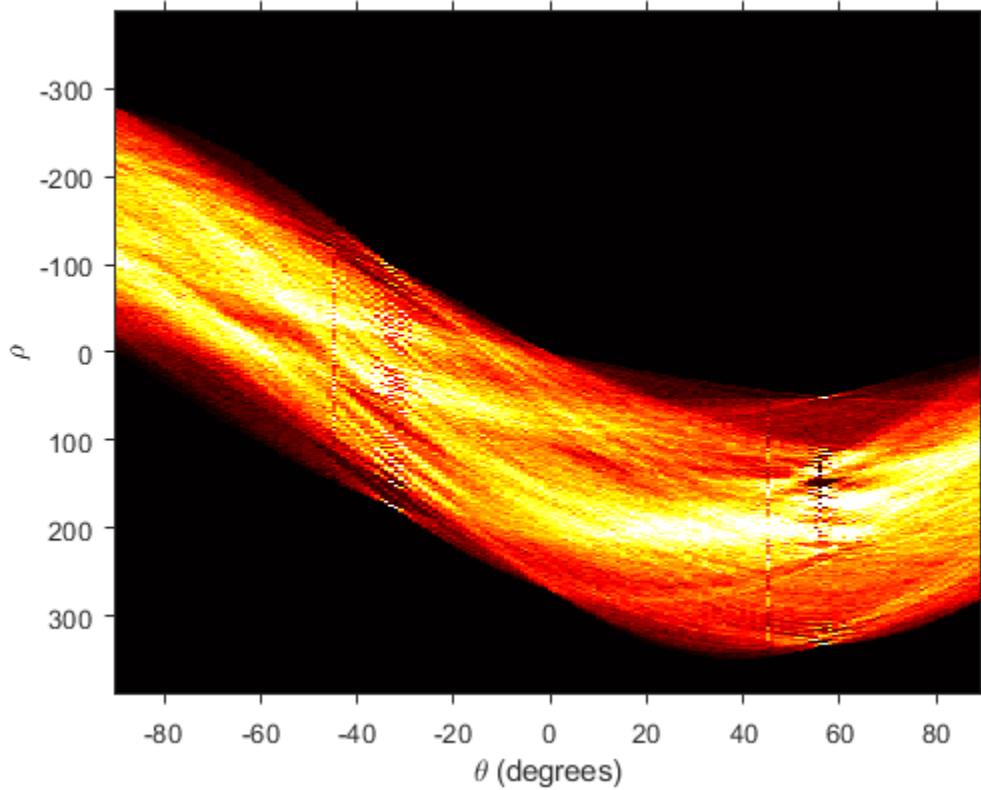


计算由 `edge` 返回的二值图像的 Hough 变换。

```
[H,theta,rho] = hough(BW);
```

显示由 `hough` 函数返回的变换 `H`。

```
figure
imshow(imadjust(rescale(H)),[],...
        'XData',theta,...
        'YData',rho,...
        'InitialMagnification','fit');
xlabel('\theta (degrees)')
ylabel('rho')
axis on
axis normal
hold on
colormap(gca,hot)
```

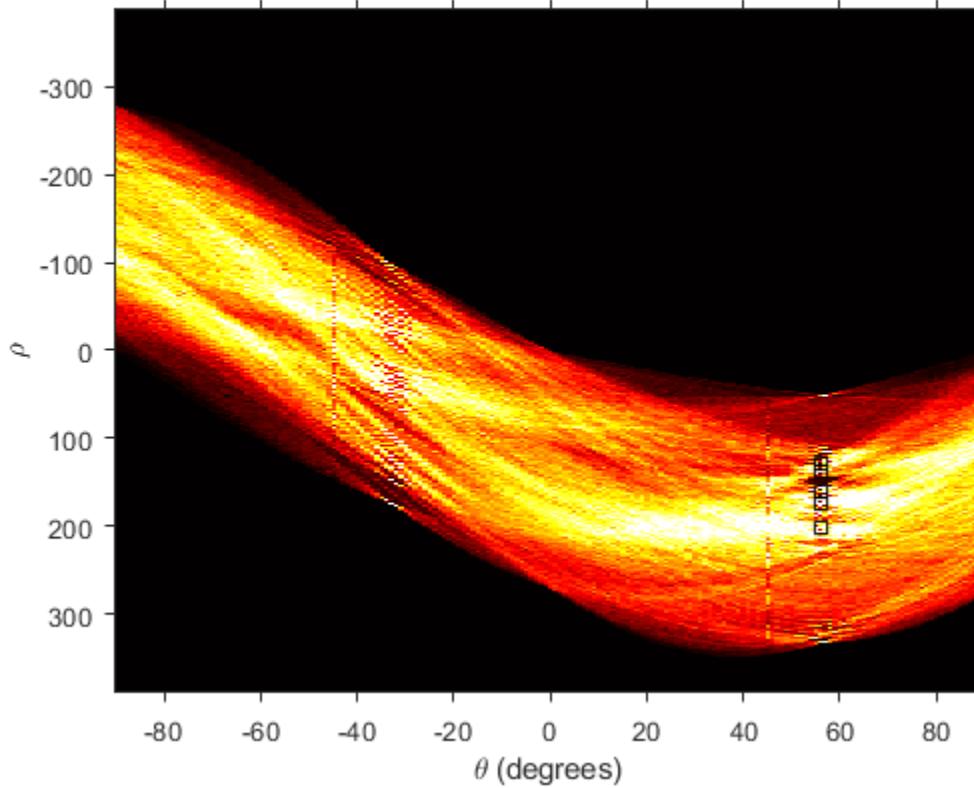


使用 `houghpeaks` 函数，在 Hough 变换矩阵 H 中找到峰值。

```
P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
```

在标识了峰值的变换图像上叠加一个绘图。

```
x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','black');
```



使用 `houghlines` 函数查找图像中的线条。

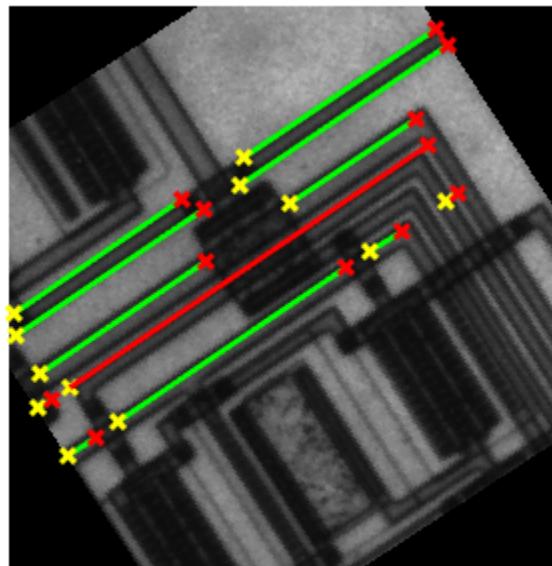
```
lines = houghlines(BW,theta,rho,P,'FillGap',5,'MinLength',7);
```

创建一个显示原始图像并在其上叠加线条的绘图。

```
figure, imshow(rotI), hold on
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

    % Plot beginnings and ends of lines
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

    % Determine the endpoints of the longest line segment
    len = norm(lines(k).point1 - lines(k).point2);
    if ( len > max_len)
        max_len = len;
        xy_long = xy;
    end
end
% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','red');
```



形态学运算

本章介绍 Image Processing Toolbox 形态学函数。您可以使用这些函数来执行常见的图像处理任务，例如对比度增强、噪声去除、精简、骨架化、填充和分割。

- “形态学运算的类型” (第 10-2 页)
- “像素连通性” (第 10-9 页)
- “泛洪填充运算” (第 10-11 页)
- “使用边缘检测和形态学检测细胞” (第 10-13 页)
- “二值图像的距离变换” (第 10-18 页)
- “标注和度量二值图像中的连通分量” (第 10-20 页)

形态学运算的类型

本节内容

“形态学膨胀和腐蚀”（第 10-2 页）

“基于膨胀和腐蚀的运算”（第 10-4 页）

形态学是基于形状处理图像的一组广泛的图像处理运算。形态学运算将结构元素应用于输入图像，从而创建相同大小的输出图像。在形态学运算中，输出图像中每个像素的值基于输入图像中对应像素与其相邻像素的比较。

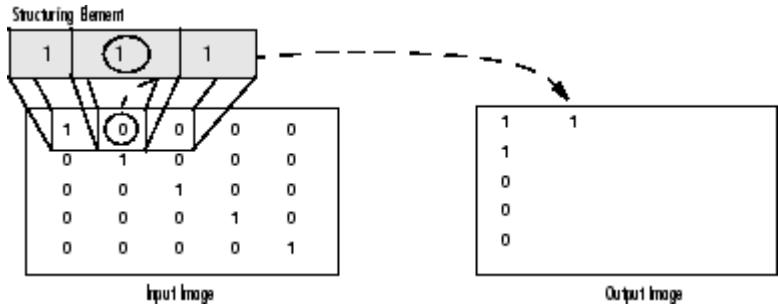
形态学膨胀和腐蚀

最基本的形态学运算是膨胀和腐蚀。膨胀指将像素添加到图像中对象的边界，而腐蚀指删除对象边界上的像素。对图像中对象添加或删除的像素数量取决于用于处理图像的结构元素的大小和形状。在形态学膨胀和腐蚀运算中，输出图像中任何给定像素的状态通过对输入图像中的对应像素及其相邻像素应用规则来确定。用于处理像素的规则将运算定义为膨胀或腐蚀。下表列出了膨胀和腐蚀的规则。

膨胀和腐蚀的规则

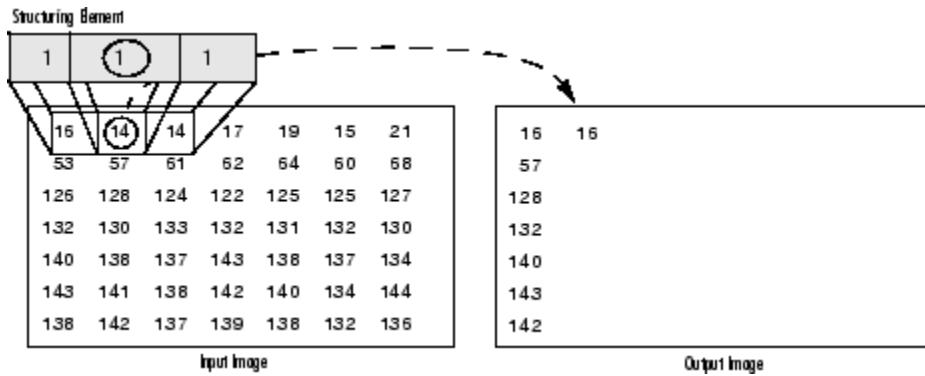
操作	规则	示例 (原始图像和处理后的图像)
膨胀	<p>输出像素的值是邻域中所有像○值。在二值图像中，如果一个○何相邻像素的值为 1，则该像○1。</p> <p>形态学膨胀使对象更加明显可○对对象中的小孔。</p>	
腐蚀	<p>输出像素的值是邻域中所有像○值。在二值图像中，如果一个○何相邻像素的值为 0，则该像○0。</p> <p>形态学腐蚀去除了孤立点和小○此只剩下重要对象。</p>	

下图说明了二值图像的膨胀。请注意结构元素如何定义关注像素的邻域，该邻域带圆圈。膨胀函数将适当的规则应用于邻域中的像素，并为输出图像中的对应像素赋值。在图中，形态学膨胀函数将输出像素的值设置为 1，因为由结构元素定义的邻域中的元素之一处于打开状态。有关详细信息，请参阅“Structuring Elements”。



二值图像的形态学膨胀

下图说明灰度图像的这种处理。该图显示输入图像中特定像素的处理。注意函数如何将规则应用于输入像素的邻域，并使用邻域中所有像素的最高值作为输出图像中对应像素的值。

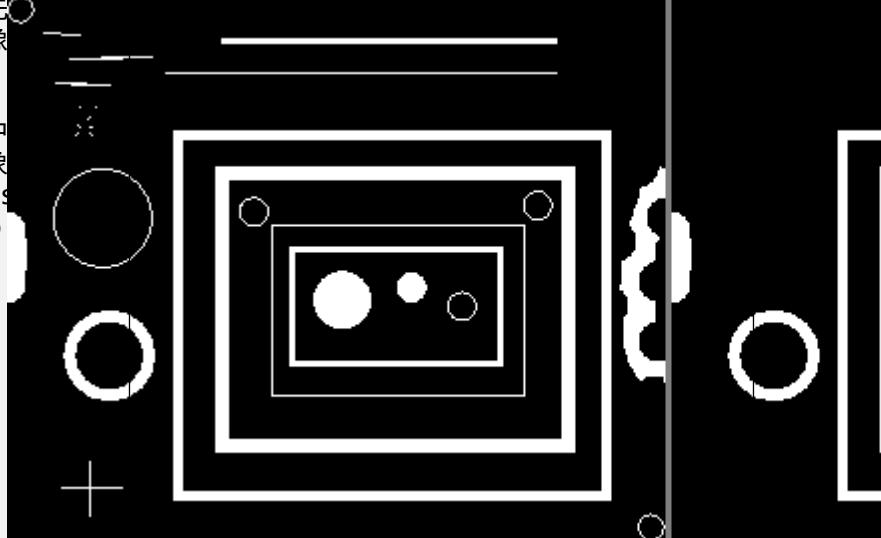
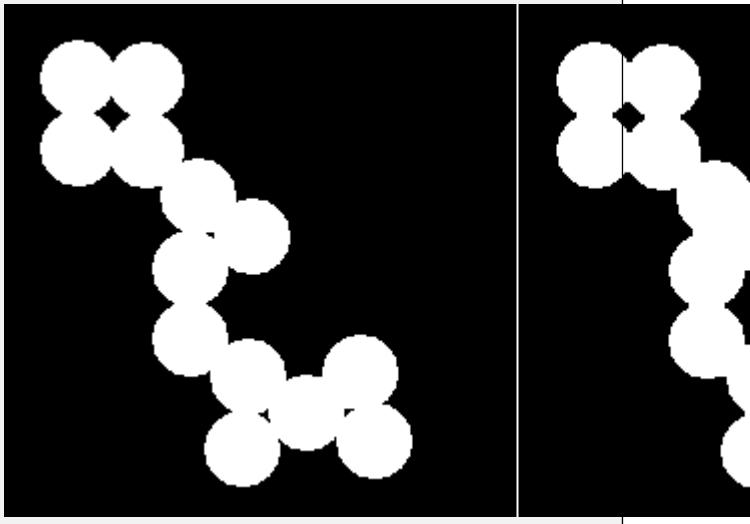


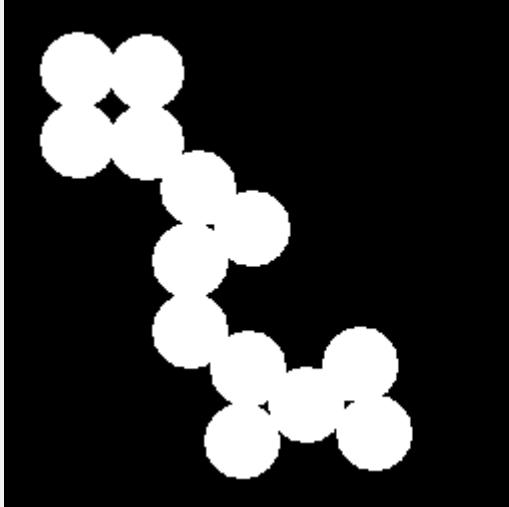
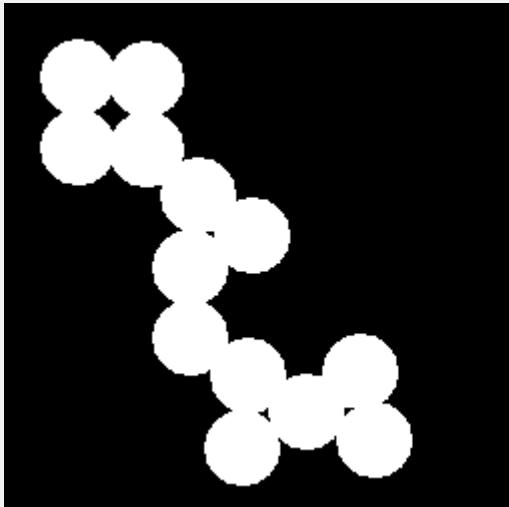
灰度图像的形态学膨胀

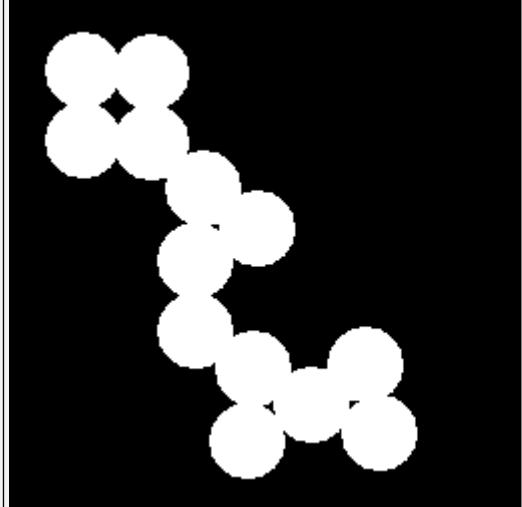
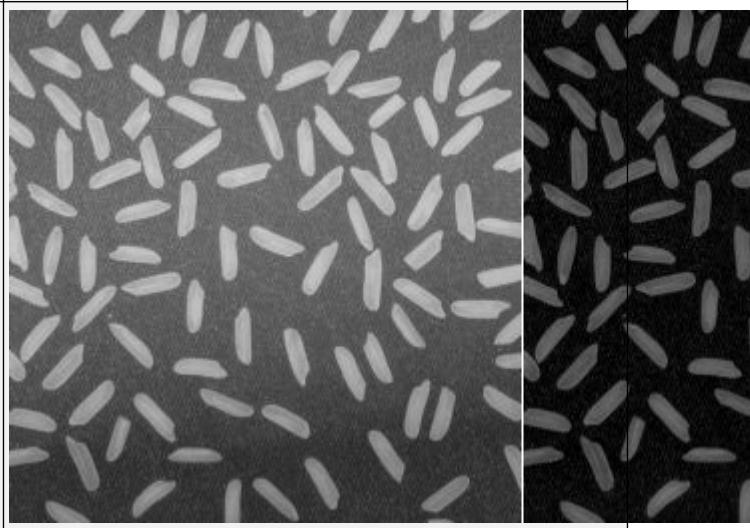
基于膨胀和腐蚀的运算

膨胀和腐蚀经常结合使用来实现图像处理运算。例如，图像的形态学开运算的定义是先腐蚀再膨胀，这两种运算使用相同的结构元素。您可以将膨胀和腐蚀结合起来，以从图像中删除小对象，并对大对象的边框进行平滑处理。

下表列出了工具箱中执行基于膨胀和腐蚀的常见形态学运算的函数。

函数	形态学定义	示例 (原始图像和处理后的图像)
imopen	<p>执行形态学开运算。开运算先膨胀图像，然后腐蚀经过膨胀的图像，两种运算使用相同的结构元素。</p> <p>形态学开运算有助于从图像中移除较小的对象，同时保持图像中较大对象的大小。有关示例，请参阅 “Using Morphological Opening to Remove Small Image Features”。</p>	
imclose	<p>执行形态学闭运算。闭运算先膨胀图像，然后腐蚀经过膨胀的图像，两种运算使用相同的结构元素。</p> <p>形态学闭运算可用于填充图像中的小孔，同时保留图像中对象的形状和大小。</p>	

函数	形态学定义	示例 (原始图像和处理后的图像)
bwskel	骨架化二值图像中的对象。骨架化处理过程会将所有对象腐蚀到中心线，而不会改变对象的基本结构，例如存在的孔洞和分支。	 
bwperim	求二值图像中对象的边界。如果某像素非零并且与至少一个零值像素连通，则该像素是边界的一部分。	 

函数	形态学定义	示例 (原始图像和处理后的图像)
bwhitmiss	<p>执行二值命中与否变换。对二值图像进行命中与否变换时，如果像素的邻域匹配某结构元素的形状，而不匹配另一个不相交结构元素的形状，则保留该像素。</p> <p>命中与否变换可用于检测图像中的图案。</p>	
imtophat	<p>执行形态学顶帽变换。顶帽变换先对一个图像进行开运算，然后从原始图像中减去经过开运算的图像。</p> <p>顶帽变换可用于增强亮度不均匀的灰度图像中的对比度。该变换还可以隔离图像中较小的明亮对象。</p>	

函数	形态学定义	示例（原始图像和处理后的图像）
imbothat	执行形态学底帽变换。底帽变换先对图像进行闭合运算，然后从经过闭合运算的图像中减去原始图像。 底帽变换可用于在灰度图像中找到光强谷。	

另请参阅

[imclose](#) | [imopen](#) | [imdilate](#) | [imerode](#)

详细信息

- “Structuring Elements”
- “像素连通性”（第 10-9 页）
- “Border Padding for Morphology”

像素连通性

形态学处理从标记图像的峰值开始，并基于像素的连通性扩展到图像的其余部分。连通性定义哪些像素与其他像素相连。二值图像中形成一个连通组的一组像素称为一个对象或连通分量。

确定哪些像素构成一个连通分量取决于如何定义像素连通性。例如，此二值图像包含一个或两个前景对象，具体取决于如何定义连通性。如果前景为 4 连通，则整个图像就是一个对象 - 前景对象和背景之间没有区别。但是，如果前景为 8 连通，则设置为 1 的像素连通而形成一个闭环，图像有两个单独的对象：环内的像素和环外的像素。

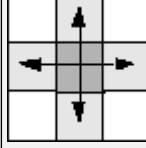
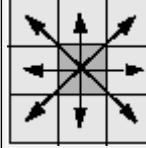
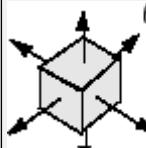
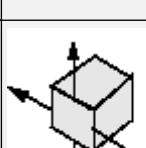
```

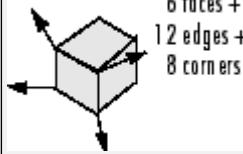
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0
0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

定义图像中的连通性

下表列出了工具箱支持的所有标准二维和三维连通性。

值	意义	
二维连通		
4 连通	如果像素的边缘相互接触，则这些像素具有连通性。如果两个相邻像素都为 on 并在水平或垂直方向上连通，则它们是同一对象的一部分。	
8 连通	如果像素的边缘或角相互接触，则这些像素具有连通性。如果两个相邻像素都为 on 并在水平、垂直或对角线方向上连通，则它们是同一对象的一部分。	
三维连通		
6 连通	如果像素的面接触，则这些像素具有连通性。如果两个相邻像素都为 on 并以如下方式连通，则它们是同一对象的一部分： <ul style="list-style-type: none"> 在所列方向之一上连通：内、外、左、右、上、下 	
18 连通	如果像素的面或边缘接触，则这些像素具有连通性。如果两个相邻像素都为 on 并以如下方式连通，则它们是同一对象的一部分： <ul style="list-style-type: none"> 在所列方向之一上连通：内、外、左、右、上、下 在两个方向的组合上连通，如右下或内上 	

值	意义
26 连通	<p>如果像素的面、边缘或角接触，则这些像素具有连通性。如果两个相邻像素都为 on 并以如下方式连通，则它们是同一对象的一部分：</p> <ul style="list-style-type: none"> 在所列方向之一上连通：内、外、左、右、上、下 在两个方向的组合上连通，如右下或内上 在三个方向的组合上连通，如内右上或内左下 

选择连通性

您选择的邻域类型会影响在图像中找到的对象数量以及这些对象的边界。因此，许多形态学运算的结果通常因指定的连通性类型而异。

例如，如果指定 4 连通的邻域，则此二值图像包含两个对象；如果指定 8 连通的邻域，则图像只有一个对象。

```
0   0   0   0   0   0
0   1   1   0   0   0
0   1   1   0   0   0
0   0   0   1   1   0
0   0   0   1   1   0
```

指定自定义连通性

您也可以通过指定由 0 和 1 组成的 $3 \times 3 \times \dots \times 3$ 数组来定义自定义邻域。值为 1 元素定义邻域相对于中心元素的连通性。

例如，以下数组定义“北/南”连通性，可用于将图像拆分成独立的列。

```
CONN = [ 0 1 0; 0 1 0; 0 1 0 ]
CONN =
    0   1   0
    0   1   0
    0   1   0
```

注意 连通性数组必须关于其中心元素对称。您还可以对三维图像使用二维连通性数组；连通性会影响三维图像中的每个“页”。

另请参阅

[conndef](#) | [iptcheckconn](#) | [bwconncomp](#) | [bwconncomp](#) | [imfill](#) | [bwareaopen](#) | [boundarymask](#)

详细信息

- “Morphological Reconstruction”

泛洪填充运算

`imfill` 函数对二值图像和灰度图像执行泛洪填充运算。此运算对于从图像中删除不相关的伪影非常有用。

- 对于二值图像，`imfill` 将连通的背景像素 (0) 更改为前景像素 (1)，在到达对象边界时停止。
- 对于灰度图像，`imfill` 将较亮区域包围的较暗区域的强度值提高到与周围像素相同的强度级别。实际上，`imfill` 会删除未与图像边界连通的区域最小值。有关详细信息，请参阅 “[Finding Areas of High or Low Intensity](#)”。

指定连通性

对于二值图像和灰度图像，填充运算的边界由您指定的像素连通性（第 10-9 页）确定。

注意 与其他基于对象的运算不同，`imfill` 对背景像素进行运算。当您使用 `imfill` 指定连通性时，指定的是背景的连通性，而不是前景的连通性。

连通性的含义可以用以下矩阵来说明。

```
BW = logical([0 0 0 0 0 0 0 0;
    0 1 1 1 1 1 0 0;
    0 1 0 0 0 1 0 0;
    0 1 0 0 0 1 0 0;
    0 1 0 0 0 1 0 0;
    0 1 1 1 1 0 0 0;
    0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0]);
```

如果背景是 4 连通的，此二值图像包含两个单独的背景元素（环内部分和环外部分）。如果背景是 8 连通的，则像素以对角方式连通，且只有一个背景元素。

指定起点

对于二值图像，您可以通过传入位置下标或在交互模式下使用 `imfill`，用鼠标选择起始像素来指定填充运算的起点。

例如，如果您调用 `imfill`，将像素 `BW(4,3)` 指定为起点，则 `imfill` 仅填充环内部分，因为默认情况下背景是 4 连通的。

```
imfill(BW,[4 3])
```

```
ans =
 0 0 0 0 0 0 0 0
 0 1 1 1 1 1 0 0
 0 1 1 1 1 1 0 0
 0 1 1 1 1 1 0 0
 0 1 1 1 1 1 0 0
 0 1 1 1 1 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
```

如果指定相同的起点但使用 8 连通的背景连通性，则 `imfill` 会填充整个图像。

```
imfill(BW,[4 3],8)
```

```
ans =
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

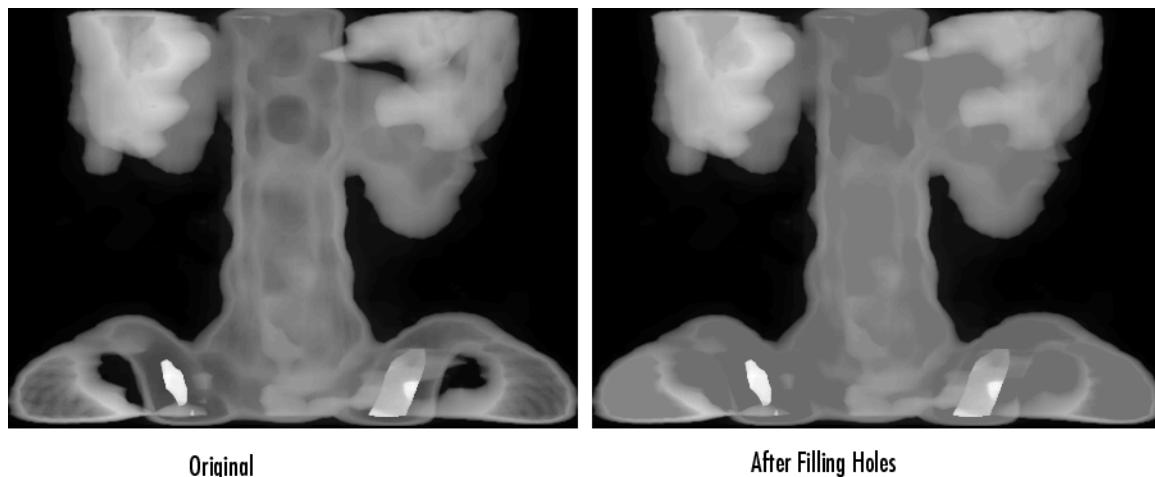
填充孔

泛洪填充运算的常见用途是填充图像中的孔。例如，假设您有一个二值图像或灰度图像，其中前景对象表示球体。在该图像中，这些对象应显示为圆盘，但由于原始照片中的反射，它们呈环形。在对该图像进行任何进一步处理之前，您可能希望首先使用 `imfill` 填充“环形孔”。

由于使用泛洪填充来填充孔非常常见，`imfill` 包括特殊语法来支持二值图像和灰度图像的泛洪填充孔操作。在此语法中，您只需指定参数 '`holes`'；不必指定每个孔的起始位置。

为了演示，此示例填充脊柱灰度图像中的孔。

```
[X,map] = imread('spine.tif');
I = ind2gray(X,map);
Ifill = imfill(I,'holes');
imshow(I);figure, imshow(Ifill)
```



另请参阅

`imfill`

详细信息

- “Morphological Reconstruction”

使用边缘检测和形态学检测细胞

此示例说明如何使用边缘检测和基本形态学检测细胞。如果某对象与背景有足够的对比度，则可以在图像中轻松检测到该对象。

步骤 1：读取图像

在 `cell.tif` 图像中读取，这是一个前列腺癌细胞的图像。此图像中存在两个细胞，但只有一个细胞完整显示。目标是检测或分割完整显示的细胞。

```
I = imread('cell.tif');
imshow(I)
title('Original Image');
text(size(I,2),size(I,1)+15, ...
    'Image courtesy of Alan Partin', ...
    'FontSize',7,'HorizontalAlignment','right');
text(size(I,2),size(I,1)+25, ...
    'Johns Hopkins University', ...
    'FontSize',7,'HorizontalAlignment','right');
```



Image courtesy of Alan Partin
Johns Hopkins University

步骤 2：检测整个细胞

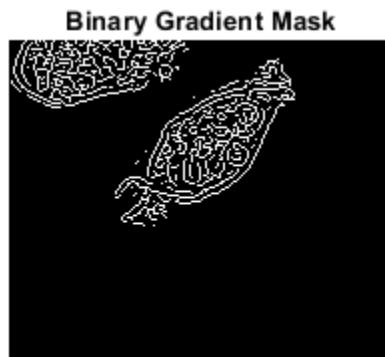
要分割的对象与背景图像的对比度相差很大。计算图像梯度的算子可以检测到对比度的变化。要创建包含分割后的细胞的二值掩膜，请计算梯度图像并应用一个阈值。

使用 `edge` 和 `Sobel` 算子计算阈值。调整阈值，再次使用 `edge` 获得包含分割后的细胞的二值掩膜。

```
[~,threshold] = edge(I,'sobel');
fudgeFactor = 0.5;
BW = edge(I,'sobel',threshold * fudgeFactor);
```

显示生成的二元梯度掩膜。

```
imshow(BW)
title('Binary Gradient Mask')
```



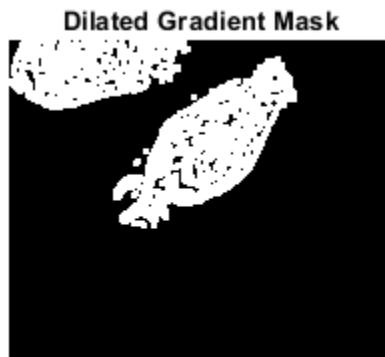
步骤 3：膨胀图像

二元梯度掩膜显示图像中高对比度的线条。这些线条没有很好地描绘出关注对象的轮廓。与原始图像相比，梯度掩膜中对象周围的线条有间隙。如果使用线性结构元素膨胀 Sobel 图像，这些线性间隙将消失。使用 `strel` 函数创建两个垂直接构元素。

```
se90 = strel('line',3,90);
se0 = strel('line',3,0);
```

先后使用垂直结构元素和水平结构元素，来膨胀二元梯度掩膜。使用 `imdilate` 函数膨胀图像。

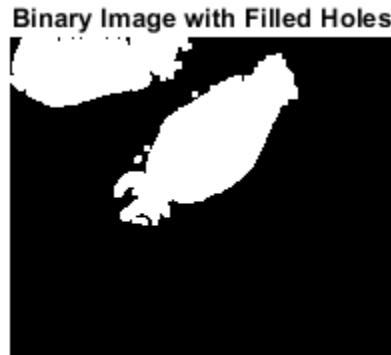
```
BWsdil = imdilate(BWs,[se90 se0]);
imshow(BWsdil)
title('Dilated Gradient Mask')
```



步骤 4：填补内部间隙

膨胀的梯度掩膜很好地显示了细胞的轮廓，但细胞内部仍有小孔。要填充这些孔洞，请使用 `imfill` 函数。

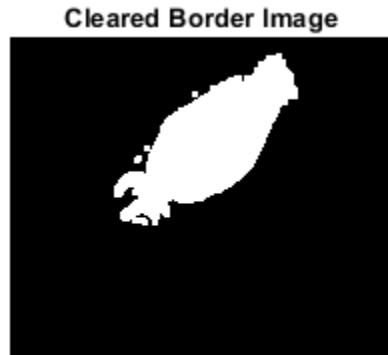
```
BWdfill = imfill(BWsdl,'holes');
imshow(BWdfill)
title('Binary Image with Filled Holes')
```



步骤 5：删除边界上的连通对象

关注的细胞已成功分割，但它不是被发现的唯一对象。可以使用 `imclearborder` 函数删除任何与图像边界连通的对象。要删除对角线连通，请将 `imclearborder` 函数中的连通性设置为 4。

```
BWnobord = imclearborder(BWdfill,4);
imshow(BWnobord)
title('Cleared Border Image')
```

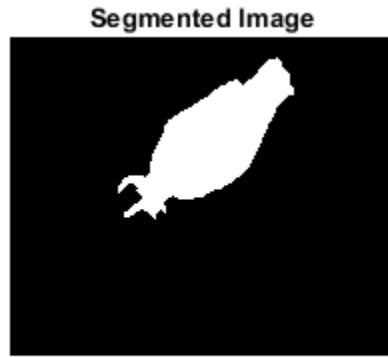


步骤 6：平滑处理对象

最后，为了使分割后的对象看起来自然，用菱形结构元素对图像腐蚀两次来平滑处理对象。使用 `strel` 函数创建菱形结构元素。

```
seD = strel('diamond',1);
BWfinal = imerode(BWnobord,seD);
BWfinal = imerode(BWfinal,seD);
```

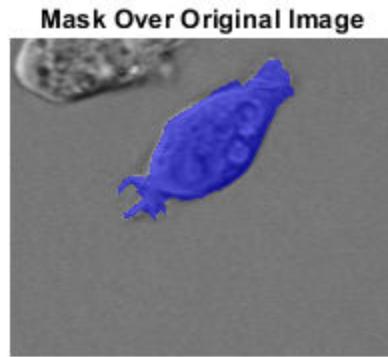
```
imshow(BWfinal)
title('Segmented Image');
```



步骤 7：可视化分割

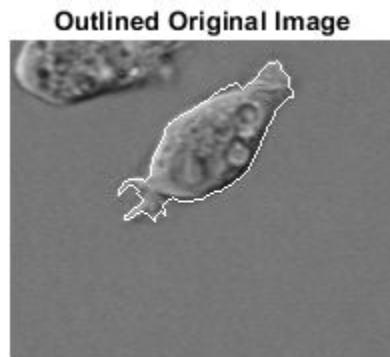
您可以使用 `labeloverlay` 函数在原始图像上显示掩膜。

```
imshow(labeloverlay(I,BWfinal))
title('Mask Over Original Image')
```



显示分割后的对象的另一种方法是在分割的细胞周围绘制轮廓。使用 `bwperim` 函数绘制轮廓。

```
BWoutline = bwperim(BWfinal);
Segout = I;
Segout(BWoutline) = 255;
imshow(Segout)
title('Outlined Original Image')
```



另请参阅

`imfill` | `imclearborder` | `edge` | `imdilate` | `imerode` | `bwperim` | `strel`

详细信息

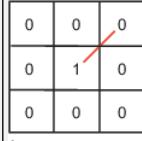
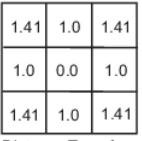
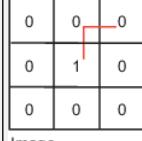
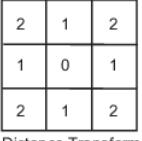
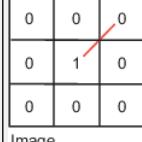
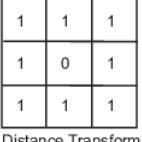
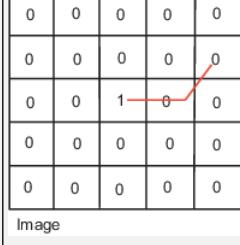
- “形态学运算的类型” (第 10-2 页)

二值图像的距离变换

距离变换提供图像中点间距的一种指标或度量。**bwdist** 函数计算二值图像中设置为 **off (0)** 的每个像素与最近的非零像素之间的距离。

bwdist 函数支持多种距离度量。

距离度量

距离度量	说明	示意图
欧几里德	欧几里德距离是两个像素之间的直线距离。	 
City block	City block 距离度量基于 4 连通邻域来测量像素之间的路径。边缘接触的像素相隔 1 个单位；以对角方式接触的像素相隔 2 个单位。	 
棋盘	棋盘距离度量基于 8 连通邻域来测量像素之间的路径。边缘或角接触的像素相隔 1 个单位。	 
准欧几里德	准欧几里德度量测量沿一组水平、垂直和对角线段的总欧几里德距离。	 

此示例创建一个包含两个相交圆形目标的二值图像。

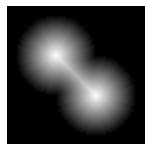
```

center1 = -10;
center2 = -center1;
dist = sqrt(2*(2*center1)^2);
radius = dist/2 * 1.4;
lims = [floor(center1-1.2*radius) ceil(center2+1.2*radius)];
[x,y] = meshgrid(lims(1):lims(2));
bw1 = sqrt((x-center1).^2 + (y-center1).^2) <= radius;
bw2 = sqrt((x-center2).^2 + (y-center2).^2) <= radius;
bw = bw1 | bw2;
figure
imshow(bw)

```

要计算该二值图像的补码的距离变换，请使用 **bwdist** 函数。在距离变换的图像中，请注意两个圆形区域的中心是白色的。

```
D = bwdist(~bw);
figure
imshow(D,[])
```



标注和度量二值图像中的连通分量

本节内容

- “检测连通分量” (第 10-20 页)
- “标注连通分量” (第 10-21 页)
- “选择二值图像中的对象” (第 10-22 页)
- “测量连通分量的属性” (第 10-22 页)

检测连通分量

二值图像中的一个连通分量是一组像素，它们构成一个连通组。例如，以下二值图像有三个连通分量。

```
BW = [0 0 0 0 0 0 0 0 0;
      0 1 1 0 0 1 1 1 1;
      0 1 1 0 0 0 0 1 1;
      0 1 1 0 0 0 0 0 0;
      0 0 0 1 1 0 0 0 0;
      0 0 0 1 1 0 0 0 0;
      0 0 0 1 1 0 0 0 0;
      0 0 0 0 0 0 0 0 0];
```

Connected Components

使用 **bwconncomp** 计算连通分量。在以下示例代码中，**BW** 是以上图像中所示的二值矩阵。将连通性指定为 4，从而当两个相邻像素都为 on 并在水平或垂直方向上连通，则它们是同一对象的一部分。

PixelIdxList 字段标识属于每个连通分量的像素列表。

```
BW = zeros(8,9);
BW(2:4,2:3) = 1;
BW(5:7,4:5) = 1;
BW(2,7:9) = 1;
BW(3,8:9) = 1;
BW
```

BW =

```
0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 1 1 1
0 1 1 0 0 0 0 1 1
0 1 1 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0
```

```
cc = bwconncomp(BW,4)
```

cc =

Connectivity: 8

```
ImageSize: [8 9]
NumObjects: 3
PixelIdxList: {[6x1 double] [6x1 double] [5x1 double]}
```

标注连通分量

连通分量标注是识别图像中的连通分量并为每个分量分配唯一标签的过程。生成的矩阵称为标签矩阵。

```
BW = [0 0 0 0 0 0 0 0 0;
       0 1 1 0 0 0 3 3 3;
       0 1 1 0 0 0 0 3 3;
       0 1 1 0 0 0 0 0 0;
       0 0 0 0 2 2 0 0 0;
       0 0 0 0 2 2 0 0 0;
       0 0 0 0 2 2 0 0 0;
       0 0 0 0 0 0 0 0 0];
```

Labeled Connected Components

使用 `labelmatrix` 函数创建一个标签矩阵。此示例代码继续使用前一节中定义的连通分量结构体 `cc`。

```
labeled = labelmatrix(cc)
```

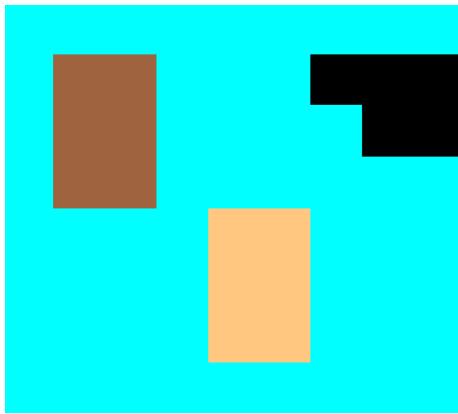
```
labeled =
```

```
8×9 uint8 matrix

0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 3 3 3
0 1 1 0 0 0 0 3 3
0 1 1 0 0 0 0 0 0
0 0 0 2 2 0 0 0 0
0 0 0 2 2 0 0 0 0
0 0 0 2 2 0 0 0 0
0 0 0 0 0 0 0 0 0
```

要可视化连通分量，请使用 `label2rgb` 函数将标签矩阵显示为伪彩色图像。用于标识标签矩阵中每个对象的标签映射到相关联的颜色图中的不同颜色。您可以指定颜色图、背景颜色以及标签矩阵中的对象如何映射到颜色图中的颜色。

```
RGB_label = label2rgb(labeled,@copper,'c','shuffle');
imshow(RGB_label)
```



选择二值图像中的对象

您可以使用 **bwselect** 函数选择二值图像中的单个对象。以编程方式或用鼠标以交互方式指定输入图像中的像素。**bwselect** 返回一个二值图像，该图像仅包含输入图像中包含指定像素之一的那些对象。

例如，使用以下命令选择在当前坐标区中显示的图像中的对象。

```
BW2 = bwselect;
```

光标在位于图像上方时会变为十字准线。点击要选择的对象；**bwselect** 在您点击的每个像素上显示一个小星形。完成后，按 **Return**。**bwselect** 返回由您选择的对象组成的二值图像，并删除小星形。

测量连通分量的属性

regionprops 函数可以返回连通分量的若干属性的测量值。其他函数测量单个属性。例如，**bwarea** 函数返回二值图像的面积。

此示例使用 **bwarea** 来确定由膨胀运算导致的 **circbw.tif** 中面积增加的百分比。面积是图像前景大小的一种度量，大致等于图像中 **on** 像素的数量。但是，**bwarea** 不会简单地对设置为 **on** 的像素数进行计数。**bwarea** 在计算面积时会对不同像素模式进行不一致的加权。这种加权可以补偿用离散像素表示连续图像所固有的失真。例如，包含 50 个像素的对角线比包含 50 个像素的水平线长。由于 **bwarea** 使用加权，水平线的面积为 50，而对角线的面积为 62.5。

```
BW = imread('circbw.tif');
SE = ones(5);
BW2 = imdilate(BW,SE);
increase = (bwarea(BW2) - bwarea(BW))/bwarea(BW)
```

```
increase =
0.3456
```

另请参阅

[bwconncomp](#) | [labelmatrix](#) | [label2rgb](#) | [bwselect](#) | [regionprops](#)

相关示例

- “Calculate Properties of Image Regions Using Image Region Analyzer”
- “校正亮度不均匀问题并分析前景对象” (第 1-7 页)

详细信息

- “像素连通性” (第 10-9 页)

分析和增强图像

本主题说明支持用于分析和增强图像的一系列标准图像处理运算的函数。

- “像素值” (第 11-2 页)
- “图像均值、标准差和相关系数” (第 11-3 页)
- “边缘检测” (第 11-4 页)
- “检测和测量图像中的圆形目标” (第 11-6 页)
- “标识圆形目标” (第 11-18 页)
- “使用灰度共生矩阵 (GLCM) 的纹理分析” (第 11-26 页)
- “创建灰度共生矩阵” (第 11-27 页)
- “对比度增强方法” (第 11-28 页)
- “直方图均衡化” (第 11-32 页)
- “将高斯平滑滤波器应用于图像” (第 11-35 页)
- “去除噪声” (第 11-41 页)
- “使用附加功能资源管理器安装示例数据” (第 11-47 页)

像素值

要确定图像中一个或多个像素的值并在变量中返回值，请使用 `impixel` 函数。您可以通过将像素的坐标作为输入参数传递来指定像素，也可以使用鼠标以交互方式选择像素。`impixel` 在 MATLAB 工作区的变量中返回指定像素的值。

注意 您也可以使用图像工具以交互方式获取像素值信息 - 请参阅“Get Pixel Information in Image Viewer App”。

确定图像中单个像素的值

此示例说明如何以交互方式使用 `impixel` 来获取像素值。

显示一个图像。

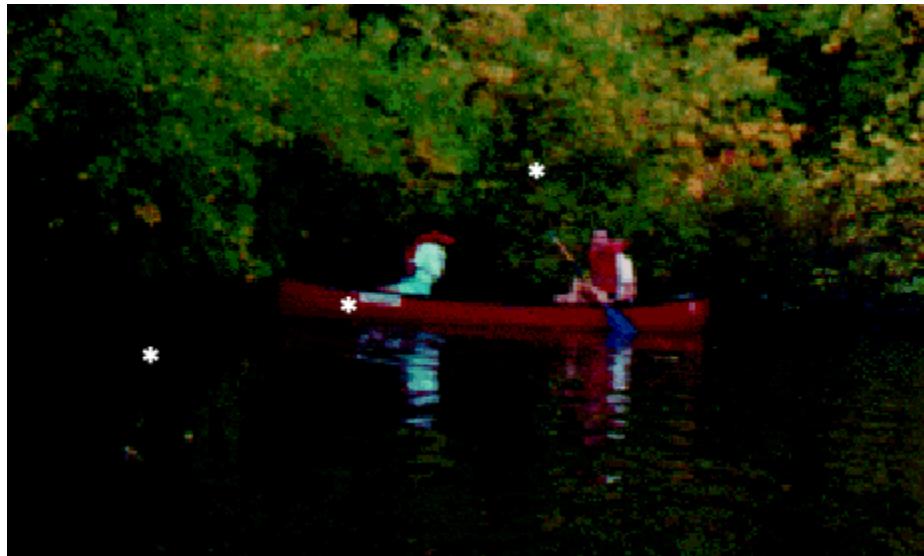
```
imshow canoe.tif
```

调用 `impixel`。在不带输入参数的情况下调用 `impixel` 时，它会将其自身与当前坐标区中的图像相关联。

```
pixel_values = impixel
```

通过点击鼠标选择图像中要检查的点。`impixel` 在所选的每个点上放置一个星形。

```
imshow canoe.tif
```



完成点选择后，按 **Return**。`impixel` 以 $n \times 3$ 数组形式返回像素值，其中 n 是所选点的数目。`impixel` 会删除用于指示所选点的星形。

```
pixel_values =  
  
0.1294 0.1294 0.1294  
0.5176 0 0  
0.7765 0.6118 0.4196
```

图像均值、标准差和相关系数

您可以使用 **mean2**、**std2** 和 **corr2** 函数计算图像的标准统计量。**mean2** 和 **std2** 计算矩阵元素的均值和标准差。**corr2** 计算两个相同大小的矩阵之间的相关系数。

这些函数是 MATLAB 函数参考中所述的 **mean**、**std** 和 **corrcoef** 函数的二维版本。

边缘检测

在图像中，边缘是一条曲线，其走势与图像中强度快速变化的路径一致。边缘通常与场景中目标的边界相关联。边缘检测用于确定图像中的边缘。

要寻找边缘，您可以使用 `edge` 函数。此函数使用以下两个标准之一来寻找图像中强度迅速变化的位置：

- 强度的一阶导数的模大于某个阈值的位置
- 强度的二阶导数有过零点的位置

`edge` 提供几个导数估算器，其中每个都实现以下定义之一。对于其中一些估算器，您可以指定运算是否应对水平边缘、垂直边缘敏感或对两者都敏感。`edge` 返回二值图像，其中包含的 1 对应于找到边的位置，0 对应于其他位置。

`edge` 提供的最强大的边缘检测方法是 Canny 方法。Canny 方法与其他边缘检测方法的不同之处在于，它使用两种不同阈值（用于检测强边缘和弱边缘），并且仅当弱边缘连通到强边缘时才在输出中包括弱边缘。因此，这种方法不太可能受到噪声的影响，更可能检测到真正的弱边缘。

检测图像中的边缘

此示例说明如何同时使用 Canny 边缘检测器和 Sobel 边缘检测器来检测图像中的边缘。

读取图像并显示它。

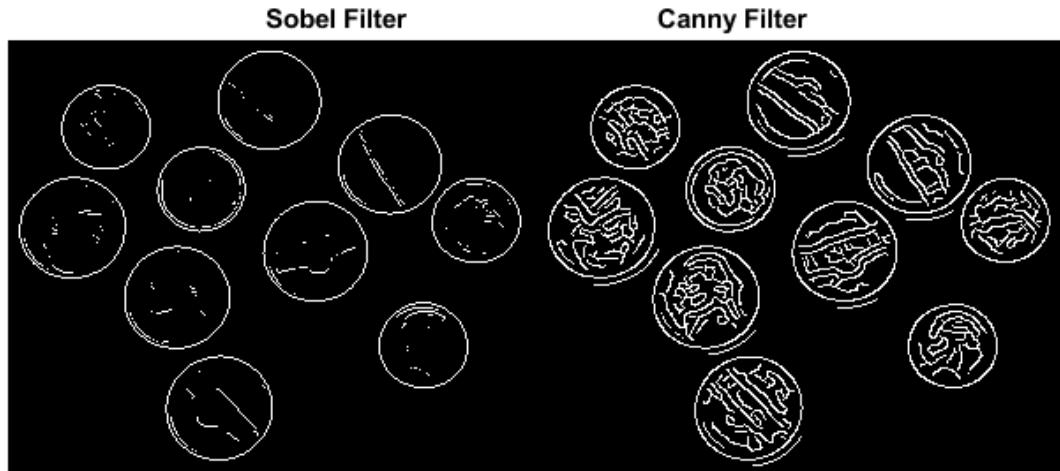
```
I = imread('coins.png');  
imshow(I)
```



将 Sobel 和 Canny 边缘检测器同时应用于图像，并显示它们以进行比较。

```
BW1 = edge(I,'sobel');  
BW2 = edge(I,'canny');
```

```
figure;
imshowpair(BW1,BW2,'montage')
title('Sobel Filter' Canny Filter');
```



检测和测量图像中的圆形目标

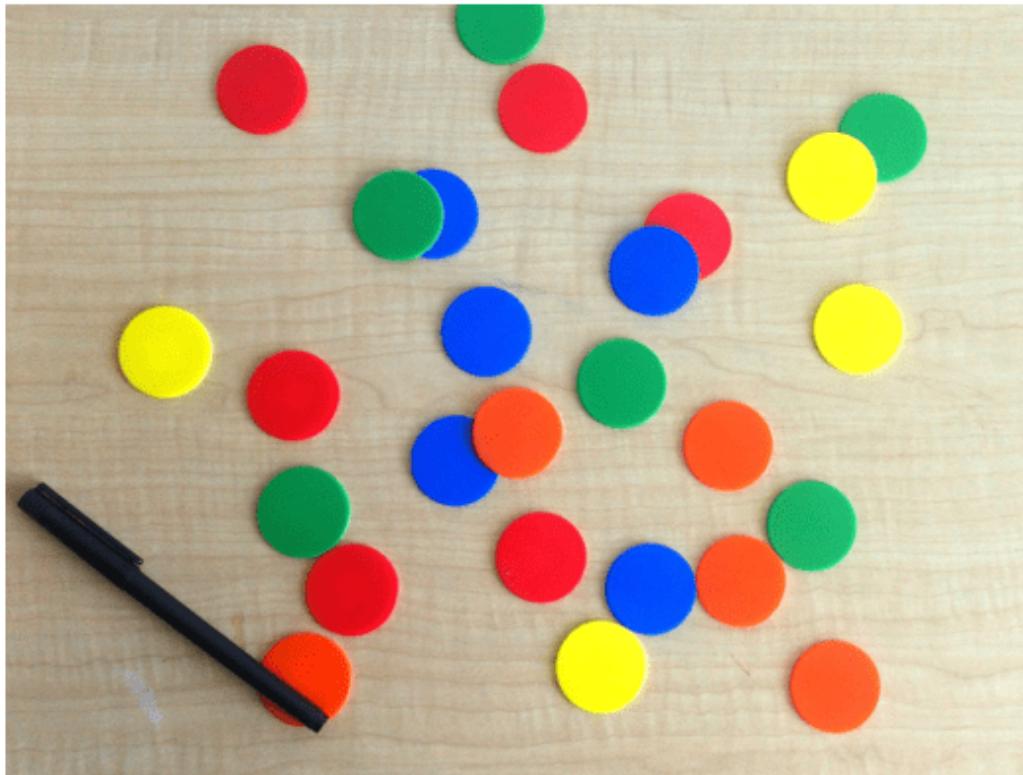
此示例说明如何自动检测图像中的圆或圆形目标并可视化检测到的圆。

步骤 1：加载图像

读取并显示包含各种颜色的圆形塑料片的图像。除了有大量要检测的圆之外，从圆检测的角度来看，此图像还有一些有趣的特点：

- 1 有不同颜色的塑料片，它们相对于背景有不同对比度。一方面，蓝色和红色塑料片在此背景上形成强烈的对比。另一方面，一些黄色塑料片与背景的对比不明显。
- 2 请注意一些塑料片重叠在一起，而另一些塑料片则靠得很近，几乎互相接触。对于目标检测来说，场景中存在重叠的对象边缘和对象遮挡通常具有挑战性。

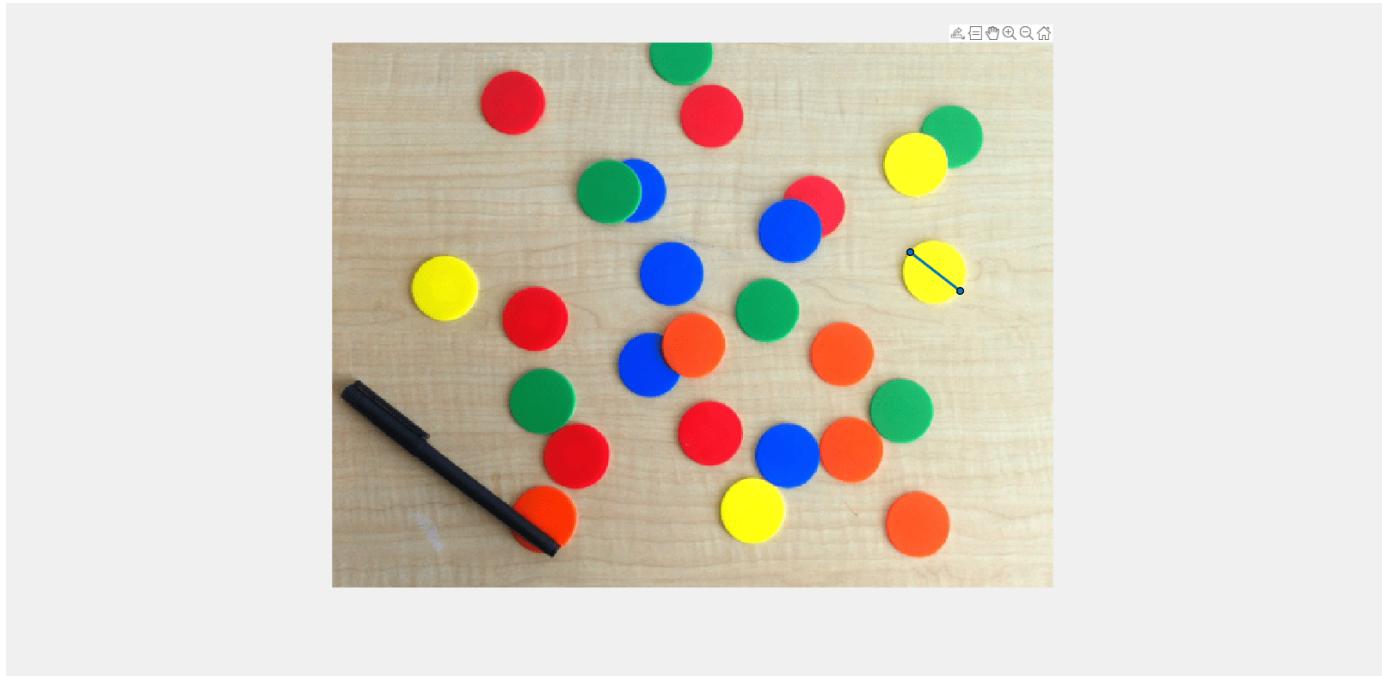
```
rgb = imread('coloredChips.png');  
imshow(rgb)
```



步骤 2：确定搜索圆的半径范围

使用 `drawline` 函数找到合适的圆半径范围。在塑料片的近似直径上绘制一条线。

```
d = drawline;
```



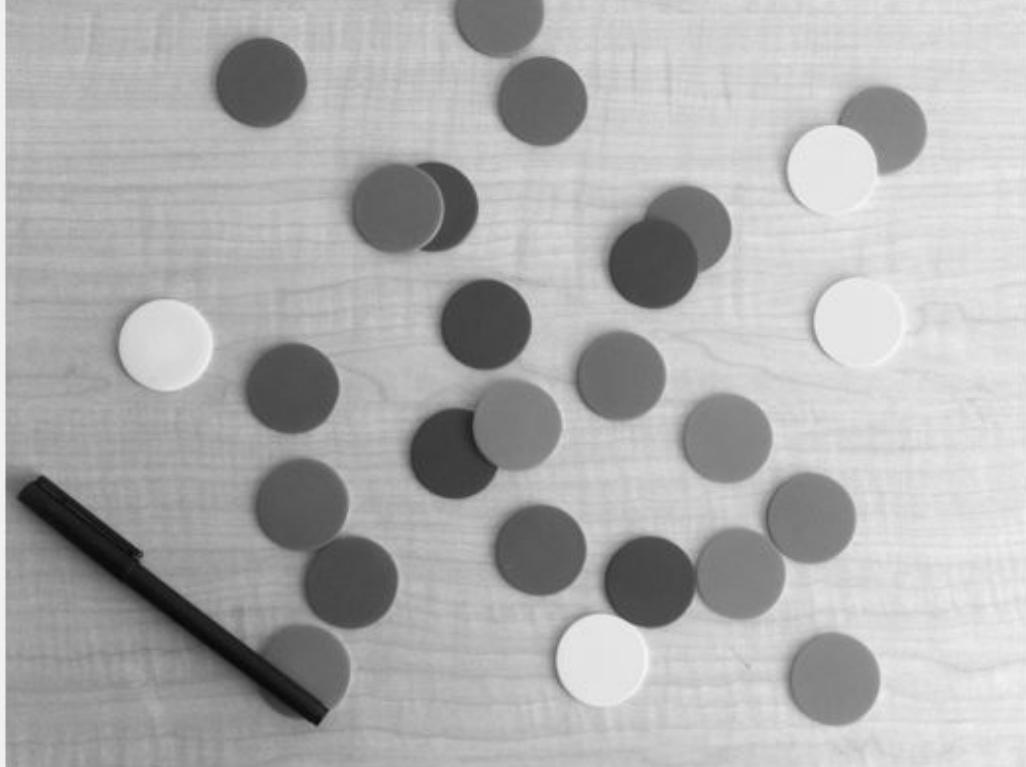
线 ROI 的长度就是塑料片的直径。通常的塑料片的直径在 40 到 50 个像素的范围内。

```
pos = d.Position;  
diffPos = diff(pos);  
diameter = hypot(diffPos(1),diffPos(2))  
  
diameter = 45.3448
```

步骤 3：寻找圆的初步尝试

`imfindcircles` 函数搜索符合半径范围的圆。搜索半径在 20 到 25 个像素范围内的圆。在此之前，最好要清楚对象是比背景亮还是比背景暗。要回答该问题，请看此图像的灰度版本。

```
gray_image = rgb2gray(rgb);  
imshow(gray_image)
```



背景相当亮，大多数塑料片比背景暗。但是，默认情况下，`imfindcircles` 会找到比背景亮的圆形目标。因此，在 `imfindcircles` 中将参数 '`ObjectPolarity`' 设置为 '`dark`' 以搜索较暗的圆。

```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark')
```

```
centers =
```

```
[]
```

```
radii =
```

```
[]
```

请注意，输出 `centers` 和 `radii` 为空，这意味着未找到圆。这种情况经常发生，因为 `imfindcircles` 是圆形检测器，与大多数检测器类似，`imfindcircles` 有内部检测阈值决定其敏感度。简而言之，这意味着检测器对某个（圆形）检测的信心必须大于某个水平，才将其视为有效检测。`imfindcircles` 有参数 '`Sensitivity`'，可用于控制此内部阈值，从而控制算法的敏感度。较高的 '`Sensitivity`' 值会将检测阈值设置得较低，并导致检测到更多圆。这类似于家庭安全系统中使用的运动检测器的敏感度控制。

步骤 4：提高检测敏感度

回到塑料片图像，在默认敏感度水平下，可能所有圆都低于内部阈值，因此未检测到圆。`'Sensitivity'` 是介于 0 和 1 之间的数字，默认设置为 0.85。将 `'Sensitivity'` 提高到 0.9。

```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark', ...
    'Sensitivity',0.9)
```

```
centers = 8×2
```

```
146.1895 198.5824
328.8132 135.5883
130.3134 43.8039
175.2698 297.0583
312.2831 192.3709
327.1316 297.0077
243.9893 166.4538
271.5873 280.8920
```

```
radii = 8×1
```

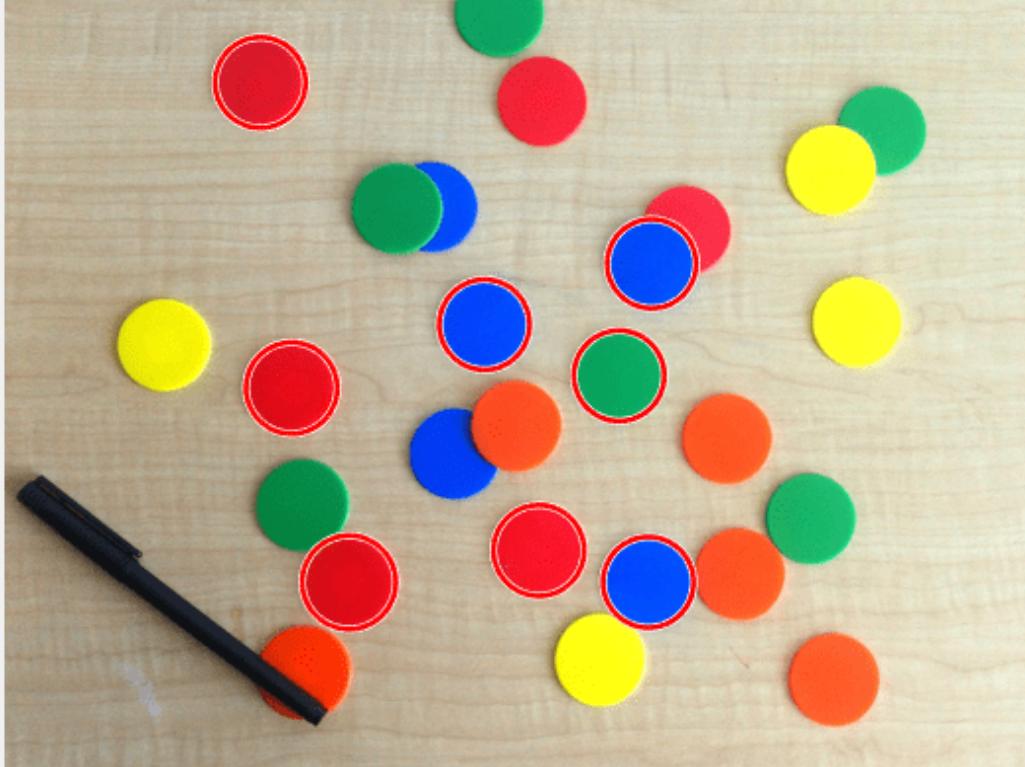
```
23.1604
22.5710
22.9576
23.7356
22.9551
22.9995
22.9055
23.0298
```

这次，`imfindcircles` 发现了一些圆 - 准确地说是八个。`centers` 包含圆心的位置，`radii` 包含这些圆的估计半径。

步骤 5：在图像上绘制圆

函数 `viscircles` 可用于在图像上绘制圆。来自 `imfindcircles` 的输出变量 `centers` 和 `radii` 可以直接传递给 `viscircles`。

```
imshow(rgb)
h = viscircles(centers,radii);
```



圆心似乎定位正确，它们对应的半径似乎与实际塑料片匹配良好。但仍未检测到相当多的塑料片。请尝试将 'Sensitivity' 提高到 0.92。

```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark',...
    'Sensitivity',0.92);
```

```
length(centers)
```

```
ans = 16
```

增加 'Sensitivity' 会让我们找到更多圆。再次在图像上绘制这些圆。

```
delete(h) % Delete previously drawn circles
h = viscircles(centers,radii);
```

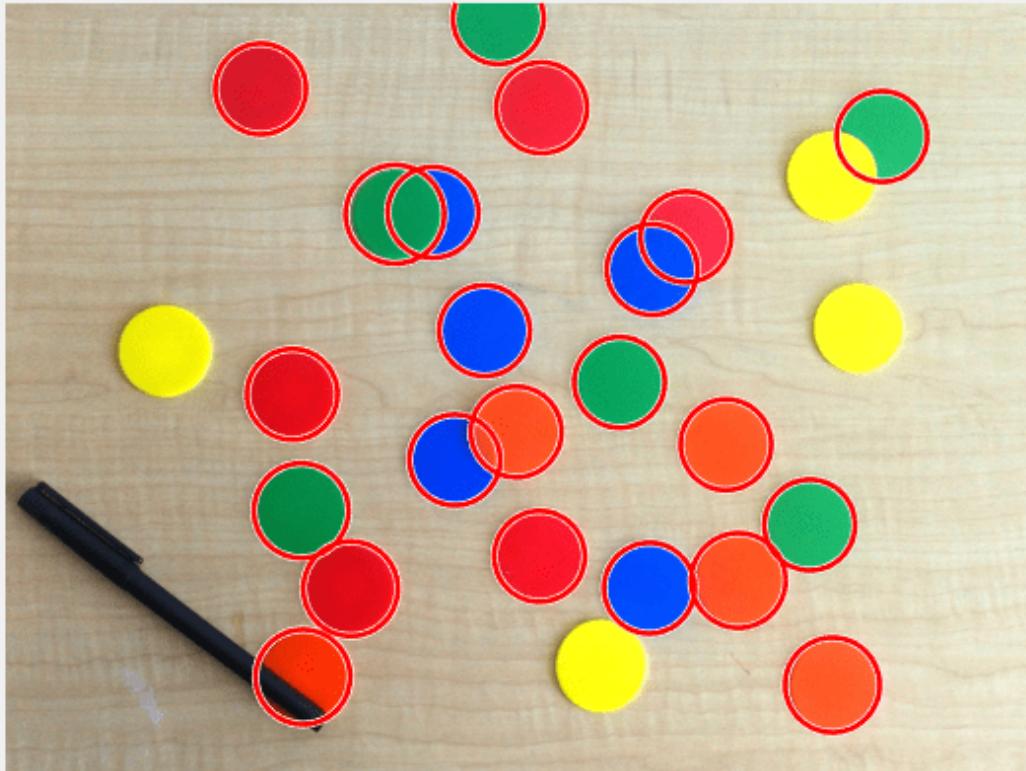


步骤 6：使用第二种方法（两阶段）寻找圆

此方法的结果看起来更好。imfindcircles 有两种不同寻找圆的方法。到目前为止，默认方法（称为相位编码方法）用于检测圆。在 imfindcircles 中还可以使用另一种方法，通常称为两阶段方法。使用两阶段方法并显示结果。

```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark',...
    'Sensitivity',0.92,'Method','twostage');

delete(h)
h = viscircles(centers,radii);
```



两阶段方法使用敏感度 0.92 检测更多圆。一般来说，这两种方法是互补的，因为它们有不同优点。相位编码方法通常比两阶段方法更快，抗噪声的稳定性稍强。但是，它也可能需要更高的 'Sensitivity' 水平才能实现与两阶段方法相同数量的检测。例如，如果 'Sensitivity' 水平提高到 0.95，相位编码方法也会找到相同的塑料片。

```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark',...
    'Sensitivity',0.95);

delete(h)
viscircles(centers,radii);
```

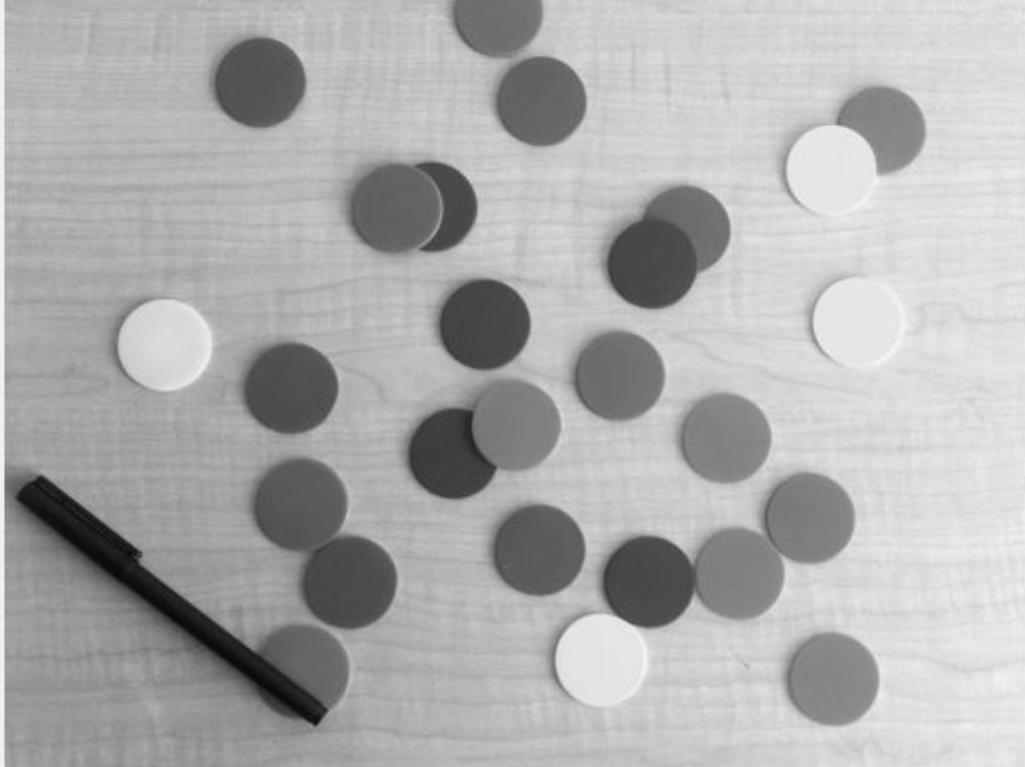


请注意，imfindcircles 中的两种方法都能准确找到部分可见（遮挡）塑料片的中心和半径。

步骤 7：为什么有些圆仍检测不到？

查看最后一个结果，很奇怪 imfindcircles 没有在图像中找到黄色塑料片。黄色塑料片与背景的对比不够强烈。事实上，它们看起来和背景的强度非常相似。是不是黄色塑料片并没有想象中的那样比背景“更暗”？要确认这一点，请再次显示该图像的灰度版本。

```
imshow(gray_image)
```



步骤 8：在图像中找到“明亮”的圆

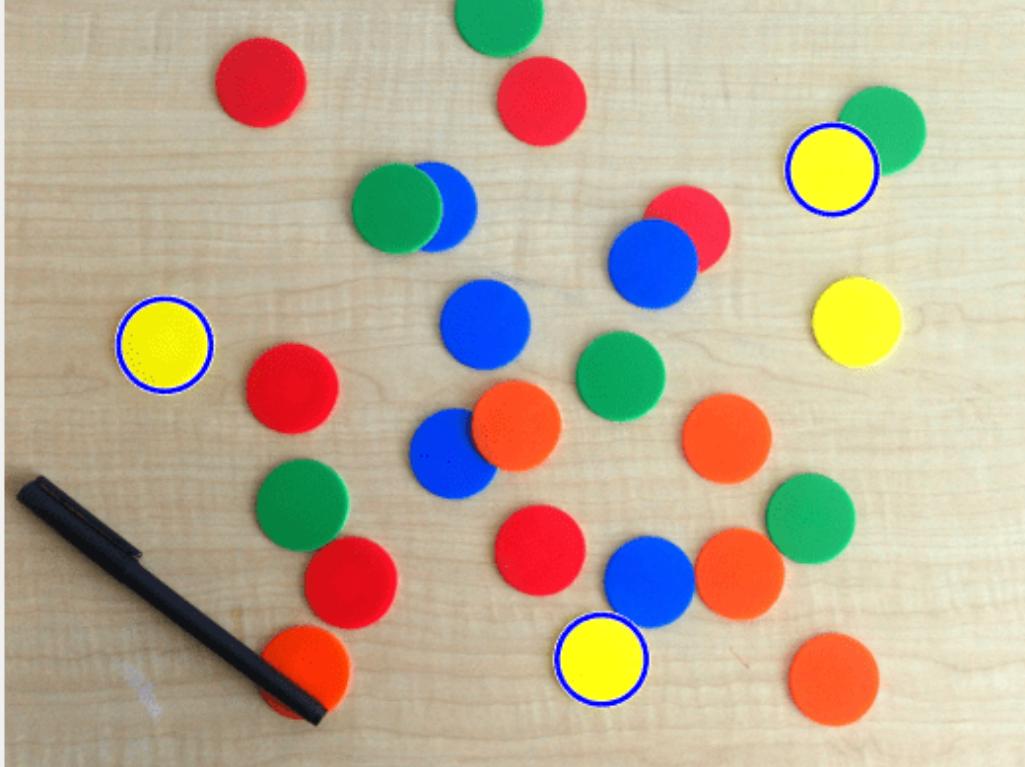
与背景相比，黄色塑料片的强度几乎相同，甚至更亮。因此，要检测黄色塑料片，请将 'ObjectPolarity' 更改为 'bright'。

```
[centersBright,radiiBright] = imfindcircles(rgb,[20 25], ...
    'ObjectPolarity','bright','Sensitivity',0.92);
```

步骤 9：用不同颜色绘制 'Bright' 圆

通过更改 `viscircles` 中的 'Color' 参数，以不同颜色绘制明亮的圆。

```
imshow(rgb)
hBright = viscircles(centersBright, radiiBright,'Color','b');
```



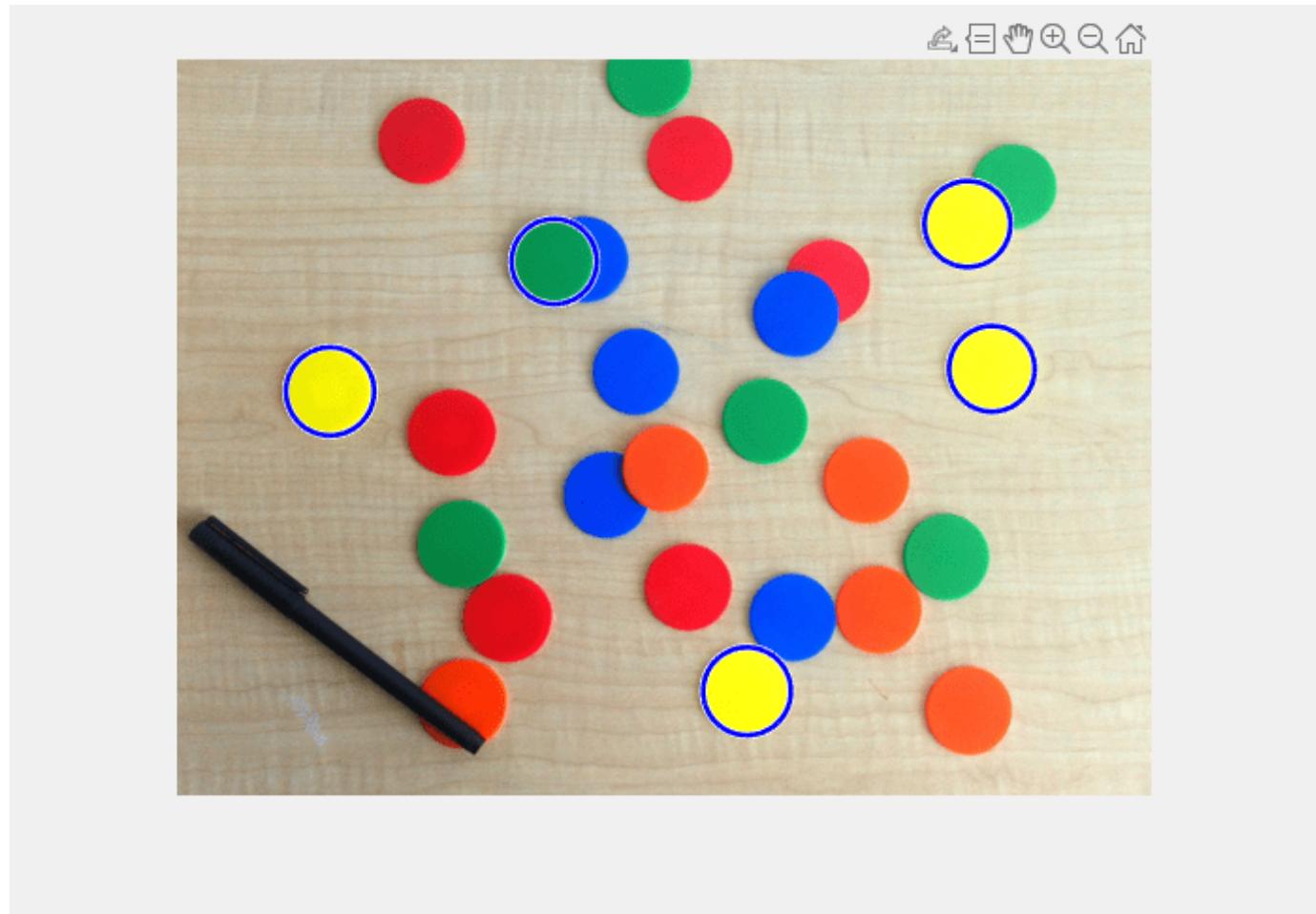
请注意，找到了三个原先未检测到的黄色塑料片，但仍有黄色塑料片未检测到。这些黄色塑料片很难检测到，因为在这种背景下，它们没有呈现出与众不同。

步骤 10：降低 'EdgeThreshold' 的值

在这里还可以使用 `imfindcircles` 中的另一个参数，即 '`EdgeThreshold`'。要查找圆，`imfindcircles` 仅使用图像中的边缘像素。这些边缘像素基本上是具有高梯度值的像素。`'EdgeThreshold'` 参数控制像素的梯度值必须有多高，才能将其视为边缘像素并包含在计算中。该参数的高值（更接近 1）只允许包含强边缘（较高梯度值），而低值（更接近 0）的宽容度更高，可在计算中包含较弱的边缘（较低梯度值）。对于检测不到黄色塑料片的情况，是因为对比度低，一些边界像素（在塑料片的圆周上）预期具有低梯度值。因此，请降低 '`EdgeThreshold`' 参数，以确保黄色塑料片的大多数边缘像素都包含在计算中。

```
[centersBright,radiiBright,metricBright] = imfindcircles(rgb,[20 25], ...
    'ObjectPolarity','bright','Sensitivity',0.92,'EdgeThreshold',0.1);

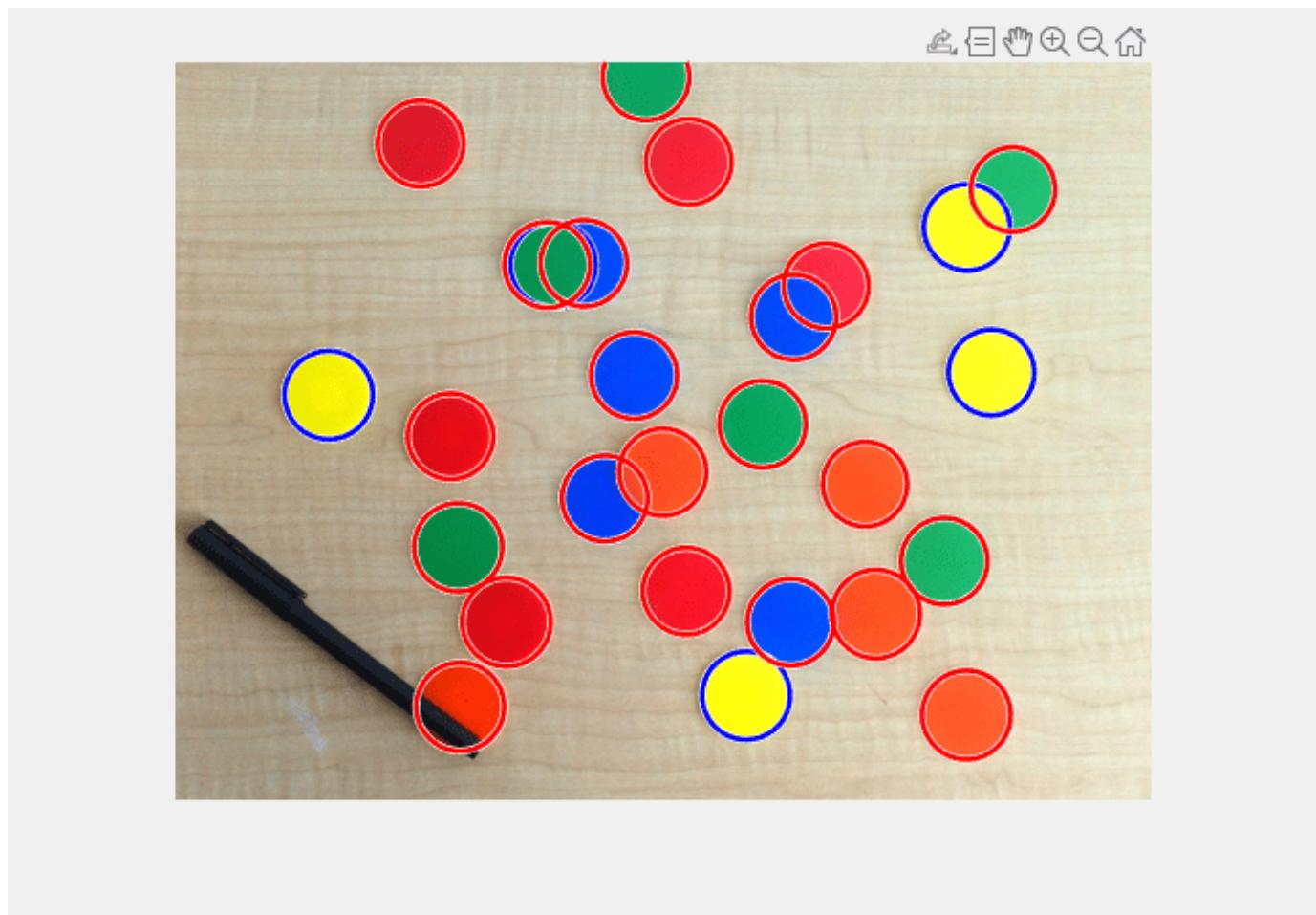
delete(hBright)
hBright = viscircles(centersBright, radiiBright,'Color','b');
```



步骤 11：同时绘制“暗”和“亮”圆

现在 `imfindcircles` 找到了所有黄色圆，还找到了一个绿色圆。用蓝色绘制这些塑料片，用红色绘制之前发现的其他塑料片 ('ObjectPolarity' 设置为 'dark') 。

```
h = viscircles(centers,radii);
```



所有圆都被检测到。最后 - 应注意，在检测中更激进地更改参数可能会发现更多圆，但也会增加检测到假圆的可能性。需要在可找到的真圆数量（检测率）和用它们找到的假圆数量（虚警率）之间实现某种平衡。

祝您能顺利检测到圆！

另请参阅

[imfindcircles](#) | [viscircles](#)

相关示例

- “[标识圆形目标](#)” (第 11-18 页)
- “[Measuring the Radius of a Roll of Tape](#)”

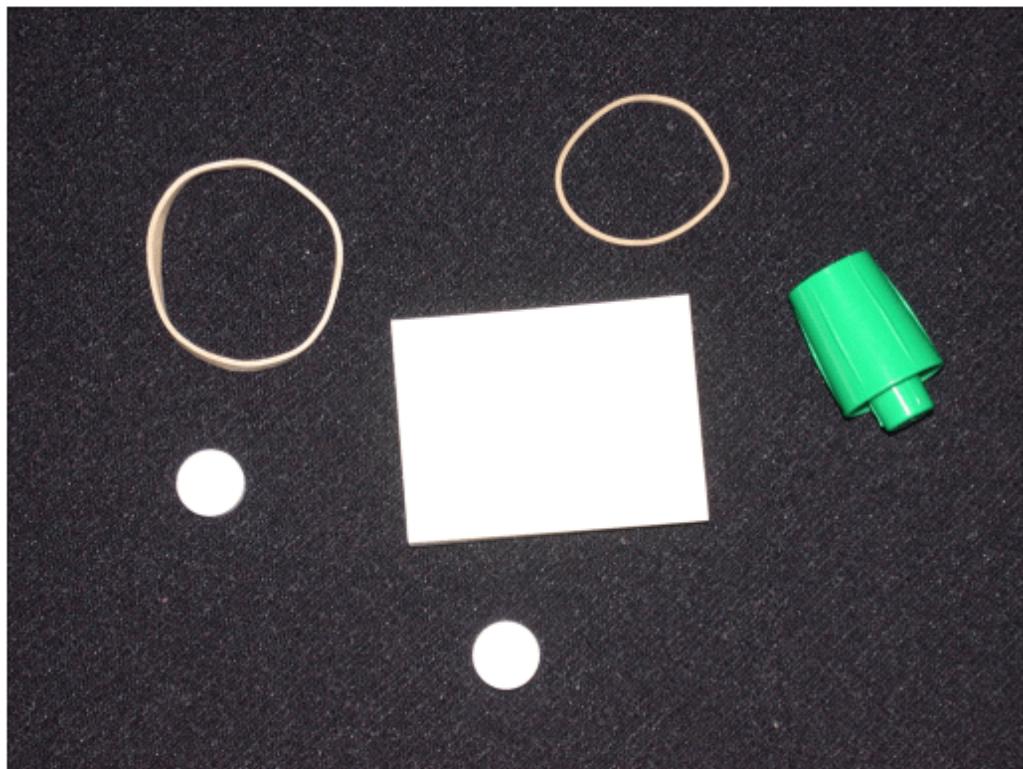
标识圆形目标

此示例说明如何使用边界跟踪例程 **bwboundaries** 根据对象的圆度对其进行分类。

步骤 1：读取图像

在 **pills/etc.png** 中进行读取。

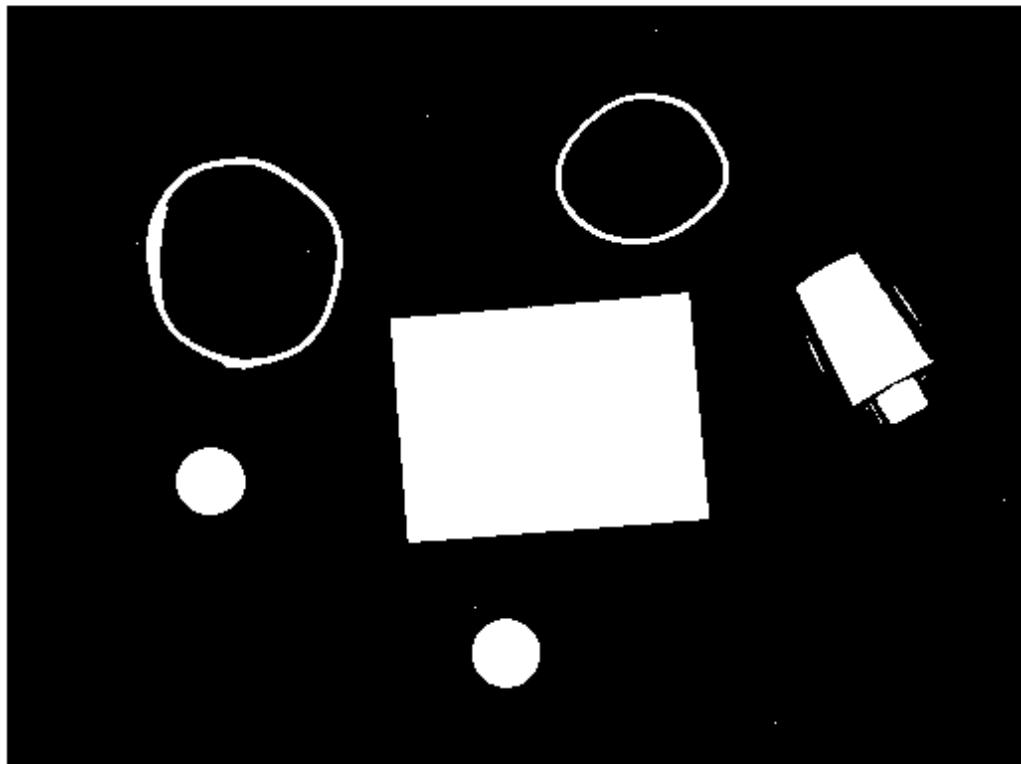
```
RGB = imread('pillsetc.png');  
imshow(RGB)
```



步骤 2：阈值化图像

将图像转换为黑白，以便使用 **bwboundaries** 为边界跟踪做准备。

```
I = rgb2gray(RGB);  
bw = imbinarize(I);  
imshow(bw)
```

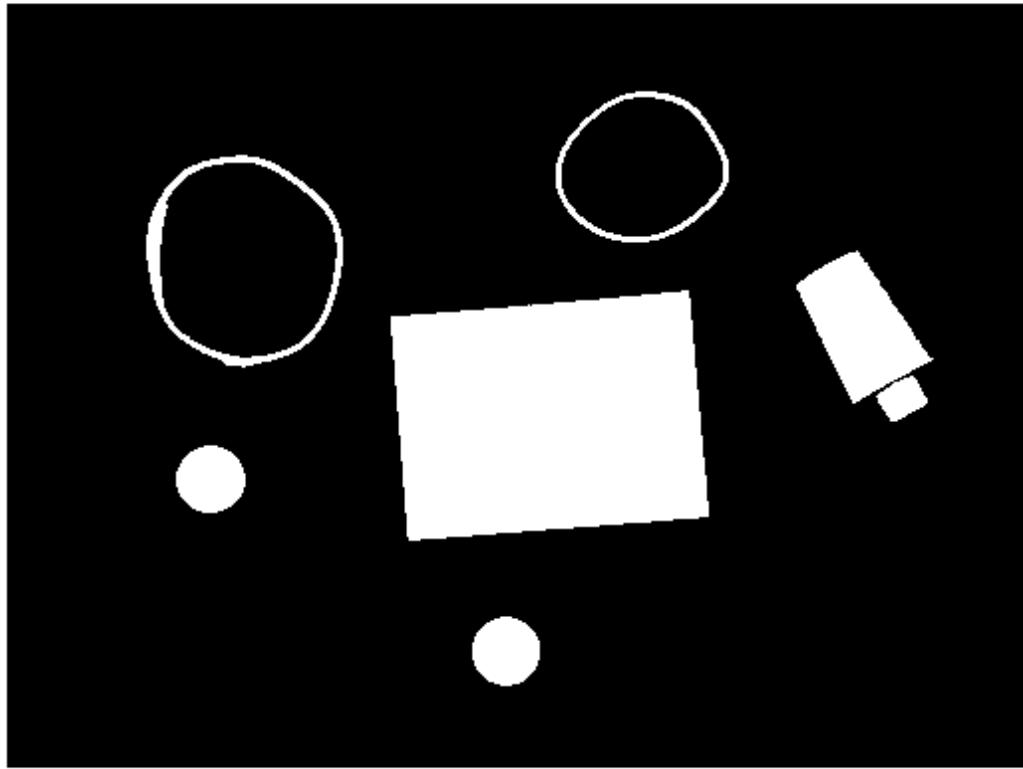


步骤 3：去除噪声

使用形态学函数，删除不属于关注对象的像素。

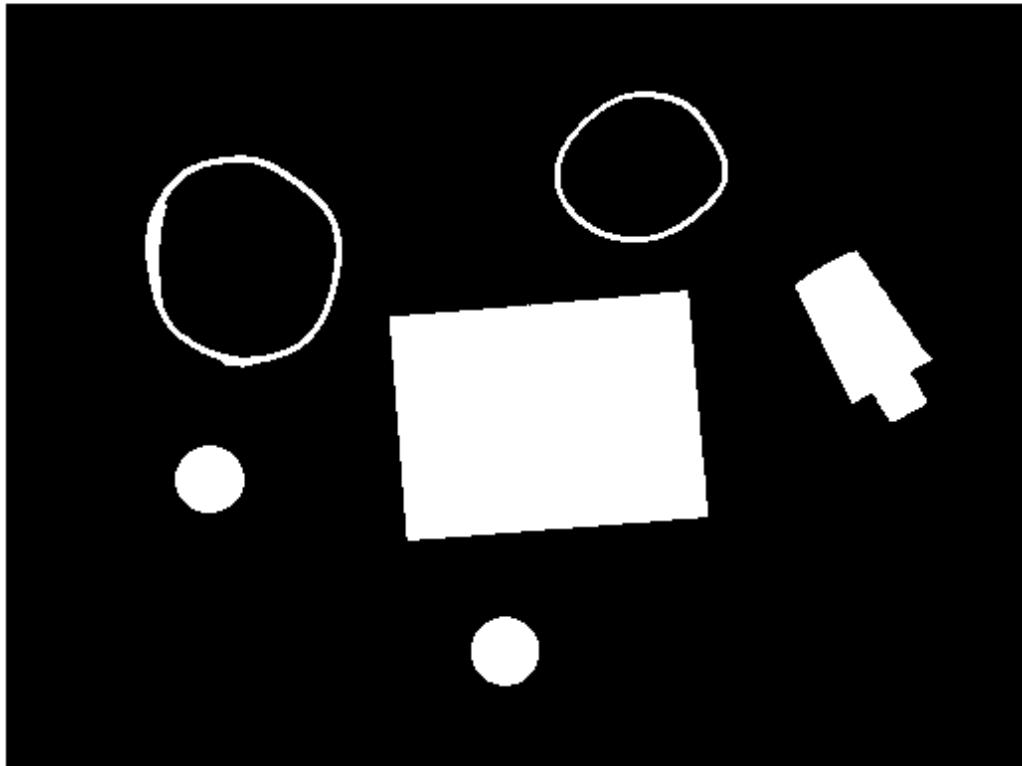
删除包含少于 30 个像素的所有对象。

```
bw = bwareaopen(bw,30);  
imshow(bw)
```



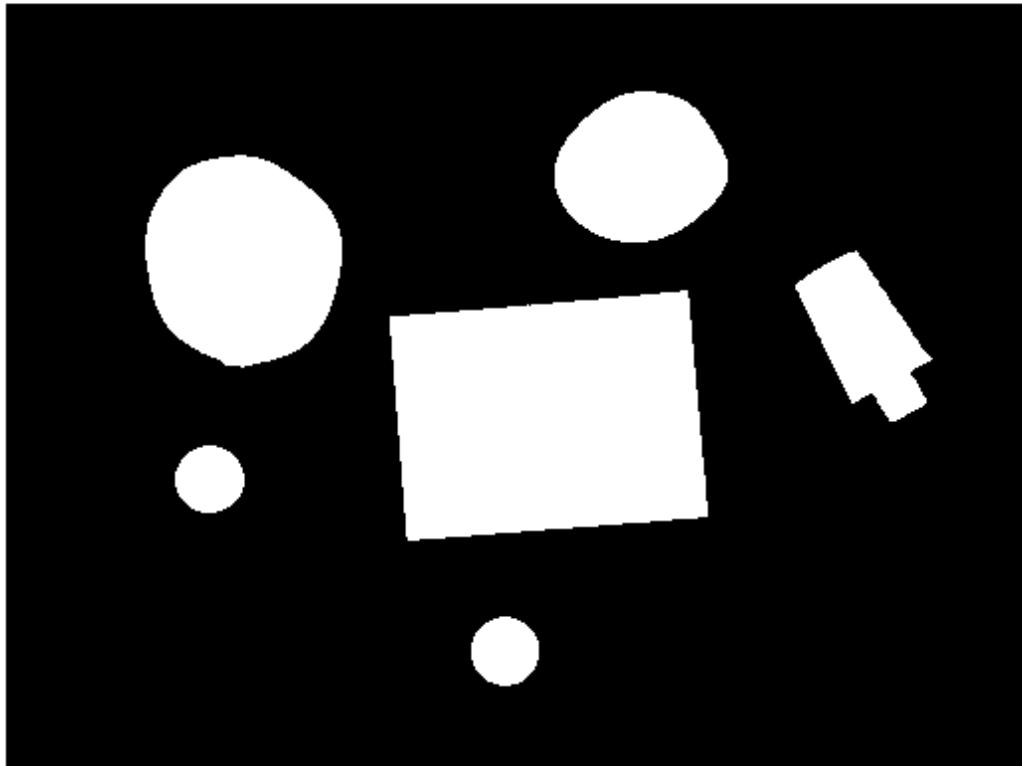
填充笔帽中的间隙。

```
se = strel('disk',2);
bw = imclose(bw,se);
imshow(bw)
```



填充任何孔洞，以便可以使用 regionprops 来估计每个边界所包围的面积

```
bw = imfill(bw,'holes');  
imshow(bw)
```



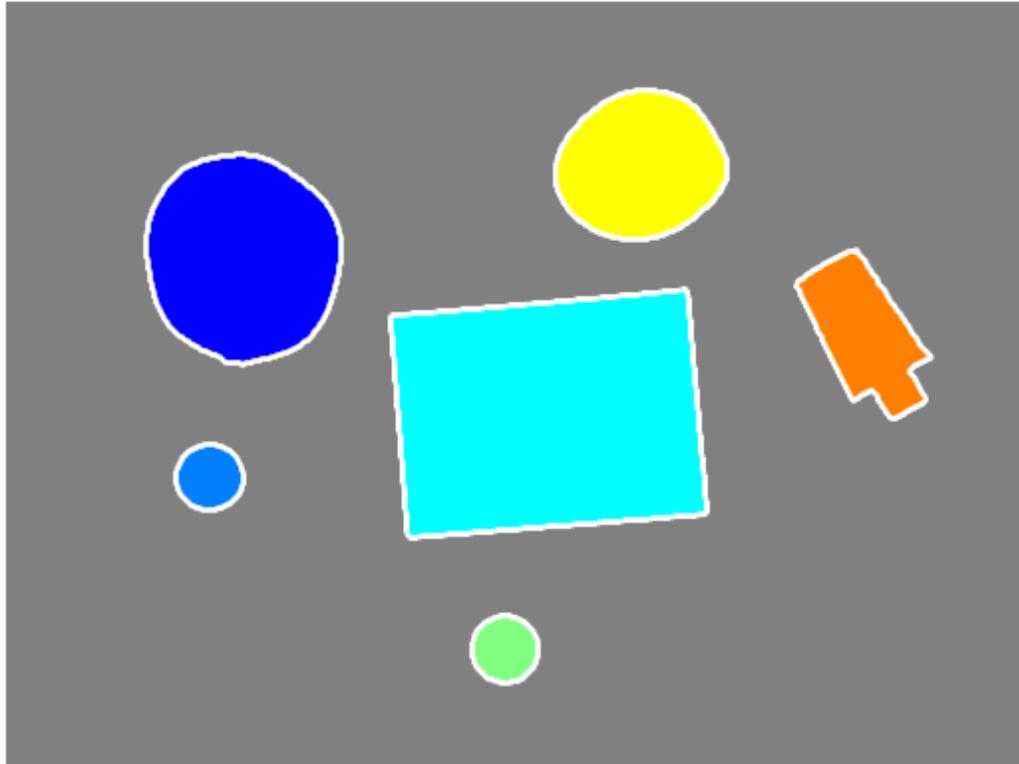
步骤 4：找到边界

只关注外边界。选项 'noholes' 将通过阻止 **bwboundaries** 搜索内部轮廓来加快处理速度。

```
[B,L] = bwboundaries(bw,'noholes');
```

显示标签矩阵并绘制每个边界。

```
imshow(label2rgb(L,@jet,[.5 .5 .5]))
hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2),boundary(:,1),'w','LineWidth',2)
end
```



步骤 5：确定哪些对象为圆形

估计每个对象的面积和周长。使用这些结果形成简单的度量来表示对象的圆度：

$$\text{metric} = \frac{4\pi * \text{area}}{\text{perimeter}^2}$$

只有对于圆，该度量值等于 1；而对于任何其他形状，该度量值都小于 1。可以通过设置适当的阈值来控制判别过程。此示例中使用 0.94 的阈值，以便只将药丸分类为圆形。

使用 **regionprops** 获得所有对象的面积估计值。请注意，**bwboundaries** 返回的标签矩阵可以被 **regionprops** 重用。

```
stats = regionprops(L,'Area','Centroid');  
threshold = 0.94;  
  
% loop over the boundaries  
for k = 1:length(B)  
  
    % obtain (X,Y) boundary coordinates corresponding to label 'k'  
    boundary = B{k};
```

```
% compute a simple estimate of the object's perimeter
delta_sq = diff(boundary).^2;
perimeter = sum(sqrt(sum(delta_sq,2)));

% obtain the area calculation corresponding to label 'k'
area = stats(k).Area;

% compute the roundness metric
metric = 4*pi*area/perimeter^2;

% display the results
metric_string = sprintf('%2.2f',metric);

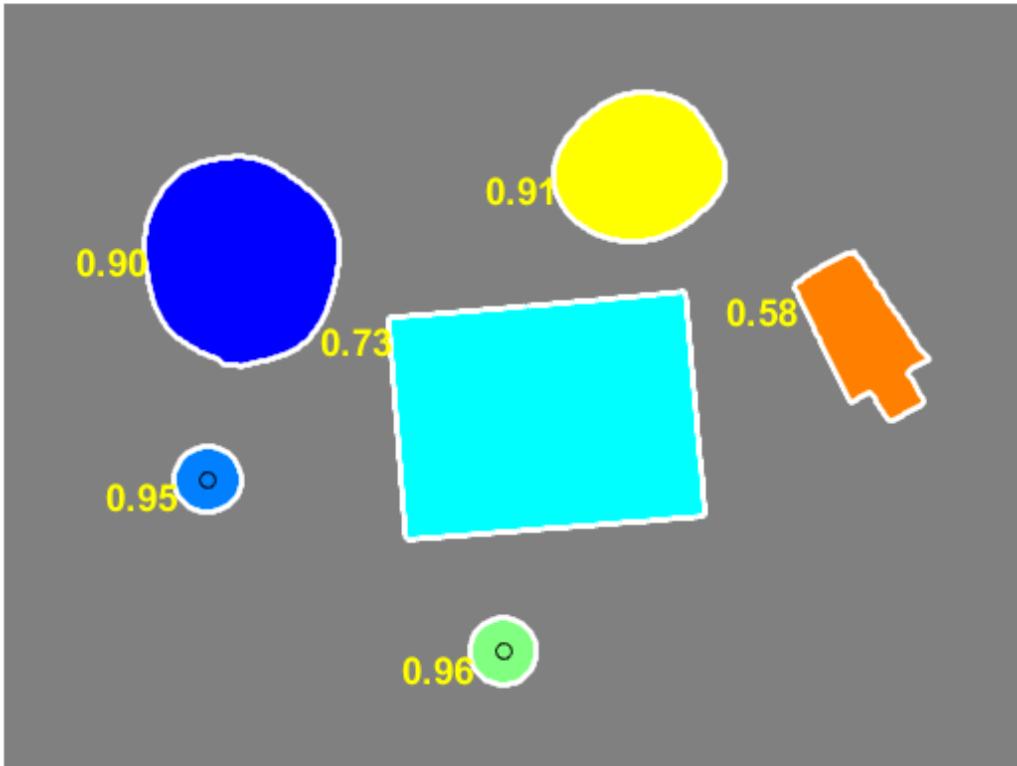
% mark objects above the threshold with a black circle
if metric > threshold
    centroid = stats(k).Centroid;
    plot(centroid(1),centroid(2),'ko');
end

text(boundary(1,2)-35,boundary(1,1)+13,metric_string,'Color','y',...
    'FontSize',14,'FontWeight','bold')

end

title(['Metrics closer to 1 indicate that',...
    'the object is approximately round'])
```

Metrics closer to 1 indicate that the object is approximately round



另请参阅

[bwboundaries](#) | [imbinarize](#) | [bwareaopen](#) | [imclose](#) | [strel](#) | [imfill](#) | [label2rgb](#) | [regionprops](#)

相关示例

- “检测和测量图像中的圆形目标” (第 11-6 页)
- “Measuring the Radius of a Roll of Tape”

使用灰度共生矩阵 (GLCM) 的纹理分析

一种考虑像素空间关系的纹理检查统计方法是灰度共生矩阵 (GLCM)，也称为灰度级空间依赖性矩阵。

GLCM 函数通过计算具有特定值和特定空间关系的像素对组在图像中出现的频率来表征图像的纹理，从而创建 GLCM，然后从该矩阵中提取统计度量。（“Calculate Statistical Measures of Texture” 中所述的纹理滤波器函数无法提供关于形状的信息，即图像中像素的空间关系。）

在使用 `graycomatrix` 创建 GLCM 后，您可以使用 `graycoprops` 从它们中派生几个统计量。这些统计量提供关于图像纹理的信息。下表列出了统计量。

统计量	说明
对比度	测量灰度共生矩阵的局部变化。
相关性	测量指定像素对组出现的联合概率。
能量	提供 GLCM 中平方元素的总和。也称为均匀性或角二阶矩。
同质性	测量 GLCM 中的元素分布与 GLCM 对角线的接近程度。

另请参阅

相关示例

- “Derive Statistics from GLCM and Plot Correlation”

详细信息

- “创建灰度共生矩阵”（第 11-27 页）

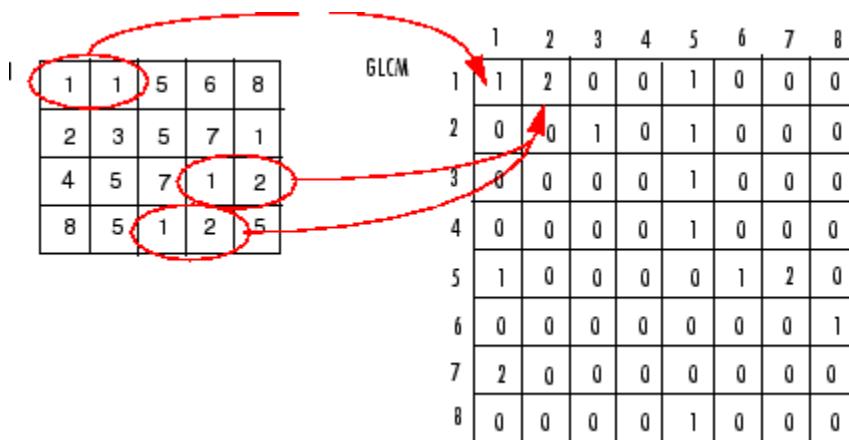
创建灰度共生矩阵

要创建 GLCM，[请使用 `graycomatrix`](#) 函数。该函数通过计算具有强度（灰度）值 i 的像素与具有值 j 的像素在特定空间关系中出现的频率，创建灰度共生矩阵 (GLCM)。默认情况下，空间关系定义为感兴趣的像素与其右侧紧邻的像素（水平相邻），但您可以指定两个像素之间的其他空间关系。生成的 `glcm` 中的每个元素 (i,j) 均为输入图像中值为 i 的像素与值为 j 的像素在指定空间关系中出现的次数之和。

图像中灰度级的数量确定 GLCM 的大小。默认情况下，`graycomatrix` 使用缩放将图像中的强度值数目减少到 8，但您可以使用 `NumLevels` 和 `GrayLimits` 参数来控制灰度级的缩放。有关详细信息，请参阅 [graycomatrix 参考页](#)。

灰度共生矩阵可以揭示有关纹理图像中灰度级空间分布的某些属性。例如，如果 GLCM 中的大多数条目都集中在对角线上，则纹理相对于指定的偏移量是粗糙的。您还可以从 GLCM 中派生几个统计度量。有关详细信息，请参阅 “Derive Statistics from GLCM and Plot Correlation”。

为了说明，下图显示 `graycomatrix` 如何计算 GLCM 中的前三个值。在输出 GLCM 中，元素 $(1,1)$ 包含值 1，因为在输入图像中两个水平相邻的像素分别具有值 1 和 1 的情况只有一处。 $glcm(1,2)$ 包含值 2，因为两个水平相邻的像素分别具有值 1 和 2 的情况有两处。GLCM 中的元素 $(1,3)$ 的值为 0，因为不存在两个水平相邻像素的值分别为 1 和 3 的情况。`graycomatrix` 继续处理输入图像，扫描图像中的其他像素对组 (i,j) 并将总和记录在 GLCM 的对应元素中。



用于创建 GLCM 的过程

对比度增强方法

此示例说明几种图像增强方法。以下三个函数特别适用于对比度增强：**imadjust**、**histeq** 和 **adapthisteq**。此示例比较它们在增强灰度和真彩色图像方面的用法。

增强灰度图像

使用默认设置，比较以下三种方法的有效性：

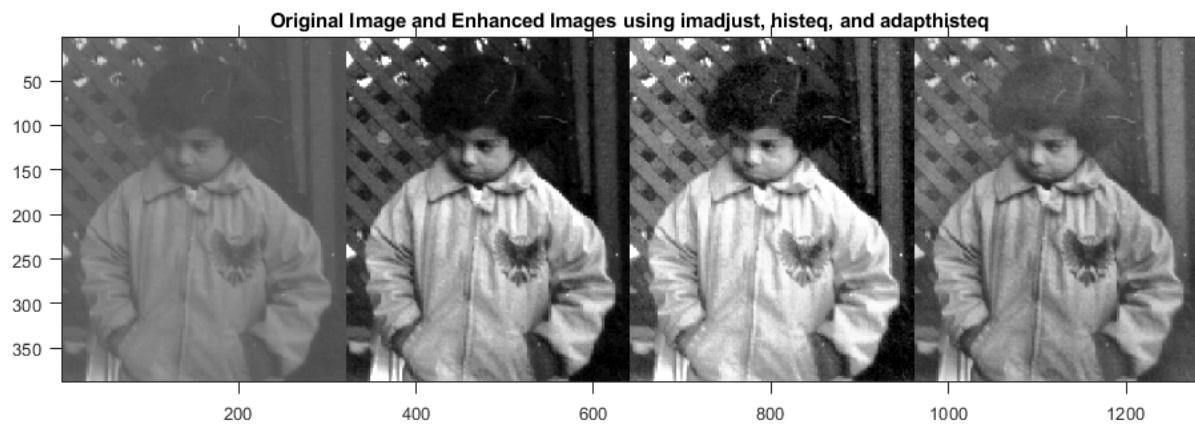
- **imadjust** 将输入强度图像的值映射到新值，以对输入数据中强度最低和最高的 1%（默认值）数据进行饱和处理，从而提高图像的对比度。
- **histeq** 执行直方图均衡化。它变换强度图像中的值，以使输出图像的直方图近似匹配指定的直方图（默认情况下为均匀分布），从而增强图像的对比度。
- **adapthisteq** 执行对比度受限的自适应直方图均衡化。与 **histeq** 不同，它对小数据区域（图块）而不是整个图像执行运算。它会增强每个图块的对比度，使得每个输出区域的直方图近似匹配指定的直方图（默认情况下为均匀分布）。可以限制对比度增强，以避免放大图像中可能存在的噪声。

将灰度图像读入工作区。使用三种对比度调整方法增强图像。

```
pout = imread('pout.tif');
pout_imadjust = imadjust(pout);
pout_histeq = histeq(pout);
pout_adapthisteq = adapthisteq(pout);
```

以蒙太奇方式显示原始图像和三个对比度调整后的图像。

```
montage({pout,pout_imadjust,pout_histeq,pout_adapthisteq},[1 4])
title("Original Image and Enhanced Images using imadjust, histeq, and adapthisteq")
```

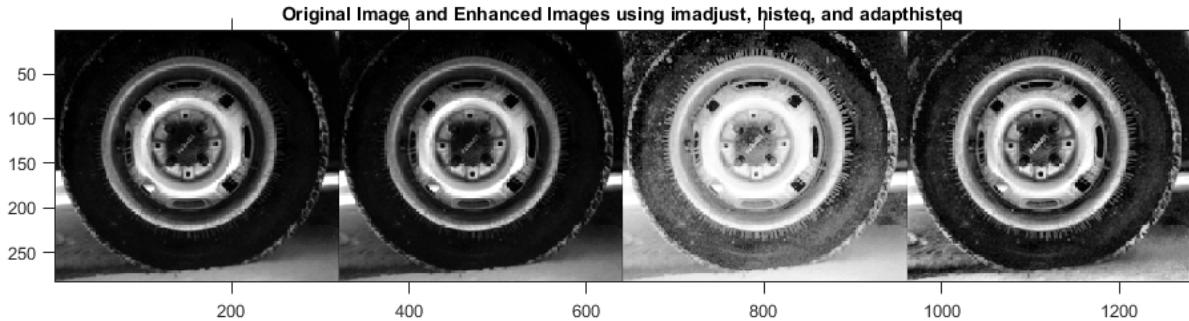


将另一个灰度图像读入工作区，并使用三种对比度调整方法增强该图像。

```
tire = imread('tire.tif');
tire_imadjust = imadjust(tire);
tire_histeq = histeq(tire);
tire_adapthisteq = adapthisteq(tire);
```

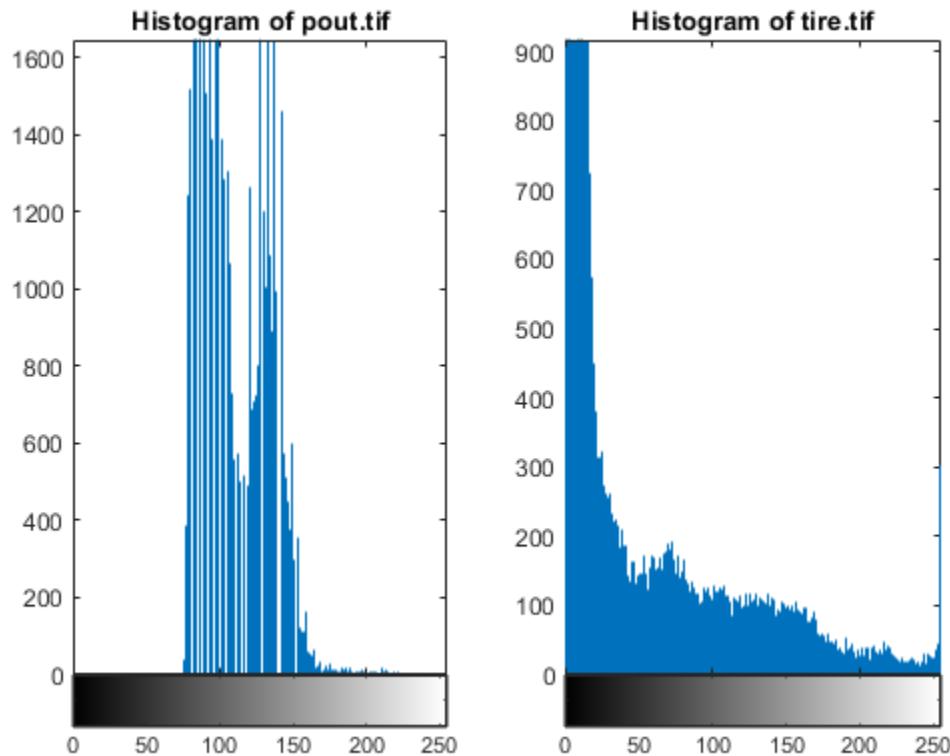
以蒙太奇方式显示原始图像和三个对比度调整后的图像。

```
montage({tire,tire_imadjust,tire_histeq,tire_adapthisteq},'Size',[1 4])
title("Original Image and Enhanced Images using imadjust, histeq, and adapthisteq")
```



请注意，**imadjust** 对轮胎的图像影响不大，但在 pout 图中产生了明显变化。绘制 **pout.tif** 和 **tire.tif** 的直方图表明，第一个图像中的大部分像素集中在直方图的中心，而对于 **tire.tif**，这些值已分布在最小值 0 和最大值 255 之间，从而阻止 **imadjust** 有效地调整图像的对比度。

```
figure
subplot(1,2,1)
imhist(pout)
title('Histogram of pout.tif')
subplot(1,2,2)
imhist(tire)
title('Histogram of tire.tif');
```



而直方图均衡化明显改变了两个图像。许多先前隐藏的特征暴露出来，尤其是轮胎上的碎屑颗粒。遗憾的是，这种增强同时使两个图像的几个区域都出现过饱和。注意轮胎的中心、孩子脸部的一部分和外套都泛白。

对于轮胎图像的处理，车轮的中心最好保持大约相同的亮度，同时增强图像其他区域的对比度。为了实现此点，必须对图像的不同部分应用不同变换。在 `adapthisteq` 中实现的对比度受限自适应直方图均衡化方法可以实现这一点。该算法分析图像的各部分并计算适当的变换。它还可以设置对比度增强水平的限制，从而防止由 `histeq` 的基本直方图均衡化方法引起的过饱和。这是此示例中最复杂的方法。

增强彩色图像

彩色图像的对比度增强通常是通过将图像转换为以图像光度作为其分量之一的颜色空间（例如 L*a*b* 颜色空间）来完成的。对比度调整仅对光度层 'L*' 执行，然后将图像转换回 RGB 颜色空间。操作光度只影响像素的强度，会保留原始颜色。

将图像读入工作区。`'shadow.tif'` 图像是索引图像，因此将图像转换为真彩色 (RGB) 图像。然后，将图像从 RGB 颜色空间转换为 L*a*b* 颜色空间。

```
[X,map] = imread('shadow.tif');
shadow = ind2rgb(X,map);
shadow_lab = rgb2lab(shadow);
```

光度值的范围是从 0 到 100。将值缩放到范围 [0 1]，这是数据类型为 `double` 的图像的预期范围。

```
max_luminosity = 100;
L = shadow_lab(:,:,1)/max_luminosity;
```

对光度通道执行三种类型的对比度调整，并保持 a* 和 b* 通道不变。将图像转换回 RGB 颜色空间。

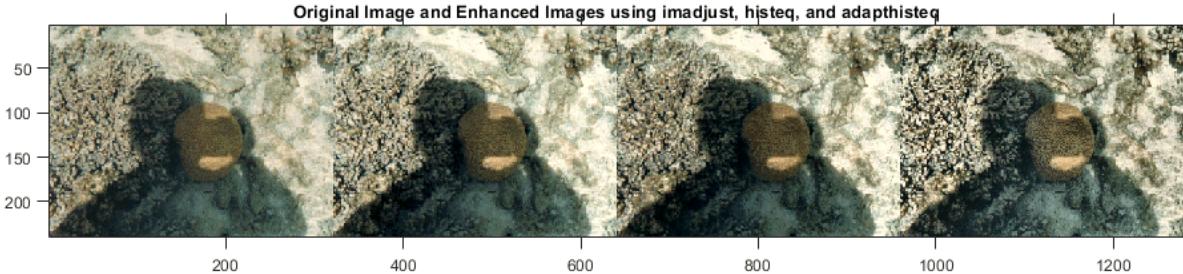
```
shadow_imadjust = shadow_lab;
shadow_imadjust(:,:,1) = imadjust(L)*max_luminosity;
shadow_imadjust = lab2rgb(shadow_imadjust);

shadow_histeq = shadow_lab;
shadow_histeq(:,:,1) = histeq(L)*max_luminosity;
shadow_histeq = lab2rgb(shadow_histeq);

shadow_adapthisteq = shadow_lab;
shadow_adapthisteq(:,:,1) = adapthisteq(L)*max_luminosity;
shadow_adapthisteq = lab2rgb(shadow_adapthisteq);
```

以蒙太奇方式显示原始图像和三个对比度调整后的图像。

```
figure
montage({shadow,shadow_imadjust,shadow_histeq,shadow_adapthisteq}, 'Size',[1 4])
title("Original Image and Enhanced Images using imadjust, histeq, and adapthisteq")
```



另请参阅

[imadjust](#) | [histeq](#) | [adapthisteq](#)

详细信息

- “Adaptive Histogram Equalization”
- “直方图均衡化”（第 11-32 页）

直方图均衡化

您可以使用直方图均衡化自动调整图像像素的强度值。直方图均衡化包括变换强度值，以使输出图像的直方图近似匹配指定的直方图。默认情况下，直方图均衡化函数 `histeq` 会尝试匹配到具有 64 个 bin 的平坦直方图，但您可以改为指定不同直方图。

使用直方图均衡化调整强度值

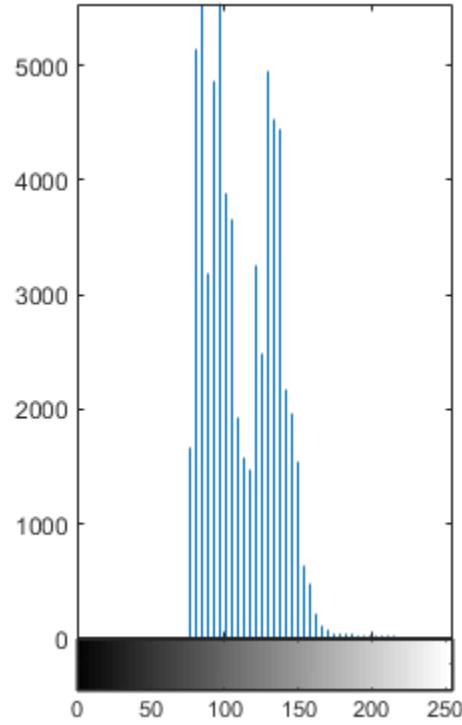
此示例说明如何使用直方图均衡化来调整灰度图像的对比度。原始图像对比度低，大多数像素值在强度范围的中间位置。`histeq` 生成一个输出图像，其像素值在完整范围内均匀分布。

将图像读入工作区。

```
I = imread('pout.tif');
```

显示图像及其直方图。

```
figure
subplot(1,2,1)
imshow(I)
subplot(1,2,2)
imhist(I,64)
```

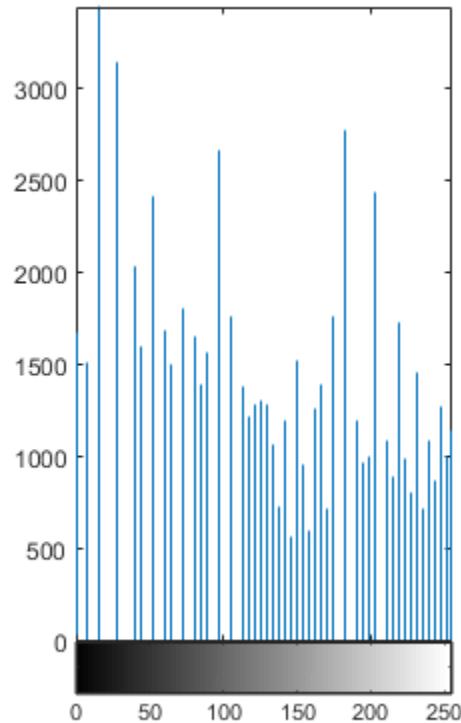


使用直方图均衡化调整对比度。在此示例中，直方图均衡化函数 `histeq` 尝试匹配到具有 64 个 bin 的平坦直方图，这是默认行为。您可以改为指定不同直方图。

```
J = histeq(I);
```

显示对比度调整后的图像及其新直方图。

```
figure
subplot(1,2,1)
imshow(J)
subplot(1,2,2)
imhist(J,64)
```



绘制直方图均衡化的变换曲线

此示例说明如何绘制直方图均衡化的变换曲线。`histeq` 可以返回 1×256 向量，该向量显示每个可能的输入值的结果输出值。不管输入图像是什么类型，此向量中的值都在 $[0,1]$ 范围内。您可以绘制这些数据来获得变换曲线。

将图像读入工作区。

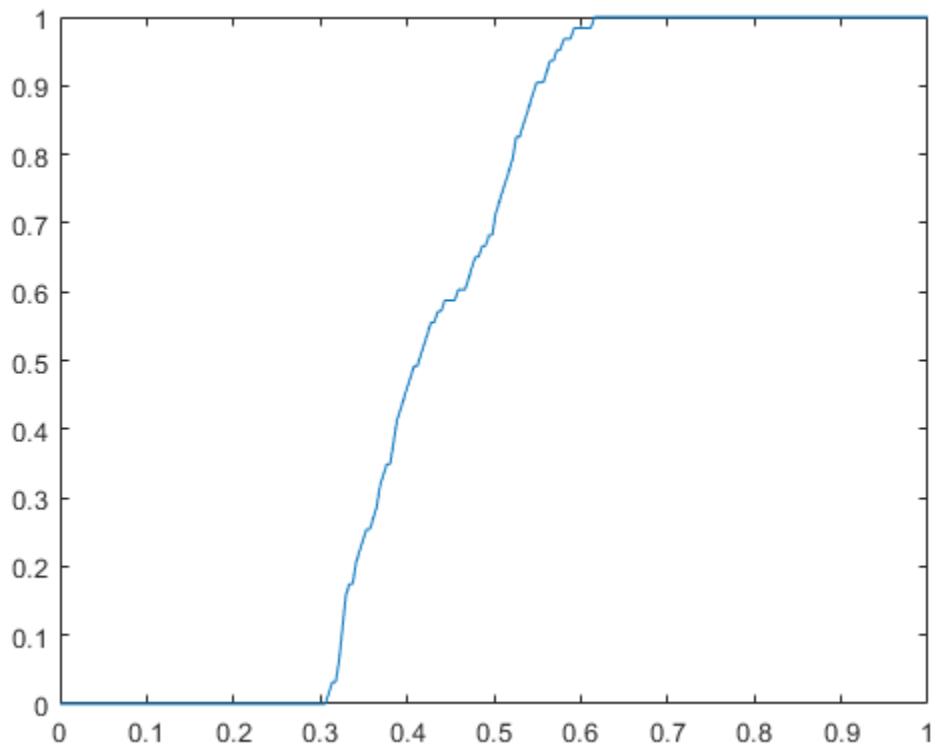
```
I = imread('pout.tif');
```

使用 `histeq` 函数通过直方图均衡化调整对比度。指定灰度变换返回值 `T`，它是一个向量，用于将强度图像 `I` 中的灰度级映射到 `J` 中的灰度级。

```
[J,T] = histeq(I);
```

绘制变换曲线。注意这条曲线如何反映上图中的直方图，输入值大多在 0.3 和 0.6 之间，而输出值均匀分布在 0 和 1 之间。

```
figure  
plot((0:255)/255,T);
```



将高斯平滑滤波器应用于图像

此示例说明如何使用 `imgaussfilt` 对图像应用不同高斯平滑滤波器。高斯平滑滤波器通常用于降低噪声。

将图像读入工作区。

```
I = imread('cameraman.tif');
```

用标准差递增的各向同性高斯平滑核对图像进行滤波。高斯滤波器通常为各向同性，也就是说，它们在两个维度上具有相同的标准差。通过为 `sigma` 指定标量值，可以用各向同性高斯滤波器对图像进行滤波。

```
Iblur1 = imgaussfilt(I,2);
Iblur2 = imgaussfilt(I,4);
Iblur3 = imgaussfilt(I,8);
```

显示原始图像和所有滤波后的图像。

```
figure
imshow(I)
title('Original image')
```



```
figure
imshow(Iblur1)
title('Smoothed image, \sigma = 2')
```

Smoothed image, $\sigma = 2$



```
figure  
imshow(Iblur2)  
title('Smoothed image, \sigma = 4')
```

Smoothed image, $\sigma = 4$



```
figure
```

```
imshow(Iblur3)
title('Smoothed image, \sigma = 8')
```



用各向异性高斯平滑核对图像进行滤波。`imgaussfilt` 允许高斯核在行和列维度上有不同标准差。这些称为轴对齐的各向异性高斯滤波器。使用各向异性滤波器时，请为 `sigma` 指定二元素向量。

```
IblurX1 = imgaussfilt(I,[4 1]);
IblurX2 = imgaussfilt(I,[8 1]);
IblurY1 = imgaussfilt(I,[1 4]);
IblurY2 = imgaussfilt(I,[1 8]);
```

显示滤波后的图像。

```
figure
imshow(IblurX1)
title('Smoothed image, \sigma_x = 4, \sigma_y = 1')
```

Smoothed image, $\sigma_x = 4, \sigma_y = 1$



```
figure  
imshow(IblurX2)  
title('Smoothed image, \sigma_x = 8, \sigma_y = 1')
```

Smoothed image, $\sigma_x = 8, \sigma_y = 1$



```
figure
```

```
imshow(IblurY1)
title('Smoothed image, \sigma_x = 1, \sigma_y = 4')
```

Smoothed image, $\sigma_x = 1$, $\sigma_y = 4$



```
figure
imshow(IblurY2)
title('Smoothed image, \sigma_x = 1, \sigma_y = 8')
```

Smoothed image, $\sigma_x = 1$, $\sigma_y = 8$



消除原始图像的天空区域中可见的水平条带。各向异性高斯滤波器可以消除图像中的水平或垂直特征。提取图像的一部分天空区域，并使用沿 X 轴（递增列的方向）具有较高标准差的高斯滤波器。

```
I_sky = imadjust(I(20:50,10:70));  
IblurX1_sky = imadjust(IblurX1(20:50,10:70));
```

显示原始的天空区块的以及经过滤波的版本。

```
figure  
imshow(I_sky), title('Sky in original image')
```

Sky in original image



```
figure  
imshow(IblurX1_sky), title('Sky in filtered image')
```

Sky in filtered image



去除噪声

本节内容

- “通过线性滤波去除噪声”（第 11-41 页）
- “使用平均值滤波器和中位数滤波器去除噪声”（第 11-41 页）
- “通过自适应滤波去除噪声”（第 11-44 页）

数字图像容易受到各种噪声的影响。噪声是图像采集过程中存在误差的结果，这些误差导致像素值无法反映真实场景的真实强度。根据图像的创建方式，图像引入噪声的方式有若干种。例如：

- 如果从用胶片拍摄的照片中扫描图像，胶片颗粒就是噪声源。噪声也可能源于胶片损坏，或由扫描仪本身引起。
- 如果图像是以数字格式直接获取的，则采集数据的机制（例如 CCD 检测器）可能会引入噪声。
- 图像数据的电子传输会引入噪声。

为了模拟上面列出的一些问题的影响，工具箱提供了 `imnoise` 函数，您可以使用该函数将各种类型的噪声添加到图像中。本节中的示例使用该函数。

通过线性滤波去除噪声

您可以使用线性滤波来去除某些类型的噪声。某些滤波器，如平均值滤波器或高斯滤波器，适用于此目的。例如，平均值滤波器可用于从照片中去除颗粒噪声。由于每个像素设置为其邻域中像素的平均值，因此会减少由颗粒引起的局部变化。

有关使用 `imfilter` 进行线性滤波的详细信息，请参阅 “什么是空间域中的图像滤波？”（第 8-2 页）。

使用平均值滤波器和中位数滤波器去除噪声

此示例说明如何使用平均值滤波器和中位数滤波器从图像中去除椒盐噪声，以便比较结果。这两种类型的滤波都将输出像素的值设置为对应输入像素周围邻域中像素值的平均值。然而，使用中位数滤波时，输出像素的值由邻域像素的中位数（而不是均值）决定。中位数远不如均值对极值（称为离群值）敏感。因此，中位数滤波能够更好地去除这些离群值，而不会降低图像的锐度。

注意：中位数滤波是顺序统计量滤波的一种特殊情况，也称为秩滤波。有关顺序统计量滤波的信息，请参阅 `ordfilt2` 函数的参考页。

将图像读入工作区并显示它。

```
I = imread('eight.tif');
figure
imshow(I)
```



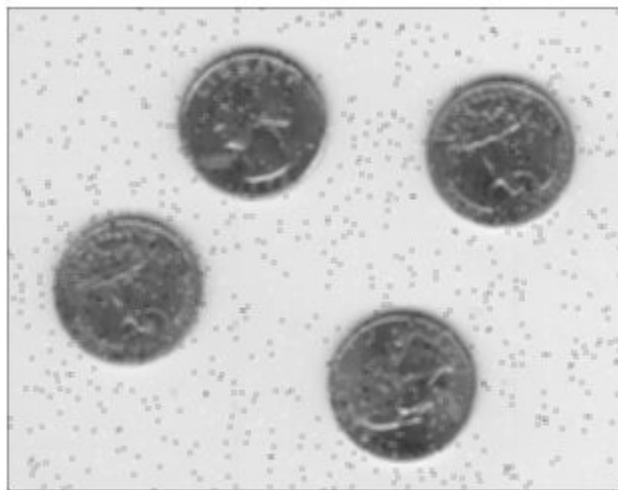
对于此示例，向图像中添加椒盐噪声。这种类型的噪声由设置为黑色或白色（数据范围的极值）的随机像素组成。

```
J = imnoise(I,'salt & pepper',0.02);
figure
imshow(J)
```



使用平均值滤波器对含噪图像 J 进行滤波并显示结果。该示例使用 3×3 邻域。

```
Kaverage = filter2(fspecial('average',3),J)/255;  
figure  
imshow(Kaverage)
```



现在使用中位数滤波器对含噪图像 J 进行滤波。该示例还使用 3×3 邻域。将两个滤波后的图像并排显示，以进行比较。请注意，`medfilt2` 在去除噪声方面做得更好，硬币边缘的模糊更少。

```
Kmedian = medfilt2(J);  
imshowpair(Kaverage,Kmedian,'montage')
```



通过自适应滤波去除噪声

此示例说明如何使用 `wiener2` 函数将 Wiener 滤波器（一种线性滤波器）以自适应方式应用于图像。Wiener 滤波器可自行适应图像局部方差。当方差较大时，`wiener2` 几乎不执行平滑处理。当方差较小时，`wiener2` 执行更多平滑处理。

这种方法通常比线性滤波产生更好的结果。自适应滤波器相比类似的线性滤波器更具选择性，它可保留图像的边缘和其他高频部分。此外，它没有设计任务；`wiener2` 函数处理所有初步计算，并对输入图像实现滤波器。然而，与线性滤波相比，`wiener2` 确实需要更多计算时间。

当噪声是恒定功率（“白色”）加性噪声（如高斯噪声）时，`wiener2` 效果最佳。以下示例将 `wiener2` 应用于添加了高斯噪声的土星图像。

将图像读入工作区中。

```
RGB = imread('saturn.png');
```

将图像从真彩色转换为灰度。

```
I = im2gray(RGB);
```

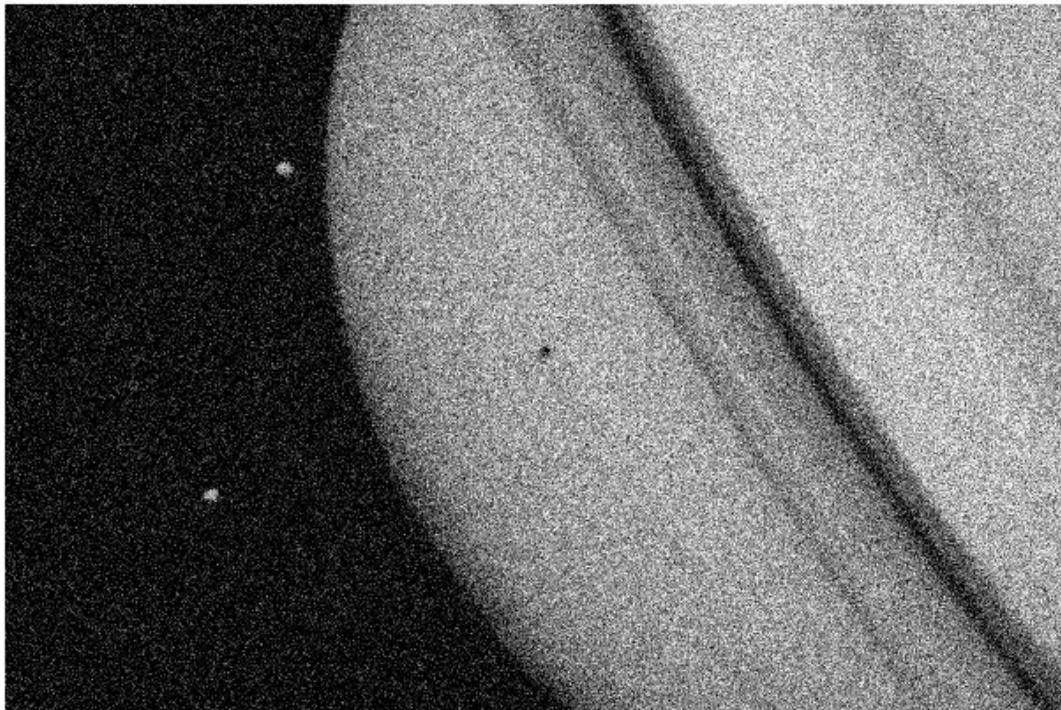
向图像中添加高斯噪声

```
J = imnoise(I,'gaussian',0,0.025);
```

显示含噪图像。由于图像相当大，因此只显示图像的一部分。

```
imshow(J(600:1000,1:600));
title('Portion of the Image with Added Gaussian Noise');
```

Portion of the Image with Added Gaussian Noise



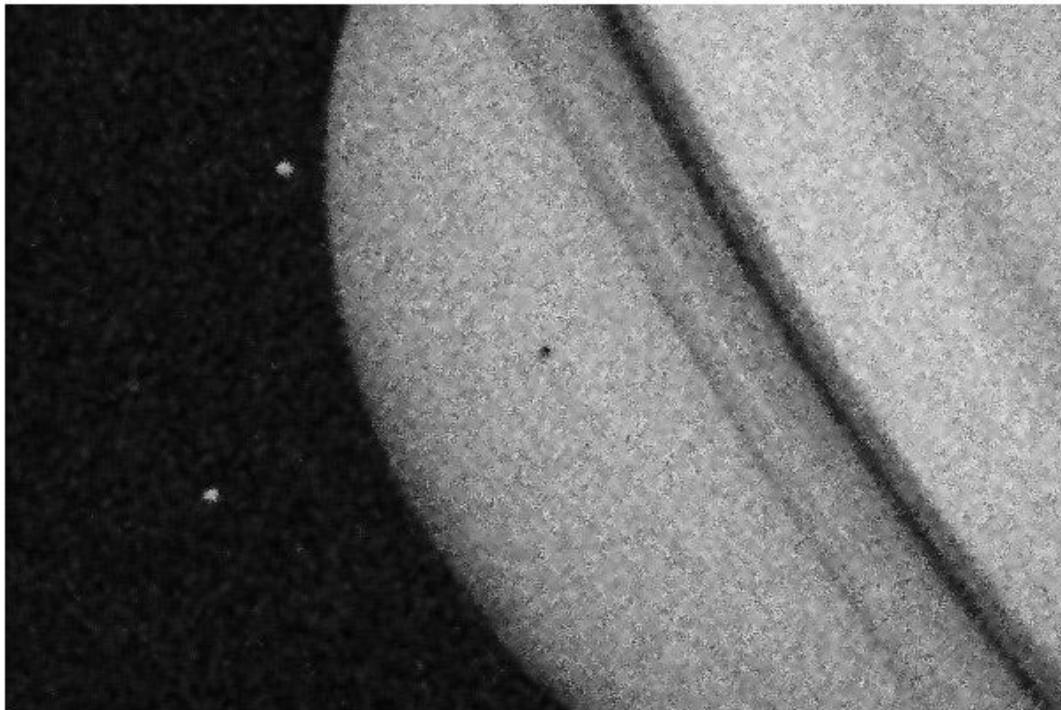
使用 `wiener2` 函数去除噪声。

```
K = wiener2(J,[5 5]);
```

显示处理后的图像。由于图像相当大，因此只显示图像的一部分。

```
figure  
imshow(K(600:1000,1:600));  
title('Portion of the Image with Noise Removed by Wiener Filter');
```

Portion of the Image with Noise Removed by Wiener Filter



另请参阅

[imfilter](#) | [imguidedfilter](#) | [imgaussfilt](#) | [locallapfilt](#) | [nlfilt](#) | [imbilatfilt](#)

详细信息

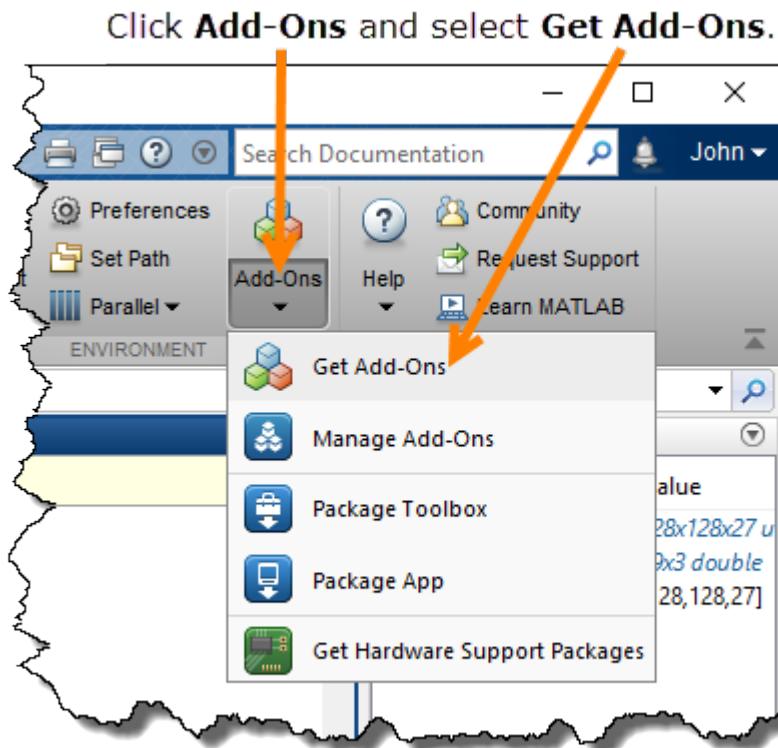
- “什么是空间域中的图像滤波？” (第 8-2 页)

使用附加功能资源管理器安装示例数据

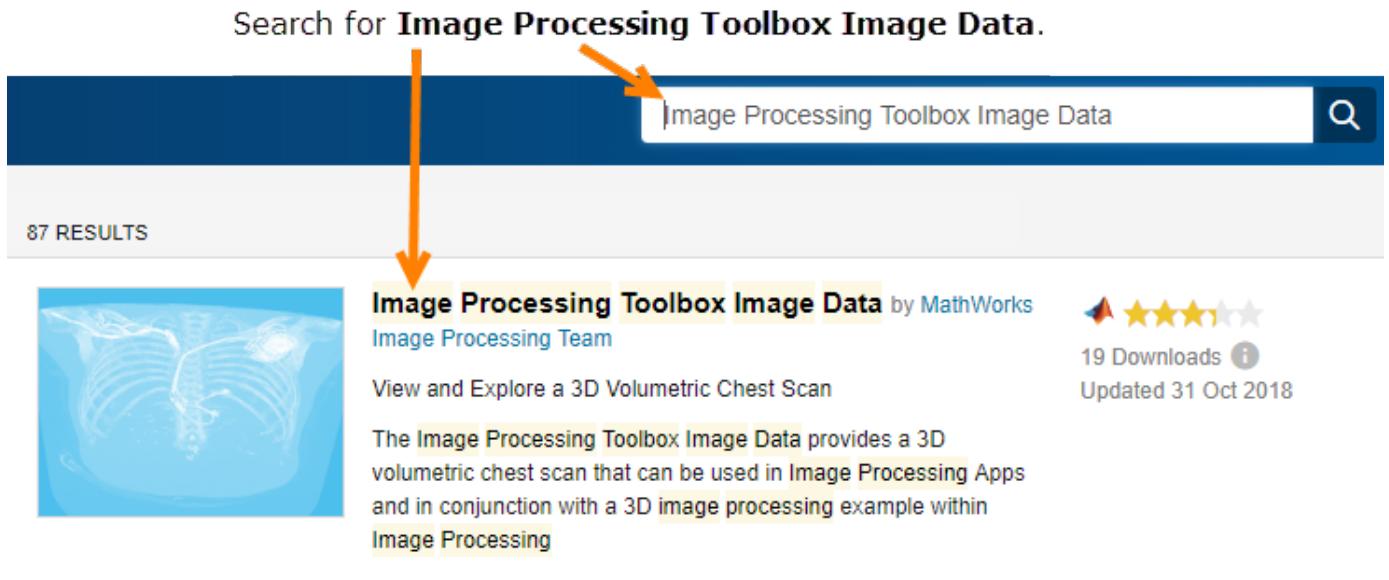
Image Processing Toolbox 以可选功能形式提供三维 MRI 胸部扫描数据集示例。示例“Segment Lungs from 3-D Chest Scan”使用此三维体数据集。

要获取此数据集，请使用附加功能资源管理器下载它。

- 1 从 MATLAB 桌面上的附加功能下拉菜单中选择**获取附加功能**。附加功能资源管理器将打开。



- 2 在附加功能资源管理器中，搜索数据包 **Image Processing Toolbox Image Data**。数据包是一项 **MathWorks 可选功能**。



- 3 在搜索结果中点击该数据包。在数据包页上，点击**安装**。按照安装程序提供的说明进行操作。

基于关注区域的处理

图像分割

本主题说明一系列用于分割图像的方法和 App。

- “使用纹理滤波器的纹理分割” (第 13-2 页)
- “使用 L*a*b* 颜色空间实现基于颜色的分割” (第 13-10 页)
- “使用 K 均值聚类实现基于颜色的分割” (第 13-16 页)
- “标记控制的分水岭分割” (第 13-21 页)

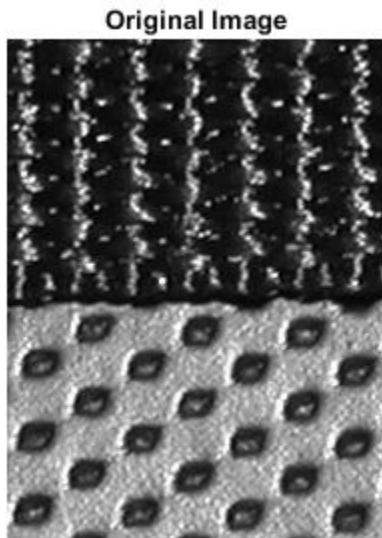
使用纹理滤波器的纹理分割

此示例说明如何根据纹理识别和分割区域。

读取图像

读取并显示一个挎包上的纹理图案的灰度图像。

```
I = imread('bag.png');
imshow(I)
title('Original Image')
```



创建纹理图像

使用 `entropyfilt` 创建纹理图像。函数 `entropyfilt` 返回一个数组，其中每个输出像素包含输入图像 `I` 中对应像素周围 9×9 邻域的熵值。熵是随机性的统计度量。

您还可以使用 `stdfilt` 和 `rangefilt` 来获得类似的分割结果。为了与局部熵的纹理图像进行比较，创建分别显示局部标准差和局部范围的纹理图像 `S` 和 `R`。

```
E = entropyfilt(I);
S = stdfilt(I,ones(9));
R = rangefilt(I,ones(9));
```

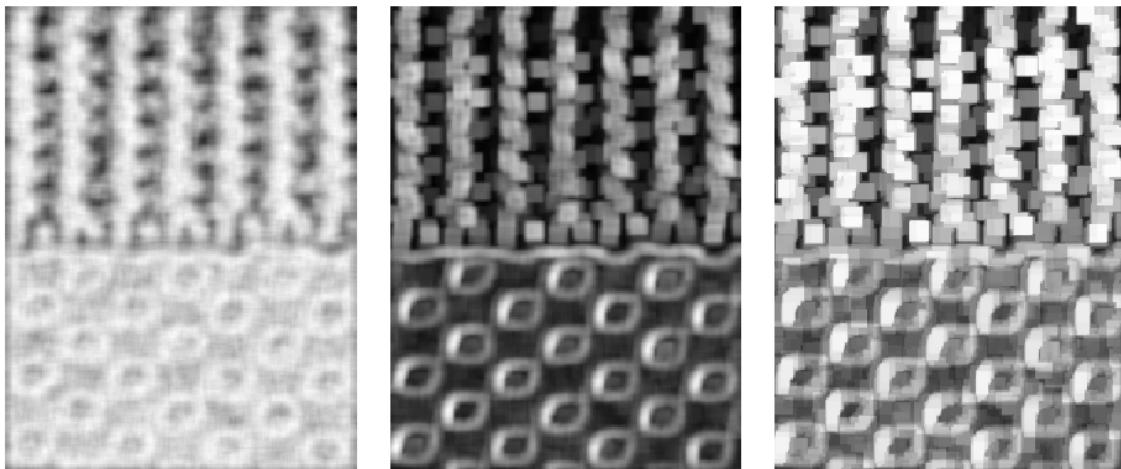
使用 `rescale` 重新缩放纹理图像 `E` 和 `S`，使像素值在数据类型为 `double` 的图像的预期范围 [0, 1] 内。

```
Eim = rescale(E);
Sim = rescale(S);
```

以蒙太奇方式显示三个纹理图像。

```
montage({Eim,Sim,R}, 'Size',[1 3], 'BackgroundColor','w', "BorderSize",20)
title('Texture Images Showing Local Entropy, Local Standard Deviation, and Local Range')
```

Texture Images Showing Local Entropy, Local Standard Deviation, and Local Range



为底部纹理创建掩膜

此示例继续处理熵纹理图像 **Eim**。您可以对其他两种类型的具有其他形态学函数的纹理图像重复类似的过程，以获得类似的分割结果。

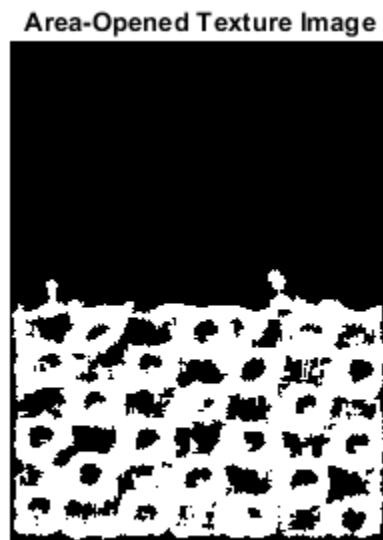
对重新缩放后的图像 **Eim** 设置阈值以分割纹理。选择 0.8 的阈值，因为它大致是纹理之间边界上像素的强度值。

```
BW1 = imbinarize(Eim,0.8);
imshow(BW1)
title("Thresholded Texture Image")
```



二值图像 BW1 中的分割对象是白色。如果您比较 BW1 和 I，会注意到顶部纹理过度分割（多个白色对象），底部纹理几乎整体分割。使用 bwareaopen 删除顶部纹理中的对象。

```
BWao = bwareaopen(BW1,2000);  
imshow(BWao)  
title('Area-Opened Texture Image')
```



使用 `imclose` 对边缘进行平滑处理，并使 `BWao` 中对象的任何开放孔洞闭合。指定 `entropyfilt` 使用的相同 9×9 邻域。

```
nhood = ones(9);
closeBWao = imclose(BWao,nhood);
imshow(closeBWao)
title('Closed Texture Image')
```



使用 `imfill` 填充 `closeBWao` 中对象的孔洞。底部纹理的掩膜并不完美，因为掩膜没有延伸到图像的底部。但是，您可以使用该掩膜来分割纹理。

```
mask = imfill(closeBWao,'holes');
imshow(mask);
title('Mask of Bottom Texture')
```

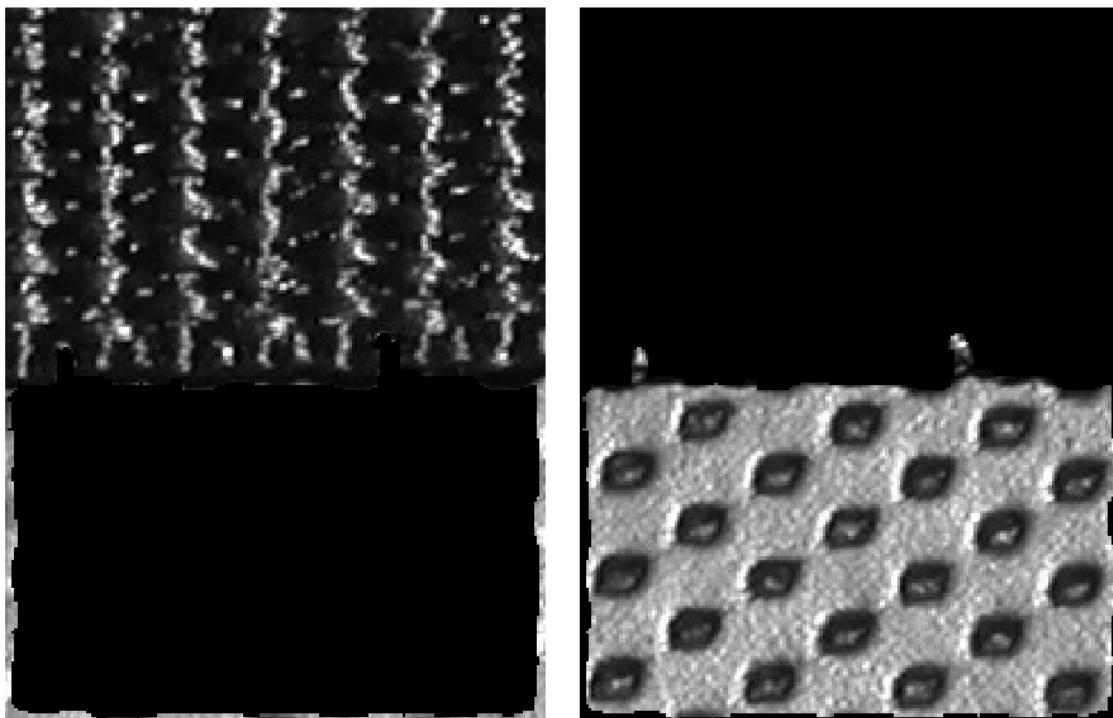


使用掩膜分割纹理

将纹理分成两个不同图像。

```
textureTop = I;
textureTop(mask) = 0;
textureBottom = I;
textureBottom(~mask) = 0;
montage({textureTop,textureBottom},'Size',[1 2],'BackgroundColor','w',"BorderSize",20)
title('Segmented Top Texture (Left) and Segmented Bottom Texture (Right)')
```

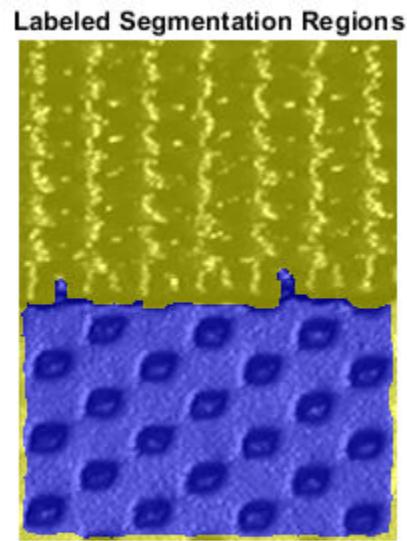
Segmented Top Texture (Left) and Segmented Bottom Texture (Right)



显示分割结果

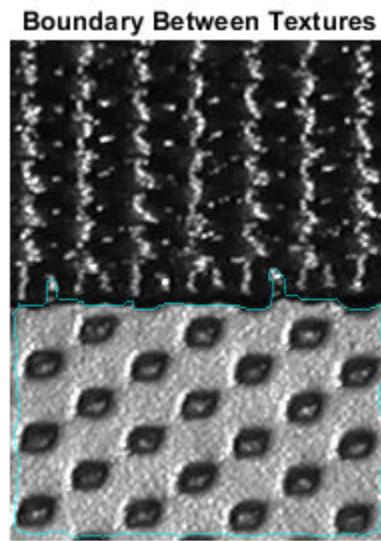
创建一个标签矩阵，其中标签 1 的掩膜是 `false`，标签 2 的掩膜是 `true`。在原始图像上叠加标签矩阵。

```
L = mask+1;  
imshow(labeloverlay(I,L))  
title('Labeled Segmentation Regions')
```



以青色显示两个纹理之间边界的轮廓。

```
boundary = bwperim(mask);
imshow(labeloverlay(I,boundary,"Colormap",[0 1 1]))
title('Boundary Between Textures')
```



另请参阅

[entropyfilt](#) | [bwareaopen](#) | [imclose](#) | [imbinarize](#) | [imfill](#) | [bwperim](#) | [rangefilt](#)

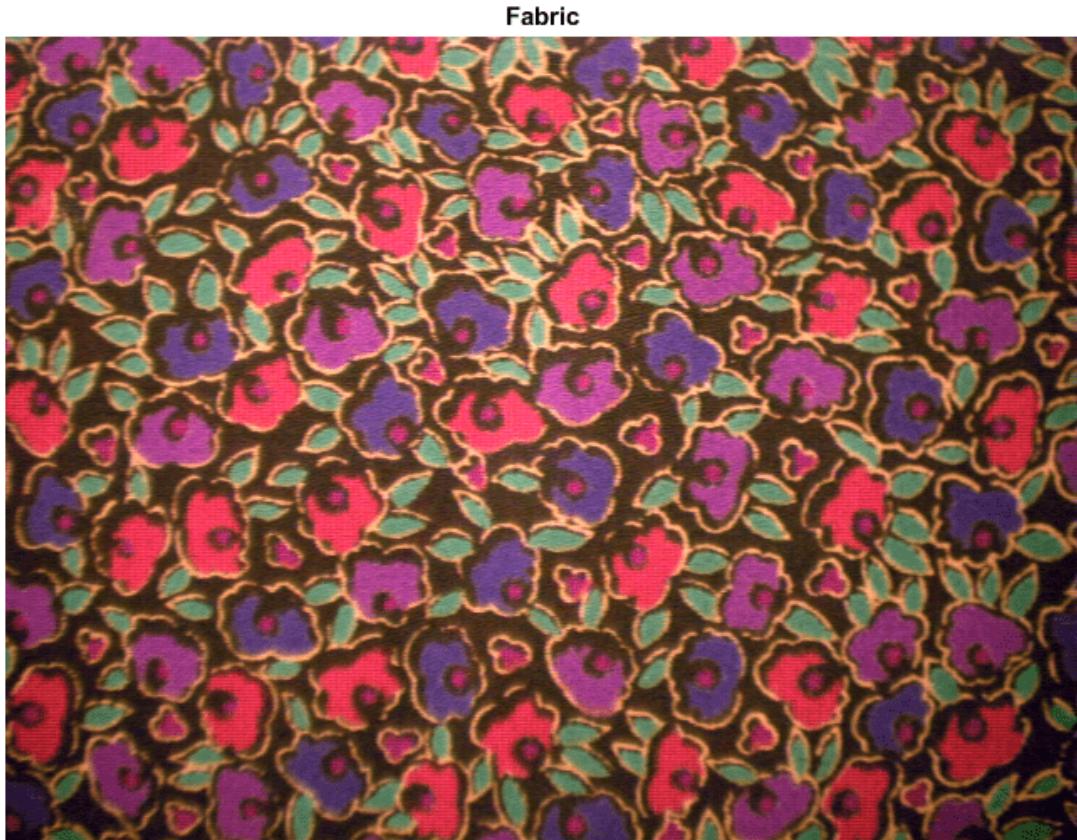
使用 L*a*b* 颜色空间实现基于颜色的分割

此示例说明如何通过分析 L*a*b* 颜色空间来识别织物中的不同颜色。织物图像是使用 Image Acquisition Toolbox™ 采集的。

步骤 1：采集图像

在 **fabric.png** 图像中读取，这是一个彩色织物图像。您可以使用 Image Acquisition Toolbox 中的以下函数来采集图像，而不是使用 **fabric.png**。

```
% Access a Matrox(R) frame grabber attached to a Pulnix TMC-9700 camera, and  
% acquire data using an NTSC format.  
% vidobj = videoinput('matrox',1,'M_NTSC_RGB');  
  
% Open a live preview window. Point camera onto a piece of colorful fabric.  
% preview(vidobj);  
  
% Capture one frame of data.  
% fabric = getsnapshot(vidobj);  
% imwrite(fabric,'fabric.png','png');  
  
% Delete and clear associated variables.  
% delete(vidobj)  
% clear vidobj;  
  
fabric = imread('fabric.png');  
imshow(fabric)  
title('Fabric')
```



步骤 2：基于 L*a*b* 颜色空间计算每个区域的样本颜色

您可以在图像中看到六种主要颜色：背景色、红色、绿色、紫色、黄色和洋红色。请注意，您可以很轻松地在视觉上区分这些颜色。L*a*b* 颜色空间（也称为 CIELAB 或 CIE L*a*b*）使您能够量化这些视觉差异。

L*a*b* 颜色空间是从 CIE XYZ 三色值派生的。L*a*b* 空间包含光度 'L*'（即亮度层）、色度层 'a*'（表示颜色落在沿红-绿轴的位置）和色度层 'b*'（表示颜色落在沿蓝-黄轴的位置）。

您的方法是为每种颜色选择一个小样本区域，并计算每个样本区域的基于 'a*b*' 空间的平均颜色。您将使用这些颜色标记对每个像素进行分类。

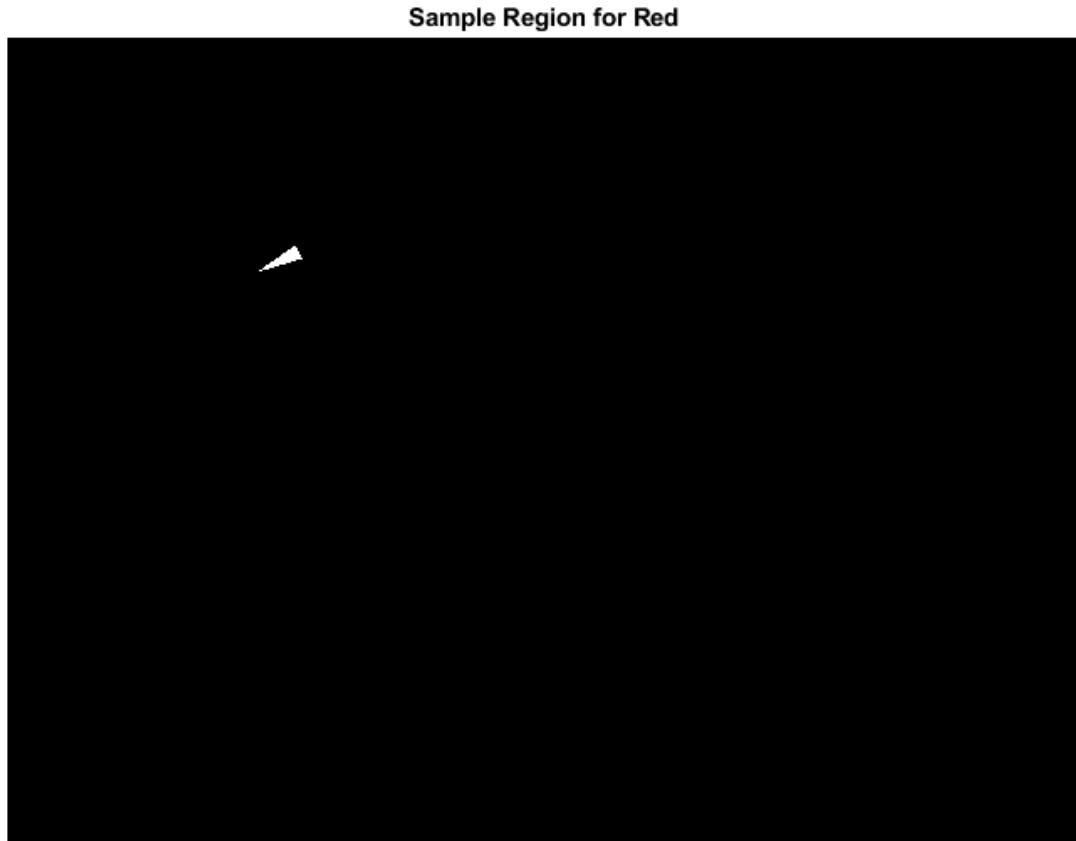
为了简化此示例，请加载存储在 MAT 文件中的区域坐标。

```
load regioncoordinates;

nColors = 6;
sample_regions = false([size(fabric,1) size(fabric,2) nColors]);

for count = 1:nColors
    sample_regions(:, :, count) = roipoly(fabric, region_coordinates(:, 1, count), ...
        region_coordinates(:, 2, count));
end
```

```
imshow(sample_regions(:,:,2))
title('Sample Region for Red')
```



使用 `rgb2lab` 将您的织物 RGB 图像转换为 L*a*b* 图像。

```
lab_fabric = rgb2lab(fabric);
```

计算您用 `roipoly` 提取的每个区域的 'a*' 和 'b*' 均值。这些值用作基于 'a*b*' 空间的颜色标记。

```
a = lab_fabric(:,:,2);
b = lab_fabric(:,:,3);
color_markers = zeros([nColors, 2]);

for count = 1:nColors
    color_markers(count,1) = mean2(a(sample_regions(:,:,count)));
    color_markers(count,2) = mean2(b(sample_regions(:,:,count)));
end
```

例如，基于 'a*b*' 空间的红色样本区域平均颜色为

```
fprintf(['%0.3f,%0.3f\n',color_markers(2,1),color_markers(2,2));
```

```
[69.828,20.106]
```

步骤 3：使用最近邻规则对每个像素进行分类

现在每个颜色标记都有一个 'a*' 和一个 'b*' 值。您可以通过计算 `lab_fabric` 图像中每个像素与每个颜色标记之间的欧几里德距离对该像素进行分类。最小距离表示像素最接近该颜色标记。例如，如果某像素和红色标记之间的距离最小，则该像素将被标记为红色像素。

创建一个包含颜色标签的数组，即 0 表示背景，1 表示红色，2 表示绿色，3 表示紫色，4 表示洋红色，5 表示黄色。

```
color_labels = 0:nColors-1;
```

初始化用于最近邻分类的矩阵。

```
a = double(a);
b = double(b);
distance = zeros([size(a), nColors]);
```

执行分类

```
for count = 1:nColors
    distance(:, :, count) = ( (a - color_markers(count, 1)).^2 + ...
        (b - color_markers(count, 2)).^2 ).^0.5;
end

[~, label] = min(distance, [], 3);
label = color_labels(label);
clear distance;
```

步骤 4：显示最近邻分类的结果

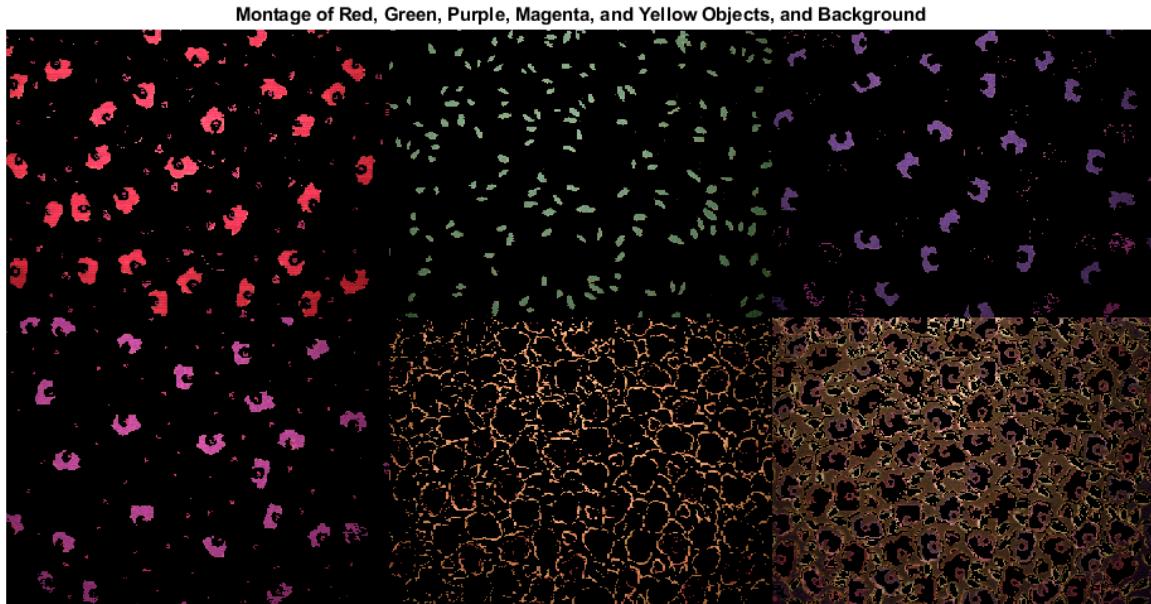
标签矩阵包含织物图像中每个像素的颜色标签。使用标签矩阵按颜色分离原始织物图像中的对象。

```
rgb_label = repmat(label, [1 1 3]);
segmented_images = zeros([size(fabric), nColors], 'uint8');

for count = 1:nColors
    color = fabric;
    color(rgb_label ~= color_labels(count)) = 0;
    segmented_images(:, :, :, count) = color;
end
```

以蒙太奇方式显示五种分割颜色。同时显示图像中未归类为颜色的背景像素。

```
montage({segmented_images(:, :, :, 2), segmented_images(:, :, :, 3) ...
    segmented_images(:, :, :, 4), segmented_images(:, :, :, 5) ...
    segmented_images(:, :, :, 6), segmented_images(:, :, :, 1)});
title("Montage of Red, Green, Purple, Magenta, and Yellow Objects, and Background")
```



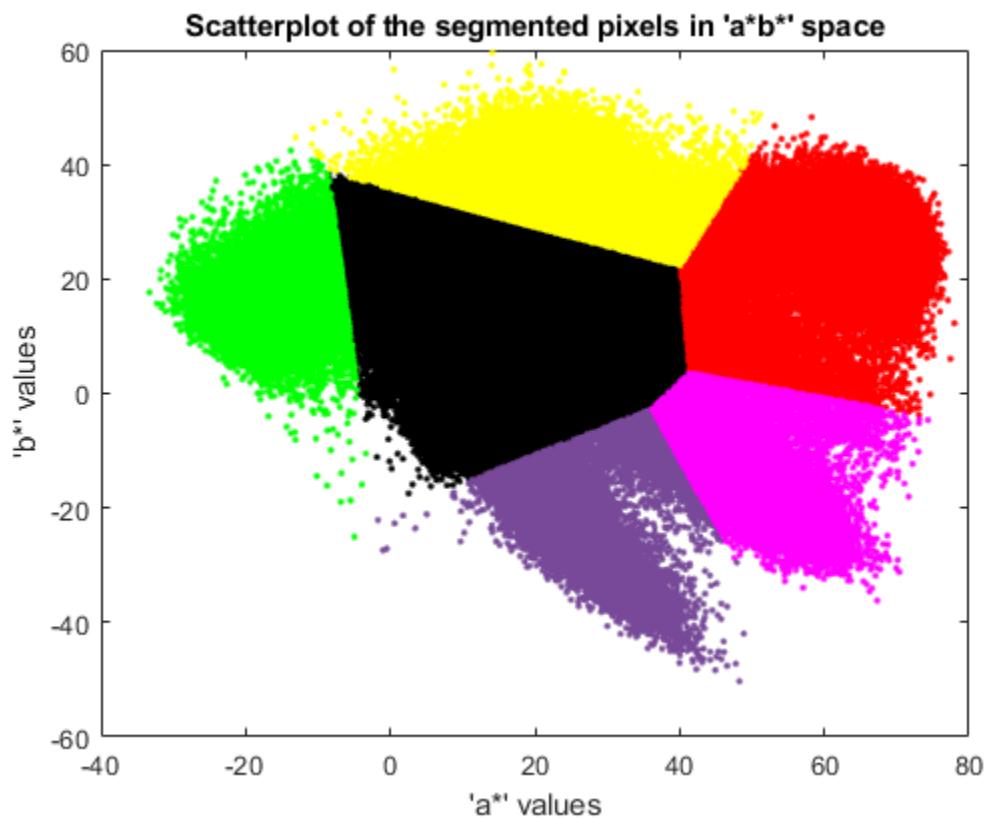
步骤 5：显示已标注颜色的 'a*' 和 'b*' 值

通过绘制分类为不同颜色的像素的 'a*' 和 'b*' 值，您可以看到最近邻分类对不同颜色总体的区分情况。出于显示目的，用颜色标签标注每个点。

```
purple = [119/255 73/255 152/255];
plot_labels = {'k', 'r', 'g', purple, 'm', 'y'};

figure
for count = 1:nColors
    plot(a(label==count-1),b(label==count-1),[],'MarkerEdgeColor', ...
        plot_labels{count}, 'MarkerFaceColor', plot_labels{count});
    hold on;
end

title('Scatterplot of the segmented pixels in "a*b*" space');
xlabel('"a*" values');
ylabel('"b*" values');
```



使用 K 均值聚类实现基于颜色的分割

此示例说明如何使用 L*a*b* 颜色空间和 K 均值聚类自动分割颜色。

步骤 1：读取图像

在 `hestain.png` 中读取，这是一个带有苏木精和曙红染色组织 (H&E) 的图像。这种染色方法有助于病理学家区分不同组织类型。

```
he = imread('hestain.png');
imshow(he), title('H&E image');
text(size(he,2),size(he,1)+15,...
    'Image courtesy of Alan Partin, Johns Hopkins University',...
    'FontSize',7,'HorizontalAlignment','right');
```

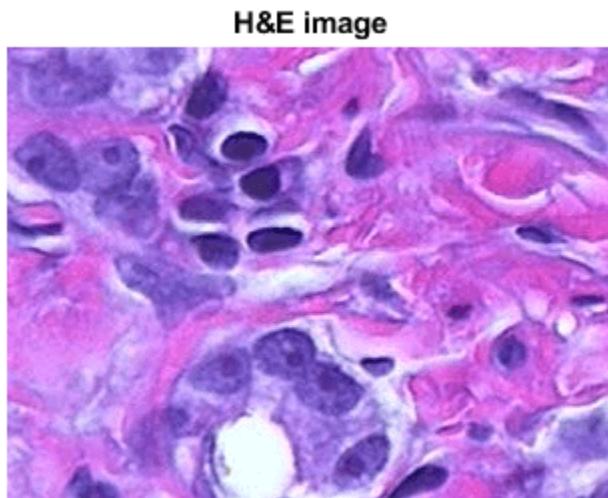


Image courtesy of Alan Partin, Johns Hopkins University

步骤 2：将图像从 RGB 颜色空间转换为 L*a*b* 颜色空间

如果忽略亮度的变化，您会在图像中看到多少种颜色？有三种颜色：白色、蓝色和粉色。请注意，您可以很轻松地在视觉上区分这些颜色。 $L^*a^*b^*$ 颜色空间（也称为 CIELAB 或 CIE $L^*a^*b^*$ ）使您能够量化这些视觉差异。

$L^*a^*b^*$ 颜色空间是从 CIE XYZ 三色值派生的。 $L^*a^*b^*$ 空间包含光度层 ' L^* '、色度层 ' a^* '（表示颜色落在沿红-绿轴的位置）和色度层 ' b^* '（表示颜色落在沿蓝-黄轴的位置）。所有颜色信息都在 ' a^* ' 和 ' b^* ' 层。您可以使用欧几里得距离度量来测量两种颜色之间的差异。

使用 `rgb2lab` 将图像转换为 $L^*a^*b^*$ 颜色空间。

```
lab_he = rgb2lab(he);
```

步骤 3：用 K 均值聚类对基于 ' a^*b^* ' 空间的颜色进行分类

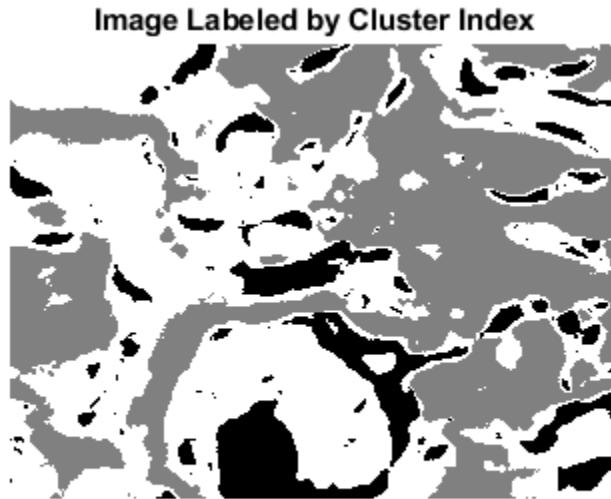
聚类是一种分离对象组的方法。K 均值聚类将每个对象视为在空间中有一个位置。它将对象划分为若干分区，使每个簇中的对象尽可能彼此靠近，并尽可能远离其他簇中的对象。K 均值聚类要求您指定要划分的簇数和用于量化两个对象之间距离的距离度量。

由于颜色信息基于 'a*b*' 颜色空间，因此您的对象是具有 'a*' 和 'b*' 值的像素。将数据转换为数据类型 `single`，以便与 `imsegkmeans` 结合使用。使用 `imsegkmeans` 对对象进行聚类以分为三个簇。

```
ab = lab_he(:,:,2:3);
ab = im2single(ab);
nColors = 3;
% repeat the clustering 3 times to avoid local minima
pixel_labels = imsegkmeans(ab,nColors,'NumAttempts',3);
```

对于您的输入中的每个对象，`imsegkmeans` 会返回一个对应于簇的索引或标签。用像素的标签标注图像中的每个像素。

```
imshow(pixel_labels,[])
title('Image Labeled by Cluster Index');
```

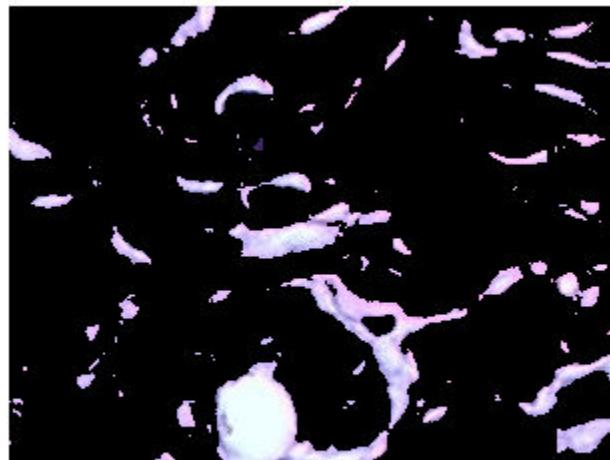


步骤 4：创建按颜色分割 H&E 图像的图像

使用 `pixel_labels`，您可以按颜色分离 `hestain.png` 中的对象，这将产生三个图像。

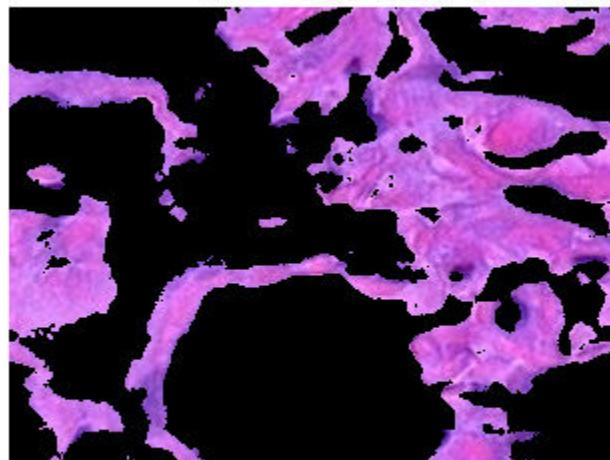
```
mask1 = pixel_labels==1;
cluster1 = he .* uint8(mask1);
imshow(cluster1)
title('Objects in Cluster 1');
```

Objects in Cluster 1

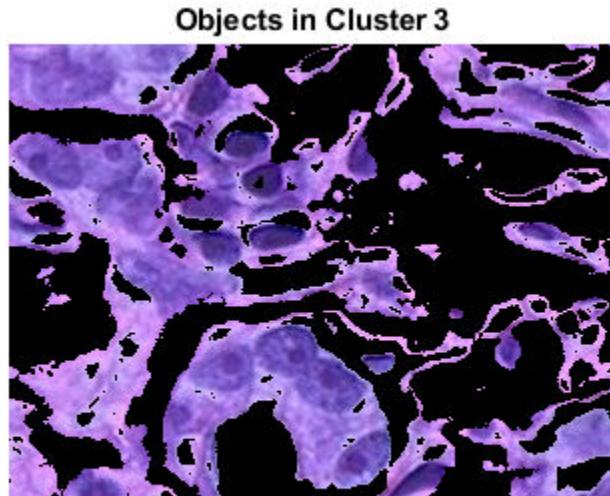


```
mask2 = pixel_labels==2;
cluster2 = he .* uint8(mask2);
imshow(cluster2)
title('Objects in Cluster 2');
```

Objects in Cluster 2



```
mask3 = pixel_labels==3;
cluster3 = he .* uint8(mask3);
imshow(cluster3)
title('Objects in Cluster 3');
```



步骤 5：分割核

簇 3 包含蓝色对象。请注意，有深蓝色和浅蓝色对象。您可以使用 L*a*b* 颜色空间中的 'L*' 层来分离深蓝色和浅蓝色。细胞核为深蓝色。

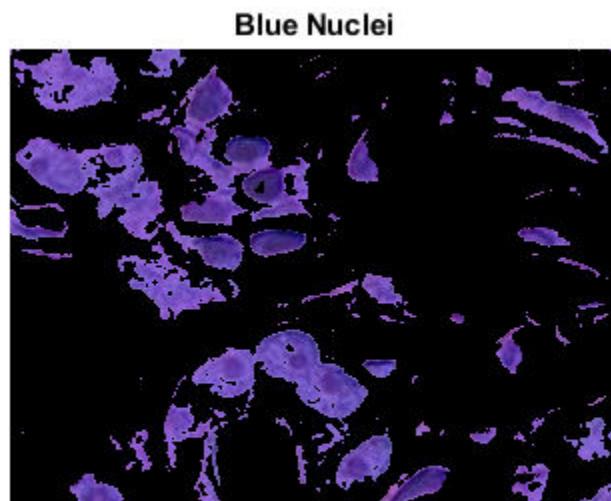
前面提到过，'L*' 层包含每种颜色的亮度值。提取此簇中像素的亮度值，并使用 `imbinarize` 用全局阈值对其设置阈值。掩膜 `is_light_blue` 给出了浅蓝色像素的索引。

```
L = lab_he(:,:,1);
L_blue = L .* double(mask3);
L_blue = rescale(L_blue);
idx_light_blue = imbinarize(nonzeros(L_blue));
```

复制蓝色对象的掩膜 `mask3`，然后从掩膜中删除浅蓝色像素。将新掩膜应用于原始图像并显示结果。只有深蓝色细胞核可见。

```
blue_idx = find(mask3);
mask_dark_blue = mask3;
mask_dark_blue(blue_idx(idx_light_blue)) = 0;

blue_nuclei = he .* uint8(mask_dark_blue);
imshow(blue_nuclei)
title('Blue Nuclei');
```



标记控制的分水岭分割

此示例说明如何使用分水岭分割来分离图像中相互接触的对象。分水岭变换通过将图像视为一个曲面，其中亮像素表示较高处，暗像素表示较低处，从而找出图像中的“汇水盆地”和“分水岭脊线”。

如果您能够标识或“标记”前景对象和背景位置，使用分水岭变换的分割效果会更好。标记控制的分水岭分割遵循以下基本过程：

- 1 计算分割函数。这得到是一个图像，其中暗区域是您尝试分割的对象。
- 2 计算前景标记。这些是每个对象中连通的像素斑点。
- 3 计算背景标记。这些像素不是任何对象的一部分。
- 4 修正分割函数，使其仅在前景和背景标记位置具有最小值。
- 5 计算修正分割函数的分水岭变换。

步骤 1：读入彩色图像并将其转换为灰度

```
rgb = imread('pears.png');
I = rgb2gray(rgb);
imshow(I)

text(732,501,'Image courtesy of Corel(R)',...
    'FontSize',7,'HorizontalAlignment','right')
```

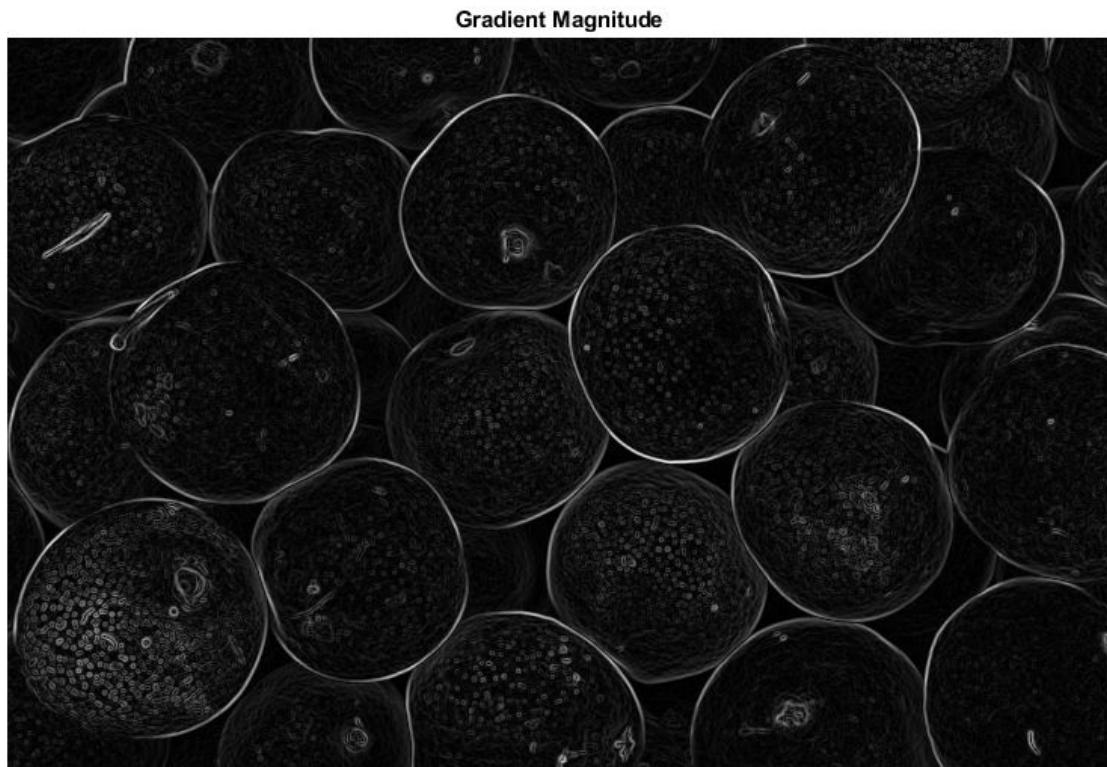


Image courtesy of Corel(R)

步骤 2：使用梯度幅值作为分割函数

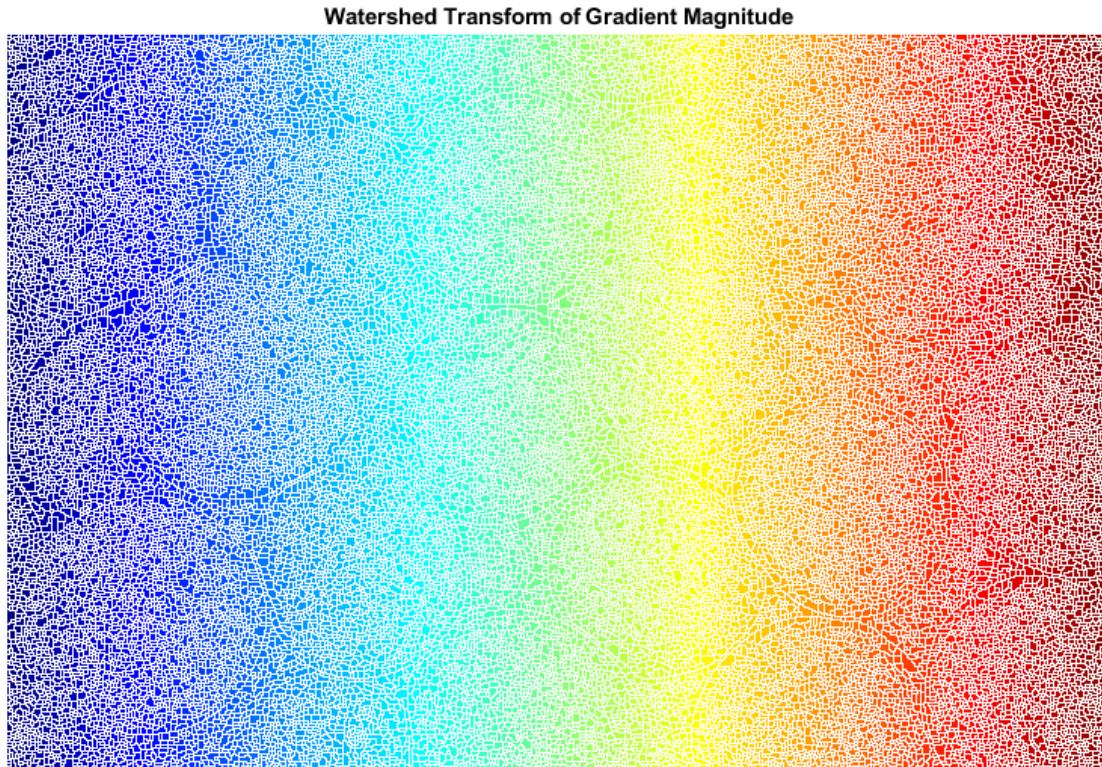
计算梯度幅值。对象边界处的梯度较高，对象内部的梯度较低（大多数情况下）。

```
gmag = imgradient(I);
imshow(gmag,[])
title('Gradient Magnitude')
```



您能通过直接对梯度幅值使用分水岭变换来分割图像吗？

```
L = watershed(gmag);
Lrgb = label2rgb(L);
imshow(Lrgb)
title('Watershed Transform of Gradient Magnitude')
```



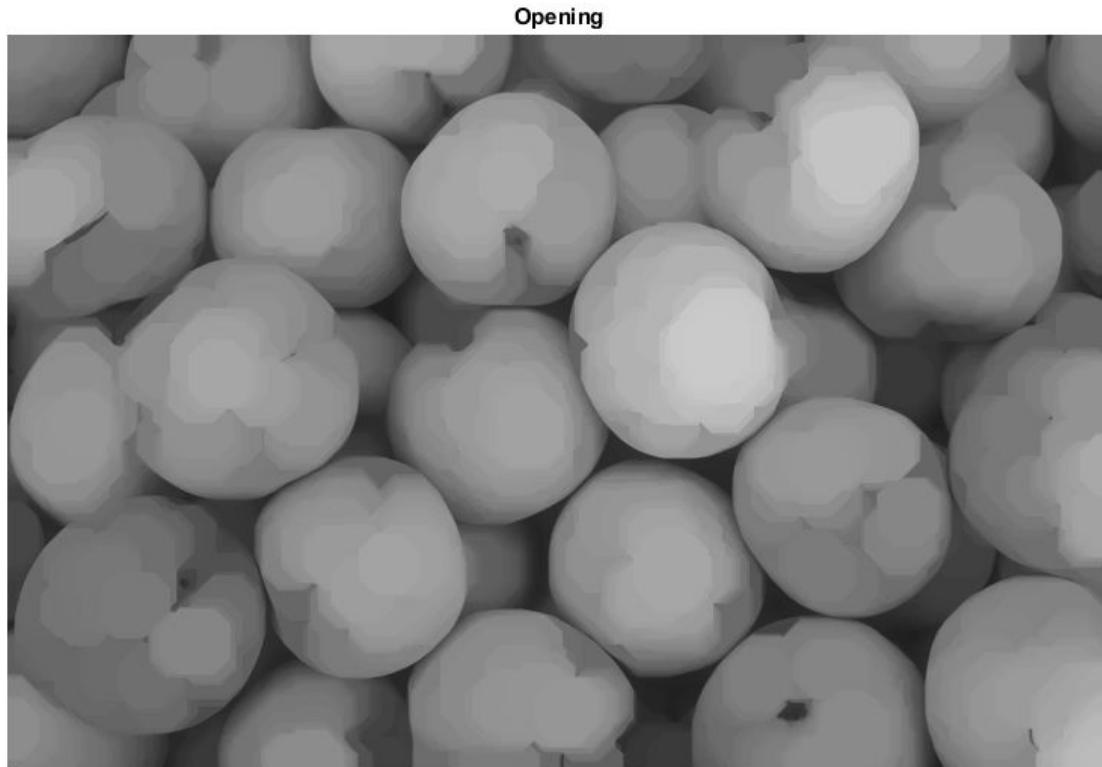
不能。如果没有额外的预处理，例如以下标记计算，直接使用分水岭变换通常会导致“过度分割”。

步骤 3：标记前景对象

此处可以应用多种过程来找到前景标记，这些标记必须是每个前景对象内部的连通像素斑点。在此示例中，您将使用称为“开运算重构”和“闭运算重构”的形态学方法来“清理”图像。这些运算将在每个对象内创建最大值平面，使用 `imregionalmax` 可以找出这些最大值。

开运算是先腐蚀后膨胀，而开运算重构是先腐蚀后进行形态学重构。让我们对两者进行比较。首先，使用 `imopen` 进行开运算。

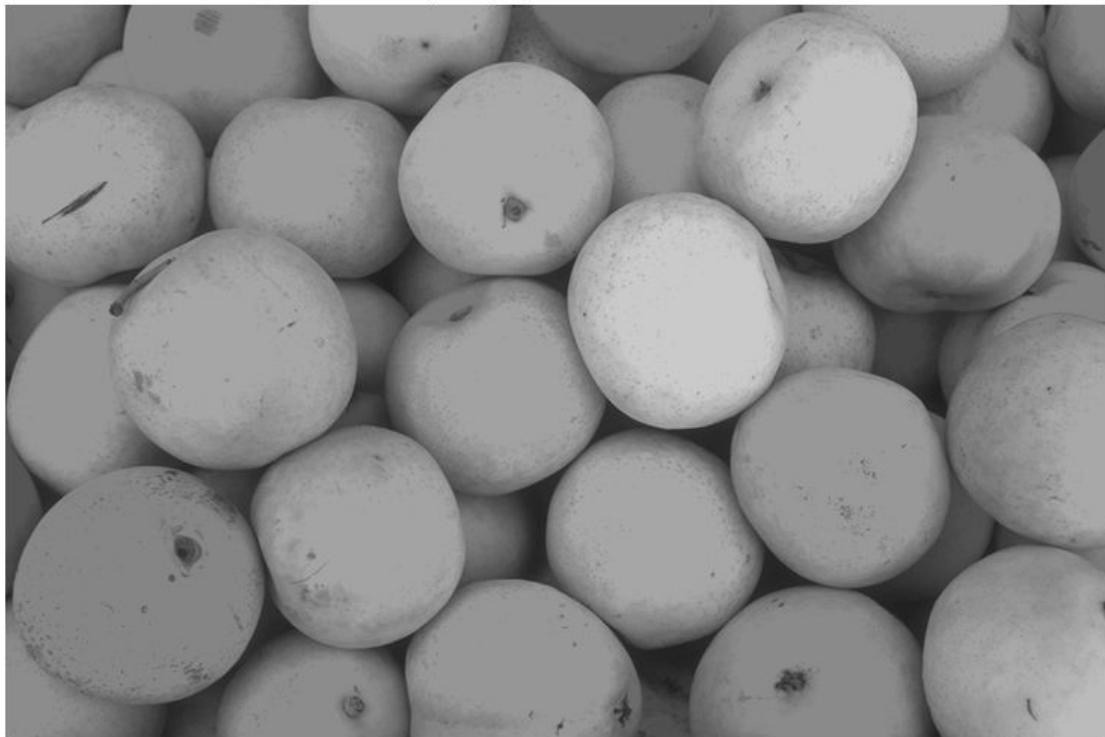
```
se = strel('disk',20);
Io = imopen(I,se);
imshow(Io)
title('Opening')
```



接下来，使用 `imerode` 和 `imreconstruct` 进行开运算重构。

```
Ie = imerode(I,se);
Iobr = imreconstruct(Ie,I);
imshow(Iobr)
title('Opening-by-Reconstruction')
```

Opening-by-Reconstruction



在开运算后闭运算可以去除暗点和针状标记。对常规形态学闭运算和闭运算重构进行比较。首先尝试 **imclose**：

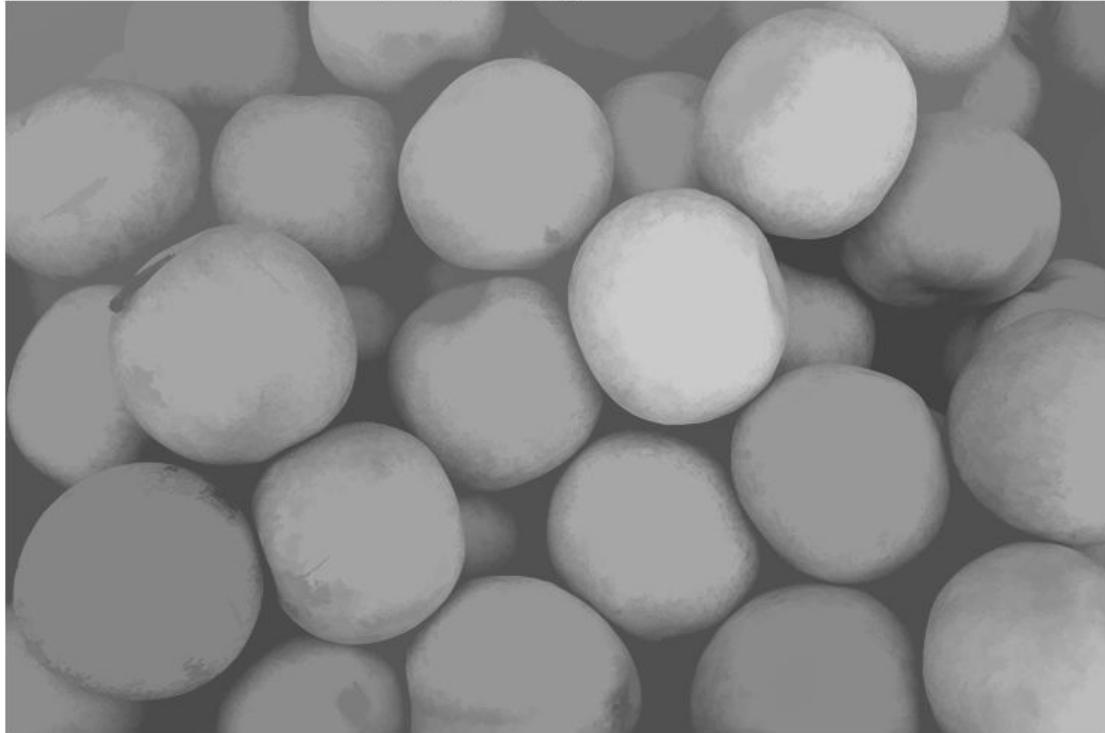
```
Ioc = imclose(Io,se);  
imshow(Ioc)  
title('Opening-Closing')
```



现在使用 `imdilate`, 然后使用 `imreconstruct`。请注意, 您必须对 `imreconstruct` 的输入和输出进行补充。

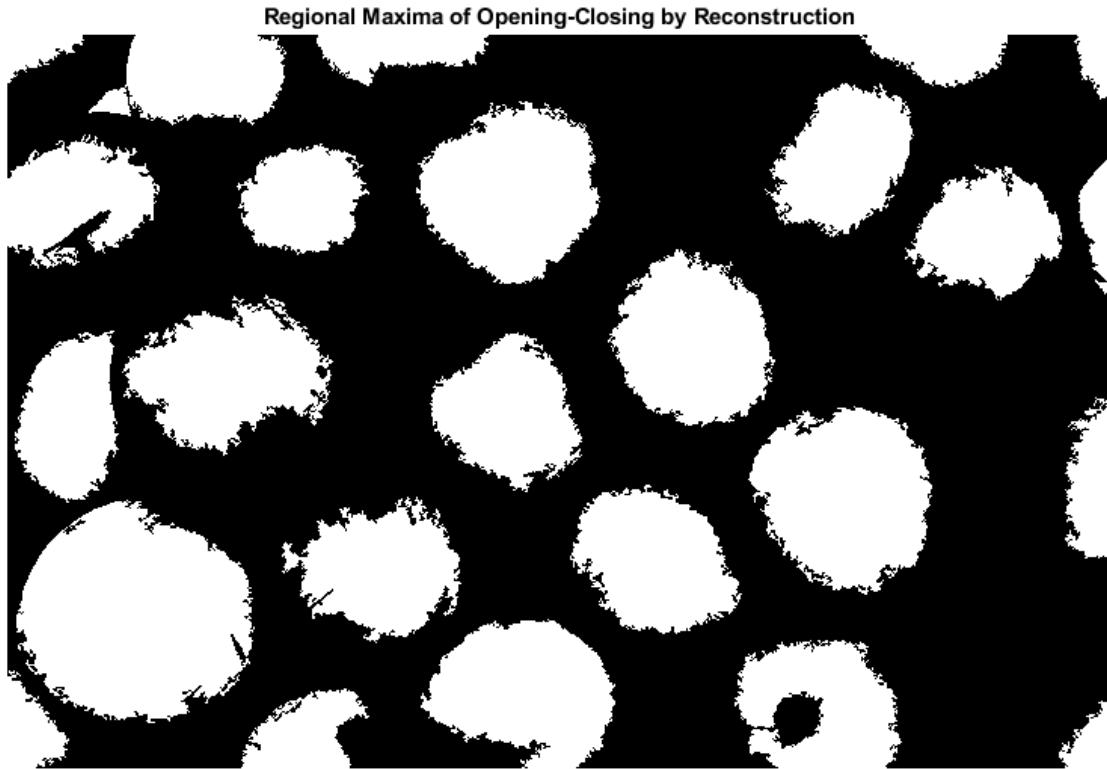
```
Iobrd = imdilate(Iobr,se);
Iobrcbr = imreconstruct(imcomplement(Iobrd),imcomplement(Iobr));
Iobrcbr = imcomplement(Iobrcbr);
imshow(Iobrcbr)
title('Opening-Closing by Reconstruction')
```

Opening-Closing by Reconstruction



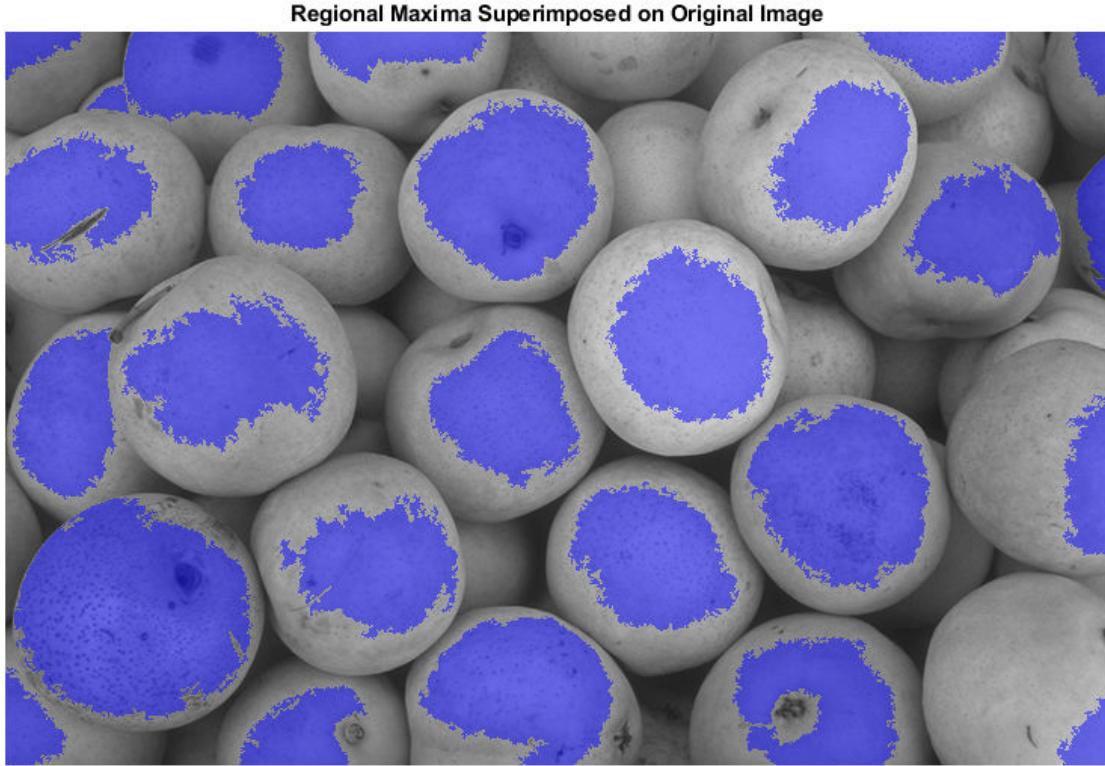
通过比较 **Iobrcbr** 和 **Loc** 可以看到，在去除小瑕疵而不影响对象整体形状方面，基于重构的开运算和闭运算比标准的开运算和闭运算更高效。计算 **Iobrcbr** 的局部最大值，以获得良好的前景标记。

```
fgm = imregionalmax(Iobrcbr);
imshow(fgm)
title('Regional Maxima of Opening-Closing by Reconstruction')
```



为了帮助解释结果，可将前景标记图像叠加在原始图像上。

```
I2 = labeloverlay(I,fgm);
imshow(I2)
title('Regional Maxima Superimposed on Original Image')
```

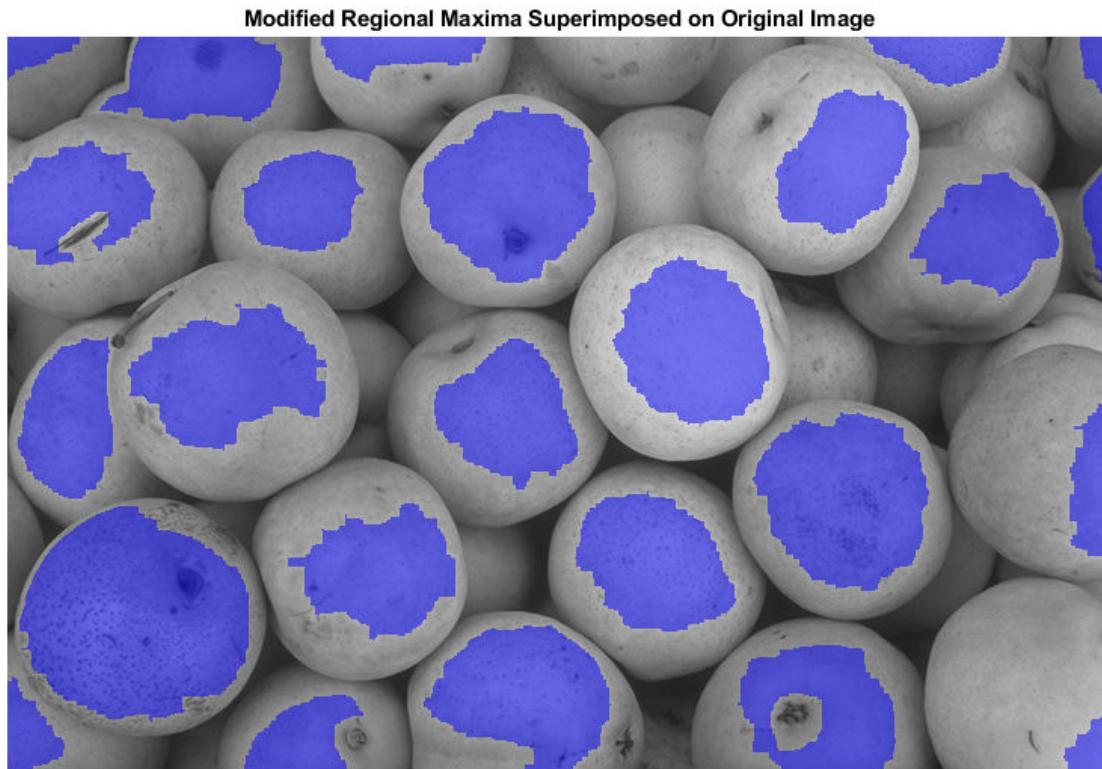


请注意，一些大部分被遮挡和遮蔽的对象未被标记，这意味着在最终结果中不会正确分割这些对象。此外，一些对象中的前景标记一直延伸到对象的边缘。这意味着您应清理标记斑点的边缘，然后将它们缩小一点。您可以通过先闭运算再腐蚀来实现这一点。

```
se2 = strel(ones(5,5));
fgm2 = imclose(fgm,se2);
fgm3 = imerode(fgm2,se2);
```

此过程往往会留下一些必须删除的杂散孤立像素。您可以使用 **bwareaopen** 来实现这一点，它可以删除像素少于特定数量的所有斑点。

```
fgm4 = bwareaopen(fgm3,20);
I3 = labeloverlay(I,fgm4);
imshow(I3)
title('Modified Regional Maxima Superimposed on Original Image')
```



步骤 4：计算背景标记

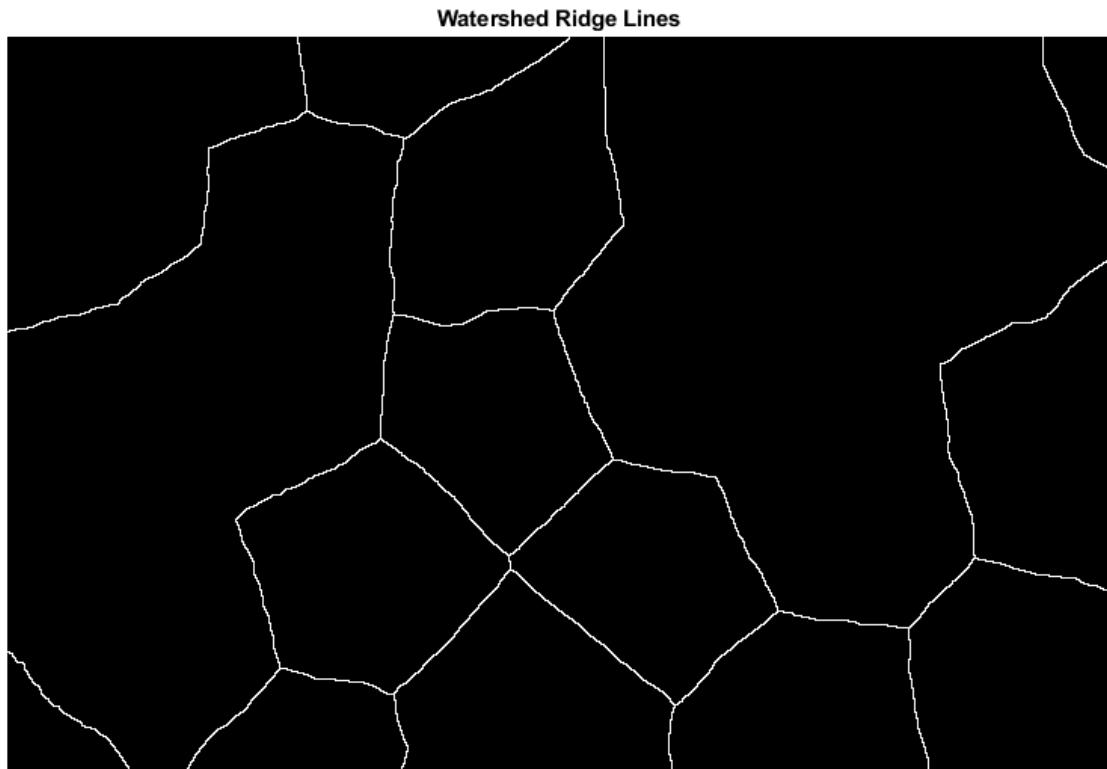
现在您需要标记背景。在清理后的图像 `Iobrcbr` 中，暗像素属于背景，因此您可以首先进行阈值化运算。

```
bw = imbinarize(Iobrcbr);  
imshow(bw)  
title('Thresholded Opening-Closing by Reconstruction')
```



背景像素为黑色，但理想情况下，我们不希望背景标记太靠近我们尝试分割的对象的边缘。我们将通过计算 `bw` 前景的“影响区骨架”（即 SKIZ）来“精简”背景。通过计算 `bw` 的距离变换的分水岭变换，然后寻找结果的分水岭脊线 (`DL == 0`)，可以实现这一目标。

```
D = bwdist(bw);
DL = watershed(D);
bgm = DL == 0;
imshow(bgm)
title('Watershed Ridge Lines')
```



步骤 5：计算分割函数的分水岭变换。

函数 `imimposemin` 可用于修正图像，使其仅在特定所需位置具有局部最小值。在此处，您可以使用 `imimposemin` 来修正梯度幅值图像，使其局部最小值只出现在前景和背景标记像素上。

```
gmag2 = imimposemin(gmag, bgm | fgm4);
```

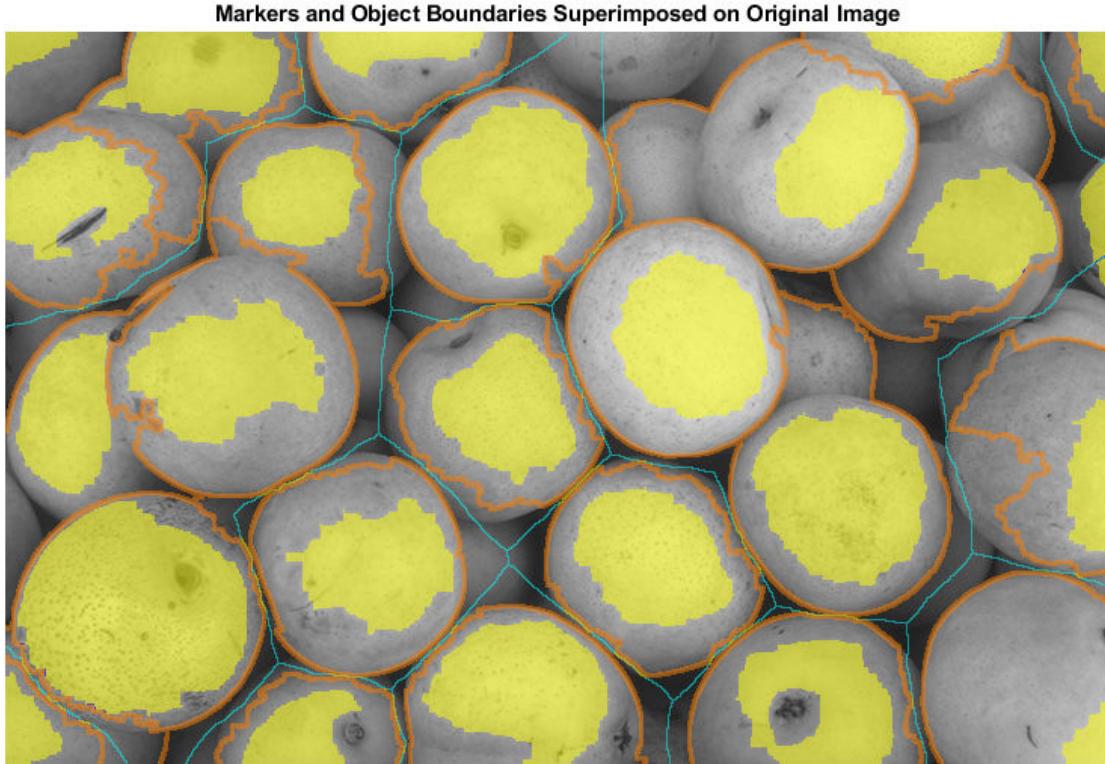
最后，计算基于分水岭的分割。

```
L = watershed(gmag2);
```

步骤 6：可视化结果

一种可视化方法是在原始图像上叠加前景标记、背景标记和分割的对象边界。您可以根据需要使用膨胀来使某些方面（如对象边界）更明显可见。对象边界位于 `L == 0` 的位置。二值前景和背景标记缩放为不同整数值，以便对它们分配不同标签。

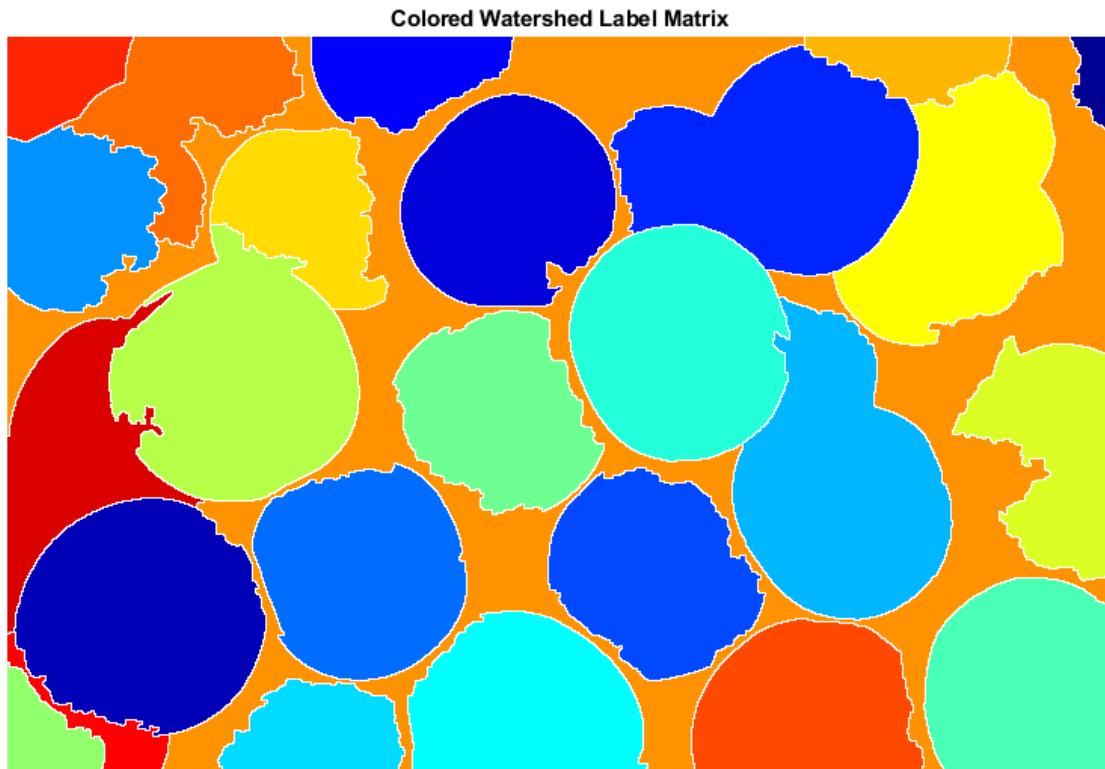
```
labels = imdilate(L==0,ones(3,3)) + 2*bgm + 3*fgm4;
I4 = labeloverlay(I,labels);
imshow(I4)
title('Markers and Object Boundaries Superimposed on Original Image')
```



此可视化图像说明前景和背景标记的位置如何影响结果。在一些位置，部分被遮挡的较暗对象与其较亮的相邻对象已合并，因为被遮挡的对象没有前景标记。

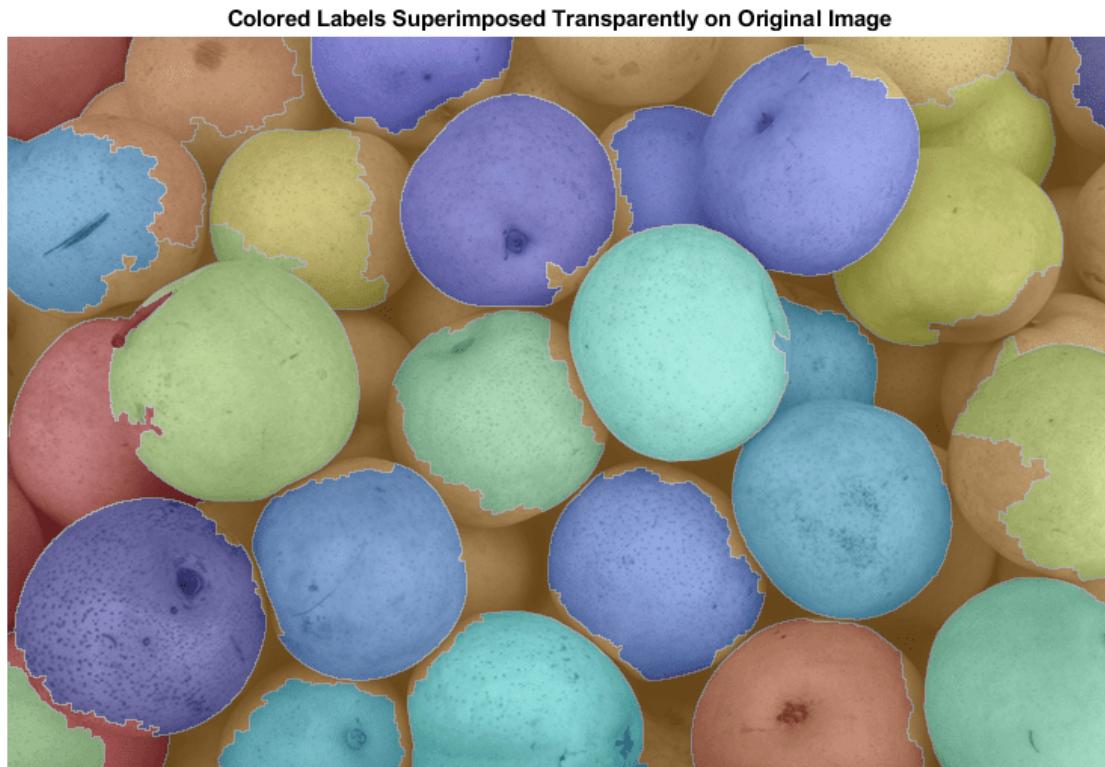
另一种有用的可视化方法是将标签矩阵显示为彩色图像。标签矩阵，如 `watershed` 和 `bwlabel` 生成的矩阵，可通过使用 `label2rgb` 转换为真彩色图像，以实现可视化目的。

```
Lrgb = label2rgb(L,'jet','w','shuffle');  
imshow(Lrgb)  
title('Colored Watershed Label Matrix')
```



您可以使用透明方式将此伪颜色标签矩阵叠加到原始强度图像上。

```
figure
imshow(I)
hold on
himage = imshow(Lrgb);
himage.AlphaData = 0.3;
title('Colored Labels Superimposed Transparently on Original Image')
```



另请参阅

[watershed](#) | [imopen](#) | [imreconstruct](#) | [imclose](#) | [imdilate](#) | [imregionalmax](#) | [imerode](#) |
[bwareaopen](#) | [bwdist](#) | [label2rgb](#) | [imcomplement](#) | [labeloverlay](#) | [imgradient](#)

图像去模糊

本章说明如何使用工具箱的去模糊函数对图像进行去模糊。

- “使用 Wiener 滤波器对图像进行去模糊”（第 14-2 页）
- “使用盲反卷积算法对图像进行去模糊”（第 14-9 页）

使用 Wiener 滤波器对图像进行去模糊

此示例说明如何使用 Wiener 反卷积对图像进行去模糊。当图像的频率特征和加性噪声已知时，至少在一定程度上可以有效地使用 Wiener 反卷积。

读取原始图像

读取并显示没有模糊或噪声的原始图像。

```
Ioriginal = imread('cameraman.tif');
imshow(Ioriginal)
title('Original Image')
```



模拟和还原无噪声运动模糊

模拟照相机移动可能导致的模糊图像。首先，创建点扩散函数 PSF，方法是使用 `fspecial` 函数并指定以 11 度角跨 21 个像素的线性运动。然后，使用 `imfilter` 将点扩散函数与图像进行卷积。

原始图像的数据类型为 `uint8`。如果您将 `uint8` 图像传递给 `imfilter`，则该函数将量化输出以返回另一个 `uint8` 图像。要减少量化误差，请在调用 `imfilter` 之前将图像转换为 `double`。

```
PSF = fspecial('motion',21,11);
Idouble = im2double(Ioriginal);
blurred = imfilter(Idouble,PSF,'conv','circular');
imshow(blurred)
title('Blurred Image')
```

Blurred Image

使用 **deconvwnr** 函数还原模糊图像。模糊图像没有噪声，因此您可以省略噪信比 (NSR) 输入参数。

```
wnr1 = deconvwnr(blurred,PSF);  
imshow(wnr1)  
title('Restored Blurred Image')
```

Restored Blurred Image

模拟和还原运动模糊和高斯噪声

使用 `imnoise` 函数将零均值高斯噪声添加到模糊图像中。

```
noise_mean = 0;
noise_var = 0.0001;
blurred_noisy = imnoise(blurred,'gaussian',noise_mean,noise_var);
imshow(blurred_noisy)
title('Blurred and Noisy Image')
```

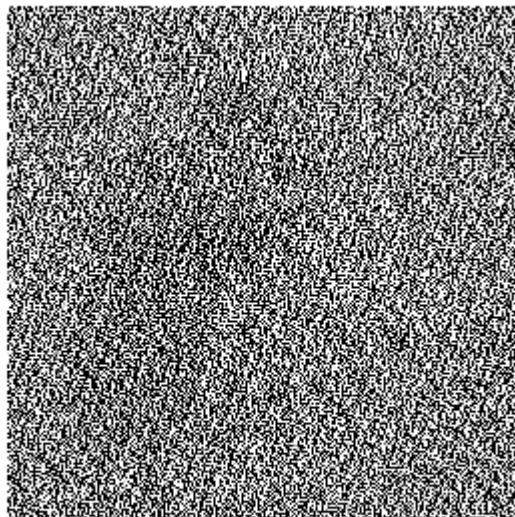
Blurred and Noisy Image



尝试使用 `deconvwnr` 还原模糊的含噪图像，但不提供噪声估计值。默认情况下，Wiener 还原滤波器假设 NSR 等于 0。在这种情况下，Wiener 还原滤波器等效于理想的逆滤波器，它对输入图像中的噪声非常敏感。

在此示例中，该还原过程将噪声放大到了使图像内容丢失的程度。

```
wnr2 = deconvwnr(blurred_noisy,PSF);
imshow(wnr2)
title('Restoration of Blurred Noisy Image (NSR = 0)')
```

Restoration of Blurred Noisy Image (NSR = 0)

尝试使用 `deconvwnr` 和更实际的估计噪声值来还原模糊的含噪图像。

```
signal_var = var(Idouble(:));
NSR = noise_var / signal_var;
wnr3 = deconvwnr(blurred_noisy,PSF,NSR);
imshow(wnr3)
title('Restoration of Blurred Noisy Image (Estimated NSR)')
```

Restoration of Blurred Noisy Image (Estimated NSR)

模拟和还原运动模糊和 8 位量化噪声

即使视觉上察觉不到的噪声也会影响结果。噪声的一个来源是在处理以 `uint8` 表示的图像中产生的量化误差。以前，为了避免量化误差，此示例基于数据类型为 `double` 的原始图像模拟模糊图像。现在，为了探究量化误差对还原的影响，基于原始 `uint8` 数据类型的原始图像模拟模糊图像。

```
blurred_quantized = imfilter(Ioriginal,PSF,'conv','circular');
imshow(blurred_quantized)
title('Blurred Quantized Image')
```

Blurred Quantized Image



尝试使用 `deconvwnr` 还原模糊量化图像，但不提供噪声估计值。尽管没有添加额外的噪声，但与数据类型为 `double` 的模糊图像的还原相比，此还原降低了质量。

```
wnr4 = deconvwnr(blurred_quantized,PSF);
imshow(wnr4)
title('Restoration of Blurred Quantized Image (NSR = 0)');
```

Restoration of Blurred Quantized Image (NSR = 0)

尝试使用 **deconvwnr** 和更实际的估计噪声值来还原模糊量化图像。

```
uniform_quantization_var = (1/256)^2 / 12;
signal_var = var(Idouble());
NSR = uniform_quantization_var / signal_var;
wnr5 = deconvwnr(blurred_quantized,PSF,NSR);
imshow(wnr5)
title('Restoration of Blurred Quantized Image (Estimated NSR)');
```

Restoration of Blurred Quantized Image (Estimated NSR)



另请参阅

[deconvwnr](#) | [fspecial](#) | [imfilter](#) | [imnoise](#)

详细信息

- “Image Deblurring”

使用盲反卷积算法对图像进行去模糊

此示例说明如何使用盲反卷积对图像进行去模糊。当不知道关于失真（模糊和噪声）的信息时，使用盲反卷积算法很有效。该算法同时还原图像和点扩散函数 (PSF)。每次迭代中都使用加速阻尼 Richardson-Lucy 算法。附加的光学系统（例如照相机）特征可以用作输入参数，这有助于提高图像还原的质量。PSF 约束可以通过用户指定的函数传递。

步骤 1：读取图像

将灰度图像读入工作区。`deconvblind` 函数可以处理任何维度的数组。

```
I = imread('cameraman.tif');
figure;imshow(I);title('Original Image');
text(size(I,2),size(I,1)+15, ...
'Image courtesy of Massachusetts Institute of Technology',...
'FontSize',7,'HorizontalAlignment','right');
```



Image courtesy of Massachusetts Institute of Technology

步骤 2：仿真模糊效果

模拟模糊的真实图像（例如，因为照相机移动或失焦而模糊）。此示例通过使用高斯滤波器对真实图像进行卷积来模拟模糊效果（使用 `imfilter`）。然后，高斯滤波器表示点扩散函数 PSF。

```
PSF = fspecial('gaussian',7,10);
Blurred = imfilter(I,PSF,'symmetric','conv');
imshow(Blurred)
title('Blurred Image')
```



步骤 3：使用不同大小的 PSF 还原模糊图像

为了说明了解真实 PSF 大小的重要性，此示例执行三次还原。对于每次还原，PSF 重构都从均匀数组（由 1 组成的数组）开始。

第一次还原（J1 和 P1）使用大小过小的数组 UNDERPSF 获得 PSF 的初始估计值。在每个维度上，UNDERPSF 数组的大小都比真实 PSF 短 4 个像素。

```
UNDERPSF = ones(size(PSF)-4);
[J1,P1] = deconvblind(Blurred,UNDERPSF);
imshow(J1)
title('Deblurring with Undersized PSF')
```

Deblurring with Undersized PSF

第二次还原 (J2 和 P2) 使用由 1 组成的数组 OVERPSF 来获得初始 PSF，该初始 PSF 在每个维度上都比真实 PSF 长 4 个像素。

```
OVERPSF = padarray(UNDERPSF,[4 4],'replicate','both');  
[J2,P2] = deconvblind(Blurred,OVERPSF);  
imshow(J2)  
title('Deblurring with Oversized PSF')
```

Deblurring with Oversized PSF

第三次还原 (J3 和 P3) 使用由 1 组成的数组 INITPSF 获得与真实 PSF 大小完全相同的初始 PSF。

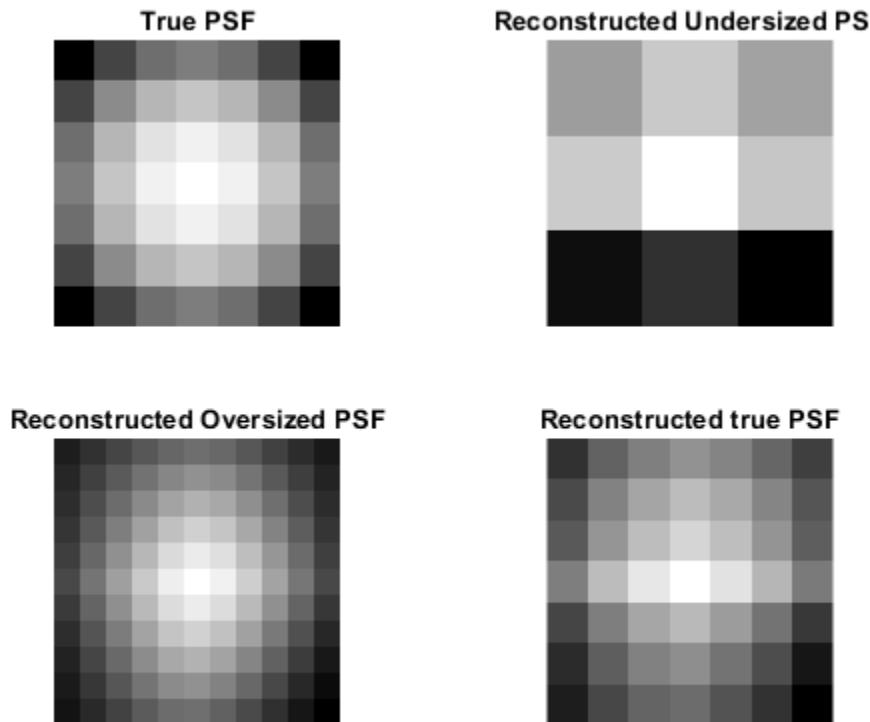
```
INITPSF = padarray(UNDERPSF,[2 2],'replicate','both');
[J3,P3] = deconvblind(Blurred,INITPSF);
imshow(J3)
title('Deblurring with INITPSF')
```



步骤 4：分析还原的 PSF

所有三个还原还会生成一个 PSF。下列各图显示对重构的 PSF 的分析如何有助于估计初始 PSF 的正确大小。在真实的 PSF (一个高斯滤波器) 中，最大值在中心 (白色)，在边界处 (黑色) 减小。

```
figure;
subplot(2,2,1)
imshow(PSF,[],'InitialMagnification','fit')
title('True PSF')
subplot(2,2,2)
imshow(P1,[],'InitialMagnification','fit')
title('Reconstructed Undersized PSF')
subplot(2,2,3)
imshow(P2,[],'InitialMagnification','fit')
title('Reconstructed Oversized PSF')
subplot(2,2,4)
imshow(P3,[],'InitialMagnification','fit')
title('Reconstructed true PSF')
```



在第一次还原中重构的 PSF P1 显然无法放入限定的大小之内。它在边界上有很强的信号变化。相对于模糊图像 Blurred，对应的图像 J1 在清晰度上并未显示出任何改进。

在第二个还原中重构的 PSF (P2) 在边缘处变得非常平滑。这意味着还原可以处理大小较小的 PSF。对应的图像 J2 显示消除了一些模糊，但被振铃效应严重破坏。

最后，在第三个还原中重构的 PSF P3 在某种程度上介于 P1 和 P2 之间。数组 P3 非常像真实的 PSF。对应的图像 J3 显示出显著的改进；然而，它仍然被振铃效应破坏。

步骤 5：改进还原

还原的图像 J3 中的振铃效应发生在图像中强度对比强烈的区域和图像边界上。此示例说明如何通过指定加权函数来减少振铃效应。该算法根据 WEIGHT 数组对每个像素进行加权，同时还原图像和 PSF。在我们的示例中，我们首先使用边缘函数找到“较高锐度”的像素。通过反复试验，我们确定理想的阈值水平是 0.08。

```
WEIGHT = edge(Blurred,'sobel',.08);
```

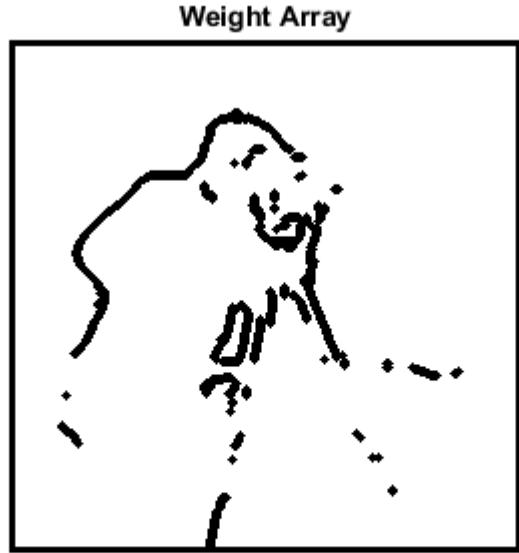
为了扩大区域，我们使用 imdilate 并传入结构元素 se。

```
se = strel('disk',2);
WEIGHT = 1-double(imdilate(WIGHT,se));
```

靠近边界的像素也被赋值为 0。

```
WEIGHT([1:3 end-(0:2)],:) = 0;
WEIGHT(:,[1:3 end-(0:2)]) = 0;
```

```
figure  
imshow(WEIGHT)  
title('Weight Array')
```



调用 deconvblind 还原图像时加上 WEIGHT 数组并增加迭代次数 (30)。几乎所有振铃效应都被消除。

```
[J,P] = deconvblind(Blurred,INITPSF,30,[],WEIGHT);  
imshow(J)  
title('Deblurred Image')
```



步骤 6：对 PSF 还原使用附加约束

此示例说明如何对 PSF 指定附加约束。以下函数 `FUN` 返回修正 PSF 数组，`deconvblind` 将该数组用于下一次迭代。

在此示例中，`FUN` 会修正 PSF，分别在每个维度中裁去 $P1$ 和 $P2$ 个像素，然后用零回填数组以恢复其原始大小。此运算不会更改 PSF 中心的值，但实际上会将 PSF 大小减小 $2*P1$ 和 $2*P2$ 个像素。

```
P1 = 2;
P2 = 2;
FUN = @(PSF) padarray(PSF(P1+1:end-P1,P2+1:end-P2),[P1 P2]);
```

匿名函数 `FUN` 最后会传入 `deconvblind` 中。有关向函数 `FUN` 提供附加参数的信息，请参阅 MATLAB Mathematics 文档中的 Parameterizing Functions 一节。

在此示例中，初始 PSF OVERPSF 的大小比真实 PSF 大 4 个像素。在 `FUN` 中将 $P1 = 2$ 和 $P2 = 2$ 设置为参数，实际上使 OVERPSF 中有效空间的大小与真实 PSF 相同。因此，结果 (`JF` 和 `PF`) 与步骤 4 中的结果 `J` 和 `P` 类似，该步骤中执行卷积时使用了合适大小的 PSF，但没用调用 `FUN`。

```
[JF,PF] = deconvblind(Blurred,OVERPSF,30,[],WEIGHT,FUN);
imshow(JF)
title('Deblurred Image')
```



如果我们使用了过大的初始 PSF OVERPSF 而没有使用约束函数 FUN，得到的图像将类似于在步骤 3 中获得的不尽如人意的结果 J2。

请注意，在 FUN 之前的任何未指定参数都可以直接省略，例如此示例中的 DAMPAR 和 READOUT，不需要补上占位符 (:)。

另请参阅

[deconvlucy](#) | [deconvblind](#)

详细信息

- “Image Deblurring”
- “Adapt Blind Deconvolution for Various Image Distortions”

颜色

本章介绍有助于您处理彩色图像数据的工具箱函数。请注意，“颜色”包括灰色；因此，本章的大部分讨论既适用于彩色图像，也适用于灰度图像。

分块图像处理

本主题说明如何处理无法放入内存的大图像数据。大图像可以有多个分辨率级别。

邻域和块运算

本章讨论这些一般块处理函数。涵盖的主题包括

深度学习

本主题说明支持使用卷积神经网络实现图像去噪的函数，并提供使用深度学习技术的其他图像处理应用的示例。

- “使用预训练的神经网络去除彩色图像中的噪声”（第 18-2 页）
- “使用深度学习的神经样式迁移”（第 18-8 页）

使用预训练的神经网络去除彩色图像中的噪声

此示例说明如何从 RGB 图像中去除高斯噪声。将图像分为单独的颜色通道，然后使用预训练的去噪神经网络 DnCNN 对每个通道去噪。

将一个彩色图像读入工作区中，并将数据转换为 **double**。显示原始彩色图像。

```
pristineRGB = imread('lighthouse.png');
pristineRGB = im2double(pristineRGB);
imshow(pristineRGB)
title('Pristine Image')
```

Pristine Image

将方差为 0.01 的零均值高斯白噪声添加到图像中。`imnoise` 分别向每个颜色通道添加噪声。显示含噪彩色图像。

```
noisyRGB = imnoise(pristineRGB,'gaussian',0,0.01);
imshow(noisyRGB)
title('Noisy Image')
```

Noisy Image



将含噪 RGB 图像分为单独的颜色通道。

```
[noisyR,noisyG,noisyB] = imsplit(noisyRGB);
```

加载预训练的 DnCNN 网络。

```
net = denoisingNetwork('dncnn');
```

使用 DnCNN 网络去除每个颜色通道的噪声。

```
denoisedR = denoiseImage(noisyR,net);
denoisedG = denoiseImage(noisyG,net);
denoisedB = denoiseImage(noisyB,net);
```

合并去噪的颜色通道以形成去噪后的 RGB 图像。显示去噪后的彩色图像。

```
denoisedRGB = cat(3,denoisedR,denoisedG,denoisedB);
imshow(denoisedRGB)
title('Denoised Image')
```



计算含噪图像和去噪图像的峰值信噪比 (PSNR)。PSNR 越大，噪声相对信号越小，说明图像质量越高。

```
noisyPSNR = psnr(noisyRGB,pristineRGB);
fprintf('\n The PSNR value of the noisy image is %0.4f!',noisyPSNR);
```

The PSNR value of the noisy image is 20.6395.

```
denoisedPSNR = psnr(denoisedRGB,pristineRGB);
fprintf('\n The PSNR value of the denoised image is %0.4f.',denoisedPSNR);
```

The PSNR value of the denoised image is 29.6857.

计算含噪图像和去噪图像的结构相似性 (SSIM) 指数。SSIM 指数接近 1 表示与参考图像相当一致，图像质量更高。

```
noisySSIM = ssim(noisyRGB,pristineRGB);
fprintf('\n The SSIM value of the noisy image is %0.4f.',noisySSIM);
```

The SSIM value of the noisy image is 0.7393.

```
denoisedSSIM = ssim(denoisedRGB,pristineRGB);
fprintf('\n The SSIM value of the denoised image is %0.4f.',denoisedSSIM);
```

The SSIM value of the denoised image is 0.9507.

实际上，图像颜色通道经常具有相关噪声。为了去除相关的图像噪声，首先将 RGB 图像转换为具有亮度通道的颜色空间，例如 L*a*b* 颜色空间。仅去除亮度通道上的噪声，然后将去噪图像转换回 RGB 颜色空间。

另请参阅

[denoisingNetwork](#) | [denoiseImage](#) | [rgb2lab](#) | [lab2rgb](#) | [psnr](#) | [ssim](#) | [imnoise](#)

详细信息

- “[Train and Apply Denoising Neural Networks](#)”

使用深度学习的神经样式迁移

此示例说明如何使用预训练的 VGG-19 网络 [1] (第 18-0 页) 将一个图像的样式外观应用于另一个图像的场景内容。

加载数据

加载样式图像和内容图像。此示例使用梵高的名作《星夜》作为样式图像，使用一个灯塔照片作为内容图像。

```
styleImage = im2double(imread('starryNight.jpg'));
contentImage = imread('lighthouse.png');
```

以蒙太奇方式显示样式图像和内容图像。

```
imshow(imtile({styleImage,contentImage},'BackgroundColor','w'));
```



加载特征提取网络

在此示例中，您使用一个经过修改的预训练 VGG-19 深度神经网络在不同层提取内容图像和样式图像的特征。这些多层特征用于计算各自的内容和样式损失。该网络使用组合损失生成样式的迁移图像。

要获得一个预训练 VGG-19 网络，请安装 `vgg19` (Deep Learning Toolbox)。如果没有安装所需的支持包，软件会提供下载链接。

```
net = vgg19;
```

要使 VGG-19 网络适合特征提取，请从网络中删除所有全连接层。

```
lastFeatureLayerIdx = 38;
layers = net.Layers;
layers = layers(1:lastFeatureLayerIdx);
```

VGG-19 网络的最大池化层会造成衰落效应。要降低衰落效应并增加梯度流，请将所有最大池化层替换为平均池化层 [1] (第 18-0 页)。

```
for l = 1:lastFeatureLayerIdx
    layer = layers(l);
```

```

if isa(layer,'nnet.cnn.layer.MaxPooling2DLayer')
    layers(l) = averagePooling2dLayer(layer.PoolSize,'Stride',layer.Stride,'Name',layer.Name);
end
end

```

使用修改后的层创建一个层次图。

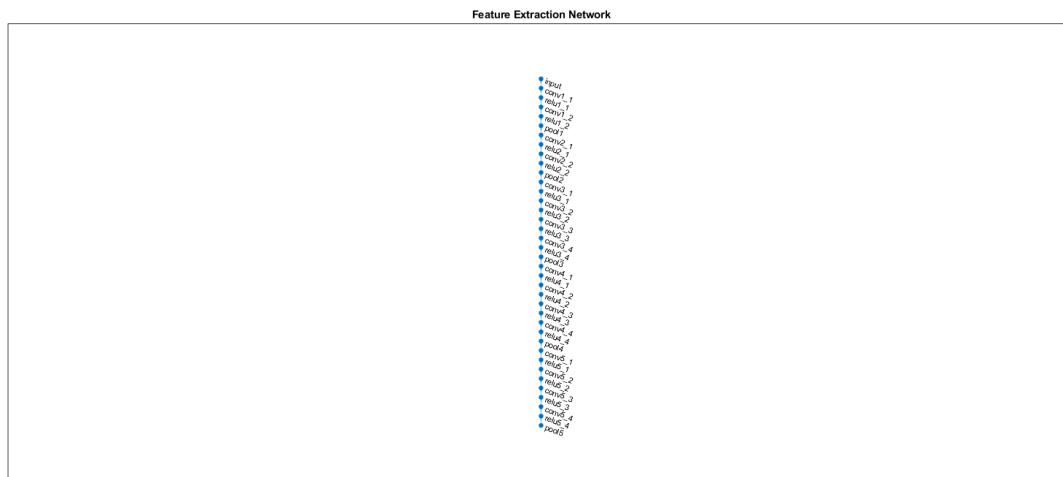
```
lgraph = layerGraph(layers);
```

在图中可视化特征提取网络。

```

plot(lgraph)
title('Feature Extraction Network')

```



要使用自定义训练循环训练网络并支持自动微分，请将层次图转换为 **dlnetwork** 对象。

```
dlnet = dlnetwork(lgraph);
```

预处理数据

将样式图像和内容图像的大小调整为较小的大小，以加快处理速度。

```

imageSize = [384,512];
styleImg = imresize(styleImage,imageSize);
contentImg = imresize(contentImage,imageSize);

```

预训练的 VGG-19 网络对减去了通道均值的图像执行分类。从图像输入层获取通道均值，图像输入层是网络中的第一层。

```

imgInputLayer = lgraph.Layers(1);
meanVggNet = imgInputLayer.Mean(1,1,:);

```

通道均值适用于像素值在 [0, 255] 范围内的浮点数据类型的图像。将样式图像和内容图像转换为范围在 [0, 255] 范围内的数据类型 **single**。然后，从样式图像和内容图像中减去通道均值。

```

styleImg = rescale(single(styleImg),0,255) - meanVggNet;
contentImg = rescale(single(contentImg),0,255) - meanVggNet;

```

初始化迁移图像

迁移图像是作为样式迁移结果的输出图像。您可以使用样式图像、内容图像或任何随机图像来初始化迁移图像。使用样式图像或内容图像初始化会使样式迁移处理发生偏置，并产生与输入图像更相似的迁移图像。相反，使用白噪声初始化可以消除偏置，但需要更长时间才能收敛于样式化图像。为了实现更好的样式化和更快的收敛，此示例将输出迁移图像初始化为内容图像和白噪声图像的加权组合。

```
noiseRatio = 0.7;
randImage = randi([-20,20],[imageSize 3]);
transferImage = noiseRatio.*randImage + (1-noiseRatio).*contentImg;
```

定义损失函数和样式迁移参数

内容损失

内容损失的目标是使迁移图像的特征与内容图像的特征相匹配。内容损失计算的是每个内容特征层的内容图像特征与迁移图像特征之间的均方差 [1]（第 18-0 页）。 \hat{Y} 是迁移图像的预测特征图， Y 是内容图像的预测特征图。 W_c^l 是 l^{th} 层的内容层权重。 H, W, C 分别是特征图的高度、宽度和通道。

$$L_{content} = \sum_l W_c^l \times \frac{1}{HWC} \sum_{i,j} (\hat{Y}_{i,j}^l - Y_{i,j}^l)^2$$

指定内容特征提取层的名称。从这些层提取的特征用于计算内容损失。在 VGG-19 网络中，使用更深层的特征比使用浅层的特征更高效。因此，将内容特征提取层指定为第四个卷积层。

```
styleTransferOptions.contentFeatureLayerNames = {'conv4_2'};
```

指定内容特征提取层的权重。

```
styleTransferOptions.contentFeatureLayerWeights = 1;
```

样式损失

样式损失的目标是使迁移图像的纹理与样式图像的纹理相匹配。图像的样式表示为一个 Gram 矩阵。因此，样式损失计算的是样式图像的 Gram 矩阵与迁移图像的 Gram 矩阵之间的均方差 [1]（第 18-0 页）。 Z 和 \hat{Z} 分别是样式图像和迁移图像的预测特征图。 G_Z 和 $G_{\hat{Z}}$ 分别是样式特征和迁移特征的 Gram 矩阵。 W_s^l 是 l^{th} 样式层的样式层权重。

$$G_{\hat{Z}} = \sum_{i,j} \hat{Z}_{i,j} \times \hat{Z}_{j,i}$$

$$G_Z = \sum_{i,j} Z_{i,j} \times Z_{j,i}$$

$$L_{style} = \sum_l W_s^l \times \frac{1}{(2HWC)^2} \sum (G_{\hat{Z}}^l - G_Z^l)^2$$

指定样式特征提取层的名称。从这些层中提取的特征用于计算样式损失。

```
styleTransferOptions.styleFeatureLayerNames = {'conv1_1','conv2_1','conv3_1','conv4_1','conv5_1'};
```

指定样式特征提取层的权重。为简单样式图像指定较小的权重，为复杂样式图像增大权重。

```
styleTransferOptions.styleFeatureLayerWeights = [0.5,1.0,1.5,3.0,4.0];
```

总损失

总损失是内容损失和样式损失的加权组合。 α 和 β 分别是内容损失和样式损失的权重因子。

$$L_{total} = \alpha \times L_{content} + \beta \times L_{style}$$

为内容损失和样式损失指定权重因子 **alpha** 和 **beta**。**alpha** 与 **beta** 的比率应在 1e-3 或 1e-4 左右 [1] (第 18-0 页)。

```
styleTransferOptions.alpha = 1;
styleTransferOptions.beta = 1e3;
```

指定训练选项

进行 2500 次迭代的训练。

```
numIterations = 2500;
```

指定 Adam 优化的选项。将学习率设置为 2 以加快收敛速度。您可以通过观察输出图像和损失来尝试调整学习率。用 [] 初始化尾部平均梯度和尾部平均梯度平方衰减率。

```
learningRate = 2;
trailingAvg = [];
trailingAvgSq = [];
```

训练网络

将样式图像、内容图像和迁移图像转换为基础类型为 **single** 和维度标签为 'SSC' 的 **dlarray** (Deep Learning Toolbox) 对象。

```
dlStyle = dlarray(styleImg,'SSC');
dlContent = dlarray(contentImg,'SSC');
dlTransfer = dlarray(transferImage,'SSC');
```

在 GPU 上 (如果有) 进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。对于 GPU 训练，需将数据转换为 **gpuArray**。

```
if canUseGPU
    dlContent = gpuArray(dlContent);
    dlStyle = gpuArray(dlStyle);
    dlTransfer = gpuArray(dlTransfer);
end
```

从内容图像中提取内容特征。

```
numContentFeatureLayers = numel(styleTransferOptions.contentFeatureLayerNames);
contentFeatures = cell(1,numContentFeatureLayers);
[contentFeatures{:}] = forward(dlnet,dlContent,'Outputs',styleTransferOptions.contentFeatureLayerNames);
```

从样式图像中提采样式特征。

```
numStyleFeatureLayers = numel(styleTransferOptions.styleFeatureLayerNames);
styleFeatures = cell(1,numStyleFeatureLayers);
[styleFeatures{:}] = forward(dlnet,dlStyle,'Outputs',styleTransferOptions.styleFeatureLayerNames);
```

使用自定义训练循环训练模型。对于每次迭代：

- 使用内容图像、样式图像和迁移图像的特征来计算内容损失和样式损失。要计算损失和梯度，请使用辅助函数 `imageGradients` (在此示例的支持函数 (第 18-0 页) 部分中定义)。
- 使用 `adamupdate` (Deep Learning Toolbox) 函数更新迁移图像。
- 选择最佳样式迁移图像作为最终输出图像。

```
figure  
  
minimumLoss = inf;  
  
for iteration = 1:numIterations  
    % Evaluate the transfer image gradients and state using dlfeval and the  
    % imageGradients function listed at the end of the example.  
    [grad,losses] = dlfeval(@imageGradients,dlnet,dlTransfer,contentFeatures,styleFeatures,styleTransferOptions  
    [dlTransfer,trailingAvg,trailingAvgSq] = adamupdate(dlTransfer,grad,trailingAvg,trailingAvgSq,iteration,learnRate);  
  
    if losses.totalLoss < minimumLoss  
        minimumLoss = losses.totalLoss;  
        dlOutput = dlTransfer;  
    end  
  
    % Display the transfer image on the first iteration and after every 50  
    % iterations. The postprocessing steps are described in the "Postprocess  
    % Transfer Image for Display" section of this example.  
    if mod(iteration,50) == 0 || (iteration == 1)  
  
        transferImage = gather(extractdata(dlTransfer));  
        transferImage = transferImage + meanVggNet;  
        transferImage = uint8(transferImage);  
        transferImage = imresize(transferImage,size(contentImage,[1 2]));  
  
        image(transferImage)  
        title(['Transfer Image After Iteration ',num2str(iteration)])  
        axis off image  
        drawnow  
    end  
  
end
```

Transfer Image After Iteration 2500

后处理迁移图像以用于显示

获取更新后的迁移图像。

```
transferImage = gather(extractdata(dlOutput));
```

为迁移图像加上网络训练后的均值。

```
transferImage = transferImage + meanVggNet;
```

某些像素值可能会超过内容和样式图像的原始范围 [0, 255]。通过将数据类型转换为 `uint8`, 您可以将值裁剪到 [0, 255] 的范围之内。

```
transferImage = uint8(transferImage);
```

将迁移图像的大小调整为内容图像的原始大小。

```
transferImage = imresize(transferImage,size(contentImage,[1 2]));
```

以蒙太奇方式显示内容图像、迁移图像和样式图像。

```
imshow(imtile({contentImage,transferImage,styleImage}, ...
    'GridSize',[1 3],'BackgroundColor','w'));
```



支持函数

计算图像损失和梯度

`imageGradients` 辅助函数使用内容图像、样式图像和迁移图像的特征来返回损失和梯度。

```
function [gradients,losses] = imageGradients(dlnet,dlTransfer,contentFeatures,styleFeatures,params)

    % Initialize transfer image feature containers.
    numContentFeatureLayers = numel(params.contentFeatureLayerNames);
    numStyleFeatureLayers = numel(params.styleFeatureLayerNames);

    transferContentFeatures = cell(1,numContentFeatureLayers);
    transferStyleFeatures = cell(1,numStyleFeatureLayers);

    % Extract content features of transfer image.
    [transferContentFeatures{:}] = forward(dlnet,dlTransfer,'Outputs',params.contentFeatureLayerNames);

    % Extract style features of transfer image.
    [transferStyleFeatures{:}] = forward(dlnet,dlTransfer,'Outputs',params.styleFeatureLayerNames);

    % Compute content loss.
    cLoss = contentLoss(transferContentFeatures,contentFeatures,params.contentFeatureLayerWeights);

    % Compute style loss.
    sLoss = styleLoss(transferStyleFeatures,styleFeatures,params.styleFeatureLayerWeights);

    % Compute final loss as weighted combination of content and style loss.
    loss = (params.alpha * cLoss) + (params.beta * sLoss);

    % Calculate gradient with respect to transfer image.
    gradients = dlgradient(loss,dlTransfer);

    % Extract various losses.
    losses.totalLoss = gather(extractdata(loss));
```

```

losses.contentLoss = gather(extractdata(cLoss));
losses.styleLoss = gather(extractdata(sLoss));

end

```

计算内容损失

contentLoss 辅助函数计算内容图像特征和迁移图像特征之间的加权均方差。

```

function loss = contentLoss(transferContentFeatures,contentFeatures,contentWeights)

loss = 0;
for i=1:numel(contentFeatures)
    temp = 0.5 .* mean((transferContentFeatures{1,i} - contentFeatures{1,i}).^2,'all');
    loss = loss + (contentWeights(i)*temp);
end
end

```

计算样式损失

styleLoss 辅助函数计算样式图像特征的 Gram 矩阵和迁移图像特征的 Gram 矩阵之间的加权均方差。

```

function loss = styleLoss(transferStyleFeatures,styleFeatures,styleWeights)

loss = 0;
for i=1:numel(styleFeatures)

    tsf = transferStyleFeatures{1,i};
    sf = styleFeatures{1,i};
    [h,w,c] = size(sf);

    gramStyle = computeGramMatrix(sf);
    gramTransfer = computeGramMatrix(tsf);
    sLoss = mean((gramTransfer - gramStyle).^2,'all') / ((h*w*c)^2);

    loss = loss + (styleWeights(i)*sLoss);
end
end

```

计算 Gram 矩阵

styleLoss 辅助函数使用 **computeGramMatrix** 辅助函数来计算特征图的 Gram 矩阵。

```

function gramMatrix = computeGramMatrix(featureMap)
    [H,W,C] = size(featureMap);
    reshapedFeatures = reshape(featureMap,H*W,C);
    gramMatrix = reshapedFeatures' * reshapedFeatures;
end

```

参考资料

[1] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge."A Neural Algorithm of Artistic Style."Preprint, submitted September 2, 2015. <https://arxiv.org/abs/1508.06576>

另请参阅

`vgg19` | `trainNetwork` | `trainingOptions` | `dlarray`

详细信息

- “Get Started with GANs for Image-to-Image Translation”
- “定义自定义训练循环、损失函数和网络” (Deep Learning Toolbox)
- “Specify Training Options in Custom Training Loop” (Deep Learning Toolbox)
- “Train Network Using Custom Training Loop” (Deep Learning Toolbox)
- “List of Functions with dlarray Support” (Deep Learning Toolbox)
- “深度学习层列表” (Deep Learning Toolbox)

高光谱图像处理

本主题说明支持高光谱图像分析的函数，并提供使用端元和丰度图进行光谱分类和异常检测的示例。

Image Processing Toolbox 函数的代码生成

- “图像处理的代码生成” (第 20-2 页)
- “为目标检测生成代码” (第 20-4 页)

图像处理的代码生成

现已支持使用特定的 Image Processing Toolbox 函数来生成 C 代码（需要 MATLAB Coder）。要将代码生成与图像处理函数结合使用，请执行以下步骤：

- 使用 Image Processing Toolbox 中的函数，像平常一样编写您的 MATLAB 函数或应用程序。
- 在函数签名的末尾添加 %#codegen 编译器指令。此指令指示 MATLAB 代码分析器诊断会阻止成功生成代码的问题。
- 打开 **MATLAB Coder**，创建一个工程，并将您的文件添加到工程中。在该 App 中，您可以检查代码的就绪情况以进行代码生成。例如，您的代码可能包含不支持代码生成的函数。针对代码生成进行必需的所有修改。
- 在 **MATLAB Coder** 的 Generate Code 页上点击 **Generate**，以生成代码。您可以选择生成 MEX 文件、共享库、动态库或可执行文件。

即使您解决了 **MATLAB Coder** 标识的所有就绪问题，您仍可能会遇到编译问题。就绪检查只查看函数依存关系。当您尝试生成代码时，**MATLAB Coder** 可能会发现代码生成不支持的编码模式。请查看错误报告并修改您的 MATLAB 代码，直到编译成功。

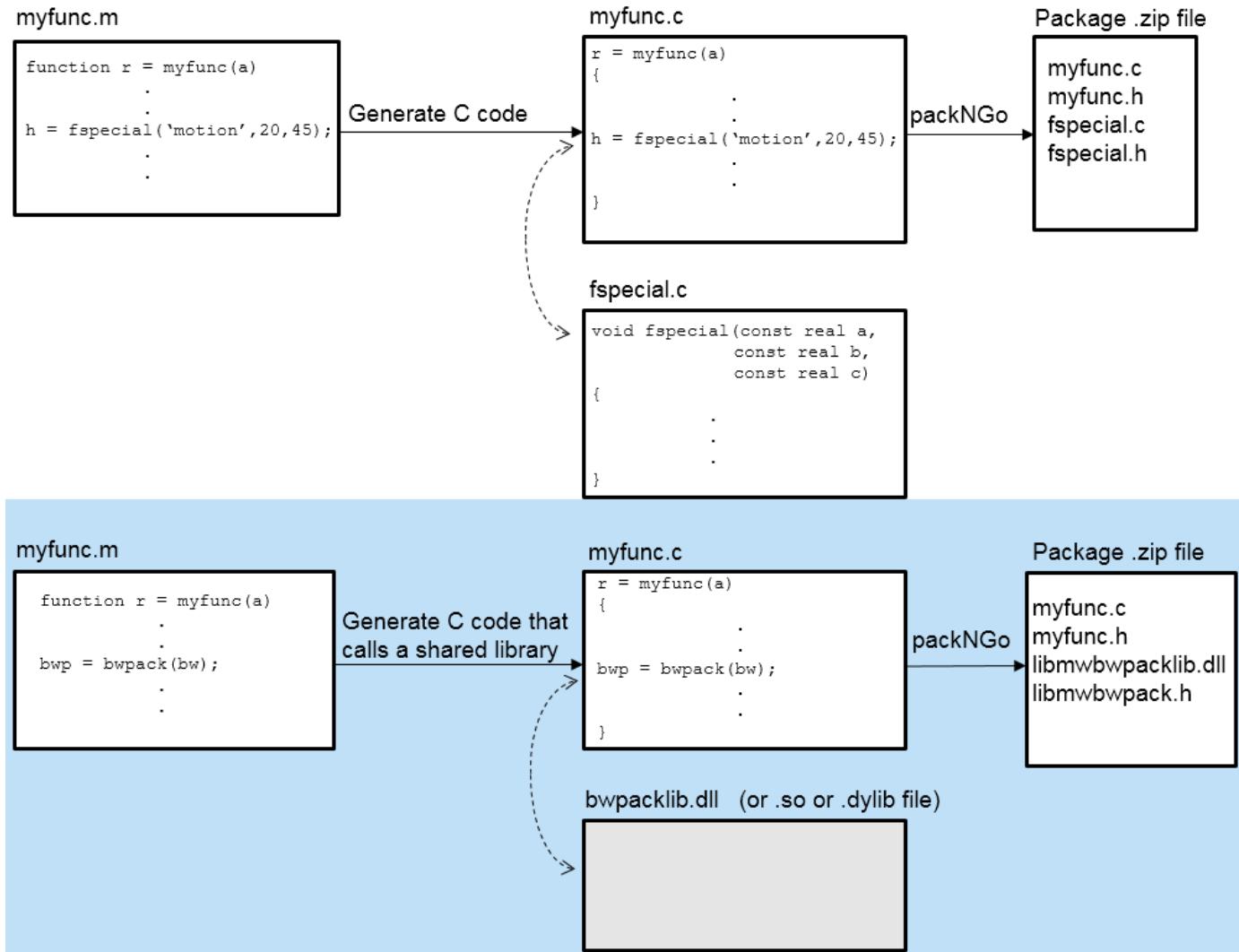
有关支持代码生成的 Image Processing Toolbox 函数的完整列表，请参阅支持代码生成的函数。有关使用代码生成的示例，请参阅“为目标检测生成代码”（第 20-4 页）。

使用共享库进行代码生成

Image Processing Toolbox 函数可以生成独立的 C 代码，也可以生成依赖于预编译的特定于平台的共享库的代码。

- 有些函数可生成独立的 C 代码，您可以将这些代码合并到可在多个平台（例如 ARM 处理器）上运行的应用程序中。
- 有些函数生成使用特定于平台的共享库的 C 代码。Image Processing Toolbox 使用这种共享库方法来保持性能优化，但这会将您的代码限制为只能在能够托管 MATLAB 的平台上运行。要查看主机平台列表，请参阅系统要求。
- 有些函数可以生成独立的 C 代码，也可以生成依赖于共享库的代码，具体取决于您在 MATLAB Coder 配置设置中选择的目标。
 - 如果选择通用的 **MATLAB Host Computer** 选项，这些函数将提供使用共享库的代码。
 - 如果您选择任何其他平台选项，这些函数将提供 C 代码。

下图说明生成 C 代码和生成使用共享库的代码之间的区别。



另请参阅

[codegen](#) | [MATLAB Coder](#)

相关示例

- “为目标检测生成代码” (第 20-4 页)

详细信息

- “代码生成工作流” (MATLAB Coder)
- “使用 MATLAB Coder App 生成 C 代码” (MATLAB Coder)
- 支持代码生成的函数

为目标检测生成代码

此示例说明如何使用 MATLAB® Coder™ 从使用 Image Processing Toolbox™ 函数的 MATLAB 应用程序中生成 C 代码。此示例说明如何设置您的 MATLAB 环境，并准备您的 MATLAB 代码以用于代码生成。

此示例还说明如何解决在 MATLAB 代码中可能遇到的阻止代码生成的问题。为了说明此过程，此示例使用的代码存在一些就绪问题和编译问题，您必须解决这些问题才能生成代码。

有关生成代码的详细信息，请参阅 MATLAB Coder 文档。

设置编译器

通过使用带 **-setup** 选项的 **mex** 函数，指定要与 MATLAB Coder 结合使用的 C/C++ 编译器来生成代码。

mex -setup

```
MEX configured to use 'MinGW64 Compiler (C)' for C language compilation.
```

To choose a different language, select one from the following:

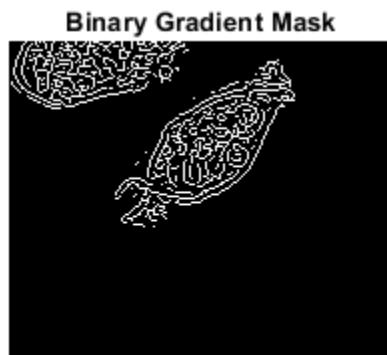
```
mex -setup C++
mex -setup FORTRAN
```

定义入口函数

入口函数是一个 MATLAB 函数，用作代码生成的源代码。首先，在不支持代码生成的情况下建立图像处理工作流原型。此示例定义名为 **detectCells.m** 的函数，该函数使用分割和形态学方法来执行细胞检测。此函数作为支持文件包含在本示例中。

使用示例图像 **cell.tif** 测试示例代码。

```
I = imread('cell.tif');
Iseg = detectCells(I);
```



Dilated Gradient Mask

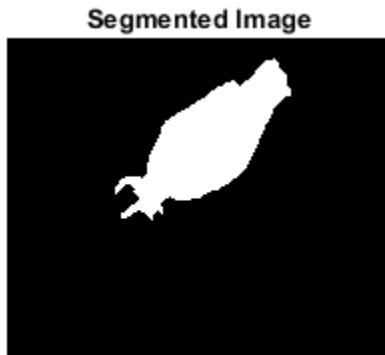


Binary Image with Filled Holes



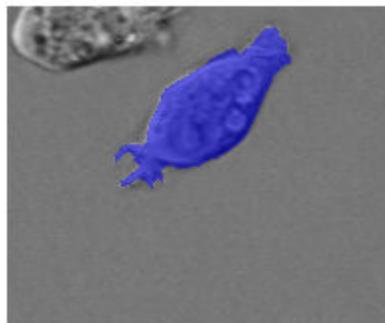
Cleared Border Image





通过在原始图像上叠加分割图像来确认分割的准确度。

```
imshow(labeloverlay(I,Iseg))
```

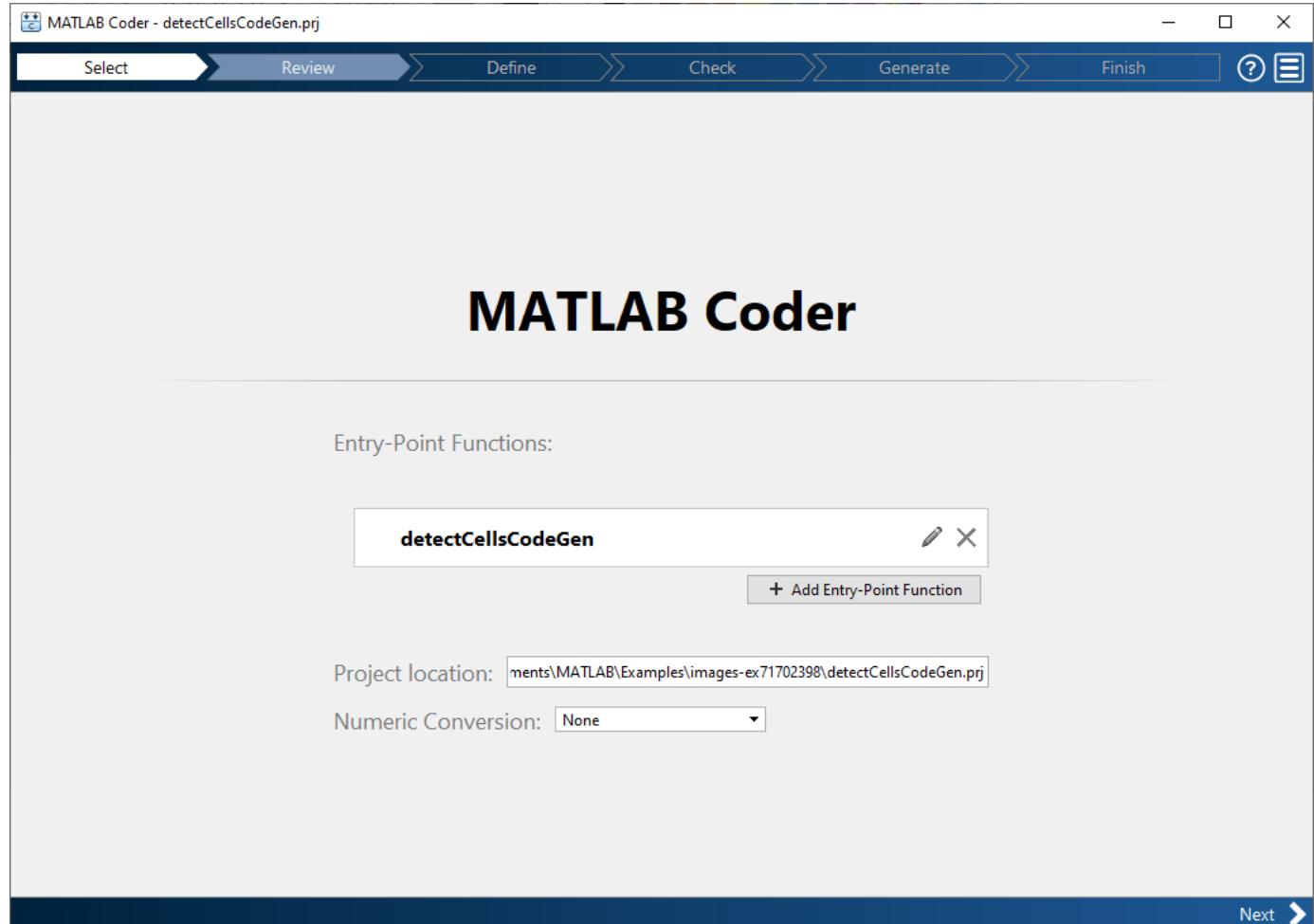


由于您要修改此代码以进行代码生成，因此最好使用代码的副本。此示例包含名为 `detectCellsCodeGen.m` 的辅助函数 `detectCells.m` 的副本。用于代码生成的函数版本包含位于函数签名末尾的 MATLAB Coder 编译指令 `%#codegen`。此指令指示 MATLAB 代码分析器诊断会阻止成功生成代码的问题。

使用 `coder` 函数打开 MATLAB Coder。（或者，在 MATLAB 中，选择 App 选项卡，导航到“代码生成”，然后点击 MATLAB Coder。）

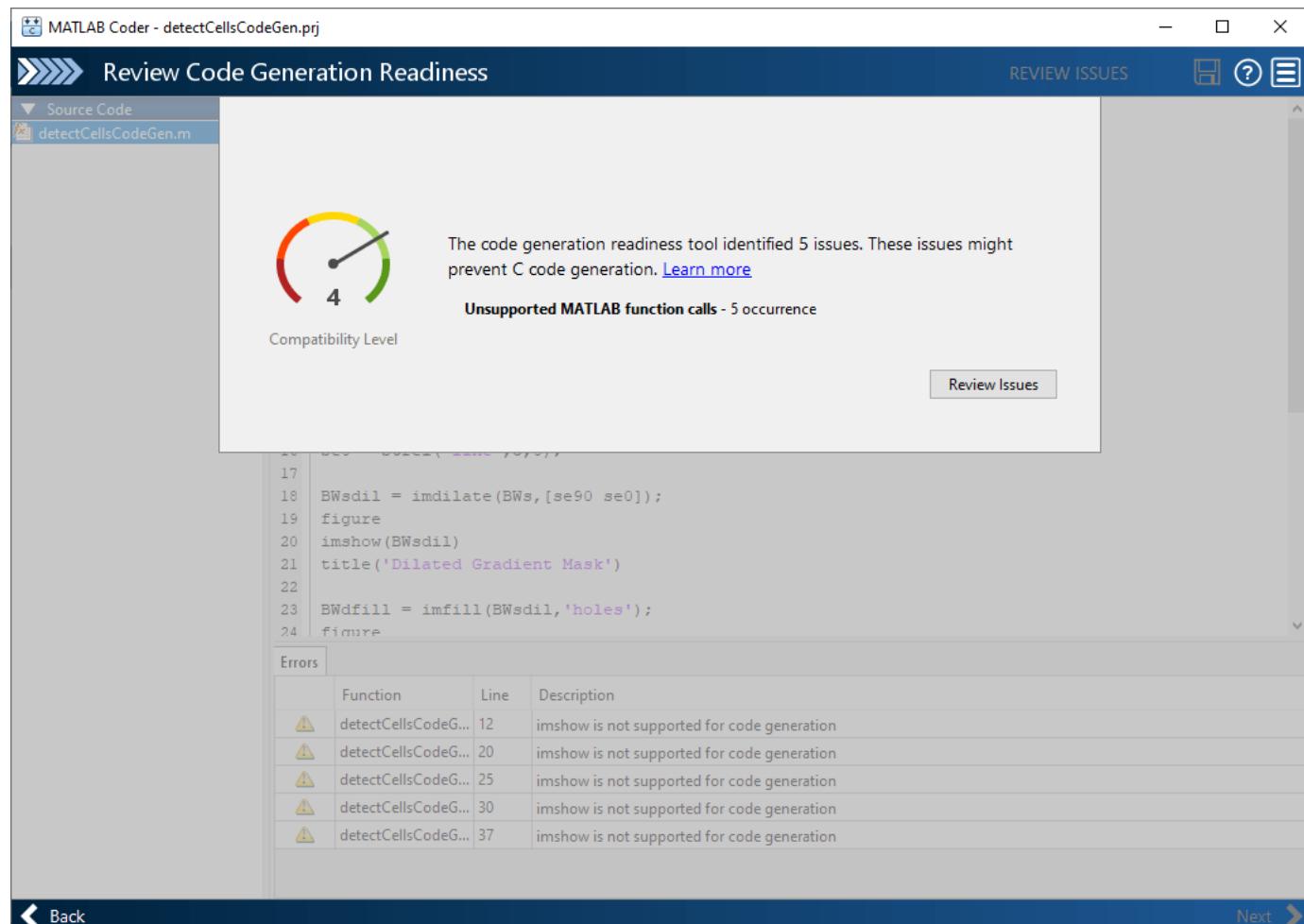
```
coder
```

将您的入口函数的名称指定为 `detectCellsCodeGen`，然后按 **Enter** 键。



确定代码就绪情况以进行代码生成

点击 **Next**。 MATLAB Coder 会标识任何可能阻止代码生成的问题。示例代码包含五个不受支持的函数调用。



检查就绪问题。点击 **Review Issues**。在报告中，MATLAB Coder 在编辑窗口中显示您的代码，并在下方列出就绪问题，标记使用了不支持代码生成的 `imshow` 函数。

The screenshot shows the MATLAB Coder interface for a project named 'detectCellsCodeGen.pj'. The main window title is 'Review Code Generation Readiness'. The left sidebar shows 'Source Code' and the file 'detectCellsCodeGen.m' is selected. The code editor displays the following MATLAB script:

```

1 function BWfinal = detectCellsCodeGen(I) %#codegen
2 % DETECTCELLSCODEGEN detects cells using image segmentation techniques and
3 % supports code generation.
4 %
5 % Note the MATLAB Coder compilation directive %#codegen at the end of the
6 % function signature
7
8 [~,threshold] = edge(I,'sobel');
9 fudgeFactor = .5;
10 BWs = edge(I,'sobel',threshold*fudgeFactor);
11 figure
12 imshow(BWs)
13 title('Binary Gradient Mask')
14
15 se90 = strel('line',3,90);
16 se0 = strel('line',3,0);
17
18 BWsdil = imdilate(BWs,[se90 se0]);
19 figure
20 imshow(BWsdil)
21 title('Dilated Gradient Mask')
22
23 BWdfill = imfill(BWsdil,'holes');
24 figure

```

The 'REVIEW ISSUES' tab is active. Below it, the 'Errors' tab is selected, showing a table with five entries:

	Function	Line	Description
⚠	detectCellsCodeGen.m	12	imshow is not supported for code generation
⚠	detectCellsCodeGen.m	20	imshow is not supported for code generation
⚠	detectCellsCodeGen.m	25	imshow is not supported for code generation
⚠	detectCellsCodeGen.m	30	imshow is not supported for code generation
⚠	detectCellsCodeGen.m	37	imshow is not supported for code generation

At the bottom, there are navigation buttons: 'Back' and 'Next'.

更正就绪问题

解决就绪问题。从您的示例中删除对 `imshow` 和相关显示代码的调用。分段操作不需要显示语句。您可以直接在 MATLAB Coder 中编辑示例代码。删除相关代码后，点击 **Save** 保存您的编辑并重新运行就绪检查。重新运行就绪检查后，MATLAB Coder 显示 **No issues found** 消息。

```

1 function BWfinal = detectCellsCodeGen(I) %#codegen
2 % DETECTCELLSCODEGEN detects cells using image segmentation techniques and
3 % supports code generation.
4 %
5 % Note the MATLAB Coder compilation directive %#codegen at the end of the
6 % function signature
7
8 [~,threshold] = edge(I,'sobel');
9 fudgeFactor = .5;
10 BWs = edge(I,'sobel',threshold*fudgeFactor);
11
12 se90 = strel('line',3,90);
13 se0 = strel('line',3,0);
14
15 BWsdil = imdilate(BWs,[se90 se0]);
16 BWdfill = imfill(BWsdil,'holes');
17 BWnobord = imclearborder(BWdfill,4);
18
19 seD = strel('diamond',1);
20 BWfinal = imerode(BWnobord,seD);
21 BWfinal = imerode(BWfinal,seD);
22
23 end

```

Errors			
	Function	Line	Description
⚠	detectCellsCodeGen...	12	imshow is not supported for code generation
⚠	detectCellsCodeGen...	20	imshow is not supported for code generation
⚠	detectCellsCodeGen...	25	imshow is not supported for code generation
⚠	detectCellsCodeGen...	30	imshow is not supported for code generation
⚠	detectCellsCodeGen...	37	imshow is not supported for code generation

✔ Save your changes to recheck the compatibility of your code.

定义函数输入的大小和数据类型

代码的每项输入都必须指定为具有固定大小、可变大小或常量。有几种方法可以指定输入参数的大小，但最简单的方法是为 MATLAB Coder 提供调用函数的示例。在文本输入字段中输入调用您的函数的脚本。对于此示例，请在 MATLAB 提示符下输入以下代码，然后按 **Autodetect Input Types**。

```
I = imread('cell.tif');
Iseg = detectCellsCodeGen(I);
```

Binary Gradient Mask



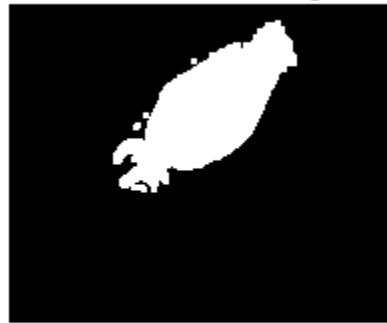
Dilated Gradient Mask



Binary Image with Filled Holes



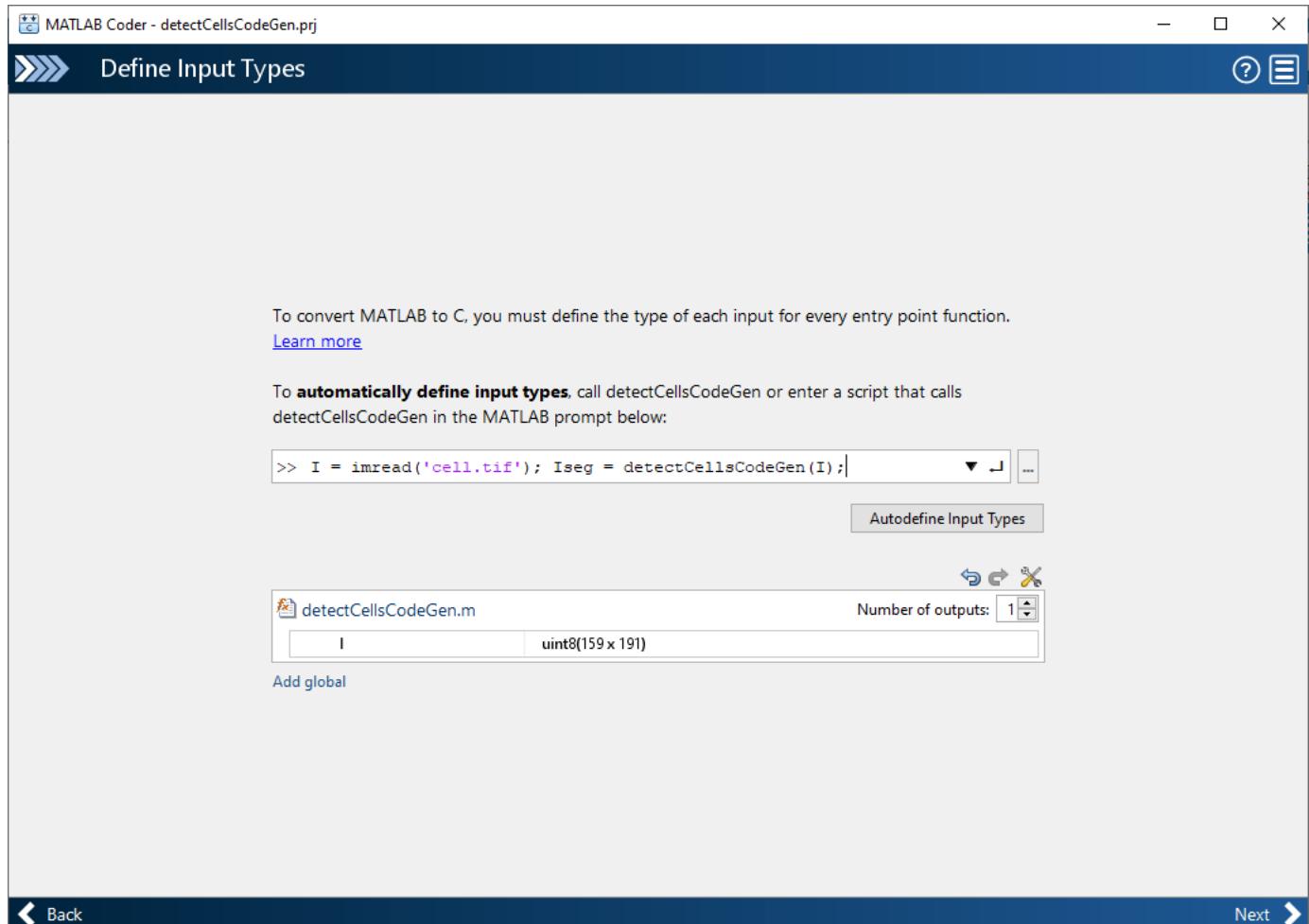
Cleared Border Image



Segmented Image

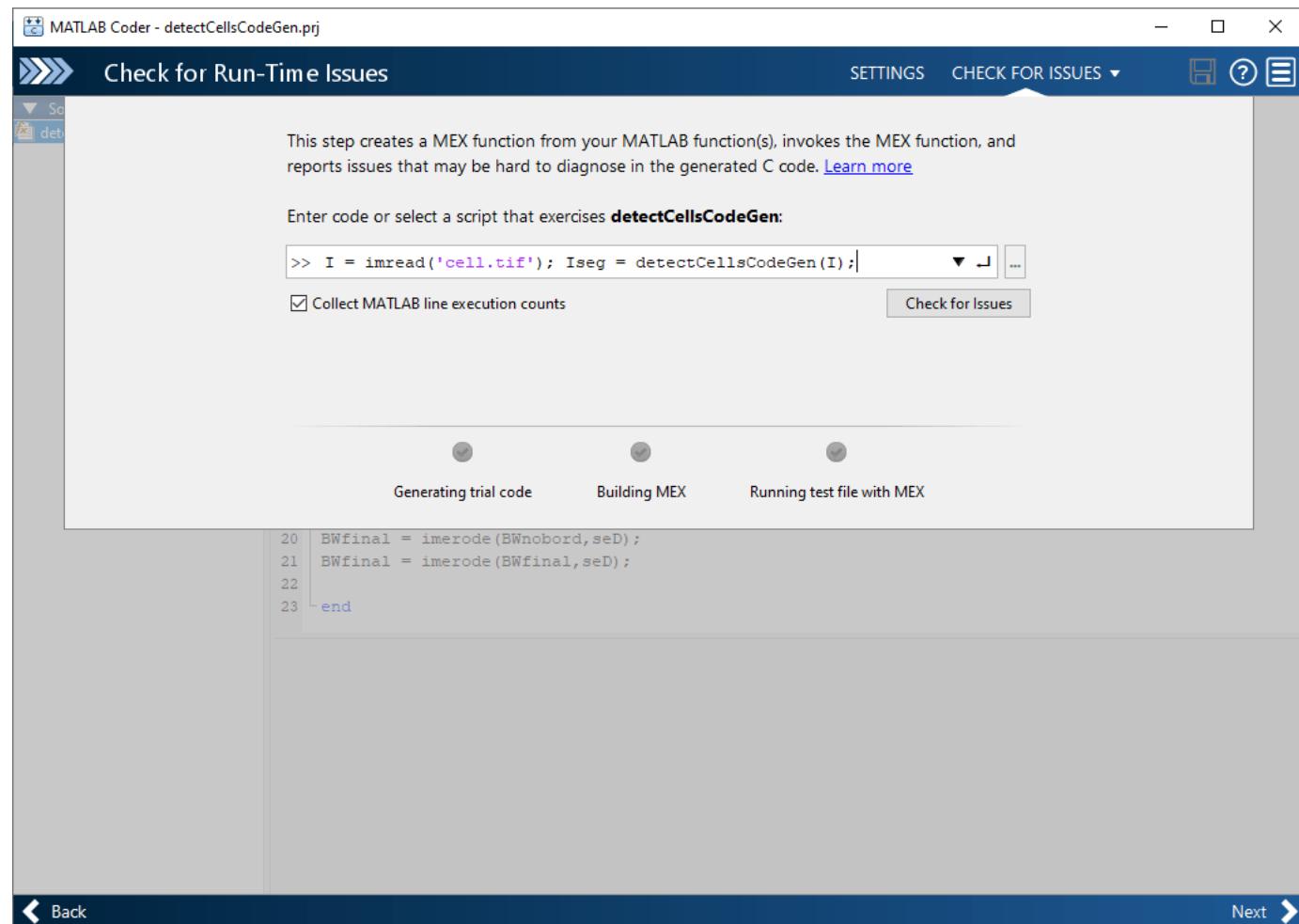


有关定义输入的详细信息，请参阅 MATLAB Coder 文档。在 MATLAB Coder 返回输入类型定义后，点击 **Next**。



检查并解决运行时问题

即使您执行了 MATLAB Coder 就绪检查，在编译过程中仍可能出现阻止代码生成的其他问题。当就绪检查查看函数依存关系以确定就绪情况时，编译过程会检查编码模式。您可以使用您输入的相同代码来定义输入类型（已预加载到对话框中）。点击 **Check for Issues**。



此示例包含一个编译问题：它将 `strel` 对象数组传递给 `imdilate`，而代码生成不支持对象数组。

The screenshot shows the MATLAB Coder interface for a project named 'detectCellsCodeGen.pj'. The main window title is 'Check for Run-Time Issues'. The code editor displays the following MATLAB script:

```

1 function BWfinal = detectCellsCodeGen(I) %#codegen
2 % DETECTCELLSCODEGEN detects cells using image segmentation techniques and
3 % supports code generation.
4 %
5 % Note the MATLAB Coder compilation directive %#codegen at the end of the
6 % function signature
7
8 [~,threshold] = edge(I,'sobel');
9 fudgeFactor = .5;
10 BWs = edge(I,'sobel',threshold*fudgeFactor);
11
12 se90 = strel('line',3,90);
13 se0 = strel('line',3,0);
14
15 BWsdil = imdilate(BWs,[se90 se0]);
16 BWdfill = imfill(BWsdil,'holes');
17 BWnobord = imclearborder(BWdfill,4);
18
19 seD = strel('diamond',1);
20 BWfinal = imerode(BWnobord,seD);
21 BWfinal = imerode(BWfinal,seD);
22
23 end

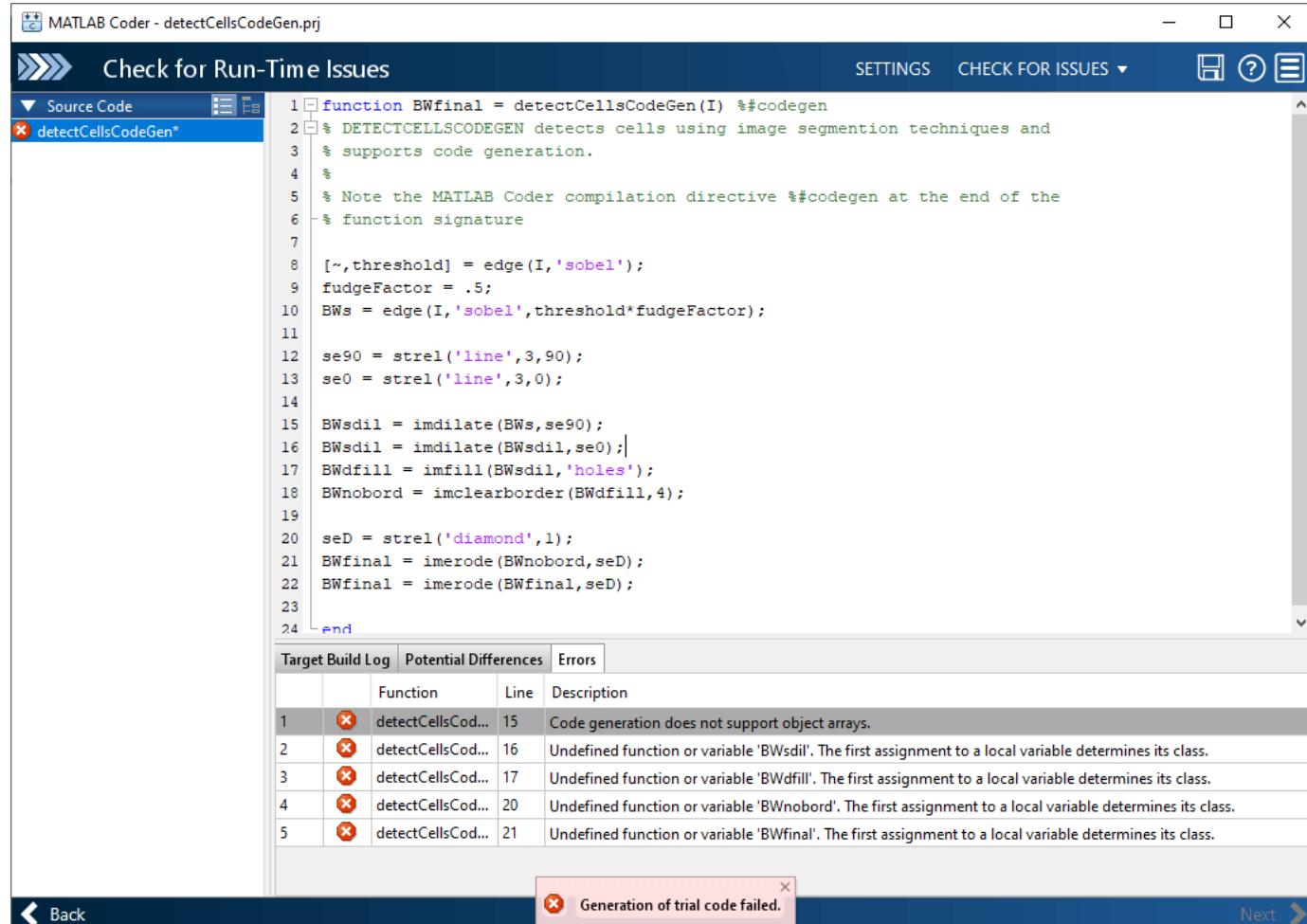
```

Below the code editor is a table titled 'Target Build Log | Potential Differences | Errors'. The 'Errors' tab is selected, showing the following entries:

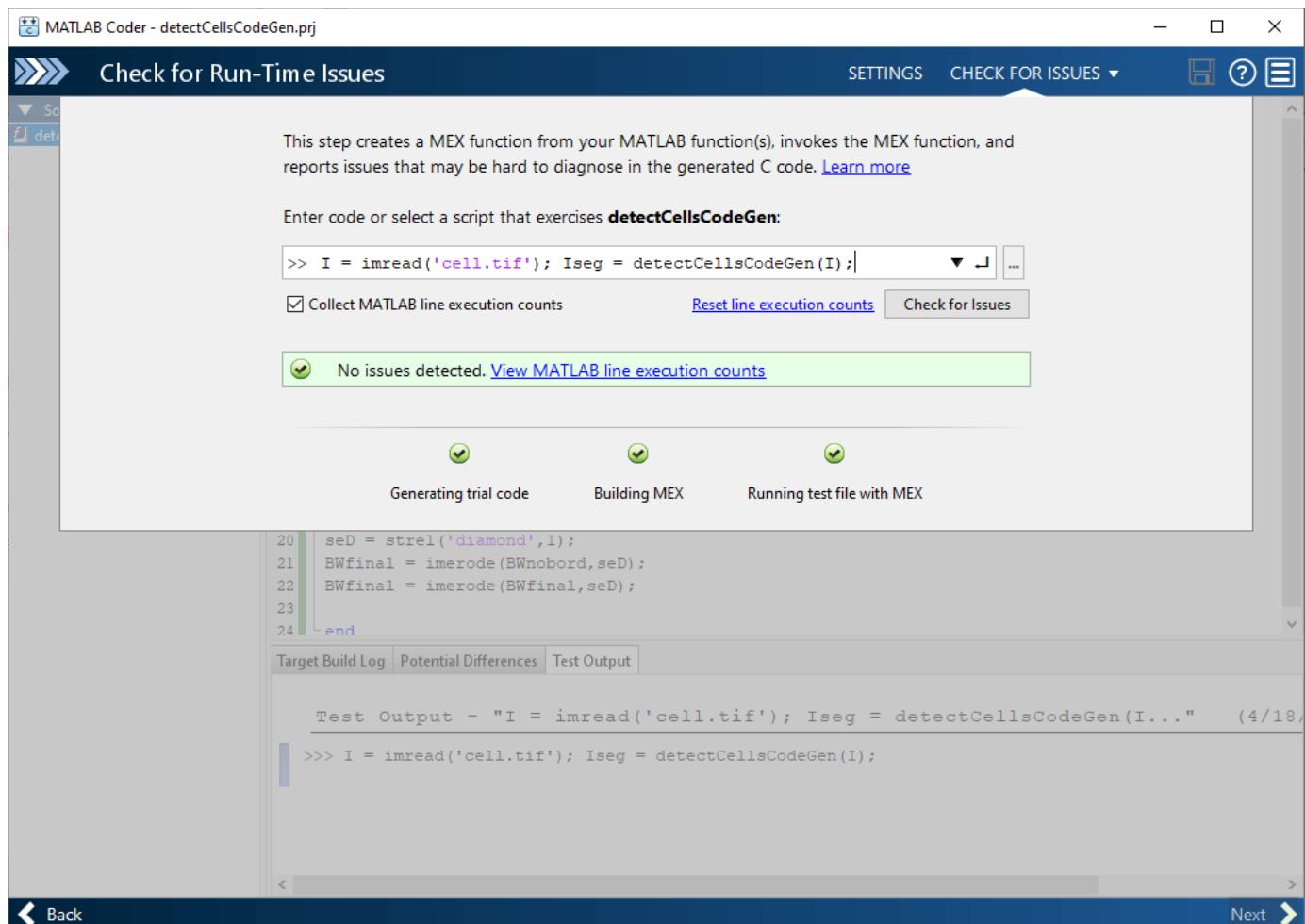
	Function	Line	Description
1	detectCellsCod...	15	Code generation does not support object arrays.
2	detectCellsCod...	16	Undefined function or variable 'BWsdil'. The first assignment to a local variable determines its class.
3	detectCellsCod...	17	Undefined function or variable 'BWdfill'. The first assignment to a local variable determines its class.
4	detectCellsCod...	20	Undefined function or variable 'BWnobord'. The first assignment to a local variable determines its class.
5	detectCellsCod...	21	Undefined function or variable 'BWfinal'. The first assignment to a local variable determines its class.

A message box at the bottom center says 'Generation of trial code failed.' with an 'X' button.

解决标识的编译问题。对于此示例，请修改对 `imdilate` 的调用，以避免传递 `strel` 对象数组。将对 `imdilate` 的单次调用替换为对 `imdilate` 的两次单独调用，每次调用传递一个 `strel` 对象。



重新运行测试编译，以确保您的更改解决了该问题。点击 **Check for Issues**。MATLAB Coder 显示一条消息，声明没有检测到问题。



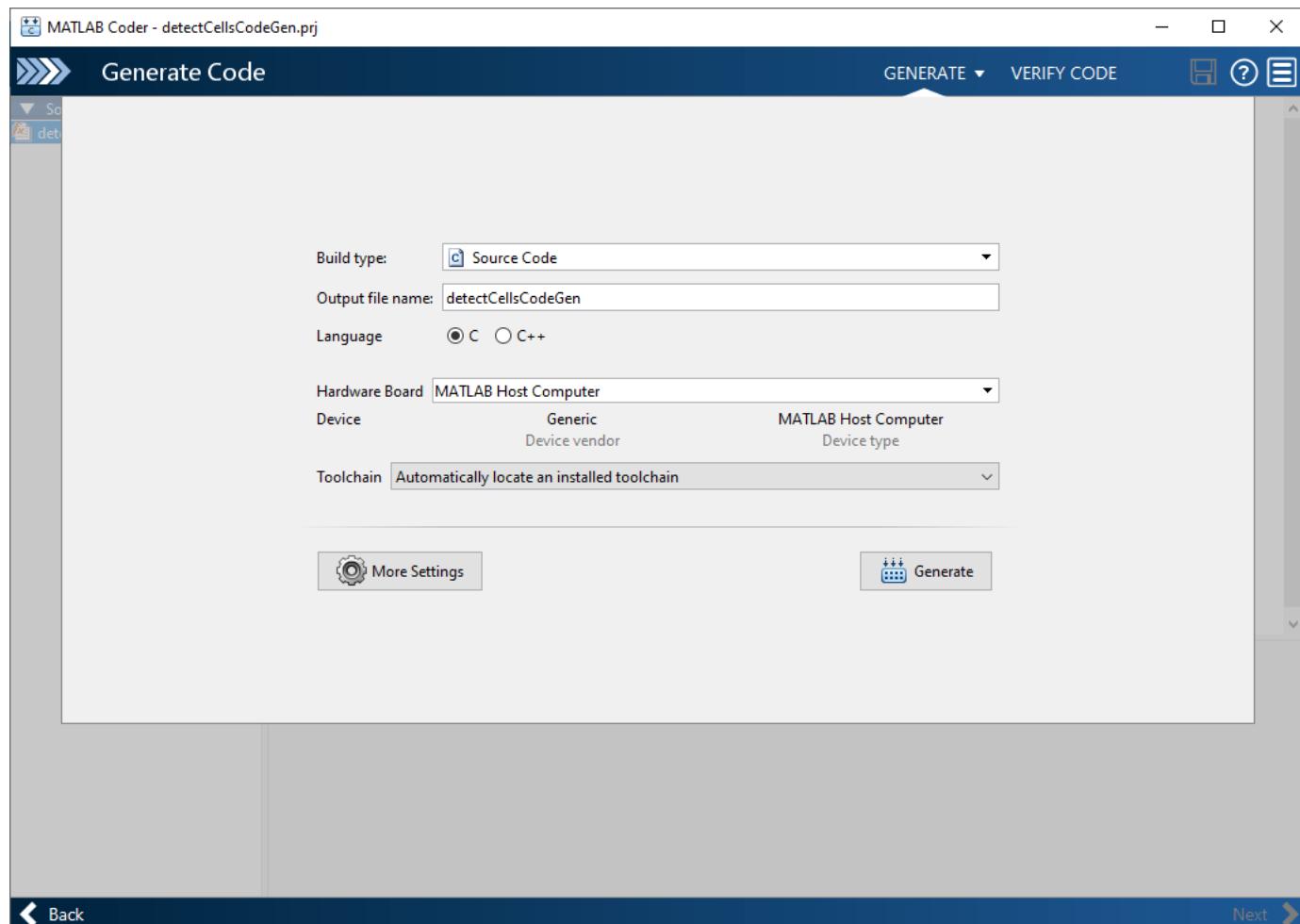
生成代码

您现在已准备好生成代码。点击 **Next**。

选择要生成的代码的类型，并选择目标平台。MATLAB Coder 可以生成 C 或 C++ 源代码、MEX 文件、静态库、共享库或独立的可执行文件。对于生产硬件，您可以从许多选项中进行选择，包括 ARM 和 Intel 处理器。

此示例使用默认选项。编译类型为 Source Code，语言为 C。对于设备选项，请指定某设备供应商的一个通用设备，并对设备类型指定一台 MATLAB 主机。当您选择 MATLAB Host Computer 时，MATLAB Coder 会生成依赖于预编译的共享库的代码。Image Processing Toolbox 函数使用共享库来保持性能优化。

点击 **Generate**。



MATLAB Coder 会显示生成的代码。

点击 **Next** 完成该过程。MATLAB Coder 显示关于它所生成代码的信息。默认情况下，MATLAB Coder 在您的工作文件夹中创建一个 **codegen** 子文件夹，其中包含生成的输出。

另请参阅

[codegen](#) | [MATLAB Coder](#)

详细信息

- “图像处理的代码生成”（第 20-2 页）
- “代码生成工作流”（MATLAB Coder）
- “使用 MATLAB Coder App 生成 C 代码”（MATLAB Coder）
- “代码生成的输入类型设定”（MATLAB Coder）
- 支持代码生成的函数

使用 Image Processing Toolbox 函数的 GPU 计算

GPU 上的图像处理

为了利用现代图形处理单元 (GPU) 提供的性能优势，某些 Image Processing Toolbox 函数支持在 GPU 上执行图像处理运算。这可以为复杂的图像处理工作流提供 GPU 加速。这些方法可以单独实现，也可以组合实现，以满足具体设计要求和性能目标。

要在图形处理单元 (GPU) 上运行图像处理代码，您必须拥有 Parallel Computing Toolbox™ 软件。要在 GPU 上执行支持的图像处理运算，请执行以下步骤：

- 将数据从 CPU 移到 GPU。使用 `gpuArray` 函数将数组从 MATLAB 传输到 GPU。有关详细信息，请参阅“Create GPU Arrays from Existing Data”(Parallel Computing Toolbox)。
- 在 GPU 上执行图像处理运算。有关支持 GPU 的所有工具箱函数的列表，请参阅支持 GPU 计算的函数。
- 将数据从 GPU 移回 CPU。使用 `gather` 函数从 GPU 中检索数组，并将该数组作为常规的 MATLAB 数组传输到 MATLAB 工作区。

如果您使用至少一个 `gpuArray` 输入参数调用支持 GPU 的函数，则该函数会在 GPU 上自动运行，并生成结果 `gpuArray`。您可以在同一个函数调用中同时使用 `gpuArray` 和 MATLAB 数组来混合输入。在这种情况下，该函数自动将 MATLAB 数组传输到 GPU 来执行。

使用 GPU 时，请注意以下几点：

- 性能的提高可能取决于 GPU 设备。
- GPU 上返回的结果与 CPU 上返回的结果可能稍有差异。

要了解如何将自定义 CUDA 内核直接集成到 MATLAB 中以加速复杂算法，请参阅“Run CUDA or PTX Code on GPU”(Parallel Computing Toolbox)。

另请参阅

`gpuArray` | `gather`

相关示例

- “Perform Thresholding and Morphological Operations on GPU”
- “Perform Pixel-Based Operations on GPU”

详细信息

- “Run MATLAB Functions on a GPU”(Parallel Computing Toolbox)
- 支持 GPU 计算的函数