MATLAB and Engineering Application

Image Processing Toolbox

Image Processing in Matlab

- Images can be conveniently represented as matrices in Matlab.
- One can open an image as a matrix using imread command.
- The matrix may simply be m x n form or it may be 3 dimensional array or it may be an indexed matrix, depending upon image type.
- The image processing may be done simply by matrix calculation or matrix manipulation.
- Image may be displayed with imshow command.
- Changes image may then be saved with imwrite command.

Image Types in Matlab

- Outside Matlab images may be of three types i.e. black & white, grey scale and colored.
- In Matlab, however, there are four types of images.
- Black & White images are called binary images, containing 1 for white and 0 for black.
- Grey scale images are called intensity images, containing numbers in the range of 0 to 255 or 0 to 1.
- Colored images may be represented as RGB Image or Indexed Image.

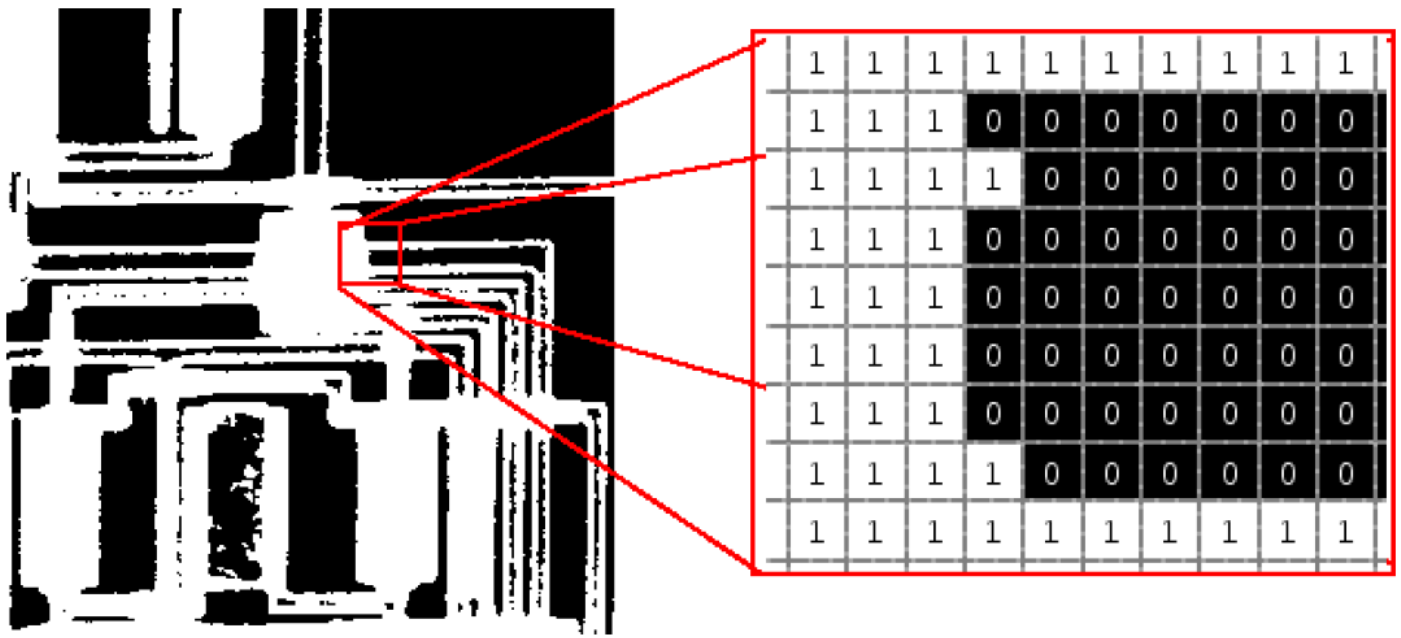
Image Types in Matlab(Continued)

- In RGB Images there exist three indexed images.
- First image contains all the red portion of the image, second green and third contains the blue portion.
- So for a 640 x 480 sized image the matrix will be 640 x 480 x 3.
- An alternate method of colored image representation is Indexed Image.
- It actually exist of two matrices namely image matrix and map matrix.
- Each color in the image is given an index number and in image matrix each color is represented as an index number.
- Map matrix contains the database of which index number belongs to which color.

Binary Images

In a binary image, each pixel assumes one of only two discrete values: 1 or 0. A binary image is stored as a `logical` array. By convention, this documentation uses the variable name `BW` to refer to binary images.

The following figure shows a binary image with a close-up view of some of the pixel values.



Pixel Values in a Binary Image

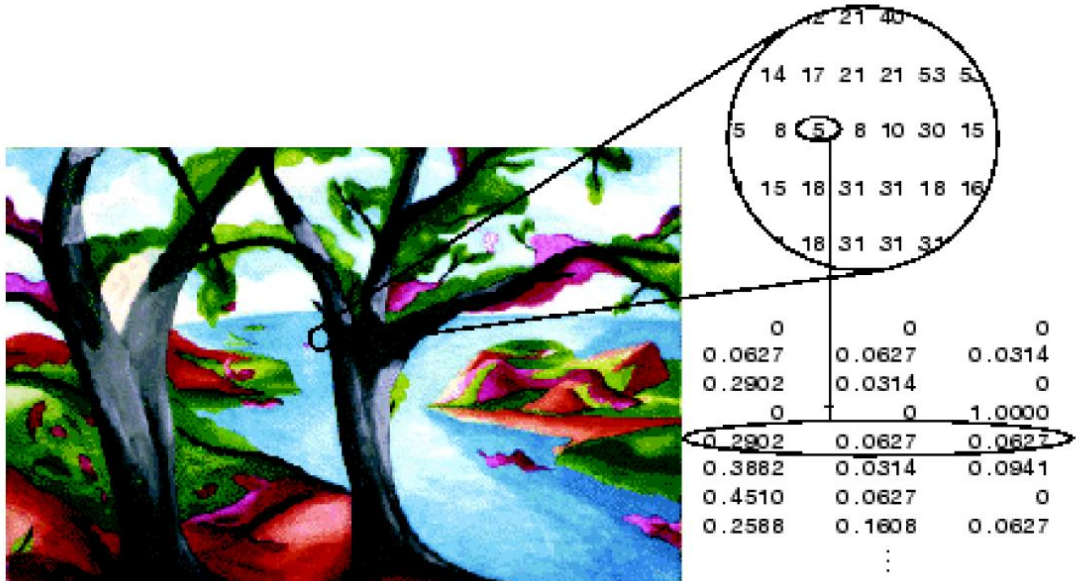
Indexed Images

An indexed image consists of an array and a colormap matrix. An indexed image uses direct mapping of pixel values in the array to colormap values. By convention, this documentation uses the variable name `X` to refer to the array and `map` to refer to the colormap.

The colormap matrix is an *m*-by-3 array of class `double` containing floating-point values in the range [0,1]. Each row of `map` specifies the red, green, and blue components of a single color.

The pixel values in the array are direct indices into a colormap. The color of each image pixel is determined by using the corresponding value of `X` as an index into `map`. The relationship between the values in the image matrix and the colormap depends on the class of the image matrix:

- If the image matrix is of class `single` or `d`
  integer values in the range [1, *p*], where *p*
  points to the first row in the colormap, the
- If the image matrix is of class `logical`, `ui`
  contains integer values in the range [0, *p–1*
  colormap, the value 1 points to the second



A colormap is often stored with an indexed im
image when you use the `imread` function. Afte
the workspace as separate variables, you must
image and colormap. However, you are not lim

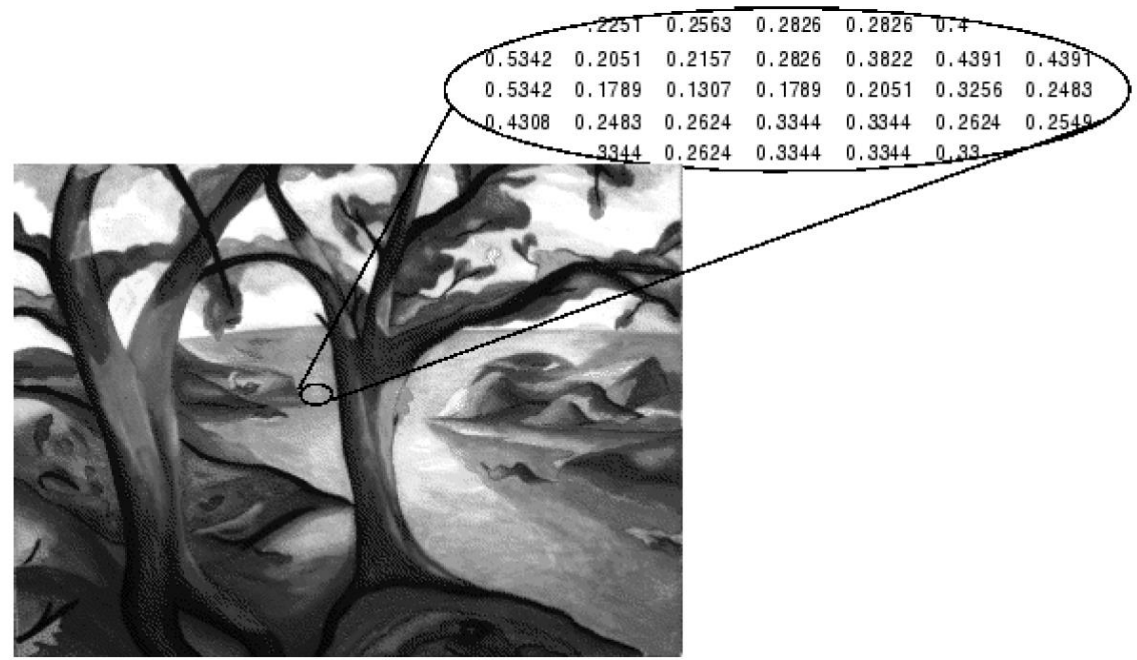
Grayscale Images

A grayscale image (also called gray-scale, gray scale, or gray-level) is a data matrix whose values represent intensities within some range. MATLAB stores a grayscale image as an individual matrix, with each element of the matrix corresponding to one image pixel. By convention, this documentation uses the variable name `I` to refer to grayscale images.

The matrix can be of class `uint8`, `uint` images are rarely saved with a colormap

For a matrix of class `single` or `double` intensity 0 represents black and the inte `uint8`, `uint16`, or `int16`, the intensity intensity `intmax(class(I))` represent

The figure below depicts a grayscale ima



Pixel Values in a Grayscale Image Define Gray Levels

Truecolor Images

A truecolor image is an image in which each pixel is specified by three values — one each for the red, blue, and green components of the pixel's color. MATLAB store truecolor images as an *m*-by-*n*-by-3 data array that defines red, green, and blue color components for each individual pixel. Truecolor images do not use a colormap. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location.

Graphics file formats store truecolor images as 24-bit ima
blue components are 8 bits each. This yields a potential of
with which a real-life image can be replicated has led to t
truecolor image.

A truecolor array can be of class `uint8`, `uint16`, `single`
array of class `single` or `double`, each color component i
pixel whose color components are (0,0,0) is displayed as b
components are (1,1,1) is displayed as white. The three co
are stored along the third dimension of the data array. For
blue color components of the pixel (10,5) are stored in RG
`RGB(10,5,3)`, respectively.

Image Type Conversion

- RGB Image to Intensity Image (rgb2gray)
- RGB Image to Indexed Image (rgb2ind)
- RGB Image to Binary Image (im2bw)
- Indexed Image to RGB Image (ind2rgb)
- Indexed Image to Intensity Image (ind2gray)
- Indexed Image to Binary Image (im2bw)
- Intensity Image to Indexed Image (gray2ind)
- Intensity Image to Binary Image (im2bw)
- Intensity Image to RGB Image (gray2ind, ind2rgb)

Image Histogram

- There are a number of ways to get statistical information about data in the image.
- Image histogram is on such way.
- An image histogram is a chart that shows the distribution of intensities in an image.
- Each color level is represented as a point on x-axis and on y-axis is the number instances a color level repeats in the image.
- Histogram may be view with imhist command.
- Sometimes all the important information in an image lies only in a small region of colors, hence it usually is difficult to extract information out of that image.
- To balance the brightness level, we carryout an image processing operation termed histogram equalization.

Basic Image Import, Processing, and Export

read and display an imag

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

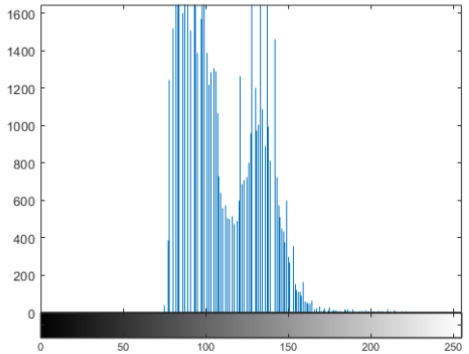
Check how the image appears in the workspace

```
whos I
```

```
  Name        Size              Bytes  Class    Attributes

  I         291x240             69840  uint8
```

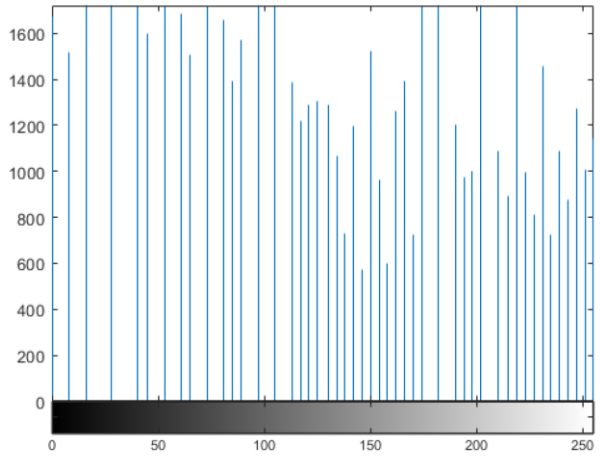
adjust Image Contrast

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



View the distribution of image pixel intensities

```
figure
imhist(I2)
```



Basic Image Import, Processing, and Export

- Read a true color image

```
RGB = imread('football.jpg');
```

- Read a grayscale image

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

```
  Name        Size                Bytes  Class    Attributes

  I         256x256               65536  uint8
  RGB       256x320x3            245760  uint8
```

- Read an indexed image

```
[X,map] = imread('trees.tif');
whos
```

Read an indexed image into the workspace. `imread` uses two variables to store an indexed image in the workspace: one for the image and another for its associated colormap. `imread` always reads the colormap into a matrix of class `double`, even though the image array itself may be of class `uint8` or `uint16`.

If the image file format uses 8-bit pixels, imread returns the image data as an m-by-n-by-3 array of uint8 values. For graphics file formats that support 16-bit data, such as PNG and TIFF, imread returns an array of uint16 values.

Read Multiple Images from a Single Graphics File

Preallocate a 4-D array to hold the images to be read from a file.

```
mri = zeros([128 128 1 27],'uint8');
```

Read the images from the file, using a loop to read each image sequentially.

```
for frame=1:27
    [mri(:,:,:,frame),map] = imread('mri.tif',frame);
end
whos
```

```
  Name          Size                     Bytes  Class     Attributes

  frame         1x1                          8  double
  map           256x3                     6144  double
  mri           128x128x1x27            442368  uint8
```

Write Image Data to File in Graphics Format

Load image data into the workspace. This example loads the indexed image `X` from a MAT-file, `trees.mat`, along with the associated colormap `map`.

```
load trees
whos
```

```
  Name           Size              Bytes  Class     Attributes

  X            258x350            722400  double
  caption        1x66                132  char
  map          128x3                3072  double
```

Export the image data as a bitmap file using `imwrite`, specifying the name of the variable and the name of the output file you want to create. If you include an extension in the filename, `imwrite` attempts to infer the desired file format from it. For example, the file extension `.bmp` specifies the Microsoft Windows Bitmap format. You can also specify the format explicitly as an argument to `imwrite`.

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

Use format-specific parameters with `imwrite` to control aspects of the export process. For example, with PNG files, you can specify the bit depth. To illustrate, read an image into the workspace in TIFF format and note its bit depth.

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

```
ans = 8
```

Write the image to a graphics file in PNG format, specifying a bit depth of 4.

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

Check the bit depth of the newly created file.

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

```
ans = 4
```

Display Multiple Images

Display Multiple Images

`imshow` always displays an image in the current figure. If you display two images in succession, the second image replaces the first image. To view multiple figures with `imshow`, use the `figure` command to explicitly create a new empty figure before calling `imshow` for the next image. The following example views the first three frames in an array of grayscale images `I`.

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

Display Multiple Images in a Montage

View Image Sequence as Montage

This example shows how to view multiple frames in a multiframe array at one time, using the `montage` function. `montage` displays all the image frames, arranging them into a rectangular grid. The montage of images is a single image object. The image frames can be grayscale, indexed, or truecolor images. If you specify indexed images, they all must use the same colormap.

Create an array of truecolor images.

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

Display all the images at once, in a montage. By default, the `montage` function displays the images in a grid. The first image frame is in the first position of the first row, the next frame is in the second position of the first row, and so on.

```
montage(onionArray);
```

Display Multiple Images

To specify a different number of rows and columns, use the `'size'` parameter. For example, to display the images in one horizontal row, specify the `'size'` parameter with the value `[1 NaN]`. Using other `montage` parameters you can specify which images you want to display and adjust the contrast of the images displayed.

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

Display Images Individually in the Same Figure

You can use the `imshow` function with the MATLAB `subplot` function to display multiple images in a single figure window. For additional options, see “Work with Image Sequences as Multidimensional Arrays” on page 2-71.

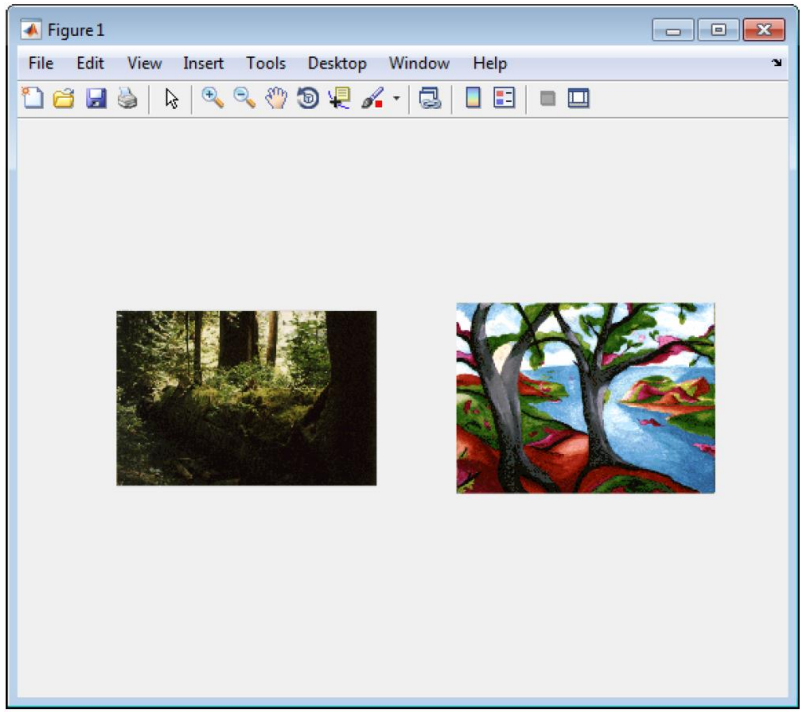
Note The Image Viewer app (`imtool`) does not support this capability.

Divide a Figure Window into Multiple Display Regions

`subplot` divides a figure into multiple display regions. Using the syntax `subplot(m,n,p)`, you define an `m`-by-`n` matrix of display regions and specify which region, `p`, is active.

For example, you can use this syntax to display two images side by side.

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



Get Image Information in Image Viewer App

To get information about the image displayed in the Image Viewer, use the Image Information tool. The Image Information tool can provide two types of information about an image:

- Basic information — Includes width, height, class, and image type. For grayscale and indexed images, this information also includes the minimum and maximum intensity values.
- Image metadata — Displays all the metadata from the graphics file that contains the image. This metadata is the same information returned by the `imfinfo` function or the `dicominfo` function.

Note The Image Information tool can display image metadata only when you specify the file name containing the image to Image Viewer, e.g., `imtool('moon.tif')`.

For example, view an image in the Image Viewer.

```
imtool('moon.tif')
```

Get Image Information

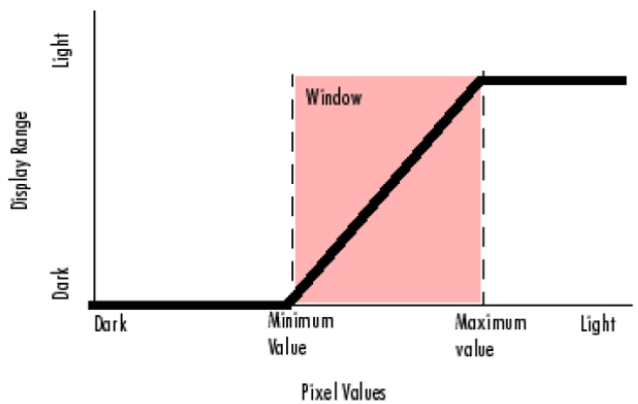
Adjust Image Contrast

Crop Image Using Image

An image lacks contrast when there are no sharp differences between black and white. Brightness refers to the overall lightness or darkness of an image.

To change the contrast or brightness of an image, the Adjust Contrast tool in the Image View app performs *contrast stretching*. In this process, pixel values below a specified value are displayed as black, pixel values above a specified value are displayed as white, and pixel values in between these two values are displayed as shades of gray. The result a linear mapping of a subset of pixel values to the entire range of grays, from black to white, producing an image of higher contrast.

The following figure shows this mapping. Note that the lower limit and upper limit mark the boundaries of the window, displayed graphically as the red-tinted window in the Adjust Contrast tool — see “Open the Adjust Contrast Tool” on page 4-63



Relationship of Pixel Values to Display Range

Explore 3-D Volumetric Data with Volume Viewer App

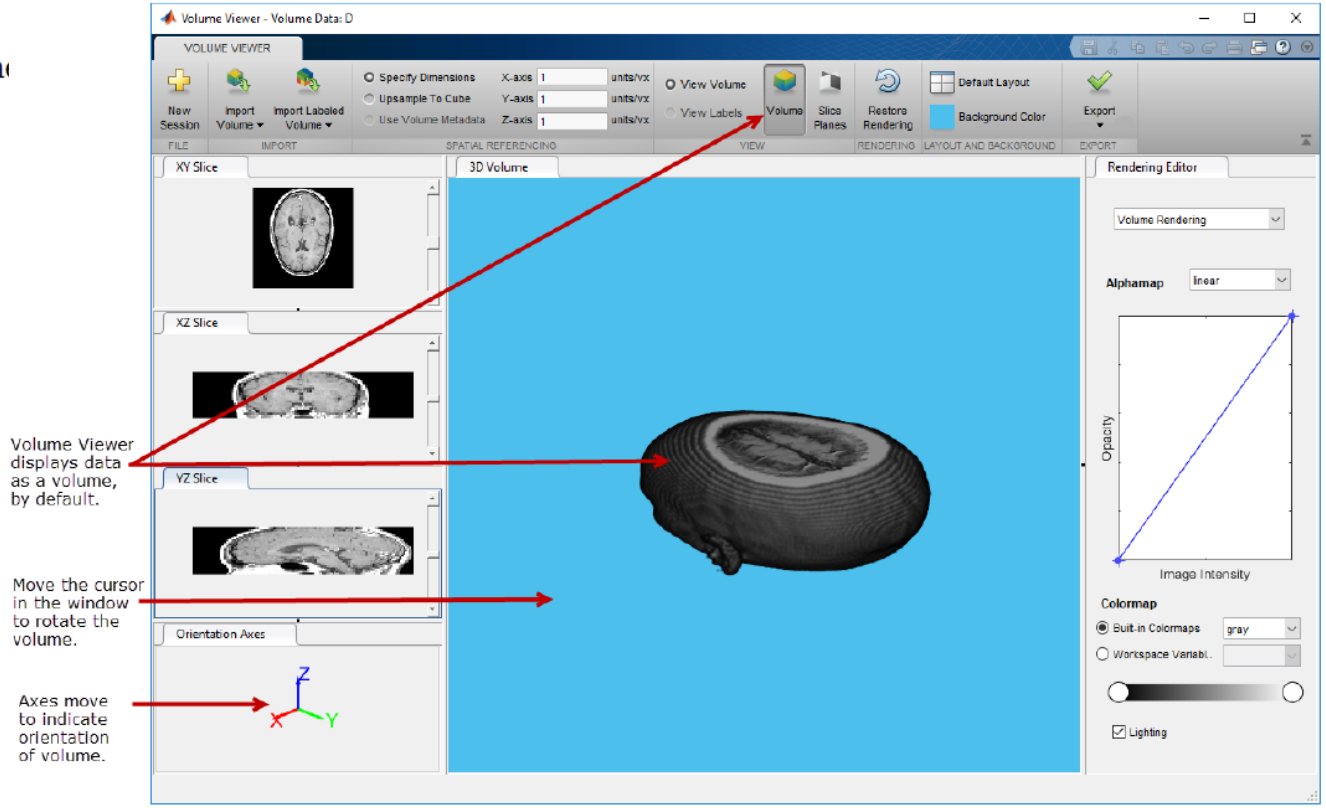
Load Volume Data into the Volume Viewer

This part of the example shows how to load volumetric data into the Volume Viewer.

Load the MRI data of a human head from a MAT-file into the workspace. This operation creates a variable named `D` in your workspace that contains the volumetric data. Use the `squeeze` command to remove the singleton dimension from the data.

```
load mri

D = squeeze(D);

whos
```

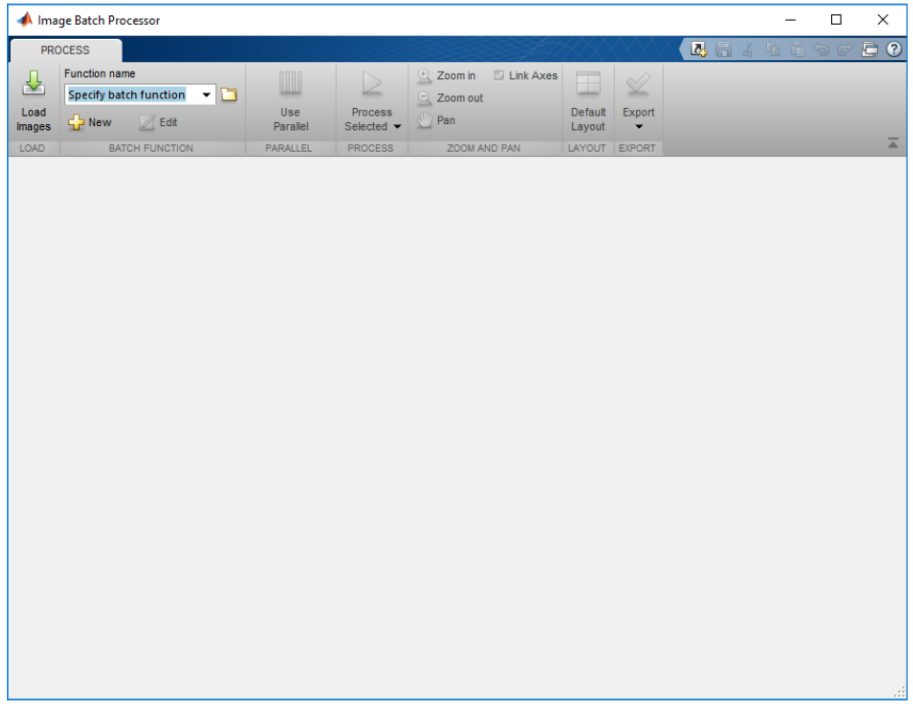
```
Name        Size                Bytes  Class     Attributes

  D         128x128x27         442368  uint8
  map        89x3                2136  double
  siz         1x3                  24  double
```

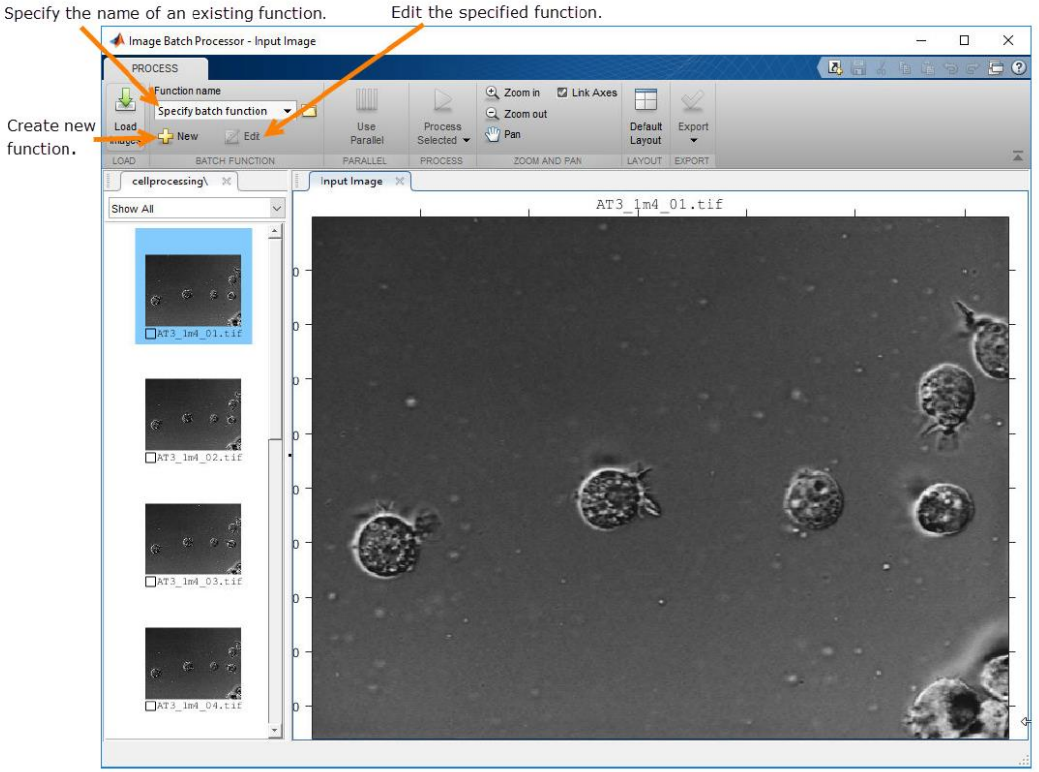


To change the background color used in the display window, click **Background Color** and select a color.

Open Image Batch Processor App

`imageBatchProcessor`





Load Images into the Image Batch Processor App

This part of the example shows how to load images into the Image Batch Processor app.

For this example, create a new folder in an area where you have write permission, and load a set of 10 images from the Image Processing Toolbox `imdata` folder.

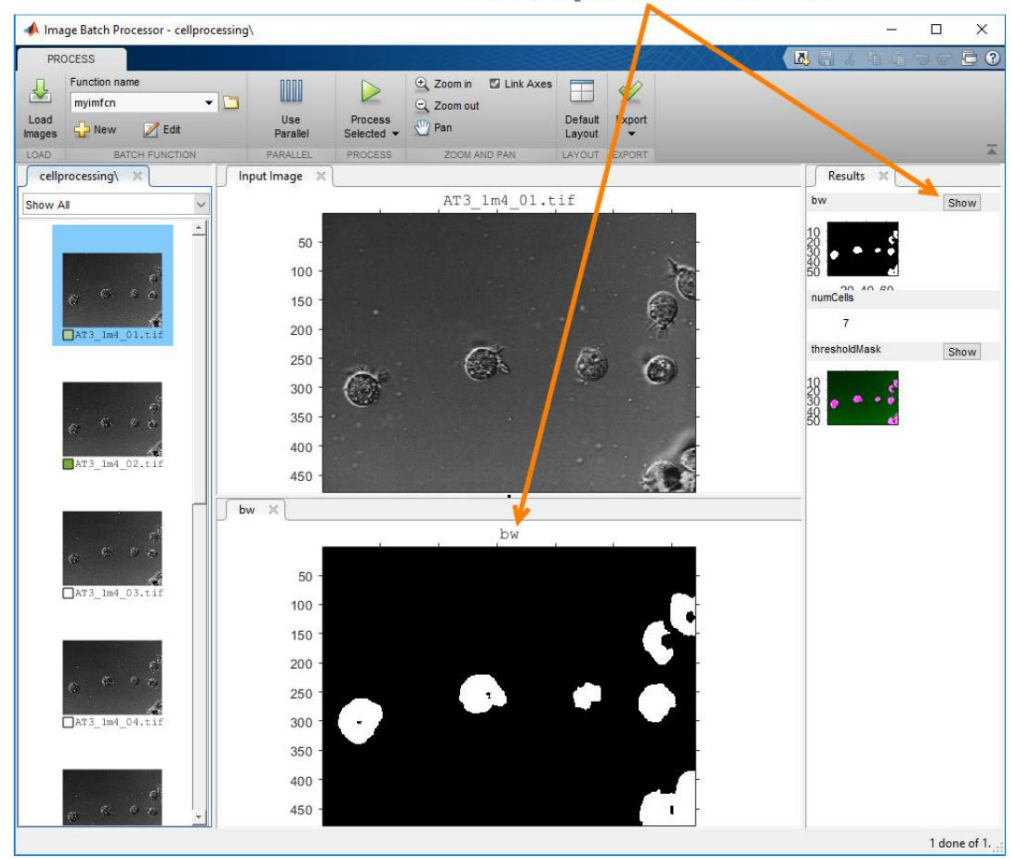
```
mkdir('cellprocessing');
copyfile(fullfile(matlabroot,'toolbox','images','imdata','AT3*.tif'),'cellprocessing')
```

Open Image Batch Processor App

```
function results = myimfcn(im)
%Image Processing Function
%
% IM      - Input image.
% RESULTS - A scalar structure with the processing results.
%

%-------------------------------------------------------------------------
% Auto-generated by imageBatchProcessor App.
%
% When used by the App, this function will be called for every input image
% file automatically. IM contains the input image as a matrix. RESULTS is
% scalar structure containing the results of this processing function.
%
%-------------------------------------------------------------------------

imstd = stdfilt(im,ones(27));
bw    = imstd>30;

thresholdMask = imfuse(im, bw);
[~, n] = bwlabel(bw);

results.bw = bw;
results.thresholdMask = thresholdMask;
results.numCells = n;
```

When you save the file, the app displays the name of your new function in the **Function name** field.

Click **Show** to get a closer look at the results.

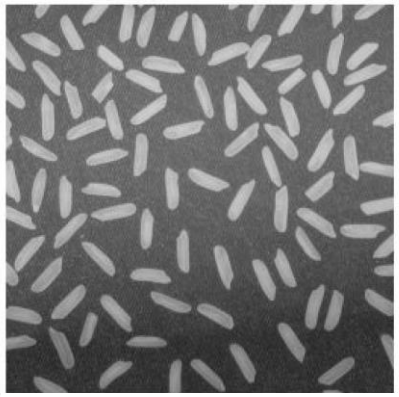


If the results of the test run are successful, execute the function on all the images in your input folder. Click the additional options menu on the **Process Selected** button, and select **Process All**. You can also select the multiple images to process using either **Ctrl-**

Morphological Operations

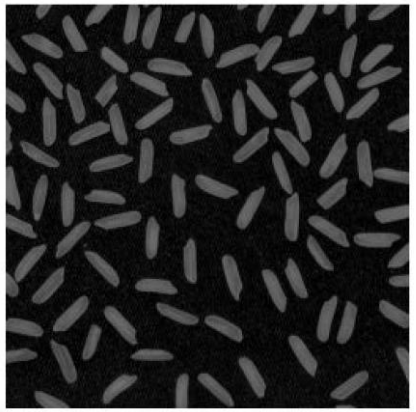
- These are image processing operations done on binary images based on certain morphologies or shapes.
- The value of each pixel in the output is based on the corresponding input pixel and its neighbors.
- By choosing appropriately shaped neighbors one can construct an operation that is sensitive to a certain shape in the input image.

Correct nonuniform illumination and analyze foreground objects



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

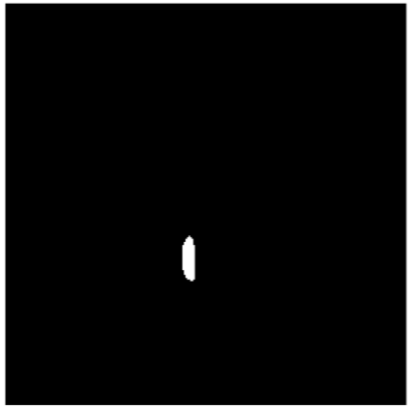
```
se = strel('disk',15)

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



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

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

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

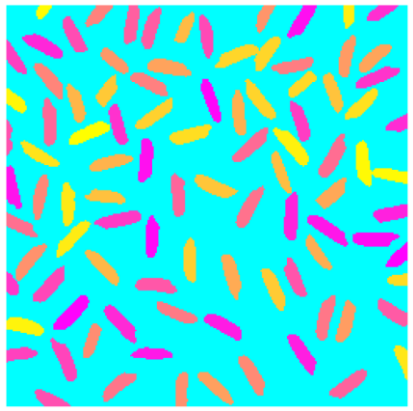


```
cc = bwconncomp(bw,4)
```

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

View the rice grain that is labeled 50 in the image.

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



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

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

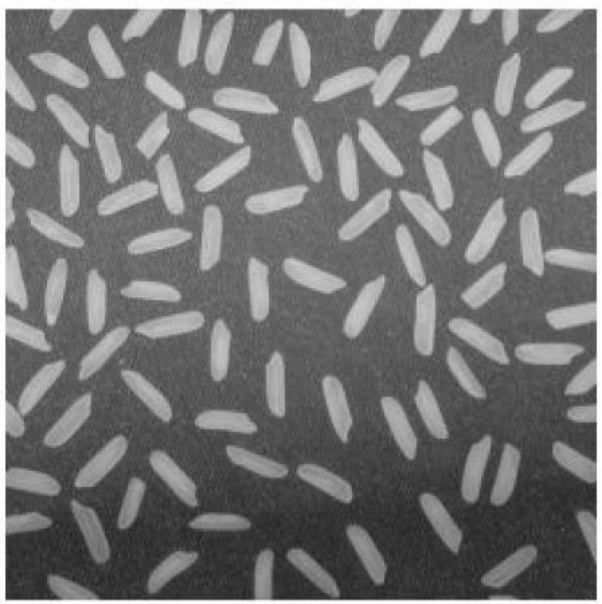
Contour Plot of Image Data

Create Contour Plot of Image Data

This example shows how to create a contour plot of an image.

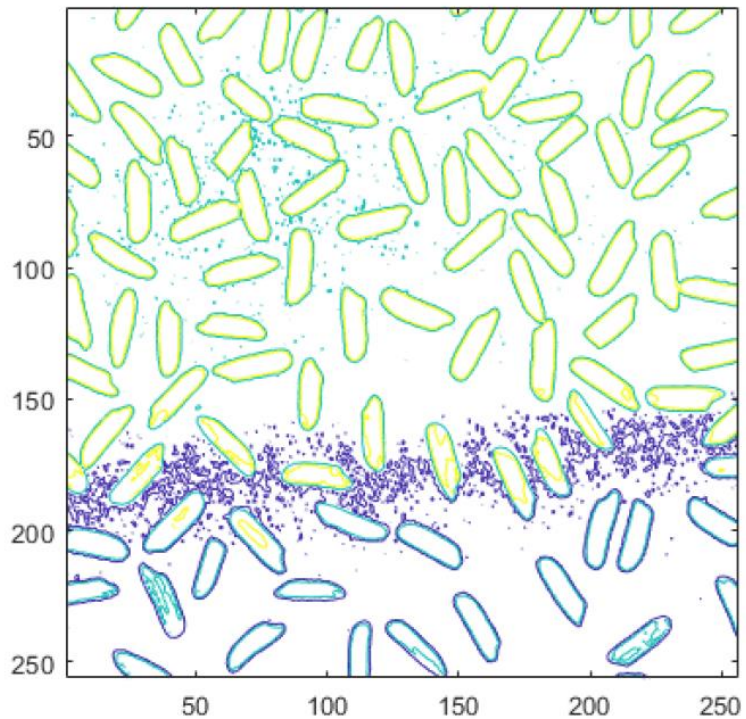
Read grayscale image and display it. The example uses an example image of grains of rice.

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



Create a contour plot of the image using `imcontour` .

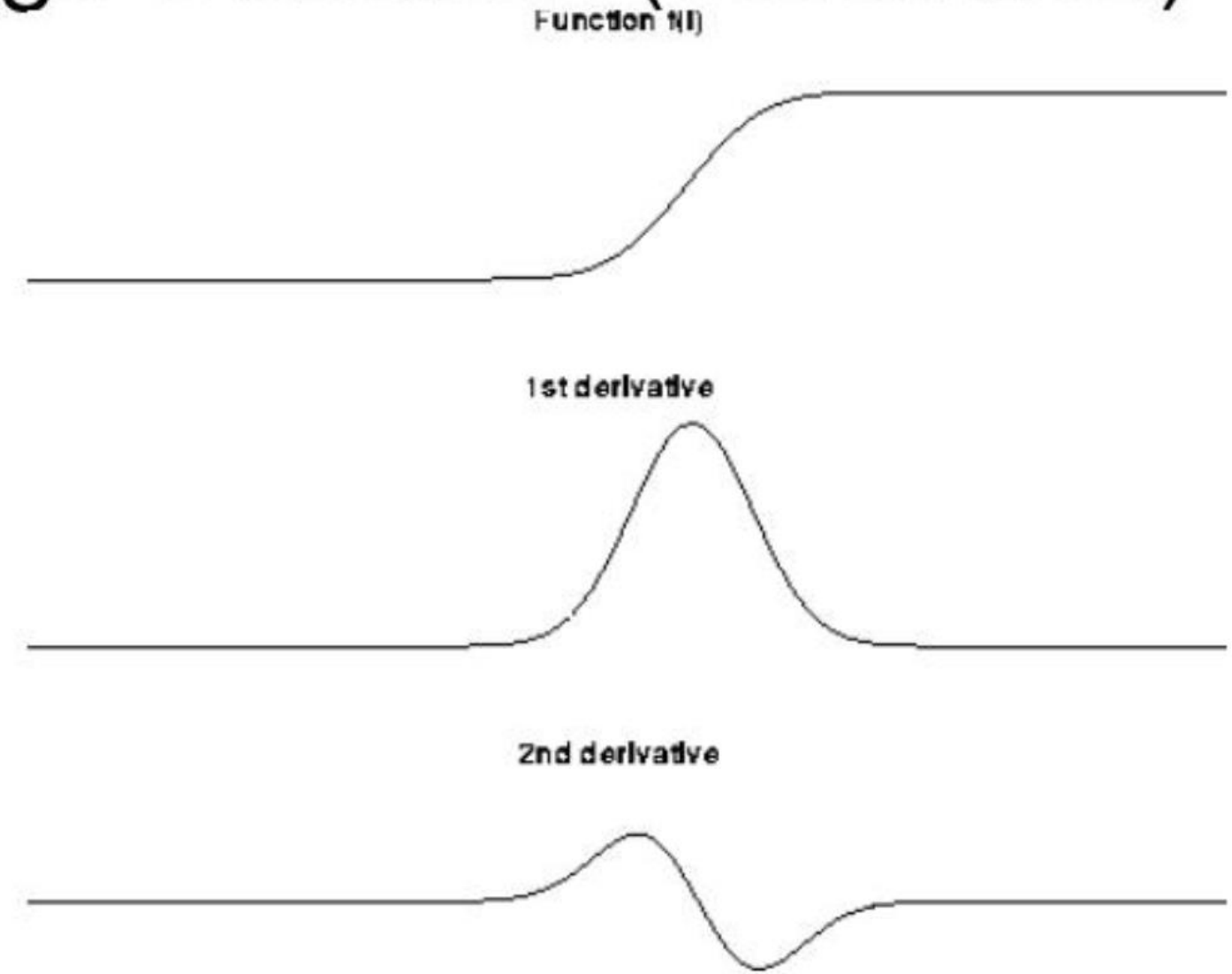
```
figure;
imcontour(I,3)
```



Edge Detection

- Edge detection extract edges of objects from an image.
- There are a number of algorithms for this, but these may be classified as derivative based or gradient based.
- In derivative based edge detection the algorithm takes first or second derivative on each pixel in the image.
- In case of first derivative at the edge of the image there is a rapid change of intensity.
- While in case of second derivative there is a zero pixel value, termed zero crossing.
- In gradient based edge detection a gradient of consecutive pixels is taken in x and y direction.

Edge Detection (Continued)



Edge Detection (Continued)

- Taking derivative on each and every pixel of the image consumes a lot of computer resources and hence is not practical.
- So usually an operation called kernel operation is carried out.
- A kernel is a small matrix sliding over the image matrix containing coefficients which are multiplied to corresponding image matrix elements and their sum is put at the target pixel.

Sobel Edge Detection

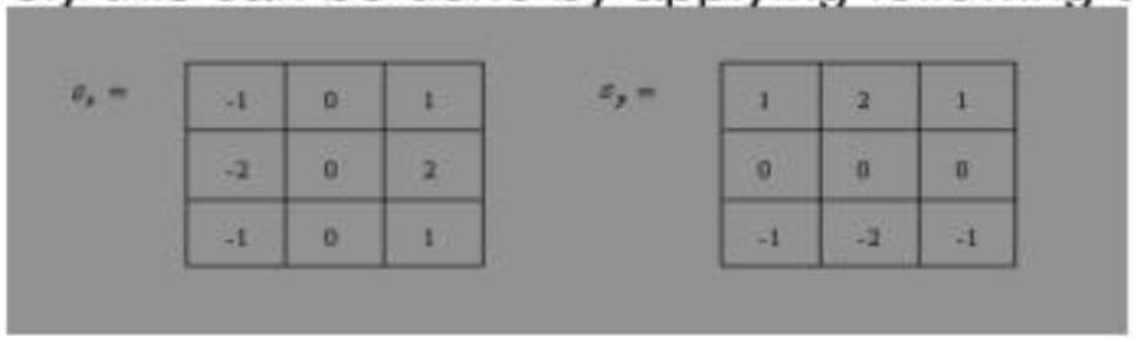
- In sobel following formulas are applied on each pixel(i,j) in the image and two matrices S_x and S_y are obtained:

$$S_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$

$$S_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4)$$

$$C = 2$$

- Alternatively this can be done by applying following two kernels:



-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

- The resultant matrix is then obtained by taking the square root of the sum of the squares of Sx and Sy, as follows:

$$M(i,j) = (S_x^2 + S_y^2)^{1/2}$$

Dilation and erosion

- Dilation adds pixels to the boundaries of objects in an image
- Erosion removes pixels on object boundaries

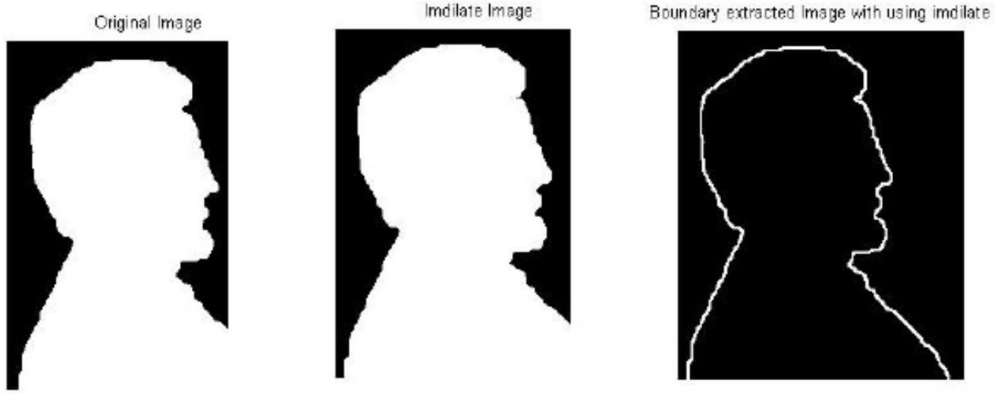


Fig 2: (a) Original Image (linkon.tif) (B) After Dilation Operation (C) Boundary Extraction with the help of Dilation.

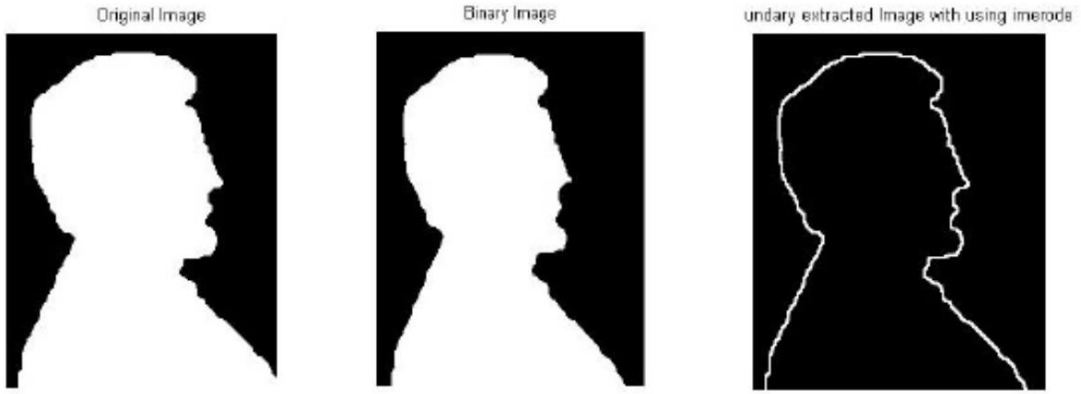
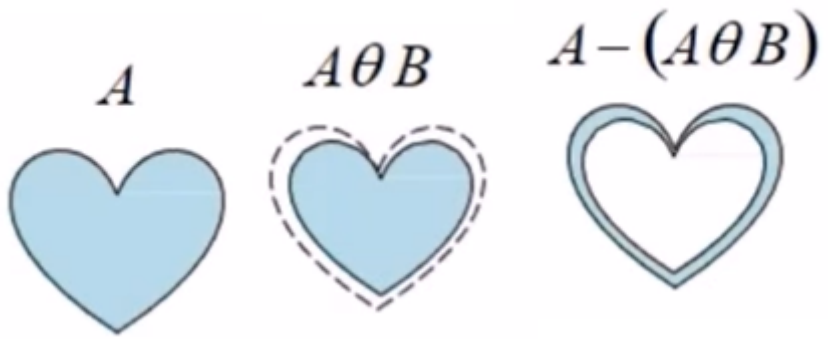


Fig 3: (a) Original Image (linkon.tif) (B) After erosion operation (C) Boundary Extraction with the help of Erosion.

Morphological algorithms - Boundary Extraction

To find the boundary of a set A, erode it by a small structuring element B
Then take the set difference between A and its erosion



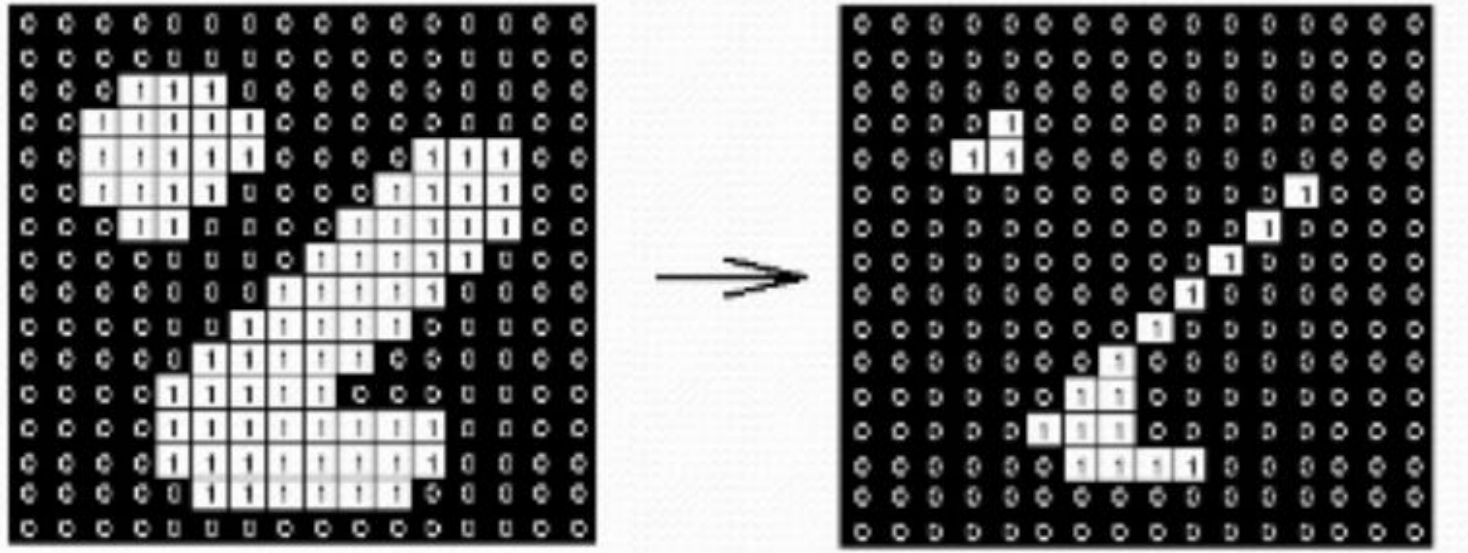
$$\beta(A) = A-(A\theta B)$$

Erosion

- Suppose that the structuring element is a 3x3 square
- Note that in subsequent diagrams, foreground pixels are represented by 1 and background pixels by 0
- The structuring element is now superimposed over each foreground pixel (input pi8xel) in the image. If all the pixels below the structuring elements are foreground pixels then the input pixel retains its value. But if any of the pixel is a background pixel then the input pixel gets the background pixel value.

1	1	1
1	1	1
1	1	1

Structuring element



Erosion

Edge Detection

Detect Edges in Images

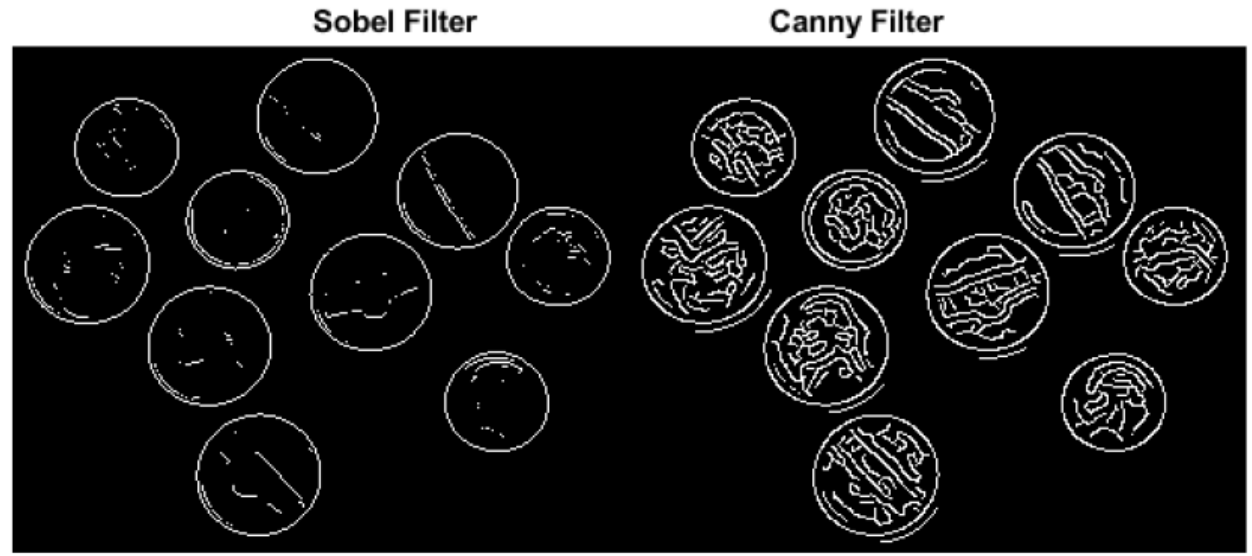
This example shows how to detect edges in an image using both the Canny edge detector and the Sobel edge detector.

Read image and display it.

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

Apply both the Sobel and Canny edge detectors to the image and display them for comparison.

```
BW1 = edge(I,'sobel');
BW2 = edge(I,'canny');
figure;
imshowpair(BW1,BW2,'montage')
title('Sobel Filter                                    Canny Filter');
```



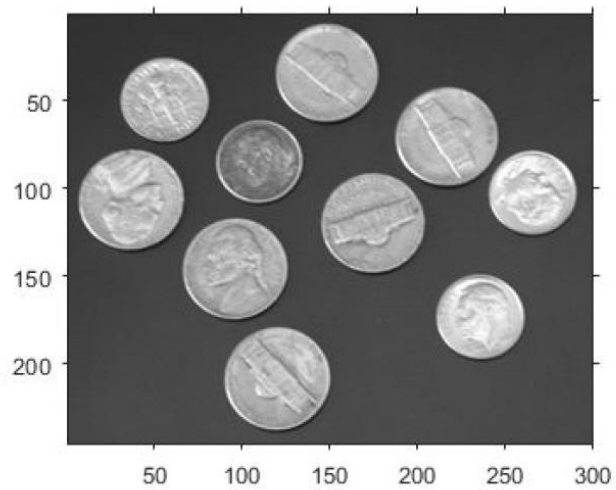
Boundary Tracing in Images

The toolbox includes two functions you can use to find the boundaries of objects in a binary image:

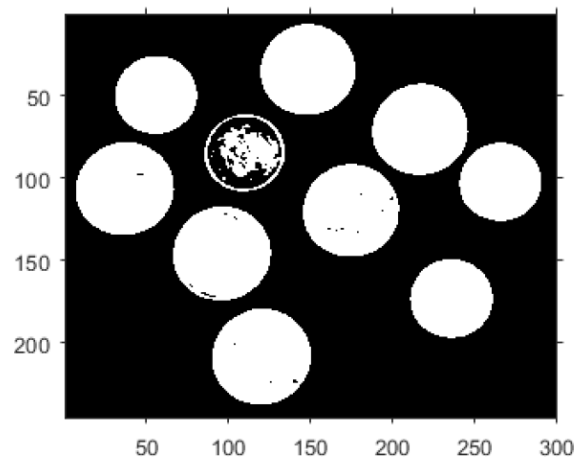
- `bwtraceboundary`
- `bwboundaries`

Trace Boundaries of Objects in Images

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




```
BW = im2bw(I);
imshow(BW)
```



Determine the row and column coordinates of a pixel on the border of the object you want to trace. `bwboundary` uses this point as the starting location for the boundary tracing.

```
dim = size(BW)
```

```
dim = 1×2

   246   300
```

```
col = round(dim(2)/2)-90;
row = min(find(BW(:,col)))
```

```
row = 27
```

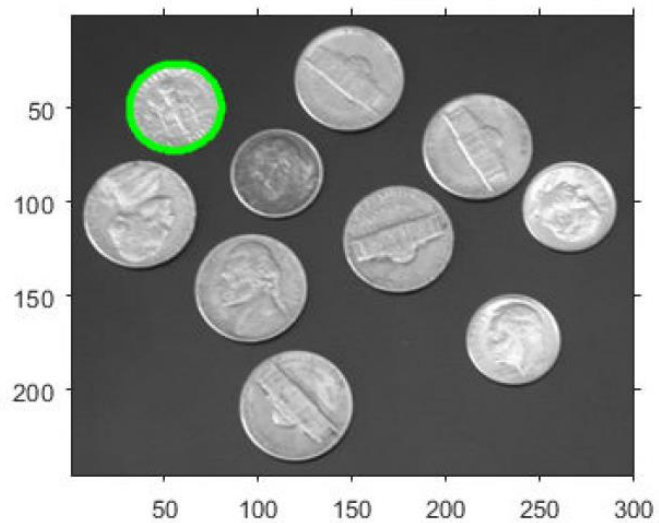
Call `bwtraceboundary` to trace the boundary from the specified point. As required arguments, you must specify a binary image, the row and column coordinates of the starting point, and the direction of the first step. The example specifies north (`'N'`).

```
boundary = bwtraceboundary(BW,[row, col],'N');
```

Boundary Tracing in Images

Display the original grayscale image and use the coordinates returned by `bwtraceboundary` to plot the border on the image.

```
imshow(I)
hold on;
plot(boundary(:,2),boundary(:,1),'g','LineWidth',3);
```

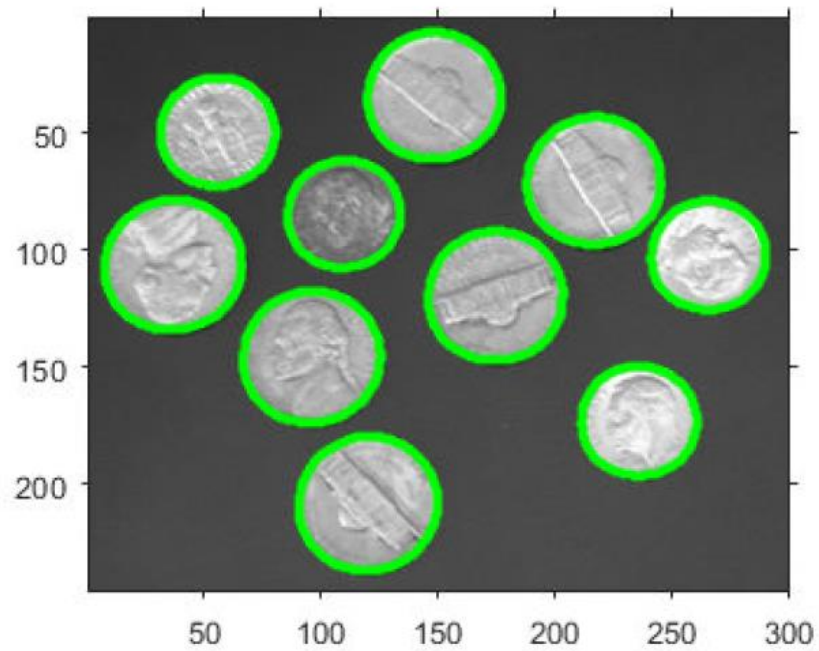


To trace the boundaries of all the coins in the image, use the `bwboundaries` default, `bwboundaries` finds the boundaries of all objects in an image, includ inside other objects. In the binary image used in this example, some of the co black areas that `bwboundaries` interprets as separate objects. To ensure tha `bwboundaries` only traces the coins, use `imfill` to fill the area inside each `bwboundaries` returns a cell array, where each cell contains the row/column for an object in the image.

```
BW_filled = imfill(BW,'holes');
boundaries = bwboundaries(BW_filled);
```

Plot the borders of all the coins on the original grayscale image using the coordinates returned by `bwboundaries` .

```
for k=1:10
   b = boundaries{k};
   plot(b(:,2),b(:,1),'g','LineWidth',3);
end
```



Detect and Measure Circular Objects in an Image

Step 1: Load Image

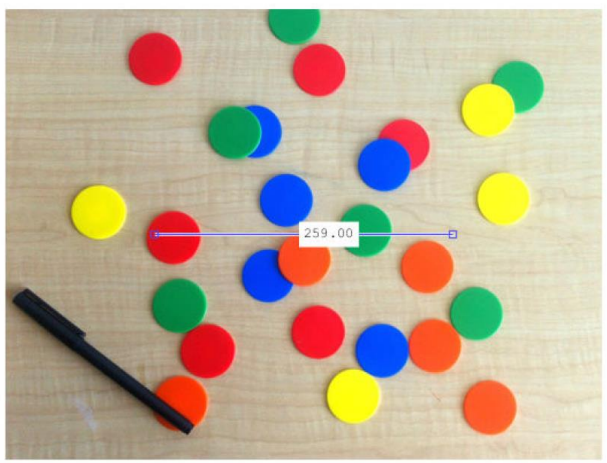
```
rgb = imread('coloredChips.png');
imshow(rgb)
```

Step 2: Determine Radius Range for Searching Circles

`imfindcircles` needs a radius range to search for the circles. A quick way to find the appropriate radius range is to use the interactive tool `imdistline` to get an approximate estimate of the radii of various objects.

```
d = imdistline;
```

```
delete(d)
```

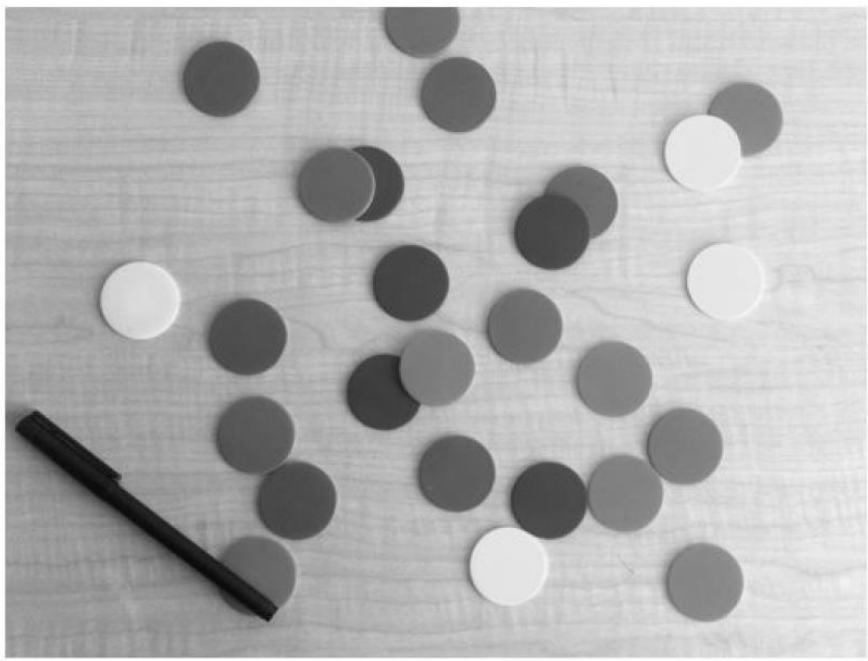


Detect and Measure Circular Objects in an Image

Step 3: Initial Attempt to Find Circles

Call `imfindcircles` on this image with the search radius of [20 25] pixels. Before that, it is a good practice to ask whether the objects are brighter or darker than the background. To answer that question, look at the grayscale version of this image.

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



The background is quite bright and most of the chips are darker than the background. But, by default, `imfindcircles` finds circular objects that are brighter than the background. So, set the parameter 'ObjectPolarity' to 'dark' in `imfindcircles` to search for dark circles.

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

```
centers =

     []


radii =
```

Step 4: Increase Detection Sensitivity

Coming back to the chip image, it is possible that at the default sensitivity level all the circles are lower than the internal threshold, which is why no circles were detected. By default, 'Sensitivity', which is a number between 0 and 1, is set to 0.85. Increase 'Sensitivity' to 0.9.

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

Detect and Measure Circular Objects in an Image

Step 5: Draw the Circles on the Image

The function `viscircles` can be used to draw circles on the image. Output variables `centers` and `radii` from `imfindcircles` can be passed directly to `viscircles`.

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

The circle centers seem correctly positioned and their corresponding radii seem to match well to the actual chips. But still quite a few chips were missed. Try increasing the 'Sensitivity' even more, to 0.92.

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

length(centers)
```

```
ans = 16
```

So increasing 'Sensitivity' gets us even more circles. Plot these circles on the image again.

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

Step 6: Use the Second Method (Two-stage) for Finding Circles

This result looks better. `imfindcircles` has two different methods for finding circles. So far the default method, called the *phase coding* method, was used for detecting circles. There's another method, popularly called the *two-stage* method, that is available in `imfindcircles`. Use the two-stage method and show the results.

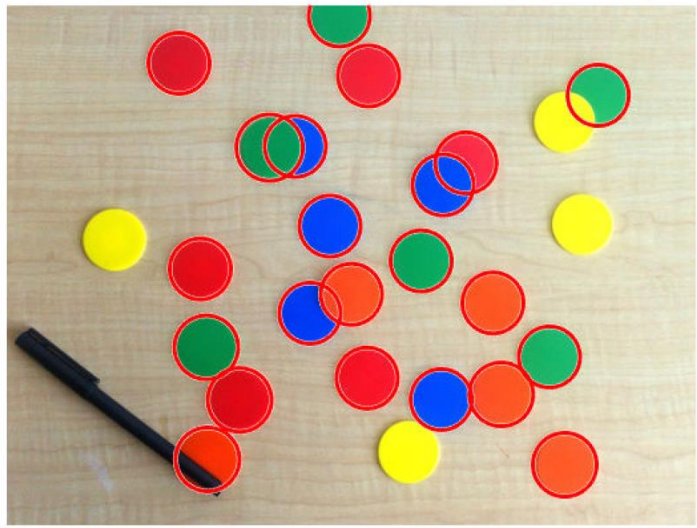
```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark', ...
          'Sensitivity',0.92,'Method','twostage');

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

Detect and Measure Circular Objects in an Image

Step 6: Use the Second Method (Two-stage) for Finding Circles

This result looks better. `imfindcircles` has two different methods for finding circles. So far the default method, called the *phase coding* method, was used for detecting circles. There's another method, popularly called the *two-stage* method, that is available in `imfindcircles`. Use the two-stage method and show the results.

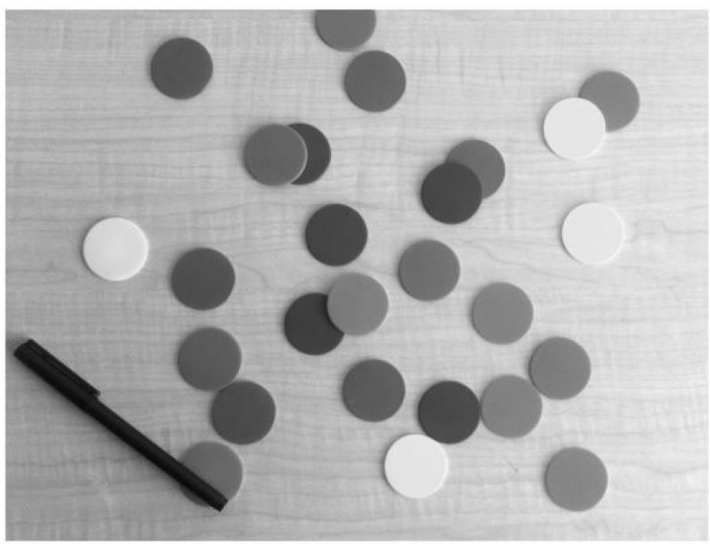
```
[centers,radii] = imfindcircles(rgb,[20 25],'ObjectPolarity','dark', ...
          'Sensitivity',0.92,'Method','twostage');

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



Step 7: Why are Some Circles Still Getting Missed?

Looking at the last result, it is curious that `imfindcircles` does not find the yellow chips in the image. The yellow chips do not have strong contrast with the background. In fact they seem to have very similar intensities as the background. Is it possible that the yellow chips are not really 'darker' than the background as was assumed? To confirm, show the grayscale version of this image again.

```
imshow(gray_image)
```



Detect and Measure Circular Objects in an Image

Step 8: Find 'Bright' Circles in the Image

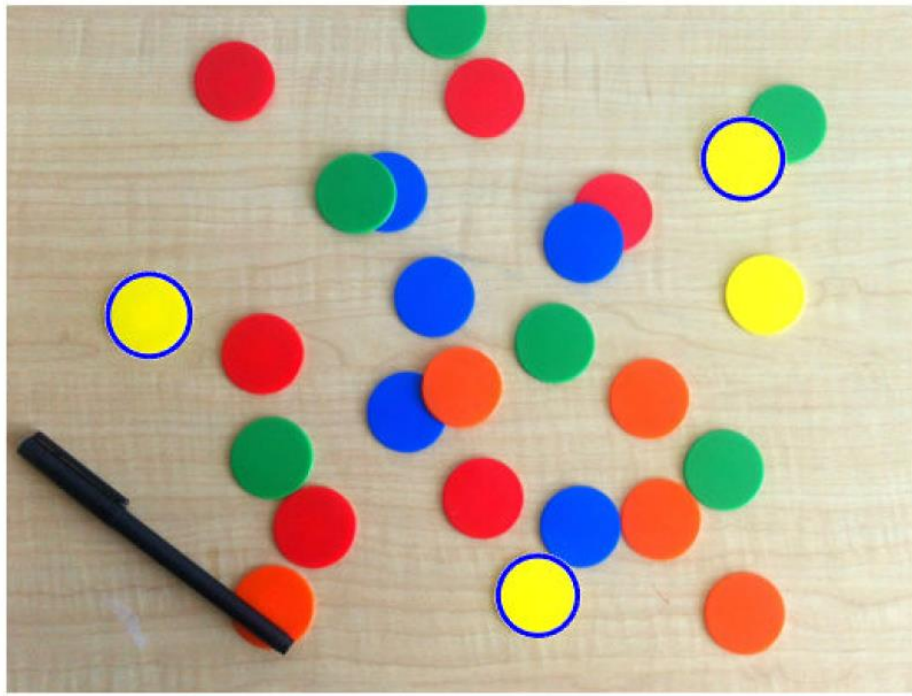
The yellow chips are almost the same intensity, maybe even brighter, as compared to the background. Therefore, to detect the yellow chips, change 'ObjectPolarity' to 'bright'.

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

Step 9: Draw 'Bright' Circles with Different Color

Draw the *bright* circles in a different color, by changing the 'Color' parameter in `viscircles`.

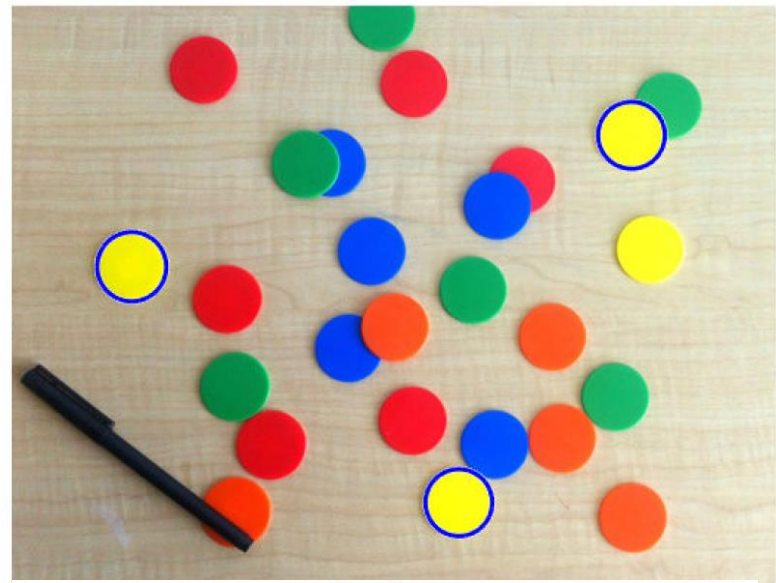
```
imshow(rgb)

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



Detect and Measure Circular Objects in an Image

Step 10: Lower the Value of 'EdgeThreshold'

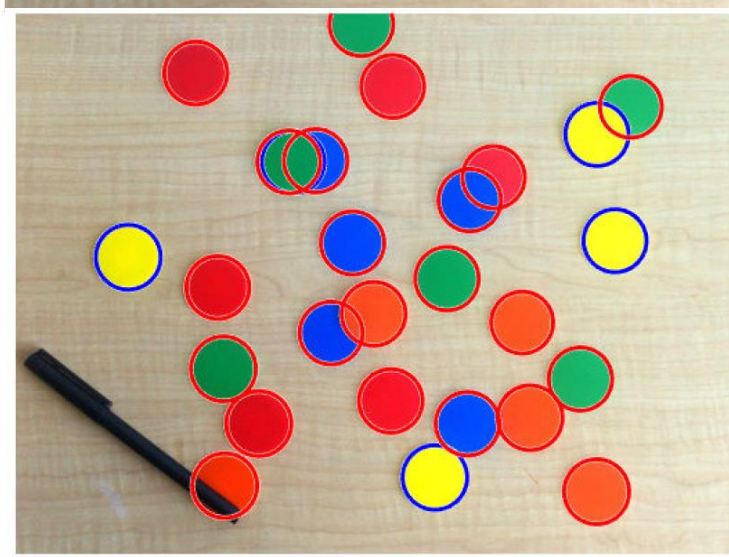
There is another parameter in `imfindcircles` which may be useful here, namely 'EdgeThreshold'. To find circles, `imfindcircles` uses only the edge pixels in the image. These edge pixels are essentially pixels with high gradient value. The 'EdgeThreshold'



Step 11: Draw 'Dark' and 'Bright' Circles Together

Now `imfindcircles` finds all of the yellow ones, and a green one too. Draw these chips in blue, together with the other chips that were found earlier (with 'ObjectPolarity' set to 'dark'), in red.

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



Detecting Cars in a Video of Traffic

Step 1: Access Video with VideoReader

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

Step 2: Explore Video with IMPLAY

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

Step 3: Develop Your Algorithm

```
darkCarValue = 50;
darkCar = rgb2gray(read(trafficVid,71));
noDarkCar = imextendedmax(darkCar, darkCarValue);
imshow(darkCar)
figure, imshow(noDarkCar)
```

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

Detecting Cars in a Video of Traffic

Step 4: Apply the Algorithm to the Video

```
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
```

Detecting Cars in a Video of Traffic

Step 5: Visualize Results

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