Correcting Nonuniform Illumination

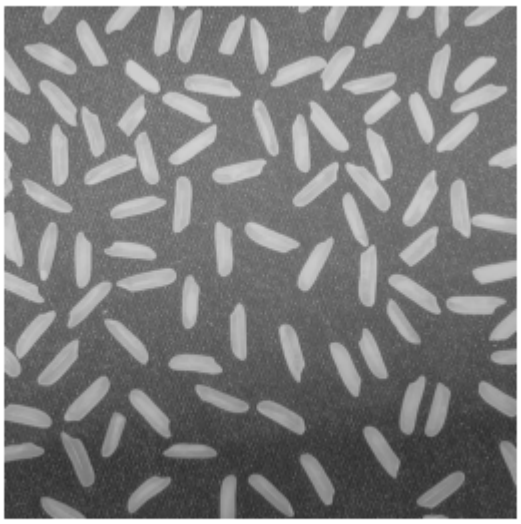
Image Processing Toolbox™ offers a variety of techniques to enhance an image prior to segmentation. Enhancement techniques can improve segmentation results by accounting for problems in the image capture process, such as non-uniform illumination, noise, or blurring.

This example shows how to correct nonuniform illumination in an image to make it easy to identify individual grains of rice in the image. You can then learn about the characteristics of the grains and easily compute statistics for all the grains in the image.

Step 1: Reading the Image

Suppose you have an image rice.png, which shows grains of rice on a varied gray background. You can get started by reading the image into MATLAB®.

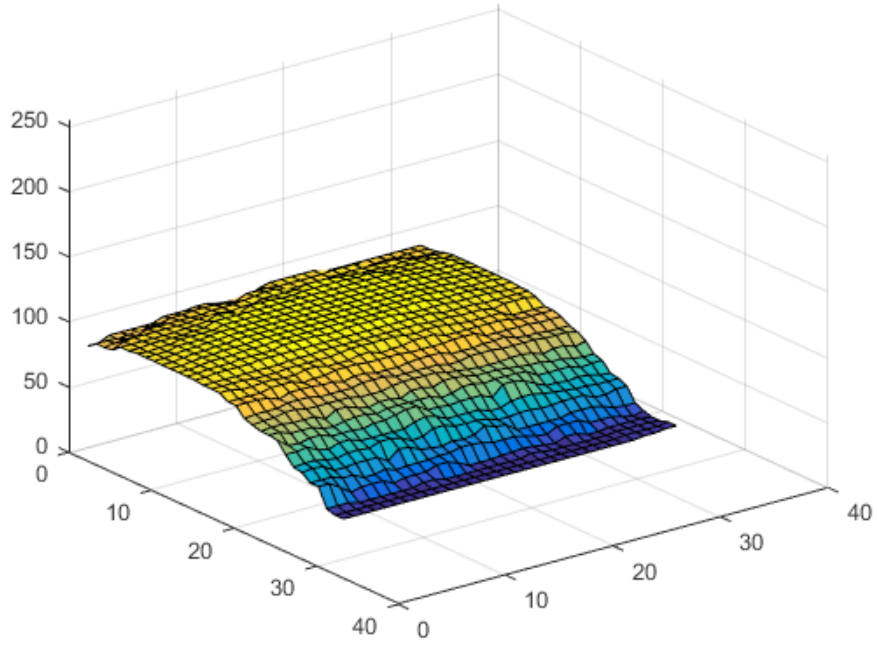
```
I = imread('rice.png');
imshow(I)
```



Step 2: Using Morphological Opening to Estimate the Background

Notice that the background illumination is brighter in the center of the image than at the bottom. You can use `imopen` to estimate the background illumination.

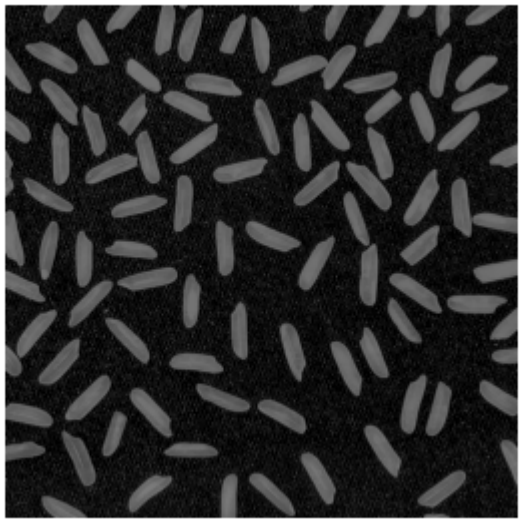
```
background = imopen(I,strel('disk',15));

% Display the Background Approximation as a Surface
figure
surf(double(background(1:8:end,1:8:end))),zlim([0 255]);
ax = gca;
ax.YDir = 'reverse';
```



Step 3: Subtracting the Background Image from the Original Image

You can then subtract the background from the image.

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



You can perform steps 2 and 3 together using the `imtophat` function, which first calculates the morphological opening and then subtracts it from the original image.

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

Step 4: Increasing the Image Contrast

You can use `imadjust` to increase the contrast of the image.

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

Step 5: Thresholding the Image

You can then create a new binary image by thresholding the adjusted image with `imbinarize` and removing background noise with `bwareaopen`.

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

Step 6: Identifying Objects in the Image

The function `bwconncomp` finds all the connected components (objects) in the binary image. The accuracy of your results depends on the size of the objects, the connectivity parameter (4, 8, or arbitrary), and whether or not any objects are touching (in which case they may be labeled as one object). Some of the rice grains in `bw` are touching.

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


     Connectivity: 4
        ImageSize: [256 256]
       NumObjects: 95
     PixelIdxList: {1x95 cell}
```

Step 7: Examining One Object

Each distinct object is labeled with the same integer value. You can see a single grain by showing the 50th connected component, for example.

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

Step 8: Viewing All Objects

One way to visualize connected components is to create a label matrix and then display it as a pseudo-color indexed image.

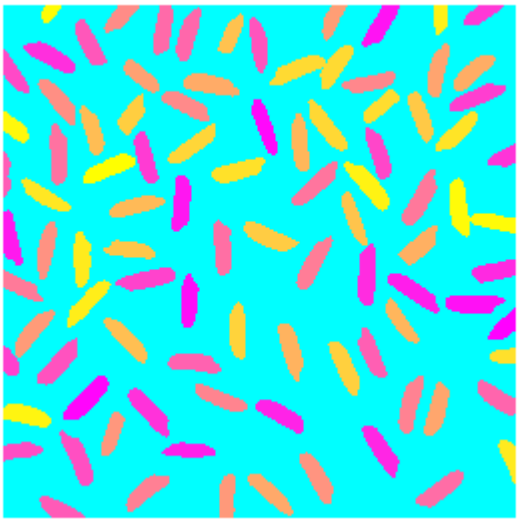
You can use `labelmatrix` to create a label matrix from the output of `bwconncomp`. Note that `labelmatrix` stores the label matrix in the smallest numeric class necessary for the number of objects.

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

```
  Name             Size          Bytes       Class     Attributes
  labeled       256x256          65536       uint8
```

In the pseudo-color image, the label identifying each object in the label matrix maps to a different color in the associated colormap matrix. You can use `label2rgb` to choose the colormap, the background color, and how objects in the label matrix map to colors in the colormap.

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



Step 9: Computing the Area of Each Object

Each rice grain is one connected component in the `cc` structure. You can use `regionprops` on cc to compute the area.

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


95x1 struct array with fields:

    Area
    Centroid
    BoundingBox
```

To find the area of the 50th component, you can use dot notation to access the `Area` field in the 50th element of the `graindata` structure array.

```
graindata(50).Area
ans =

   194
```

Step 10: Computing Area-Based Statistics

To hold the area measurement for each grain, you can create a new vector, `grain _ areas`.

```
grain _ areas = [graindata.Area];
```

You can then find the grain with the smallest area.

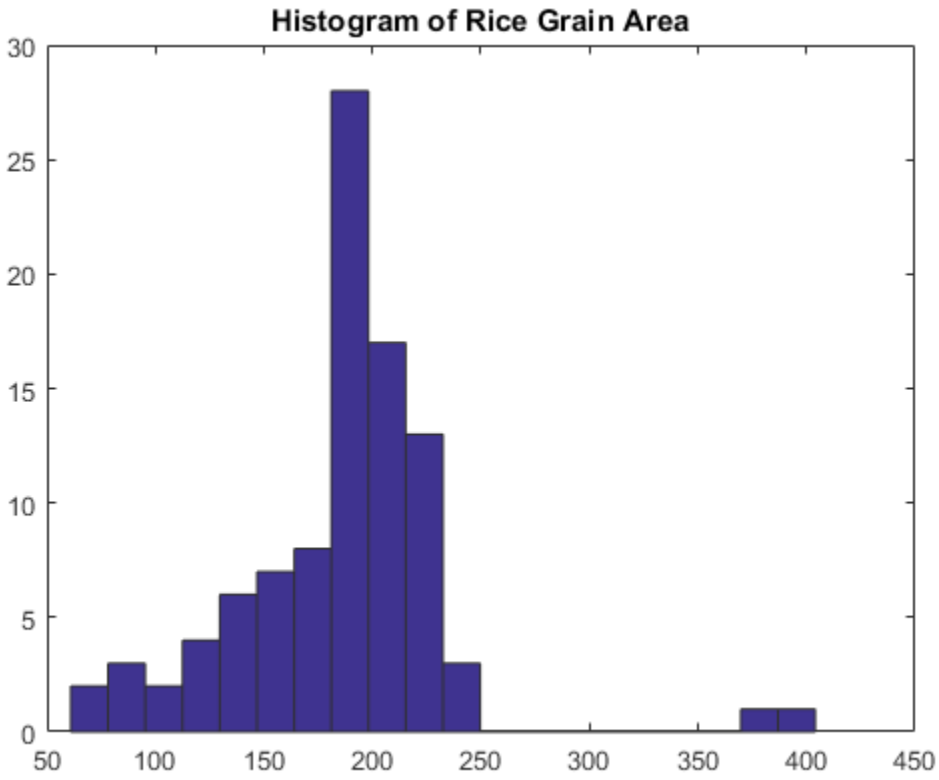
```
[min _ area, idx] = min(grain _ areas)
grain = false(size(bw));
grain(cc.PixelIdxList{idx}) = true;
imshow(grain);
min _ area =

    61
```

```
idx =

    16
```

Step 11: Creating a Histogram of the Area

To visualize the area distribution for all of the rice grains, you can create a histogram.

```
nbins = 20;
figure, histogram(grain_areas,nbins)
title('Histogram of Rice Grain Area');
```



Learn More About Image Processing

- *The Watershed Transform: Strategies for Image Segmentation* (technical article)
- *New Features for High-Performance Image Processing in MATLAB* (technical article)
- *ThresholdLocally* (download)
- *Image Processing Toolbox* (product trial)