Detecting and Measuring Circular Objects in an Image

Image segmentation is often an effective approach for identifying objects in an image. Image Processing Toolbox™ offers a variety of techniques for image segmentation.

This example shows how you can use the `imfindcircles` function in Image Processing Toolbox to automatically detect circles or circular objects in an image. It also shows the use of `viscircles` to visualize the detected circles.

Step 1: Loading the Image

Suppose you have an image of round plastic chips of various colors. You start by reading this file into MATLAB®.

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

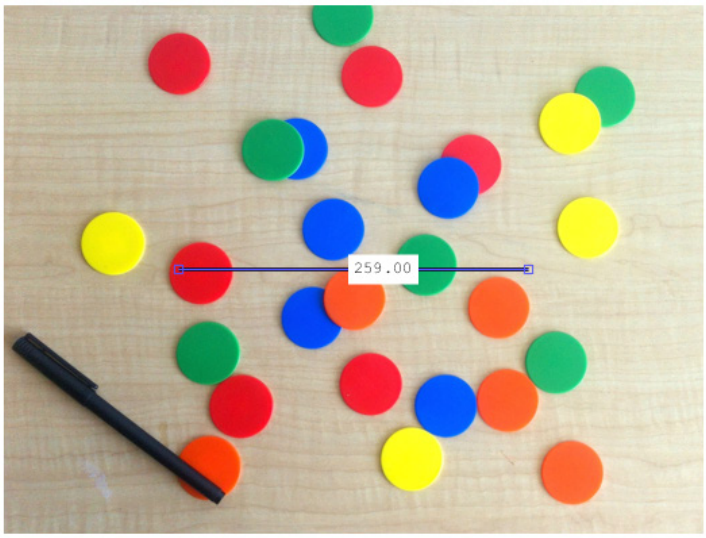
Besides having plenty of circles to detect, there are a few interesting things going on in this image from a circle detection point of view:

- There are chips of different colors, which have different contrasts with respect to the background. On one end, the blue and red ones have strong contrast on this background. On the other end, some of the yellow chips do not contrast well with the background.
- Some chips are on top of each other, and others are close together and almost touching each other. Overlapping object boundaries and object occlusion are usually challenging scenarios for object detection.

Step 2: Determining the Radius Range for Searching Circles

The `imfindcircles` function in Image Processing Toolbox 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;
```



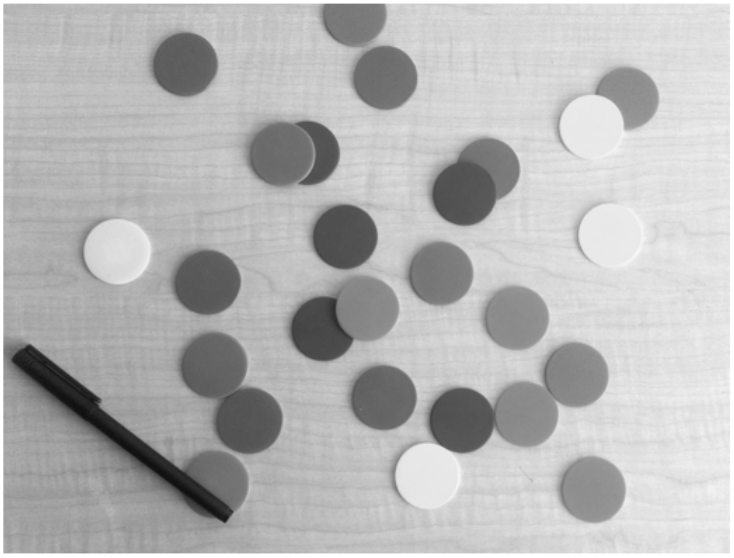
The `imdistline` tool creates a draggable line for measuring distance; you can adjust it to fit across a chip and use the number shown to estimate its radius. Most chips have radius in the range of 21–23 pixels. Using a slightly larger radius range is recommended; in this case, you can use 20–25 pixels. Before searching for the circles, you should remove the `imdistline` tool.

```
delete(d);
```

Step 3: Initial Attempt to Find Circles

Before searching for circles, it is a good practice to ask whether the objects are brighter or darker than the background. To answer that question, you can create a grayscale version of the image.

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



The background is quite bright, and most of the chips are darker than the background. By default, the `imfindcircles` function finds circular objects that are brighter than the background. To find images that are darker than the background, you should set the parameter `'ObjectPolarity'` to `'dark'` in `imfindcircles`. Using the measurement obtained earlier, you can specify the search radius of [20 25] pixels.

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

Note that the outputs `centers` and `radii` are empty, which means that no circles were found. This happens frequently because `imfindcircles` is a circle *detector*, and similar to most detectors, `imfindcircles` has an internal *detection threshold* that determines its sensitivity. In simple terms, this means that the detector's confidence in a given (circle) detection has to be greater than a certain level before it is considered a *valid* detection. You can use the `'Sensitivity'` parameter in the `imfindcircles` function to control this internal threshold and, consequently, the sensitivity of the algorithm. A higher sensitivity value sets the detection threshold lower and leads to detecting more circles. This is similar to the sensitivity control on the motion detectors used in home security systems.

Step 4: Increasing Detection Sensitivity

It is possible that at the default sensitivity level, all the circles in the chip image 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. To increase detection sensitivity, try specifying a `'Sensitivity'` parameter of 0.9.

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

  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 =

   23.1604
   22.5710
   22.9576
   23.7356
   22.9551
   22.9995
   22.9055
   23.0298
```

This time `imfindcircles` found some circles—eight to be precise. The output variable `centers` contains the locations of circle centers, and the output variable radii contains the estimated radii of those circles.

Step 5: Drawing the Circles on the Image

You can use the function `viscircles` to draw circles on the image. The 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 quite a few chips were missed. To detect additional circles, try increasing the `'Sensitivity'` even more, to 0.92.

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

```
length(centers)
```

```
ans =

    16
```

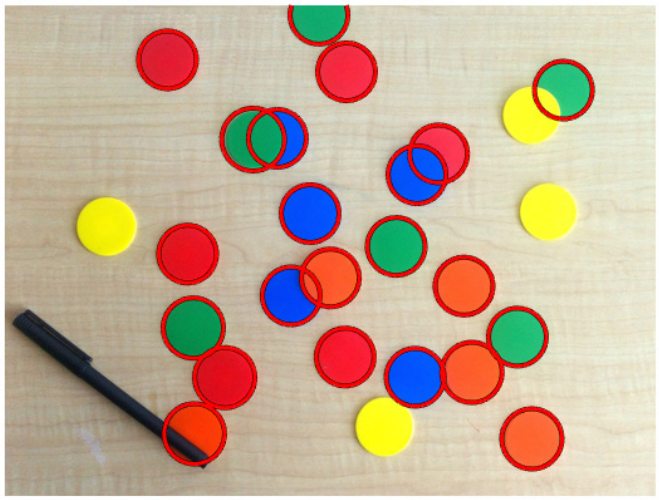
So increasing `'Sensitivity'` detects even more circles. You can use `viscircles` to plot these circles on the image again.

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

Step 6: Using the Second Method (Two-Stage) for Finding Circles

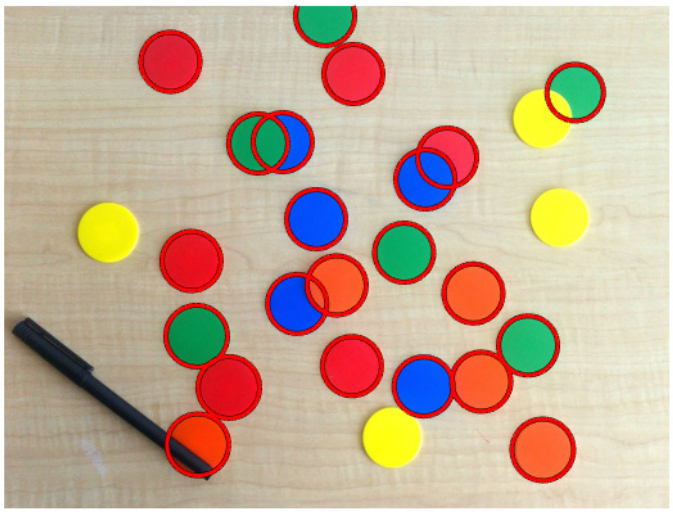
The default method, called the *phase coding* method, was used for detecting circles in this example so far. Let's compare it with the other method available in imfindcircles, called the two-stage method. Try adding the parameters `'Method'` and `'twostage'` to see the results of this method.

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


delete(h);


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



The two-stage method is detecting more circles, at the `'Sensitivity'` of 0.92. In general, these two methods are complementary in that have they have different strengths. Phase-coding method is typically faster and slightly more robust to noise than the two-stage method. But it may also need higher `'Sensitivity'` levels to get the same number of detections as the two-stage method. For example, the phase-coding method also finds the same chips if the `'Sensitivity'` level is raised higher, say to 0.95.

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

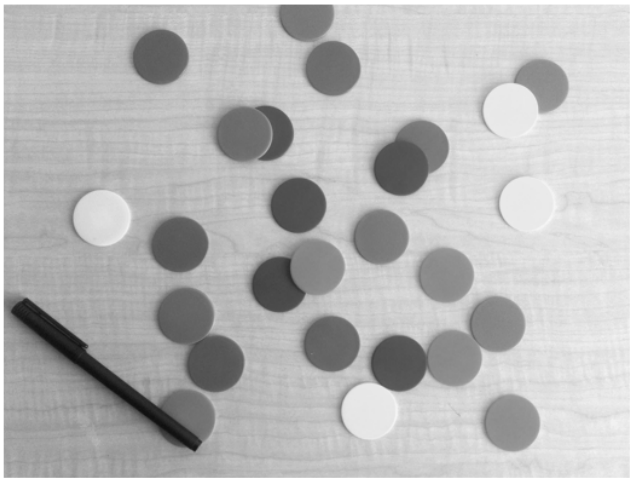


Note that both methods in `imfindcircles` find the centers and radii of the partially visible (occluded) chips accurately.

Step 7: Understanding Why Some Circles Still Get Missed

Looking at the last result, you can see that `imfindcircles` does not find the yellow chips in the image. This could be because 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, you can look at the grayscale version of this image again.

```
imshow(gray_image);
```



Step 8: Finding "Bright" Circles in the Image

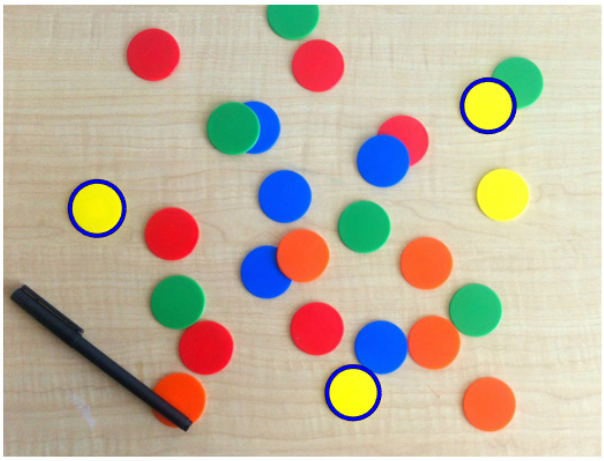
Indeed! The yellow chips are almost the same intensity, maybe even brighter, than the background. Therefore, to detect the yellow chips, try changing `'ObjectPolarity'` to `'bright'`.

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

Step 9: Draw "Bright" Circles with Different Color

You can draw the *bright* circles in a different color, say blue, by changing the `'Color'` parameter in `viscircles`.

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



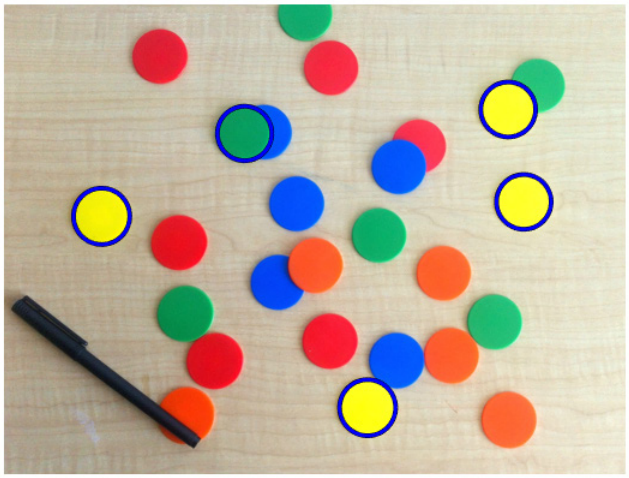
Three of the missing yellow chips were found. One yellow chip is still missing. These yellow ones are hard to find because of they don't *stand out* as well as others on this background.

Step 10: Lowering the Value of `'EdgeThreshold'`

There is another parameter in `imfindcircles` that 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'` parameter controls how *high* the gradient value at a pixel has to be before it is considered an edge pixel and included in computation. A high value (closer to 1) for this parameter will allow only the strong edges (higher gradient values) to be included, whereas a low value (closer to 0) is more permissive and includes even the weaker edges (lower gradient values) in computation. In case of the missing yellow chip, since the contrast is low, some of the boundary pixels (on the circumference of the chip) are expected to have low gradient values. Therefore, you can lower the `'EdgeThreshold'` parameter to ensure that the most of the edge pixels for the yellow chip are included in computation.

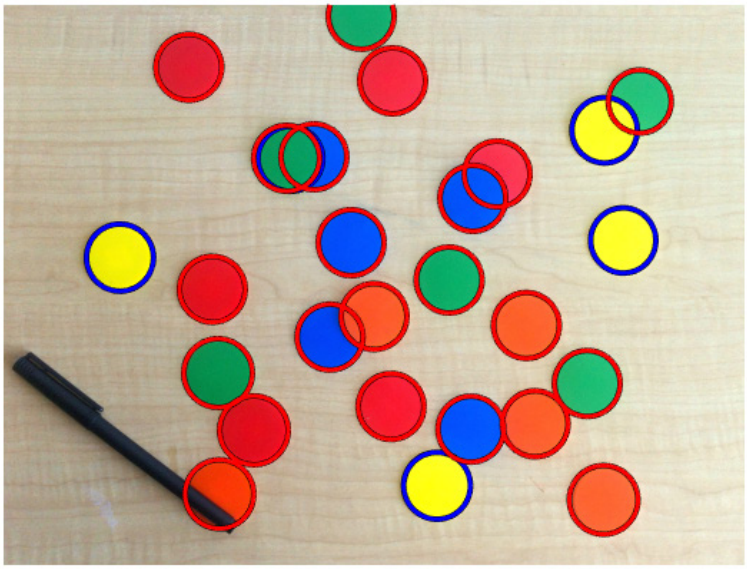
```
[centersBright, radiiBright, metricBright] = imfindcircles(rgb,[20 25], ...
    'ObjectPolarity','bright','Sensitivity',0.92,'EdgeThreshold',0.1);
delete(hBright);
hBright = viscircles(centersBright, radiiBright,'Color','b');
```



Step 11: Drawing "Dark" and "Bright" Circles Together

Now `imfindcircles` finds all of the yellow ones, and a green one too. You can draw these chips in blue to differentiate them from the chips that were found earlier (with `'ObjectPolarity'` set to `'dark'`), which are drawn in red.

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



All the circles are detected. A final word: Changing the parameters to be more aggressive in detection may find more circles, but it also increases the likelihood of detecting false circles. There is a tradeoff between the number of true circles that can be found (detection rate) and the number of false circles that are found with them (false alarm rate).

Learn More About Image Processing

- *Finding Circles in Images Using MATLAB* 4:51
- *Circle Finder* (download)
- *Image Processing Toolbox* (product trial)