Recognize Text Using Optical Character Recognition (OCR)

This example shows how to use the `ocr` function from the Computer Vision Toolbox™ to perform Optical Character Recognition.

Text Recognition Using the `ocr` Function

Recognizing text in images is useful in many computer vision applications such as image search, document analysis, and robot navigation. The `ocr` function provides an easy way to add text recognition functionality to a wide range of applications.

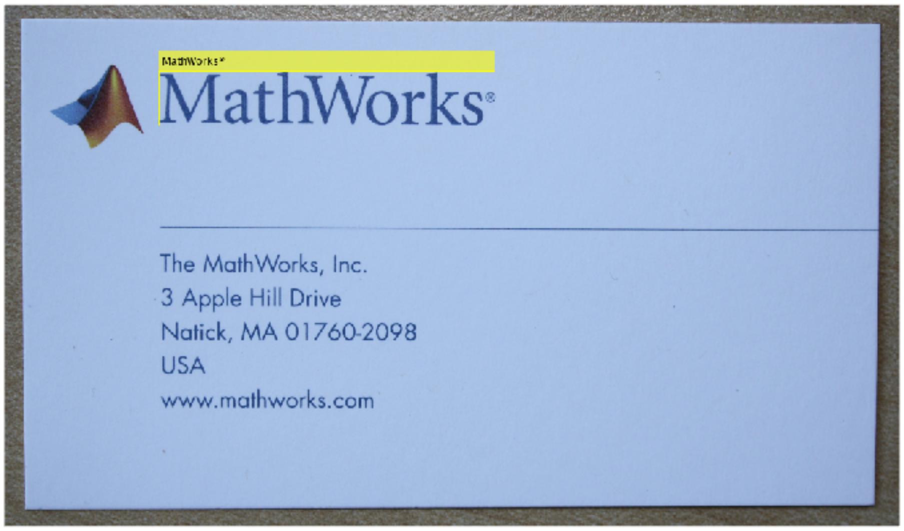
```
% Load an image.
I = imread('businessCard.png');

% Perform OCR.
results = ocr(I);

% Display one of the recognized words.
word = results.Words{2}
```

```
word =
'MathWorks®'
```

```
% Location of the word in I
wordBBox = results.WordBoundingBoxes(2,:)
```

```
wordBBox = 1×4
   173    75   376    61
```

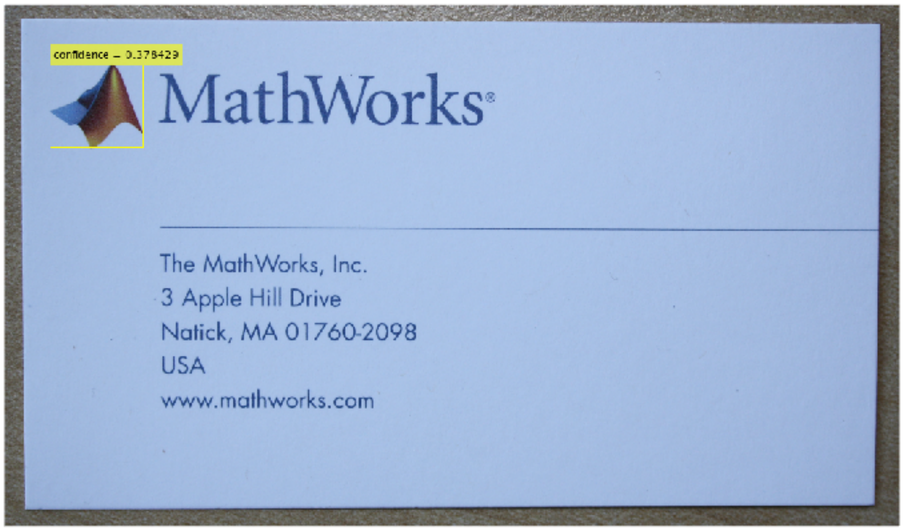
```
% Show the location of the word in the original image.
figure;
Iname = insertObjectAnnotation(I,'rectangle',wordBBox,word);
imshow(Iname);
```



Information Returned By the `ocr` Function

The `ocr` functions returns the recognized text, the recognition confidence, and the location of the text in the original image. You can use this information to identify the location of misclassified text within the image.

```
% Find characters with low confidence.
lowConfidenceIdx = results.CharacterConfidences < 0.5;

% Get the bounding box locations of the low confidence characters.
lowConfBBoxes = results.CharacterBoundingBoxes(lowConfidenceIdx, :);

% Get confidence values.
lowConfVal = results.CharacterConfidences(lowConfidenceIdx);

% Annotate image with character confidences.
str      = sprintf('confidence = %f', lowConfVal);
Ilowconf = insertObjectAnnotation(I,'rectangle',lowConfBBoxes,str);

figure;
imshow(Ilowconf);
```

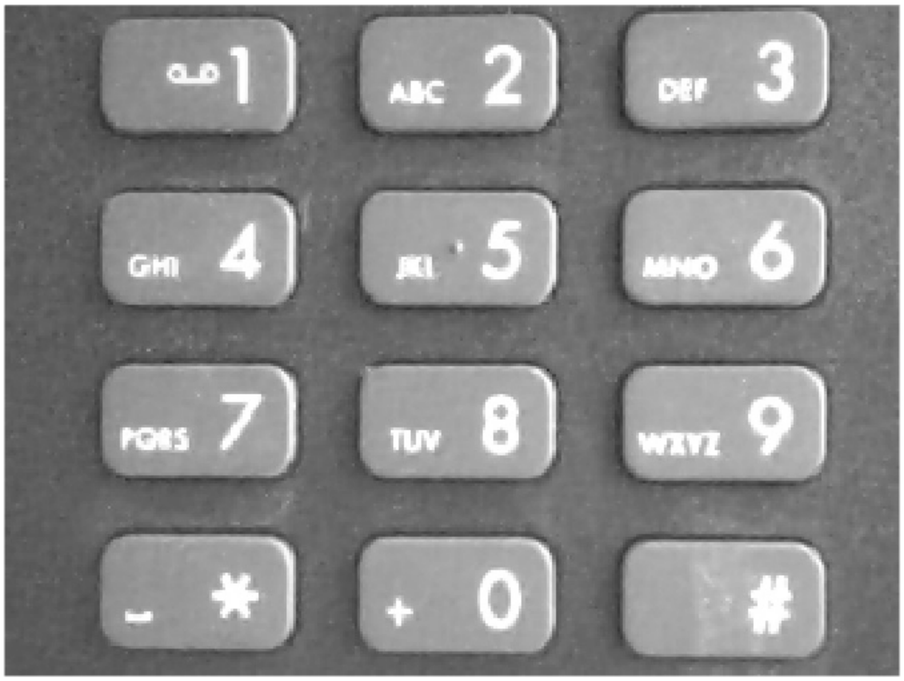


Here, the logo in the business card is incorrectly classified as a text character. These kind of OCR errors can be identified using the confidence values before any further processing takes place.

Challenges Obtaining Accurate Results

`ocr` performs best when the text is located on a uniform background and is formatted like a document. When the text appears on a non-uniform background, additional pre-processing steps are required to get the best OCR results. In this part of the example, you will try to locate the digits on a keypad. Although, the keypad image may appear to be easy for OCR, it is actually quite challenging because the text is on a non-uniform background.

```
I = imread('keypad.jpg');
I = im2gray(I);

figure;
imshow(I)
```



```
% Run OCR on the image
results = ocr(I);

results.Text
```

```
ans =
    '

     '
```

The empty `results.Text` indicates that no text is recognized. In the keypad image, the text is sparse and located on an irregular background. In this case, the heuristics used for document layout analysis within `ocr` might be failing to find blocks of text within the image, and, as a result, text recognition fails. In this situation, disabling the automatic layout analysis, using the 'TextLayout' parameter, may help improve the results.

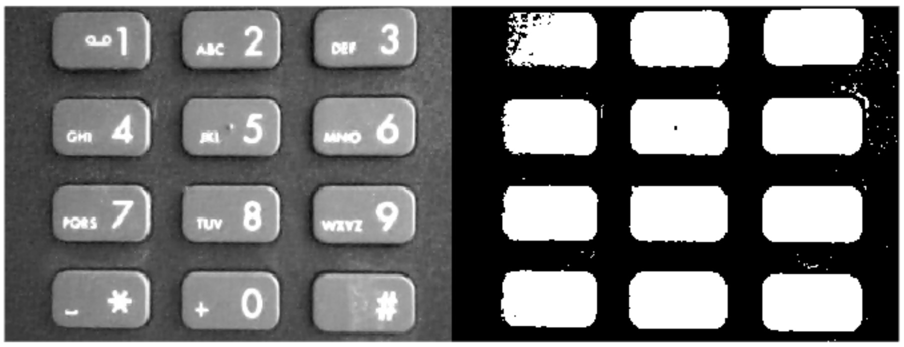
```
% Set 'TextLayout' to 'Block' to instruct ocr to assume the image
% contains just one block of text.
results = ocr(I,'TextLayout','Block');

results.Text
```

```
ans =

  0×0 empty char array
```

What Went Wrong?

Adjusting the 'TextLayout' parameter did not help. To understand why OCR continues to fail, you have to investigate the initial binarization step performed within `ocr`. You can use `imbinarize` to check this initial binarization step because both `ocr` and the default 'global' method in `imbinarize` use Otsu's method for image binarization.

```
BW = imbinarize(I);

figure;
imshowpair(I,BW,'montage');
```

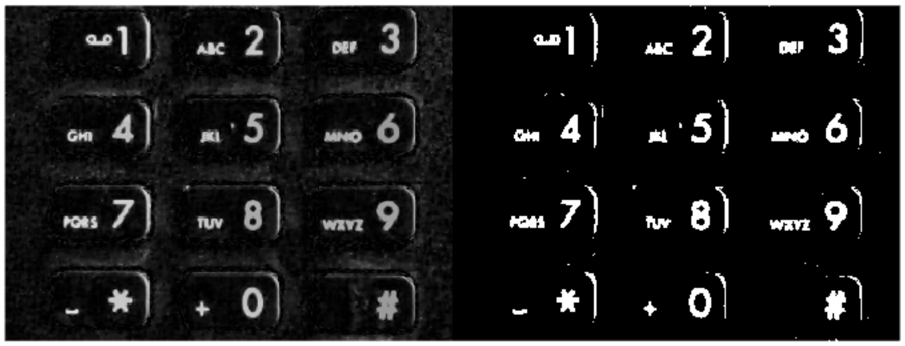


After thresholding, the binary image contains no text. This is why `ocr` failed to recognize any text in the original image. You can help improve the results by pre-processing the image to improve text segmentation. The next part of the example explores two useful pre-processing techniques.

Image Pre-processing Techniques To Improve Results

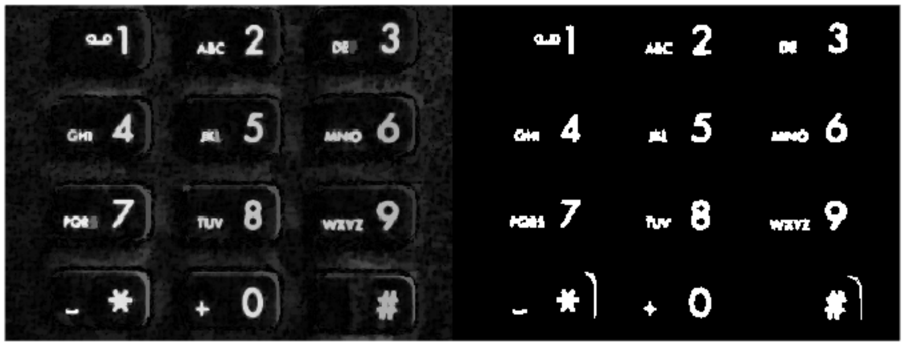
The poor text segmentation seen above is caused by the non-uniform background in the image, i.e. the light-gray keys surrounded by dark gray. You can use the following pre-processing technique to remove the background variations and improve the text segmentation. Additional details about this technique are given in the example entitled Correcting Nonuniform Illumination.

```
% Remove keypad background.
Icorrected = imtophat(I,strel('disk',15));

BW1 = imbinarize(Icorrected);

figure;
imshowpair(Icorrected,BW1,'montage');
```



After removing the background variation, the digits are now visible in the binary image. However, there are a few artifacts at the edge of the keys and the small text next to the digits that may continue to hinder accurate OCR of the whole image. Additional pre-processing using morphological reconstruction helps to remove these artifacts and produce a cleaner image for OCR.

```
% Perform morphological reconstruction and show binarized image.
marker = imerode(Icorrected, strel('line',10,0));
Iclean = imreconstruct(marker, Icorrected);

BW2 = imbinarize(Iclean);

figure;
imshowpair(Iclean,BW2,'montage');
```



After these pre-processing steps, the digits are now well segmented from the background and `ocr` produces better results.

```
results = ocr(BW2,'TextLayout','Block');

results.Text
```

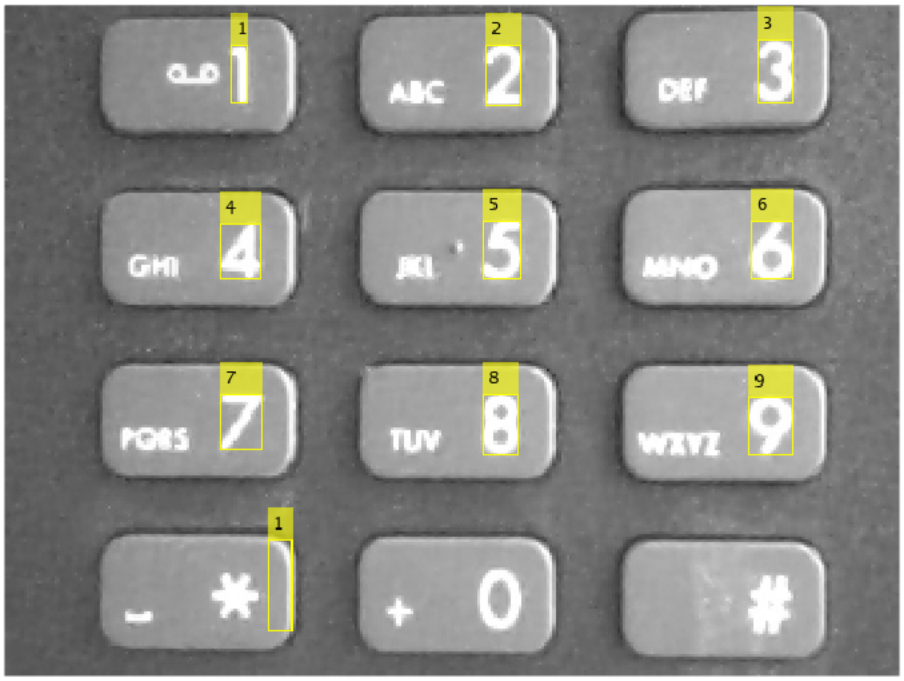
```
ans =
    '«-1 ..c2 .3
     ....4 .5 .....6
     W7 M8 M9
     -*1..o fl

     '
```

There is some "noise" in the results due to the smaller text next to the digits. Also, the digit 0, is falsely recognized as the letter 'o'. This type of error may happen when two characters have similar shapes and there is not enough surrounding text for the `ocr` function to determine the best classification for a specific character. Despite the "noisy" results, you can still find the digit locations in the original image using the `locateText` method with the OCR results.

The `locateText` method supports regular expressions so you can ignore irrelevant text.

```
% The regular expression, '\d', matches the location of any digit in the
% recognized text and ignores all non-digit characters.
```

```
regularExpr = '\d';

% Get bounding boxes around text that matches the regular expression
bboxes = locateText(results,regularExpr,'UseRegexp',true);

digits = regexp(results.Text,regularExpr,'match');

% draw boxes around the digits
Idigits = insertObjectAnnotation(I,'rectangle',bboxes,digits);

figure;
imshow(Idigits);
```



Another approach to improve the results is to leverage a priori knowledge about the text within the image. In this example, the text you are interested in contains only numeric digits. You can improve the results by constraining `ocr` to only select the best matches from the set '0123456789'.

```
% Use the 'CharacterSet' parameter to constrain OCR
results = ocr(BW2, 'CharacterSet','0123456789','TextLayout','Block');

results.Text
```

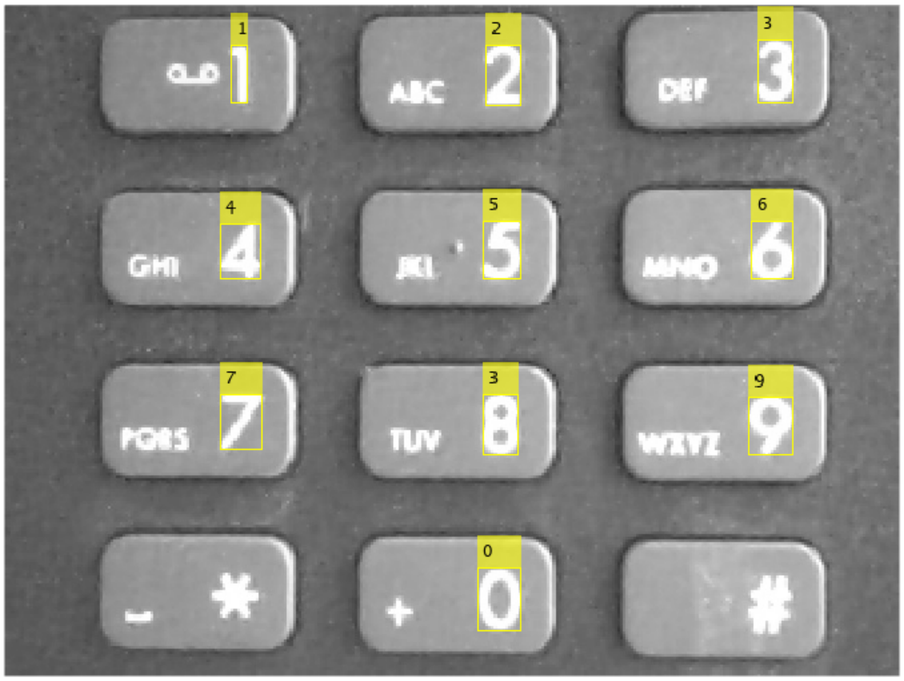
```
ans =
    ' 1 1   2   3
     5 4  5   06
      7  3  9
      4 1 0  51
```

'

The results now only have characters from the digit character set. However, you can see that several non-digit characters in the image are falsely recognized as digits. This can happen when a non-digit character closely resembles one of the digits.

You can use the fact that there are only 10 digits on the keypad along with the character confidences to find the 10 best digits.

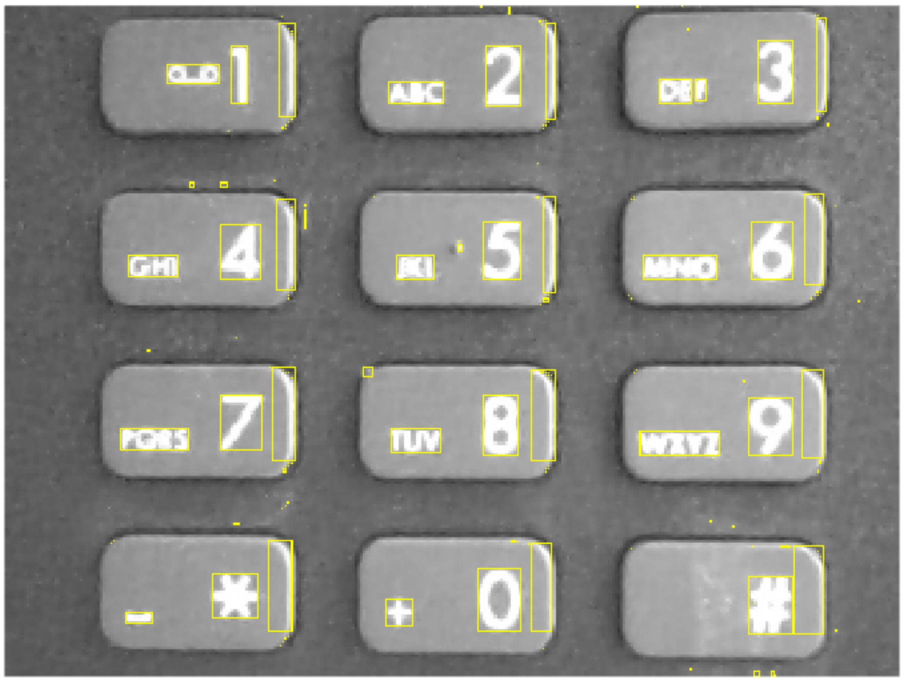
```
% Sort the character confidences.
[sortedConf, sortedIndex] = sort(results.CharacterConfidences, 'descend');

% Keep indices associated with non-NaN confidences values.
indexesNaNsRemoved = sortedIndex( ~isnan(sortedConf) );

% Get the top ten indexes.
topTenIndexes = indexesNaNsRemoved(1:10);

% Select the top ten results.
digits = num2cell(results.Text(topTenIndexes));
bboxes = results.CharacterBoundingBoxes(topTenIndexes, :);

Idigits = insertObjectAnnotation(I,'rectangle',bboxes,digits);

figure;
imshow(Idigits);
```



ROI-based Processing To Improve Results

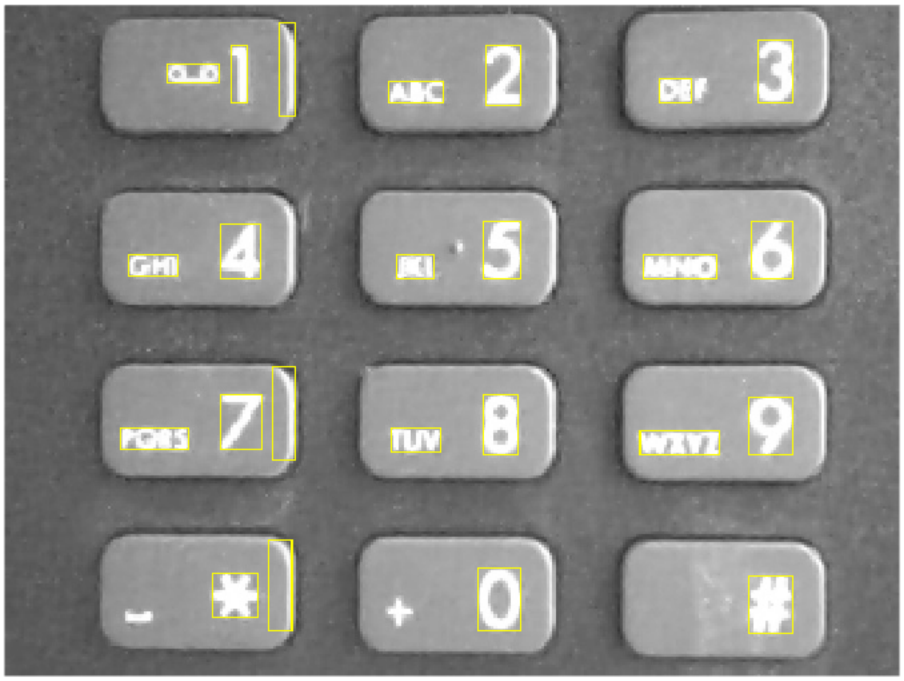
In some situations, just pre-processing the image may not be sufficient to achieve good OCR results. One approach to use in this situation, is to identify specific regions in the image that `ocr` should process. In the keypad example image, these regions would be those that just contain the digits. You may select the regions manually using `imrect`, or you can automate the process. One method for automating text detection is given in the example entitled Automatically Detect and Recognize Text in Natural Images. In this example, you will use `vision.BlobAnalysis` to find the digits on the keypad.

```
% Initialize the blob analysis System object(TM).
blobAnalyzer = vision.BlobAnalysis('MaximumCount',500);

% Run the blob analyzer to find connected components and their statistics.
[area,centroids,roi] = step(blobAnalyzer,BW1);

% Show all the connected regions.
img = insertShape(I,'rectangle',roi);
figure;
imshow(img);
```



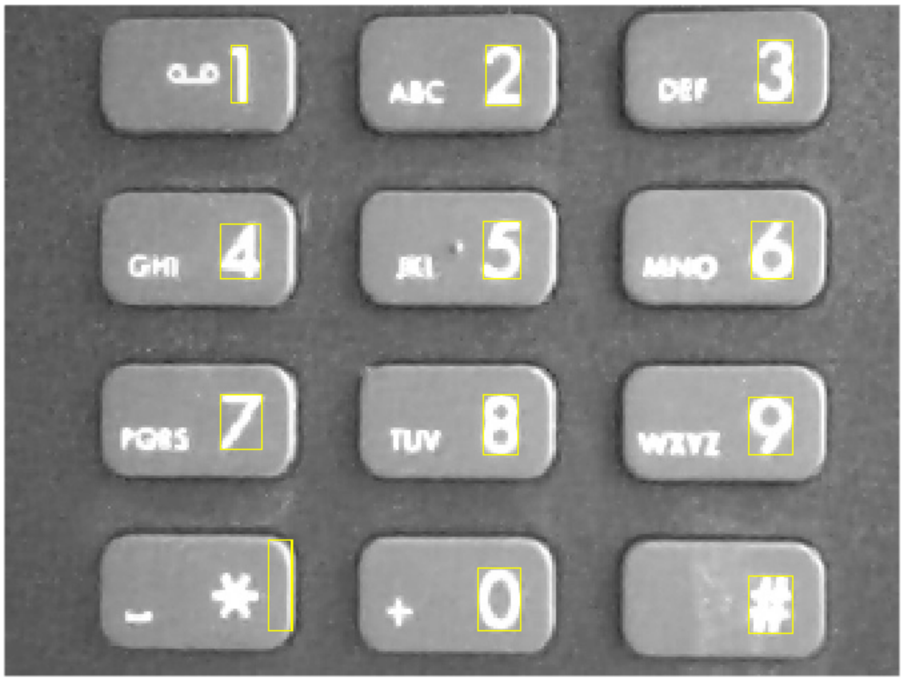
There are many connected regions within the keypad image. Small regions are not likely to contain any text and can be removed using the area statistic returned by `vision.BlobAnalysis`. Here, regions having an area smaller than 300 are removed.

```
areaConstraint = area > 300;

% Keep regions that meet the area constraint.
roi = double(roi(areaConstraint, :));

% Show remaining blobs after applying the area constraint.
img = insertShape(I,'rectangle',roi);
figure;
imshow(img);
```



Further processing based on a region's aspect ratio is applied to identify regions that are likely to contain a single character. This helps to remove the smaller text characters that are jumbled together next to the digits. In general, the larger the text the easier it is for `ocr` to recognize.

```
% Compute the aspect ratio.
width  = roi(:,3);
height = roi(:,4);
aspectRatio = width ./ height;

% An aspect ratio between 0.25 and 1 is typical for individual characters
% as they are usually not very short and wide or very tall and skinny.
roi = roi( aspectRatio > 0.25 & aspectRatio < 1 ,:);

% Show regions after applying the area and aspect ratio constraints.
img = insertShape(I,'rectangle',roi);
figure;
imshow(img);
```

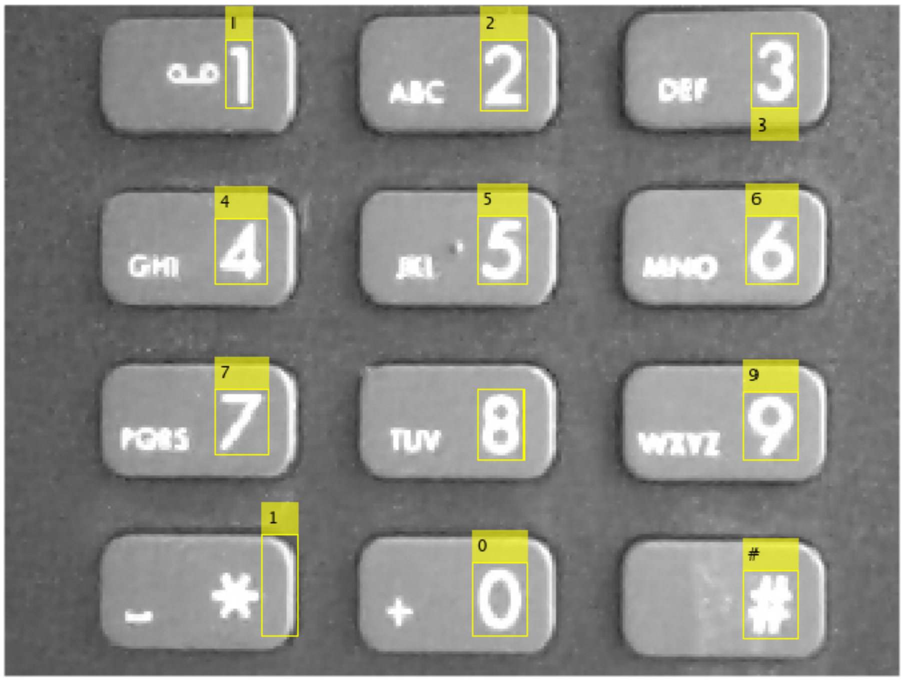


The remaining regions can be passed into the `ocr` function, which accepts rectangular regions of interest as input. The size of the regions are increased slightly to include additional background pixels around the text characters. This helps to improve the internal heuristics used to determine the polarity of the text on the background (e.g. light text on a dark background vs. dark text on a light background).

```
roi(:,1:2) = roi(:,1:2) - 4;
roi(:,3:4) = roi(:,3:4) + 8;
results = ocr(BW1, roi,'TextLayout','Block');
```

The recognized text can be displayed on the original image using `insertObjectAnnotation`. The `deblank` function is used to remove any trailing characters, such as white space or new lines. There are a few missing classifications in these results (e.g. the digit 8) that are correctable using additional pre-processing techniques.

```
text = deblank( {results.Text} );
img  = insertObjectAnnotation(I,'rectangle',roi,text);

figure;
imshow(img)
```



Although `vision.BlobAnalysis` enabled you to find the digits in the keypad image, it may not work as well for images of natural scenes where there are many objects in addition to the text. For these types of images, the technique shown in the example entitled Automatically Detect and Recognize Text in Natural Images may provide better text detection results.

Summary

This example showed how the `ocr` function can be used to recognize text in images, and how a seemingly easy image for OCR required extra pre-processing steps to produce good results.

References

[1] Ray Smith. Hybrid Page Layout Analysis via Tab-Stop Detection. Proceedings of the 10th international conference on document analysis and recognition. 2009.

Copyright 2013-2015 The MathWorks, Inc.