Introduction to Mathematical Image Processing
Image Processing Toolbox: Part 4
(image deblurring and Hough transform)

Suh-Yuh Yang (楊肅煜)

Department of Mathematics, National Central University
Jhongli District, Taoyuan City 32001, Taiwan

http://www.math.ncu.edu.tw/∼syyang/

Image deblurring

- Deblurring is the process of removing blurring artifacts (such as blur caused by defocus aberration or motion blur) from images.
- *The blur is typically modeled as a convolution point-spread function (PSF) with a hypothetical sharp input image, where both the sharp input image (which is to be recovered) and the PSF are unknown.*
- Image deblurring removes distortion from a blurry image using knowledge of the PSF.
- Image deblurring algorithms in IP Toolbox include *Wiener, and regularized filter deconvolution,* blind, Lucy-Richardson, as well as conversions between the PSF and optical transfer function.

Wiener and regularized filters

How to deblur an image using *Wiener and regularized filters?*

- `deconvwnr` function deblurs the image using Wiener filter. `deconvreg` function deblurs with a regularized filter.
- Wiener deconvolution can be used effectively when frequency characteristics of the image and additive noise are known to some extent. *In the absence of noise, the Wiener filter reduces to an ideal inverse filter.*
- `J=deconvwnr(I,PSF,NSR)`

  `I` is the input image, can be of class uint8, uint16, int16, single or double. Image `J` has the same class as image `I`.

  `PSF` the point-spread function with which `I` was convolved;

  `NSR` the noise-to-signal power ratio of the additive noise

  (`NSR` can be a scalar or a spectral-domain array of the same size as image `I`. `NSR=0` is equivalent to creating an ideal inverse filter).

Example

The following steps are taken to read 'puppy_gray.jpg', blur it, add noise to it and then restore the image using Wiener filter.

```
>>I=im2double(imread('puppy_gray.jpg'));
>>imshow(I)
>>LEN=21;
>>THETA=11;
>>PSF=fspecial('motion',LEN,THETA);
```

The function `fspecial` *returns a filter to approximate the linear motion of a camera by len pixels, with an angle of theta degrees in a counter-clockwise direction. The filter becomes a vector for horizontal and vertical motions. The default value of len is* 9 *and that of theta is* 0*, which corresponds to a horizontal motion of nine pixels.*

```
>>blurred=imfilter(I,PSF,'conv','circular');
```

```
>>figure,imshow(blurred)
```

Function: `imfilter`

`imfilter(f,h,mode,boundary_options,size_options)`

`f`: input image; `h`: filter mask;

`mode`: `'conv'` or `'corr'`, convolution or correlation (default);

`boundary_options`: how the filtering algorithm should treat border values. There are four possibilities:

(1) The boundaries of the input array (image) are extended by padding with *a value 'x', default x=0;*

(2) *'symmetric'*: extended by mirror-reflecting across its border;

(3) *'replicate'*: extended by replicating the values nearest to the image border;

(4) *'circular'*: extended by implicitly assuming the input array is periodic.

`size_options`: *'full'* (output image is the full filtered result); *'same'* (output image is of the same size as input image, is default).

Adding Gaussian white noise

`J=imnoise(I,'gaussian',M,V)` adds Gaussian white noise of mean `M` and variance `V` to image `I`.

```
>>mean=0;
>>var=0.002;
>>blurred_noise=imnoise(blurred,'gaussian',mean,var);
>>figure,imshow(blurred_noise)
```

image A

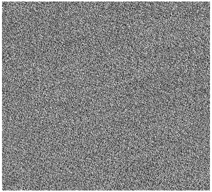
image B

image C

```
image A=puppy_gray.jpg
image B=puppy_gray_blurred.jpg
image C=puppy_gray_blurred_noise.jpg
```

Deblurring using Wiener filter

```
>>estimated_nsr=0;
>>wnr2=deconvwnr(blurred_noise,PSF,estimated_nsr);
>>estimated_nsr=0.002/var(I(:));
>>wnr3=deconvwnr(blurred_noise,PSF,estimated_nsr);
```



snr=0

snr=0.002/var(I(:))

Deblurring with a regularized filter

A regularized filter can be used effectively when limited information is known about the additive noise.

`J=deconvreg(I,PSF)` deconvolves image `I` using regularized filter algorithm and returns deblurred image `J`. The assumption is that image `I` was created by convolving a true image with a point-spread function and possibly by adding noise.

`J=deconvreg(I,PSF,NOISEPOWER)`, where NOISEPOWER is the additive noise power. The default value is 0.

`J=deconvreg(I,PSF,NOISEPOWER,LRANGE)`, where LRANGE is a vector specifying range where the search for the optimal solution is performed.

`J=deconvreg(I,PSF,NOISEPOWER,LRANGE,REGOP)`, where REGOP is the regularization operator to constrain deconvolution. The default regularization operator is Laplacian operator, to retain the image smoothness.

`[J,LAGRA]=deconvreg(I,PSF,···)` outputs the value of Lagrange multiplier LAGRA in addition to the restored image `J`.

Example

```
>>I=imread('puppy_gray.jpg');
>>PSF=fspecial('gaussian',11,5);
>>blurred=imfilter(I,PSF,'conv');
>>V=0.01;
>>blurred_noise=imnoise(blurred,'gaussian',0,V);
>>NP=V*prod(size(I));
>>[reg1 LAGRA]=deconvreg(blurred_noise,PSF,NP);
```

image A

image B

image C

```
image A=puppy_gray_blurred2.jpg
image B=puppy_gray_blurred_noise2.jpg
image C=puppy_gray_blurred_noise_reg.jpg
```

Hough transform

Hough transform (HT) is designed to identify lines and curves within an image. You can find line segments and endpoints, measure angles, find circles based on size, and detect and measure circular objects.

- Read the image and display it:

  `>>A=imread('color_circles.jpg');` `>>imshow(A)`

- Determine radius range for searching circles:

  `>>d=imdistline`

  The line can be dragged to get the size of different circles. To remove the imdistline tool, use `>>delete(d);`

image A

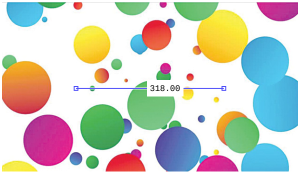


image B

Find the number of circles

Consider the image shown below. There are many circles having with different contrasts with respect to the background.

```
>>A=imread('multi_circles2.png');
```

We want to find circles using circular Hough transform.

```
>>centers=imfindcircles(A,radius)
>>[centers,radii]=imfindcircles(A,radiusRange)
>>[centers,radii,metric] =
imfindcircles(A,radiusRange)
```

Function `viscircles`

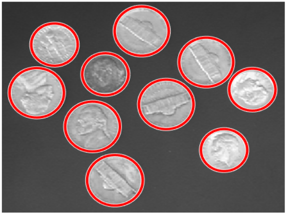
Find all the circles with radius r pixels in the range $[15, 30]$.

```
>>[centers,radii,metric]=imfindcircles(A,[15 30])
```

```
centers =

  355.0154  131.8587
  325.0334  202.4384
  145.2755   79.2986
  237.7229   63.5005
  306.0845   99.8110
  199.2026  113.8485
  209.3535  237.7137
  263.7126  149.0913
  126.5001  135.5469
  185.6071  174.9347

radii =

   24.5067
   24.5371
   24.6441
   28.6828
   28.5874
   24.4831
   29.4738
   28.8892
   28.8329
   29.0353

metric =

    0.4317
    0.3724
    0.3499
    0.3093
    0.2430
    0.2154
    0.2062
    0.1760
    0.1693
    0.1675
```

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

```
>>viscircles(centers,radii)
```



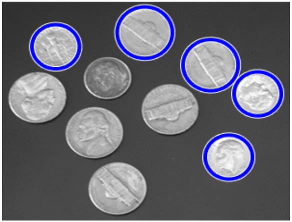
Retain the five strongest circles

Retain the five strongest circles according to the metric values.

```
>>centersStrong5=centers(1:5,:);
>>radiiStrong5=radii(1:5);
>>metricStrong5=metric(1:5);
```

Draw the five strongest circle perimeters over the original image.

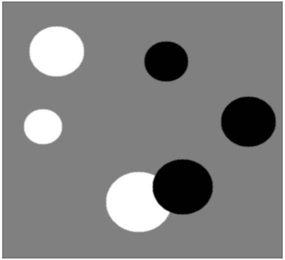
```
>>viscircles(centersStrong5,radiiStrong5,'EdgeColor','b'
```



Draw lines around bright and dark circles

Read the image into the workspace and display it:

```
>>A=imread('multi_circles3.png');
>>figure,imshow(A)
```



Define the radius range: `>>Rmin=30;Rmax=65;`

Find all the bright circles in the image within the radius range:

```
>>[centersBright,radiiBright]=imfindcircles(A,[Rmin
Rmax],'ObjectPolarity','bright');
```

Draw lines around bright and dark circles (continued)

Find all the dark circles in the image within the radius range:

```
>>[centersDark,radiiDark]=imfindcircles(A,[Rmin
Rmax],'ObjectPolarity','dark');
```

Draw blue lines around the edges of the dark circles:

```
>>viscircles(centersDark,radiiDark,'Color','b')
```