Digital Image Processing 18MIT31C

- **UNIT-V: Image Processing Tools:- Image file read/Write-Image display commands-Create image, image show, image tool, Sub image, Color bar-Image arithmetic-add, subtract, divide, multiply, complement-Spatial transformations-image rotation, image resize, cropping-Image statistics-mean, variance, standard deviation, histogram, pixel values-Image enhancement-normalized histogram-adjusting image intensity, adding/removing noise, median and order statistic filtering-contrast stretching-Linear filtering.**
- **TEXT BOOKS**

  Prathap R, “Getting started with MATLAB 7: A Quick introduction for Scientists and Engineers”, Oxford University Press, 2005.

Prepared By : Mrs. G. Shashikala, Assistant Professor, Department of Information Technology

Image Processing Tools:- Image file read/Write-Image display commands

Call special MATLAB functions to read and write image data from graphics file formats:

- To read a graphics file format image use `imread`.
- To write a graphics file format image, use `imwrite`.
- To obtain information about the nature of a graphics file format image, use `imfinfo`.

The following commands read the image `ngc6543a.jpg` into the workspace variable `RGB` and then displays the image using the `image` function:

```
RGB = imread('ngc6543a.jpg');

image(RGB)
```

You can write (save) image data using the `imwrite` function. The statements

```
load clown % An image that is included with MATLAB

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

Create image, image show, image tool, Sub image, Color bar

Read:

```
image=imread(filename);
```

Create:

```
image=zeros(300,400,3); %initialize
image(:,1:100,1)=0.5;   %Red (dark red)
image(:,101:200,1)=1;   %Red (maximum value)
image(1:100,:,2)=rand(100,400);  %Green
```

Display

```
figure, imshow(image)
```

imshow(f, [ ])' sets variable low to the minimum value of array 'f' and high to its maximum value. This helps in improving the contrast of images having a low dynamic range.

VARIOUS IMAGE TOOLS AND THEIR CAPABILITIES

Image tools	Capabilities
Pixel information	Displays information about the pixel under the mouse pointer
Pixel region	Superimposes pixel values on a zoomed-in pixel view
Measure distance	Measures the distance between two pixels
Image information	Displays information about images and image files
Adjust contrast	Adjusts the contrast of the displayed image
Crop image	Defines a crop region and crops the image
Display range	Shows the display range of the image data
Overview	Shows the currently visible image
Choose colormap	Shows the image under different colour settings

The Image tool in the image processing toolbox provides a more interactive environment for viewing and navigating within images, displaying detailed information about pixel values, measuring distances and other useful operations. To start the image tool, use the imtool function. The following statements read the image Penguins_grey.jpg saved on the desktop and then display it using 'imtool':

>>B = imread(Penguins_grey.jpg);

>>imtool(B)

`subimage(I)` displays the RGB (truecolor), grayscale, or binary image `I` in the current axes.

You can use `subimage` in conjunction with `subplot` to create figures with multiple images, even if the images have different colormaps. `subimage` converts images to RGB for display purposes, thus avoiding colormap conflicts.

`subimage(X,map)` displays the indexed image `X` with colormap `map` in the current axes.

`subimage(x,y,__)` displays an image using a nondefault spatial coordinate system, where `x` and `y` specify the image limits in the world coordinate system.

`h = subimage(__)` returns a handle to an image object.

Examples

Display Two Indexed Images in Same Figure

```
load trees
[X2,map2] = imread('forest.tif');
subplot(1,2,1), subimage(X,map)
subplot(1,2,2), subimage(X2,map2)
```

colormap

Set and get the current colormap

Syntax

- `colormap(map)`
- `colormap('default')`
- `cmap = colormap`

Description

A colormap is an *m*-by-3 matrix of real numbers between 0.0 and 1.0. Each row is an RGB vector that defines one color. The *k*th row of the colormap defines the *k*th color, where `map(k,:) = [r(k) g(k) b(k)]` specifies the intensity of red, green, and blue.

`colormap(map)` sets the colormap to the matrix `map`. If any values in `map` are outside the interval [0 1], MATLAB returns the error `Colormap must have values in [0,1]`.

`colormap('default')` sets the current colormap to the default colormap.

`cmap = colormap` retrieves the current colormap. The values returned are in the interval [0 1].

Specifying Colormaps

M-files in the `color` directory generate a number of colormaps. Each M-file accepts the colormap size as an argument. For example,

- `colormap(hsv(128))`

creates an `hsv` colormap with 128 colors. If you do not specify a size, MATLAB creates a colormap the same size as the current colormap.

colorbar

Display colorbar showing the color scale

Syntax

- `colorbar`
- `colorbar('vert')`
- `colorbar('horiz')`
- `colorbar(h)`
- `h = colorbar(...)`
- `colorbar(...,'peer',axes_handle)`

Description

The `colorbar` function displays the current colormap in the current figure and resizes the current axes to accommodate the colorbar.

`colorbar` updates the most recently created colorbar or, when the current axes does not have a colorbar, `colorbar` adds a new vertical colorbar.

`colorbar('vert')` adds a vertical colorbar to the current axes.

`colorbar('horiz')` adds a horizontal colorbar to the current axes.

`colorbar(h)` uses the axes `h` to create the colorbar. The colorbar is horizontal if the width of the axes is greater than its height, as determined by the axes `Position` property.

`h = colorbar(...)` returns a handle to the colorbar, which is an axes graphics object.

`colorbar(...,'peer',axes_handle)` creates a colorbar associated with the axes `axes_handle` instead of the current axes.

`colorbar` works with two-dimensional and three-dimensional plots.

Image arithmetic-add, subtract, divide, multiply, complement

Image Arithmetic

Add, subtract, multiply, and divide images

Image arithmetic is the implementation of standard arithmetic operations, such as addition, subtraction, multiplication, and division, on images. Image arithmetic has many uses in image processing both as a preliminary step in more complex operations and by itself. For example, image subtraction can be used to detect differences between two or more images of the same scene or object.

Functions

`imabsdiff`	Absolute difference of two images
`imadd`	Add two images or add constant to image
`imapplymatrix`	Linear combination of color channels
`imcomplement`	Complement image
`imdivide`	Divide one image into another or divide image by constant
`imlincomb`	Linear combination of images
`immultiply`	Multiply two images or multiply image by constant
`imsubtract`	Subtract one image from another or subtract constant from image

Spatial transformations-image rotation, image resize, cropping

imrotate

Rotate image

Syntax

```
J = imrotate(I,angle)
J = imrotate(I,angle,method)
J = imrotate(I,angle,method,bbox)
```

Description

`J = imrotate(I,angle)` rotates image `I` by `angle` degrees in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for `angle`. `imrotate` makes the output image `J` large enough to contain the entire rotated image. By default, `imrotate` uses nearest neighbor interpolation, setting the values of pixels in `J` that are outside the rotated image to `0`.

`J = imrotate(I,angle,method)` rotates image `I` using the interpolation method specified by `method`.

`J = imrotate(I,angle,method,bbox)` also uses the `bbox` argument to define the size of the output image. You can crop the output to the same size as the input image or return the entire rotated image.

imresize

Resize image

Syntax

```
B = imresize(A,scale)
```

Description

`B = imresize(A,scale)` returns image `B` that is `scale` times the size of `A`. The input image `A` can be a grayscale, RGB, or binary image. If `A` has more than two dimensions, `imresize` only resizes the first two dimensions. If `scale` is in the range [0, 1], `B` is smaller than `A`. If `scale` is greater than 1, `B` is larger than `A`. By default, `imresize` uses bicubic interpolation.

imcrop

Crop image

Syntax

```
Icropped = imcrop
Icropped = imcrop(I)
Xcropped = imcrop(X,cmap)
Icropped = imcrop(I,rect)
```

Description

Crop Image Interactively

`Icropped = imcrop` creates an interactive Crop Image tool associated with the grayscale, truecolor, or binary image displayed in the current figure. `imcrop` returns the cropped image, `Icropped`.

With this syntax and the other interactive syntaxes, the Crop Image tool blocks the MATLAB® command line until you complete the operation.

`Icropped = imcrop(I)` displays the grayscale, truecolor, or binary image `I` in a figure window and creates an interactive Crop Image tool associated with the image.

`Xcropped = imcrop(X,cmap)` displays the indexed image `X` in a figure using the color map `cmap`, and creates an interactive Crop Image tool associated with that image. `imcrop` returns the cropped indexed image, `Xcropped`, which also has the color map `cmap`.

Crop Image by Specifying Crop Region

`Icropped = imcrop(I,rect)` crops the image `I` according to the position and dimensions specified in the crop rectangle `rect`. The cropped image includes all pixels in the input image that are completely *or partially* enclosed by the rectangle.

The actual size of the output image does not always correspond exactly with the width and height specified by `rect`. For example, suppose `rect` is `[20 20 40 30]`, using the default spatial coordinate system. The upper left corner of the specified rectangle is the center of the pixel with spatial (x,y) coordinates (20,20). The lower right corner of the rectangle is the center of the pixel with spatial (x,y) coordinates (60,50). The resulting output image has size 31-by-41 pixels, not 30-by-40 pixels.

Image statistics-mean, variance, standard deviation

The Image Statistics block calculates the mean, variance, and standard deviation of streaming video data. Each calculation is performed over all pixels in the input region of interest (ROI).

Histogram, pixel values-Image enhancement-normalized histogram

Create Image Histogram

An image histogram is a chart that shows the distribution of intensities in an indexed or grayscale image. The `imhist` function creates a histogram plot by defining *n* equally spaced bins, each representing a range of data values, and then calculating the number of pixels within each range. You can use the information in a histogram to choose an appropriate enhancement operation. For example, if an image histogram shows that the range of intensity values is small, you can use an intensity adjustment function to spread the values across a wider range.

Read an image into the workspace and display it.

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

The `imhist` function displays the histogram, by default.

```
figure;
imhist(I);
```

adjusting image intensity

imadjust

Adjust image intensity values or color map

Syntax

```
J = imadjust(I)
J = imadjust(I,[low_in high_in])
J = imadjust(I,[low_in high_in],[low_out high_out])
```

Description

`J = imadjust(I)` maps the intensity values in grayscale image `I` to new values in `J`. By default, `imadjust` saturates the bottom 1% and the top 1% of all pixel values. This operation increases the contrast of the output image `J`.

`J = imadjust(I,[low_in high_in])` maps intensity values in `I` to new values in `J` such that values between `low_in` and `high_in` map to values between 0 and 1.

`J = imadjust(I,[low_in high_in],[low_out high_out])` maps intensity values in `I` to new values in `J` such that values between `low_in` and `high_in` map to values between `low_out` and `high_out`.

adding/removing noise, median and order statistic filtering

imnoise

Add noise to image

Syntax

```
J = imnoise(I,'gaussian')
J = imnoise(I,'gaussian',m)
J = imnoise(I,'poisson')
J = imnoise(I,'salt & pepper')
J = imnoise(I,'salt & pepper',d)
J = imnoise(I,'speckle')
J = imnoise(I,'speckle',var_speckle)
```

Description

`J = imnoise(I,'gaussian')` adds zero-mean, Gaussian white noise with variance of 0.01 to grayscale image `I`.

`J = imnoise(I,'gaussian',m)` adds Gaussian white noise with mean `m` and variance of 0.01.

`J = imnoise(I,'poisson')` generates Poisson noise from the data instead of adding artificial noise to the data.

`J = imnoise(I,'salt & pepper')` adds salt and pepper noise, with default noise density 0.05. This affects approximately 5% of pixels.

`J = imnoise(I,'salt & pepper',d)` adds salt and pepper noise, where `d` is the noise density.

Filter the noisy image, `J`, with an averaging filter and display the results. The example uses a 3-by-3 neighborhood.

```
Kaverage = filter2(fspecial('average',3),J)/255;
figure
imshow(Kaverage)
```

Now use a median filter to filter the noisy image, `J`. The example also uses a 3-by-3 neighborhood. Display the two filtered images side-by-side for comparison. Notice that `medfilt2` does a better job of removing noise, with less blurring of edges of the coins.

```
Kmedian = medfilt2(J);
imshowpair(Kaverage,Kmedian,'montage')
```

contrast stretching-Linear filtering.

Contrast stretching is a simple image enhancement technique that attempts to improve the contrast in an image by `stretching' the range of intensity values it contains to span a desired range of values, e.g. the full range of pixel values that the image type concerned allows.

Filtering of images, either by correlation or convolution, can be performed using the toolbox function `imfilter`. This example filters an image with a 5-by-5 filter containing equal weights. Such a filter is often called an *averaging filter*.

- `I = imread('coins.png');`
- `h = ones(5,5) / 25;`
- `I2 = imfilter(I,h);`
- `imshow(I), title('Original Image');`
- `figure, imshow(I2), title('Filtered Image')`