Image Processing

Fundamentals of Image Processing

Pattern Recognition and Image Processing Laboratory (Since 2012)

Fundamentals

Digital Image Representation: Coordinate Convention

(0,0)	(0,1)	...	(0,N-1)
(1,0)	(1,1)		
:			
(M-1,0)			(M-1,N-1)

(1,1)	(1,2)	...	(1,N)
(2,1)	(2,2)		
:			
(M,1)			(M,N)

Fundamentals

Digital Image Representation: Images as Matrices

$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,0) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Fundamentals

Reading Images

imread(‘filename’)

```
>> f = imread( ‘chestxray.jpg’ );
>> f = imread( ‘D:\myimages\chestxray.jpg’ );
>> f = imread( ‘.\myimages\chestxray.jpg’ );
```

Fundamentals

Displaying Images

imshow(f, G)

where G is the numbers of intensity levels.

```
>> imshow(f, [low high])
>> imshow(f, [ ])
```

Fundamentals

Displaying Images

```
>> f = imread( ‘chestxray.tif’ );
>> imshow(f)
>> figure, imshow(f, [ ])
```

Fundamentals

Writing Images

imwrite(f, ‘filename.tif’)

>> imwrite(f, ‘patient10.tif’);

Fundamentals

Writing Images

imwrite(f, ‘filename.jpg’, ‘quality’, q)

```
>> f = imread( ‘bubbles.tif’ );
>> imwrite(f,  ‘bubbles50.jpg’ ,  ‘quality’ , 50);
>> imwrite(f,  ‘bubbles25.jpg’ ,  ‘quality’ , 25);
>> imwrite(f,  ‘bubbles15.jpg’ ,  ‘quality’ , 15);
>> imwrite(f,  ‘bubbles5.jpg’ ,  ‘quality’ , 5);
>> imwrite(f,  ‘bubbles0.jpg’ ,  ‘quality’ , 0);
```

Fundamentals

Writing Images

```
>> f50 = imread( ‘bubbles50.jpg’ );
>> f25 = imread( ‘bubbles25.jpg’ );
>> f15 = imread( ‘bubbles15.jpg’ );
>> f5 = imread( ‘bubbles5.jpg’ );
>> f0 = imread( ‘bubbles0.jpg’ );
>>
>> figure, imshow(f50), figure, imshow(f25);
>> figure, imshow(f15), figure, imshow(f5);
>> figure, imshow(f0)
```

Fundamentals

Writing Images

Imfinfo filename

```
>> imfinfo bubbles25.jpg
```

Fundamentals

Compression Ratio

```
>> K = imfinfo( ‘bubbles25.jpg’ );
>> image_bytes = K.Width*K.Height*K.BitDepth/8;
>> compressed_bytes = K.FileSize;
>> compression_ratio = image_bytes/compressed_bytes
```

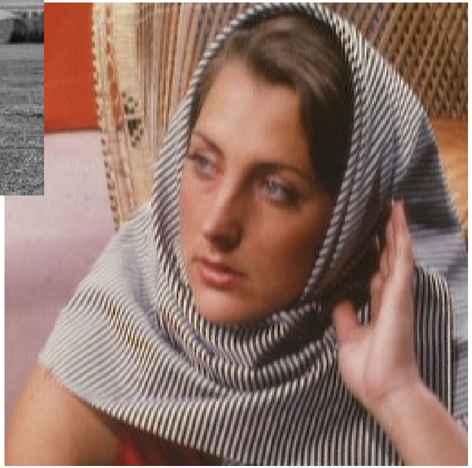
Fundamentals

Data Classes

double	-10^308 to 10^308	(8 bytes per element)
uint8	[0, 255]	(1 byte per element)
uint16	[0, 65535]	(2 bytes per element)
uint32	[0, 4294967295]	(4 bytes per element)
int8	[-128, 127]	(1 byte per element)
int16	[-32768, 32767]	(2 bytes per element)
int32	[-2147483648, 2147483647]	(4 bytes per element)
single	-10^38 to 10^38	(4 bytes per element)
char	characters	(2 bytes per element)
logical	values are 0 or 1	(1 byte per element)

Fundamentals

Data Classes



- Binary image
- Intensity image
- RGB image

Fundamentals

Converting between Data Classes and Image Types

B = data_class_name(A)

Name	Converts Input to:
im2uint8	uint8
im2uint16	uint16
mat2gray	double (in range [0,1])
im2double	double
im2bw	logical

Fundamentals

Converting between Image Classes and Types

```
>> f = [-0.5 0.5; 0.75 1.5]
>> g = im2uint8(f)
>> g1 = mat2gray(f)  % mat2gray(f, [fmin, fmax])
>> h = uint8([25 50; 128 200])
>> g2 = im2double(h)
>> f = [1 2; 3 4]
>> g = mat2gray(f)
>> gb = im2bw(g, 0.6)
>> gb1 = f > 2
>> gbv = islogical(gb)
```

Fundamentals

Converting between Image Classes and Types

```
>> gbd = im2double(gb)
>> gbd1 = im2double(im2bw(mat2gray(f), 0.6))
>> gbd2 = double(f >2)
```

Fundamentals

Array Indexing

```
>> v = [1 3 5 7 9]
>> v(2)
>> w = v.'
>> v(1:3)
>> v(2:4)
>> v(3:end)
```

Fundamentals

Array Indexing

```
>> v(:)
>> v(1:end)
>> v(1:2:end)
>> v(end:-2:1)
>> v([1 4 5])
```

Fundamentals

Matrix Indexing

```
>> A = [1 2 3; 4 5 6; 7 8 9]
>> A(2, 3)
>> C3 = A( :, 3)
>> R2 = A(2, :)
>> T2 = A(1:2, 1:3)
>> B = A; B( :, 3) = 0
>> A(end, end)
```

Fundamentals

Matrix Indexing

```
>> A(end, end - 2)
>> A(2:end, end:-2:1)
>> E = A([1 3], [2 3])
>> D = logical([1 0 0; 0 0 1; 0 0 0])
>> A(D)
>> v = T2(:)
>> s = sum(A(:))
```

Fundamentals

Matrix Indexing

```
>> f = imread( ‘lena.bmp’ );
>> fp = f(end:-1:1, :);
>> figure, imshow(fp)
>> fc = f(100:180, 100:180);
>> figure, imshow(fc)
>> fs = f(1:2:end, 1:2:end);
>> figure, imshow(fs)
>> plot(f(128, :) )
```

Fundamentals

Selecting Array Dimensions

```
>> A = [1 2 3; 4 5 6; 7 8 9]
>> k = size(A, 1)
>> d = ndims(A)
```

Fundamentals

M-Function Programming

```
function [p, pmax, pmin, pn] = improd(f, g)
% IMPROD computes the product of two images
%   [P, PMAX, PMIN, PN] = IMPROD(F,G) outputs the
%   element-by-element product of two images, F and G,
%   the product maximum and minimum values, and
%   a normalized product array with values in the range
%   [0, 1]. The input images must be of the same size.
%   They can be of class uint8, uint16, or double. The
%   outputs are of class double.

fd = double(f);
gd = double(g);
p = fd.*gd;
pmax = max(p(:));
pmin = min(p(:));
pn = mat2gray(p);

%   the end
```

Function definition line

Help text

H1 line

Function body

Comments

Fundamentals

M-Function Programming

```
function [p, pmax, pmin, pn] = improd(f, g)
% IMPROD computes the product of two images
%   [P, PMAX, PMIN, PN] = IMPROD(F,G) outputs the
%   element-by-element product of two images, F and G,
%   the product maximum and minimum values, and
%   a normalized product array with values in the range
%   [0, 1]. The input images must be of the same size.
%   They can be of class uint8, uint16, or double. The
%   outputs are of class double.

fd = double(f);
gd = double(g);
p = fd.*gd;
pmax = max(p(:));
pmin = min(p(:));
pn = mat2gray(p);

%   the end
```

Fundamentals

M-Function Programming

```
>> help improd
>> f = [1 2; 3 4]
>> g = [1 2; 2 1]
>> [p, pmax, pmin, pn] = improd(f, g)
```

Fundamentals

Code Optimization: Vectorizing Loops

```
for x1 = 1:10
        f1(x1) = 2*sin((x1-1)/(2*pi));
end
```

```
x = 0:10-1;
f = 2*sin(x/(2*pi));
```

The End