Politecnico di Milano
Master of Science program in Biomedical Engineering

# Biomedical Image Processing Lab class
(5 credits)

***Class II***

***March 23th 2012***

Enrico Caiani, PhD

## *Today's topics*

- *Multiframe arrays and movies*
- *Image Enhancement (1)*

*Caiani 23/03/2012*

## *WRITING A FUNCTION*

Functions accept arguments and produce one or more outputs.

Their structure is the following:

| | |
|---|---|
| - Function definition line | **function** [outputs]=name(inputs) |
| - Single comment line | % NAME describe here what it does |
| - Function body | commands |

```
function [s,d]=sumdif(a,b)
%SUMDIF compute sum and difference between a and b
s=a+b;
d=a-b;
```

*Test this function, then modify it in order to receive 4 numbers, and make sum and difference of them.*

*Caiani 23/03/2012*

## *FUNCTION HANDLE*

It is a data type that contains information used in referencing a function.

Advantages: it is possible to pass a function handle as argument in a call to an other function.

Named function handle:

```
>>f=@sin
>>f(pi/4)
```

Anonymous function handle: @(input argument list) expression

```
>>g=@x x.^2
>>g(3)
```

*Define a function handle that computes the Pitagora theorem, given two inputs.*

*Caiani 23/03/2012*

## AGAIN ON CELL ARRAYS…

As previously observed (see **inputdlg**), cell arrays provide a way to combine a mixed set of objects (numbers, char, matrices, other arrays,...) under one variable name.

A cell array can be created organizing the dissimilar entities into a single variable using { }:

```
>>a=zeros(512,512,'uint8');
>> b=[1 2;3 4;5 6];
>> c='pippo';
>>char_array={'area',100}
>>C={a,b,c,char_array}
```

*Try this example and explore the cell array with the different commands*

To access the single element of the cell array, we enclose the numerical location of the element in curly braces:

```
>>C{3}
```

To display the content, we can use celldisp: >>**celldisp**(C{4})
or **cellplot**: >>**cellplot**(C)

We can apply a command to every cell of the cell array using **cellfun**:
D=**cellfun**(‘lenght’,C)

*Caiani 23/03/2012*

*Write a function that has as input an image, and outputs its average intensity, its dimensions, the average intensity of its rows and columns, all as a content of a single cell array.*

## AGAIN ON STRUCT ARRAYS…

As previously observed (see **dicominfo**), struct arrays are similar to cell arrays as they allow grouping dissimilar data into a single variable. However, unlike cell arrays, the elements of the structures are addressed by user defined fields. In this way, the organization of the output data can be much clearer.

*Repeat the previous exercise assigning the output to a sigle struct array.*

*Caiani 23/03/2012*

## *Other useful command*

An image is a matrix of numbers, but not every matrix of numbers is interpreted by Matlab as an image...

To convert a matrix to an image:

I = **mat2gray**(A,[AMIN AMAX])     output between 0 and 1

```
>>R=randn(10,10);
>>I=mat2gray(R);
>>imshow(I)
```

To visualize an image with additional features:

**imtool**(I)     (only latest versions)

*Caiani 23/03/2012*

## *MULTIFRAME IMAGE ARRAYS*

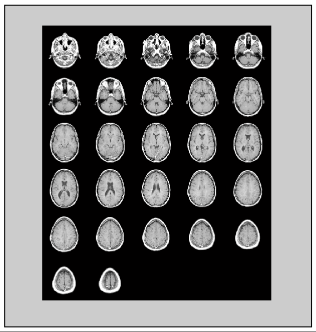
As previously observed, in several applications, it is useful to operate on a sequence of temporal (ecocardiographic loops) o spatial (MRI, CT, PET) images. It is possible to store these frames in a single array, linking them through the 4-dimension (the 3rd is 1 for intensity, 3 for RGB):

>>A=***cat***(4,A1,A2,A3,A4,A5)

>>A3=A(:,:,:,3)

When a multiframe image array has been created, the following commands can be applied:

>>**montage**(D,map) : visualize all frames at the same time (**N.B.**: each frame must have the same dimensions and, if indexed, utilize the same colormap)



*Caiani 23/03/2012*

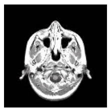
\>>mov=**immovie**(D,map) : creates the movie mov as temporal sequence of the images in D (**N.B.**: images must be indexed or RGB).

To visualize the created movie, use the command **movie**(X,n° rip,fps) or **implay**(X,fps):

>>**movie**(mov,3,6)

It is also possible to transform the movie in .avi format using **movie2avi**:

>>**movie2avi**(mov,'a:\filmatomri',parameters)



Parameters:
'FPS': default 15 fps; 'COMPRESSION':default 'Indeo3';
'QUALITY': active if there is compression; it ranges from 0 (low qualità) to 100 (high quality) – default 75;
'COLORMAP': name of the colormap to be applied in the generation of the .avi file (max 236 colors if Indeo compression is used)

*Load"mri.mat", containing images of MRI and visualize the frames one by one, using a for cycle with **pause**. Try also the utilization of the **montage** command. Then create a movie, visualizing it 2 times at 15 fps.*

*Caiani 23/03/2012*

\>>info=**aviinfo**('nome'): it provides information on the .avi file (dimensions, frame number, frame rate, colorspace, codec, compression rate).

To load a .avi file:

\>>mov=**aviread** ('nome');

mov will contain a struct array with two fields:

***cdata*** and ***colormap***

If the frames are RGB, cdata will be height x width x 3, and colormap empty;

If the frames are indexed, cdata will be height x width, with colormap not empty.

To access the different fields into the struct array:

A=mov(1).***cdata***;

*Load "pig.avi" and test the relevant commands.*
*Move all the frames from mov to a 3D array, then visualize with montage its content.*

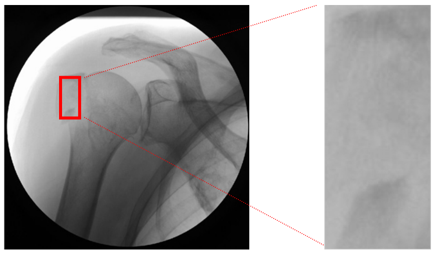
*Caiani 23/03/2012*

## ***Useful commands for raw image analysis***

Given an image, it can be useful to extract a specific region of interest (ROI) on which to apply specific operations. This is possible using the command ***I=imcrop***, that activates the cursor in order to select the ROI on the active figure. Also use the following, to memorize the ROI origins:

```
>>[C,rect]=imcrop(X);
```

*Load an image, extract some ROIs, and visualize them using the different visualization commands (imshow, imview, imtool).*



*Caiani 23/03/2012*

Matlab gives the opportunity to obtain infos on the pixel values in the image (other than using imtool and imview) :

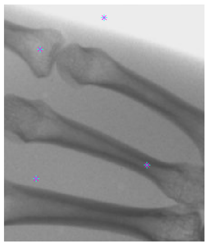
• using **impixel**, it is possible to get the R, G, B value of the selected pixels selezionati, or by mouse or by giving the spatial coordinates:

```
>>P=impixel;
```

The cursor is activated, left click to select, right click to end.

```
>>P =

  236  236  236
  165  165  165
   95   95   95
  144  144  144
```



```
>>P = impixel(I,c,r);
```

where I is the array to be analyzed, (c,r) are arrays with the pixel coordinates to be evaluated.

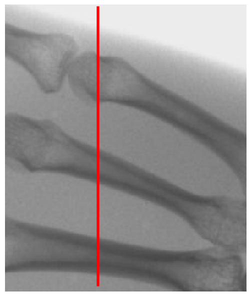
*Caiani 23/03/2012*

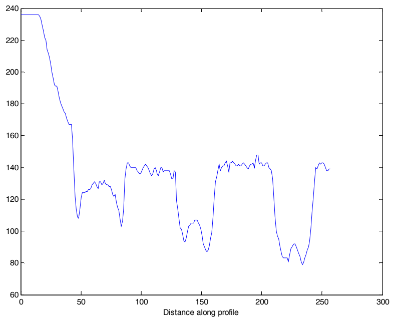
• Using ***pixval***, in the figure a window is activated, giving pixel positoin and corresponding videointensity:

>>**pixval**(‘ON’); [use **impixelinfo** in more recent Matlab versions]

• Using ***improfile***, the videointensity along the selected profile is shown:

>>P=**improfile**;





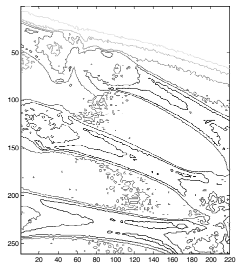
*Caiani 23/03/2012*

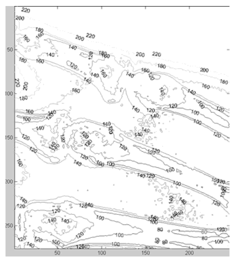
• Using ***imcontour*** it is possible to visualize the isocontour levels in an intensity image:

>>**imcontour**(I);

>> [C,H]=**imcontour**(I);

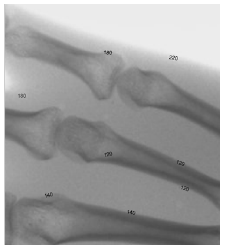
>>**clabel**(C,H)





>>**clabel**(C,H,’manual’)

With **clabel** the values of the isocontour lines are visualized in the original image.



*Load an image (repeat for an intensity and a RGB) and apply these commands, to test their performance.*

*Caiani 23/03/2012*

## Image histogram

•The histogram of a image with L possible levels in the range [0,G] is defined as the discrete function:

$h(r_k)=n_k$

Where $r_k$ is the k-intensity level in the intensity [0,G] and $n_k$ is the number of pixels in the image whose level is $r_k$.

If it is necessary to work with normalized histograms, divide by the total number of pixels in the image, n:

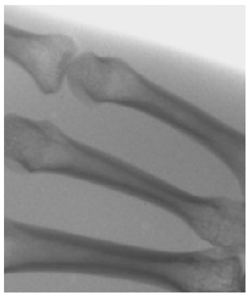
$p(r_k) = h(r_k)/n = n_k/n$ → this represents an estimate of the probability of occurrence of intensity level $r_k$ in the image.

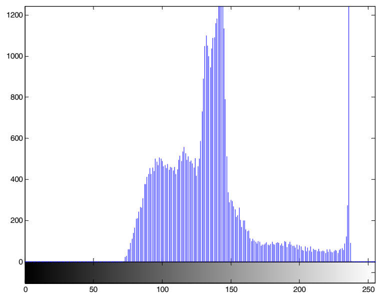
*Caiani 23/03/2012*

## Image histogram in Matlab

•The command ***imhist***(I,N) gives the histogram distribution of the videointensity levels in the image I, indexed or intensity. The available levels are divided into N equispaced bins, and the number of pixels for each bin is computed and visualized as a bar. By default, N is equal to 256.

>>**imhist**(I)



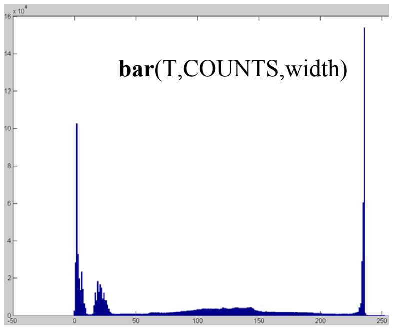


**imhist**(I,N)/**numel**(I) gives the normalized histogram.

>>[COUNTS,T]=**imhist**(X);

The number of pixels T associated to each videointensity COUNTS is memorized

*Caiani 23/03/2012*



**bar**(T,COUNTS,width)

It is possible to plot an histogram also using other commands, like **bar**.

When plotting histograms with **bar**, it can be useful to divide the horizontal axis into bands:

>>h=**imhist**(I,25);

>>horz=**linspace**(0,255,25);

>>**bar**(horz,h)

>>**axis**([0 255 0 60000]);

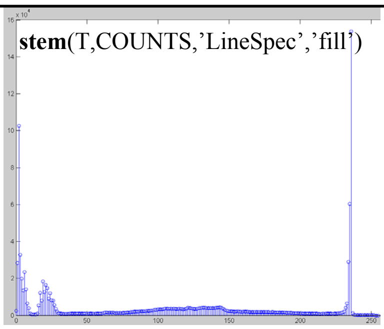
>>**set**(gca,'xtick',0:50:255)

>>**set**(gca,'ytick',0:20000:60000)

To define axis limits of the figure:
**axis**([hormin hormax vermin vermax])
**axis** tight sets the limit to the data range

**xlabel**('text string','fontsize',size)
**ylabel**('text string','fontsize',size)
**text**(xloc,yloc,'text string','fontsize',size)
**title**('titlestring')

*Caiani 23/03/2012*



**stem**(T,COUNTS,'LineSpec','fill')

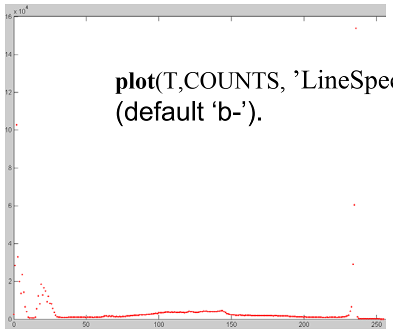
LineSpec is a triplet of values from the table (default 'b-o').

Attributes for **stem** and **plot**

| Symbol | Color | Symbol | Line Style | Symbol | Marker |
|---|---|---|---|---|---|
| k | Black | – | Solid | + | Plus sign |
| w | White | -- | Dashed | o | Circle |
| r | Red | : | Dotted | * | Asterisk |
| g | Green | -. | Dash-dot | . | Point |
| b | Blue | none | No line | x | Cross |
| c | Cyan | | | s | Square |
| y | Yellow | | | d | Diamond |
| m | Magenta | | | none | No marker |

To set the limits and ticks automatically, use:
**ylim**('auto') **xlim**('auto')

or
**ylim**([ymin ymax])
**xlim**([xmin xmax])



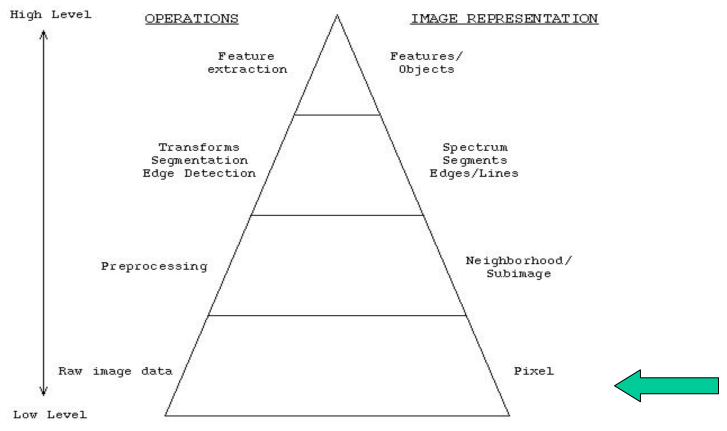
**plot**(T,COUNTS, 'LineSpec')
(default 'b-').

*Visualize the histogram using all the possible commands, of both an entire image and a selected ROI.*

*Caiani 23/03/2012*

## Image Analysis Pyramid

- Hierarchical image pyramid: Consists of levels for processing of images



Caiani 23/03/2012

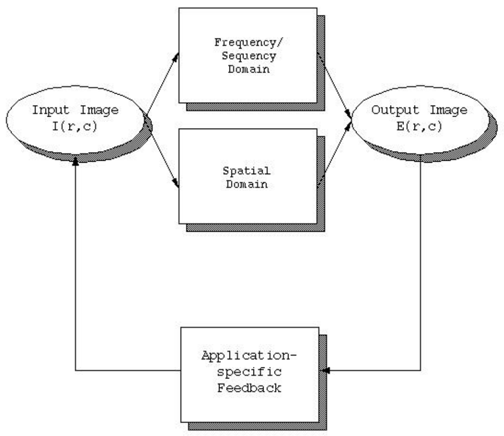
## Introduction

- Image enhancement techniques are employed to emphasize, sharpen and/or smooth image features for display and analysis
- Image enhancement is the process of applying these techniques to facilitate the development of a solution to a computer imaging problem
- Enhancement methods are **application-specific** and often developed **empirically**

Caiani 23/03/2012

## Introduction and Overview

We define the enhanced image as *E(r,c)*, and application feedback is an important part of the process.
Enhancement methods operate in both the spatial and frequency/spectral domains.



Caiani 23/03/2012

## Enhancement techniques

- Point operations : Each pixel is modified by an equation that is not dependent on other pixel values
- Mask operations : Each pixel is modified according to the values in a small neighborhood (subimage)
- Global operations : All pixel values in the image are taken into consideration
- Spatial domain processing methods include all three types, but frequency domain operations are global operations
- Frequency domain operations can become “mask operations”, by performing the transform on small image blocks instead of the entire image.

Caiani 23/03/2012

## Gray Scale Modification by Mapping Equation

- Also called gray level scaling or gray level modification
- Process of taking the original gray level values and changing them to improve the image
- Typically performed to improve image contrast (measure of the distribution and range of the gray levels) and/or image brightness (overall average, or mean, pixel value in the image)

*Caiani 23/03/2012*

## Mapping Equations

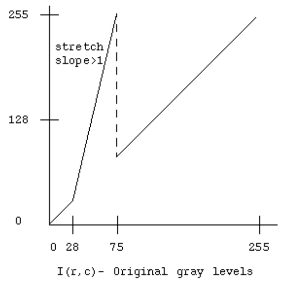
- Changes the pixel's (gray level) values based on a mathematical function
  - Uses brightness values as input
  - Outputs the enhanced pixel values
- Typically, but not necessarily, linear. Nonlinear equations can be mapped by piecewise linear models
- The use of mapping equations to modify the gray scale belong in the category of **point operations**
- Applications include contrast enhancement and feature enhancement

*Caiani 23/03/2012*

## Mapping Equations

- Primary operations applied to the gray scale of an image are to compress or stretch it
- If the slope of the line is between zero and one, it is called gray level compression, while if the slope is greater than one it is called gray level stretching

b) Original image



a) Gray level stretching

c) Image after modification

Caiani 23/03/2012

## Example

- For the ranges 0 to 28 and 75 to 255 the input equals the output.
- For the range 28 to 75, we want to stretch the range from 28 to 255.
- To do this we need a linear equation.
- Standard form for lines $y = mx + b$
  - $m$ is the slope
  - $b$ is the $y$-intercept
- Find the equation to map the input ranges to output ranges
  - $y$ corresponds to the mapping equation *M[ ],*
  - $x$ is the input image gray level values *I(r,c)* )

Caiani 23/03/2012

## Example: solution

1. We know two points on the line, (28,28) and (75,255)

$$m = \frac{y_1 - y_2}{x_1 - x_2} = \frac{255-28}{75-28} = \frac{227}{47} \approx 4.83$$

2. Solve for the intercept:

$$y = 4.83x + b$$

*Putting in a point to solve for the y - intercept, b :*

$$255 = 4.83(75) + b$$

$$b = -107.25$$

3) So the equation of the line for the range between 28 and 75 is:

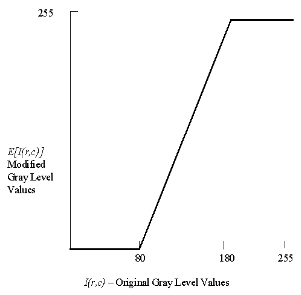
$$M[I(r,c)] = 4.83[I(r,c)] - 107.25$$

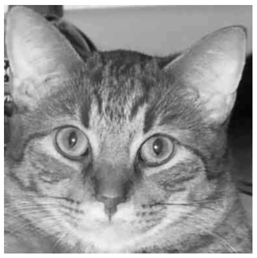
*Caiani 23/03/2012*

## Gray-level Stretching (Clipping at Both Ends)

- It is possible stretch a specific range of gray levels, while clipping the values at the low and high ends

Original image





Modified image

*Caiani 23/03/2012*