Introduction to the manipulation of visual signals with MATLAB

Objectives: This guide aims to show some of the functions of Matlab for manipulating audiovisual signals. It is an introduction to the second lab assignment (the actual work to be performed), so that the student becomes familiar with Matlab to manipulate visual signals.

Introduction

MatLab has a number of specific commands for manipulating images and for performing various image processing techniques. MatLab is particularly suited to image manipulation and processing, since its basic data structure is the matrix, and an image is nothing more than a matrix. An image is read in MatLab and stored as a matrix of type uint8 (integer value represented with 8 bits, therefore 2^8 = 256 possible values).

In order to be able to use the values of this matrix in precision number operations, as in addition, subtraction, multiplication and division, it is necessary to convert this matrix to the double type (real values of double precision). Before trying to view the result, you must convert it back to integer values (uint8).

Note that the direction of the x and y axes used in Matlab for the matrices that contain the imported images, are different from those used in Analytical Geometry. In Matlab and in general in Image Processing, the notation (x, y) must be understood as (row, column). The x for "walking" is increased vertically and the y for "walking" horizontally. It should also be noted that the indices start at 1.

Some built-in Matlab functions to import, export and retrieve image information:

`imread` - reads/imports an image file

`imwrite` - stores an image into a file

`imshow` - presents an image in a window

`imageinfo` - obtains information about an image saved in a file and creates a structure that stores that information

`imattributes` - list the attributes of the image being visualised

`image` - visualizes any matrix as if it was an image; the value of each element of the matrix is used as the color value of the spatially corresponding image element.

`im2frame` e `frame2im` - allow you to convert from image format to film format used by MATLAB and vice-versa.

Usage examples:

```
>>im = imread ('strawberries.jpg'); % reads the "strawberries" image in JPEG format for the im matrix (MxNx3)

>> [line, col] = size (im (:,:, 1)); % reads the number of rows and columns of the image that was stored in the

% im matrix and stores these values in the variables lin and col (any variables).

% Note1: the matrix im has 3 dimensions% since the image that was read is a color image.

% Note2: the command [line, col] = size (im) would give the same result.

>> imshow (im); % visualise the image stored in matrix im

>> im2 = double (im); % converts the image in the im matrix to real type of double precision and saves

% in the new matrix im2

>> im2 = im2 + 10; % increases the brightness of the image in the im2 matrix

>> im2 = uint8 (im2); % converts the image back to the int type

>> imwrite (im2, 'nome.bmp'); % saves the new image in the file "nome.bmp" with bitmap format

>> imwrite (im2, 'name2.jpg'); % saves the new image in the file "nome2.jpg" with the jpeg format
```

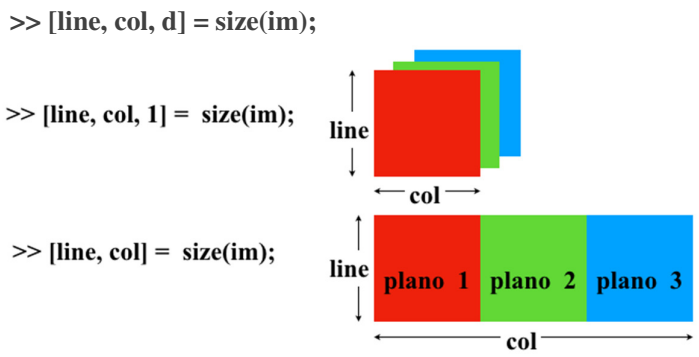
Images with the bitmap format (.bmp)

For images stored in the "bmp" format (24-bit bitmap), 3-dimensional arrays are created, in which the third dimension, depth, consists of plans to store each of the fundamental color components R, G, B. Each plane contains an MxN matrix (number of rows times number of columns) that corresponds to the matrix of one of the RGB color components (exactly in that order: R ≡ plane 1; G ≡ plane 2; B ≡ plane 3). Whenever we use the name of the matrix that was created when we import the bmp image, Matlab uses the 3 plans / components simultaneously. But we can individually access the values of each component, indicating the corresponding plan. We can use the following code:

```
% reads a 24-bit RGB image, that is, an image where each element is represented by 24 bits, 8 bits for

% the color red R, 8 bits for the color green (G) and 8 bits for the color blue (B):

>> im = imread ('peppers.bmp');

 % verifies the dimension of the matrix (to verify that it is a matrix of n pixels per m columns with depth 3,

% this is a matrix with 3 dimensions. Depth 3 indicates that in practice there is a matrix for each of the
```

```
% primary RGB colors
>> size (im)
% assigns all rows and columns of the first bi-dimensional matrix (first
plan) to variable r, those of the
% second plan to variable g and the third plan to variable b
>> r = im (:,:, 1);
>> g = im (:,:, 2);
>> b = im (:,:, 3);
% obtain the dimensions of each of the bi-dimensional matrices/plans
>> [line, col, plano] = size (im);
```

Note: The command line "`>> [line, col, 1] = size (im);`" inspects the dimensions of plane 1, that is, of the matrix where the R values of the image were stored, returning in line the number of lines or pixels vertically and in col the number of pixels horizontally. Note: the command `[line, col] = size (im)` would give the same result in line, but would give a value 3 times greater in col, because it would present the number of columns of the entire structure (composed of three two-dimensional matrices).




```
% create a new black image (matrix of zeros) with 24 bits per element,
using the dimensions of the
% image read:
>> im2 = uint8(zeros(line, col, plano)); % criada uma imagem só de zeros
% Each plan in this new matrix receives one of the planes r, g or b from
the other image, incremented and% decremented by a given value
>> im2(:, :, 1) = r+50;
>> im2(:, :, 2) = g-50;
>> im2(:, :, 3) = b+100;
% view and compare the two images that are stored in the im and im2
matrices:
```

```
>>figure; imshow(im);
>>figure; imshow(im2);
```

Some functions for changing the image representations, pixel values and for filtering:

`rgb2gray` - 24-bit conversion to grayscale

`im2bw` - conversion to black and white

`rgb2ind` - 24-bit to 256-color conversion

`rgb2hsv` - conversion between RGB and HSV color spaces

`rgb2ycbcr` - conversion between RGB and YCbCr color spaces

`imresize` - change the dimensions of an image

`imrotate` - rotates an image from a certain angle

`imhist` - calculates the histogram of an image (recording the number of times each value occurs in an image)

`filter2`, `imfilter` - apply a filter to an image and return an array of real values with the filtered image

```
>> B = imresize (A, M, 'method ')
```

The "`imresize`" function above returns an array that is M times larger (or smaller) than image A, where 'method' can be one of:

`nearest` = nearest neighbor

`bilinear` = bilinear interpolation

`bicubic` = bicubic interpolation

Identifies the approach that will be used to obtain the pixel values of the resized image.

```
>> B = imrotate (A, angle, 'method ');
```

The "`imrotate`" function above returns a matrix that is a rotated angle version of image A, where 'method' can be one of [nearest, bilinear, bicubic].

Examples:

```
>> A = imread ('fruit.bmp');
>> figure (1); imshow (A); title ('original image ');
>> B = imresize (A, 0.5, 'nearest');
>> figure (2); imshow (B); title ('resized image ');
>> B = imrotate (A, 45, 'nearest');
>> figure (3); imshow (B); title ('rotated image ');
```

```
% filter an image; after filtering, it is necessary to round and convert to uint8 to be able to
% view the filtered image; the fspecial built-in script is used to create the desired type of filter; at the
% example below, three types of filters are created: medium, motion and outline enhancement
>> im = imread('nenufares.bmp');
>> im = rgb2gray(im);
>> subplot(2,2,1);imshow(im);title('Original image in a gray scale');
>> h = fspecial ('average', 5);
>> im2 = uint8(round(filter2(h, im)));
>> subplot(2,2,2);imshow(im2);title('Image with median filter');
>> h = fspecial('motion',20,45);
>> im3 = imfilter(im,h,'replicate');
>> subplot(2,2,3);imshow(im3);title('Image with motion filter');
>> h=fspecial('unsharp');
>> im4 = imfilter(im,h,'replicate');
>> subplot(2,2,4);imshow(im4);title('Sharper image');
```

In the instruction set above, the instruction 'subplot' was used. It allows you to create a grid for orderly viewing of several images on the same screen. The first argument indicates the number of lines in that grid; the second argument, the number of columns; the third argument indicates where we want the image to be displayed. A 2 x 2 grid, has 4 positions for displaying images or graphics. The first position corresponds to row 1, column 1; the second position to row 1 column 2; the third position to row 2, column 1; the fourth position corresponds to row 2, column 2.

We can create a matrix with random values and present it as an image, using the built-in 'image' function; for example, like this:

```
>> c=round(255*rand(255));
>> image(c);
```

We can convert a color image to a black and white image, gray level or indexed color image with the built-in '`image`', '`rgb2gray`', '`rgb2ind`' functions; for example, like this:

```
>> im=imread('lena.bmp');
>> bw=im2bw(im, 0.5);
>> imshow(im);
>> grayLena=rgb2gray(im);
>> imshow(grayLena);
>> [indLena,mapa]=rgb2ind(im,256);
```

We can create a sub-image from an image already imported and just viewed:

```
>>im=imread('bird-blue.jpg');
>>imshow(im);
>>subImagem=imcrop;
% define with the mouse the desired area, on top of the original image; end with crop image
>>imshow(subImagem);
```

When performing arithmetic operations on images, it should be noted that in any image, pixels can assume values on a scale of 256 (gray tones in a colorless image, color tones in a Gif image, or intensity of each of the components. RGB of a bitmap image). Thus, the addition of two such images, for example, can result in a new image with pixels greater than 255. Or the subtraction results in values less than 0. To overcome these problems, there are basically two alternatives: (1) normalization of values and 2) zeroing of values. In the first case, calculations are performed and at the end, all values obtained are normalized (expression shown below). In the second case, as the calculations are made, all values greater than 255 are converted to 255 and all negatives are converted to 0. The second alternative is simpler than the first but can lead to worse results from the point perceptual view.

$$n = \frac{255}{f_{Max} - f_{min}} * (f - f_{min})$$

```
>> x = [125 125 150 155 ; 130 130 160 165 ; 135 135 165 175 ; 140 145 170 180 ]
>> y = 2*x;
>> z = (255/(max(max(z))-min(min(z))))*(z-min(min(z)));
```

One of the operations that is often performed with images is convolution, which consists of a set of successive multiplications, additions and displacements between two matrices, one of them being the image and the other an operator that you want to apply to the image. The matrix containing the operator is usually much smaller than the image. A similar operation is correlation.

```
>> i = imread('image.tif');
>> op = [0.25 0 -0.25 ; 0.5 0 -0.5 ; 0.25 0 -0.25]
>> res = conv2(i,op);
>> imshow(res);
```