Iniciação à manipulação de sinais visuais com o MATLAB

Objectivos: Este guião pretende dar a conhecer algumas das funções do Matlab para manipulação de sinais audiovisuais. É uma introdução ao trabalho Lab2 a realizar, para que o estudante se familiarize com o Matlab para manipular sinais visuais.

Introdução

O MatLab possui diversos comandos específicos para manipular imagens e para executar diversas técnicas de processamento de imagem. O MatLab adequa-se particularmente à manipulação e processamento de imagens, uma vez que a sua estrutura básica de dados é a matriz e uma imagem não é mais do que uma matriz. Uma imagem é lida no MatLab e armazenada como uma matriz do tipo uint8 (valor inteiro representado com 8 bits, logo com 2^8=256 valores possíveis).

Para se poder utilizar os valores dessa matriz em operações de números com precisão, tal como na adição, subtração, multiplicação e divisão, é necessário efectuar a conversão dessa matriz para o tipo double (valores reais de precisão dupla). Antes de tentar visualizar o resultado, deve-se converter de novo para valores inteiros (uint8).

De notar que a direção dos eixos x e y usado no Matlab para as matrizes que contêm as imagens importadas, são diferentes das utilizadas em Geometria Analítica. No Matlab e em geral em Processamento de Imagem, a notação (x, y) deve ser entendida como (linha, coluna). Incrementa-se o x para "andar" na vertical e o y para "andar" na horizontal. De salientar ainda que os índices começam em 1.

Agumas funções built-in do Matlab para importar, exportar e obter informação de imagens:

`imread` - lê ficheiro de imagem

`imwrite` - grava imagem num ficheiro

`imshow` - apresenta uma imagem numa janela

`imageinfo` - obtém informação sobre uma imagem gravada em ficheiro e cria uma estrutura que guarda essa informação

`imattributes` - lista atributos da imagem que está a ser visualizada

`image` - visualiza uma matriz qualquer como sendo uma imagem; o valor de cada elemento da matriz é usado como sendo o valor de cor do elemento de imagem espacialmente correspondente.

`im2frame` e `frame2im` - permitem converter de formato de imagem para formato de filme usado pelo MATLAB e vice-versa.

Exemplos de utilização:

```
>> im = imread ('strawberries.jpg'); % lê a imagem "strawberries" com o formato JPEG para a matriz im (MxNx3)

>> [line,col]=size(im(:,:,1)); % lê o número de linhas e colunas da imagem que foi armazenada na matriz im

% e armazena esses valores nas variáveis lin e col (variáveis quaisquer).

% Nota1: a matriz im tem 3 dimensões % já que a imagem que foi lida é uma imagem a cores.

% Nota2: o comando [line,col]=size(im) daria o mesmo resultado.

>> imshow (im); % visualiza a imagem que está na matriz im

>> im2 = double(im); % converte a imagem que está na matriz im para tipo real de precisão dupla e guarda

% na nova matriz im2

>> im2 = im2 + 10; % aumenta o brilho da imagem que está na matriz im2

>> im2 = uint8(im2); % converte a imagem de volta para o tipo inteiro

>> imwrite (im2, 'nome.bmp'); % guarda a nova imagem no ficheiro "nome.bmp" com o formato bitmap

>> imwrite (im2, 'nome2.jpg'); % guarda a nova imagem no ficheiro "nome2.jpg" com o formato jpeg
```

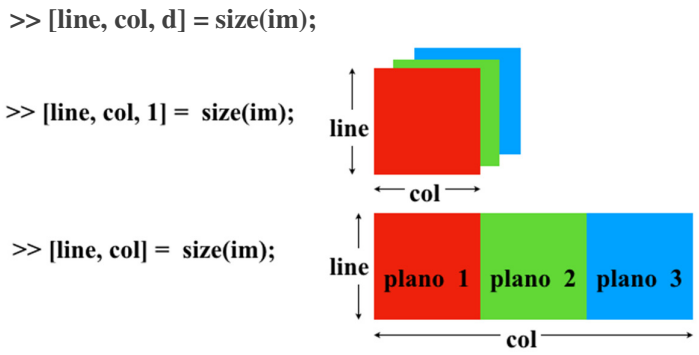
imagens com o formato bitmap (.bmp)

Para imagens armazenadas no formato "bmp" (bitmap com 24 bits) são criadas matrizes a 3 dimensões, em que a terceira dimensão, profundidade, consiste em planos para armazenar cada uma das componentes fundamentais de cor R, G, B. Cada plano contém uma matriz MxN (nº de linhas vezes nº de colunas) que corresponde à matriz de uma das componentes de cor RGB (exactamente nessa ordem: R ≡ plano 1; G ≡ plano 2; B ≡ plano 3). Sempre que usarmos o nome da matriz que foi criada quando importamos a imagem bmp, o Matlab usa os 3 planos/componentes simultaneamente. Mas podemos aceder individualmente aos valores de cada componente, indicando o plano correspondente. Podemos usar o seguinte código:

```
% ler imagem RGB de 24 bits, isto é, cada elemento de imagem é representado por 24 bits, sendo 8 bits para

% a cor vermelha R, 8 bits para a cor verde (G) e 8 bits para a cor azul (B):

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

% verificar que se trata de uma matriz de n pixels por m colunas com profundidade 3, isto é uma matriz a 3

% dimensões. a profundidade 3 indica que na prática existe uma matriz para cada uma das cores primárias

% RGB.

>>size(im)
```

```
% atribuir todas as linhas e todas as colunas do primeiro plano da matriz im à variável r , as do segundo
% plano à variável g e as do terceiro plano à variável b
>> r = im(:, :, 1);
>> g = im(:, :, 2);
>> b = im(:, :, 3);
% obter as dimensões de cada uma das matrizes
>> [line, col, plano] = size(im);
```

A linha de comando "`>> [line, col, 1] = size(im);`" inspeciona as dimensões do plano 1, ou seja, da matriz onde foram armazenados os valores de R da imagem, retornando em line o número de linhas ou de pixels na vertical e em col o número de pixels na horizontal. Nota: o comando `[line, col]=size(im)` daria o mesmo resultado em line, mas daria um valor 3 vezes maior em col, porque apresentaria o nº de colunas de toda a estrutura (composta por três matrizes bidimensionais).




```
% criar uma nova imagem preta (matriz de zeros) com 24 bits por elemento, usando as dimensões da
% imagem lida:
>> im2 = uint8(zeros(line, col, plano)); % criada uma imagem só de zeros
% Cada plano dessa nova matriz recebe um dos planos r, g ou b da outra imagem, incrementados e % decrementados de um dado valor
>> im2(:, :, 1) = r+50;
>> im2(:, :, 2) = g-50;
>> im2(:, :, 3) = b+100;
% visualizar e comparar as duas imagens que estão armazenadas nas matrizes im e im2:
>>figure; imshow(im);
>>figure; imshow(im2);
```

Agumas funções para alterar representações, valores e fazer filtragens:

`rgb2gray` - conversão de 24 bits para tons de cinza

`im2bw` - conversão para preto-e-branco

`rgb2ind` - conversão de 24 bits para 256 cores

`rgb2hsv` - conversão entre os espaços de cor RGB e HSV

`rgb2ycbcr` - conversão entre os espaços de cor RGB e YCbCr

`imresize` - altera as dimensões de uma imagem

`imrotate` - roda uma imagem de um determinado ângulo

`imhist` - calcula o histograma de uma imagem (registo do nº de vezes que cada valor ocorre numa imagem)

`filter2`, `imfilter` - aplica um filtro a uma imagem e devolve uma matriz de valores reais com a imagem filtrada

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

A função "`imresize`" acima retorna uma matriz que é M vezes maior (ou menor) que a imagem A, onde 'method' pode ser um de:

`nearest` = vizinho mais próximo

`bilinear` = interpolação bilinear

`bicubic` = interpolação bicúbica

Identifica a abordagem que vai ser usada para obter os valores dos pixels da imagem re-dimensionada.

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

A função "`imrotate`" acima retorna uma matriz que é uma versão rodada de â*ngulo* da imagem A, onde 'method' pode ser um de [`nearest`, `bilinear`, `bicubic`].

Exemplos:

```
>> A = imread ('fruta.bmp');
>> figure(1); imshow(A); title('imagem original');
>> B = imresize (A, 0.5, 'nearest');
>> figure(2); imshow(B); title('imagem redimensionada');
>> B = imrotate (A, 45, 'nearest');
>> figure(3); imshow(B); title('imagem rodada');


% filtrar uma imagem; após efectuar a filtragem, é necessário arredondar
e converter para uint8 para se poder
% visualizar a imagem filtrada; usa-se o script built-in fspecial para
criar o tipo de filtro desejado; no
% exemplo abaixo são criados três tipos de filtro: de média, de movimento
e de realce de contornos
>> im = imread('nenufares.bmp');
>> im = rgb2gray(im);
```

```
>> subplot(2,2,1);imshow(im);title('Imagem original escala cinza');
>> h = fspecial ('average', 5);
>> im2 = uint8(round(filter2(h, im)));
>> subplot(2,2,2);imshow(im2);title('Imagem filtro media');
>> h= fspecial('motion',20,45);
>> im3= imfilter(im,h,'replicate');
>> subplot(2,2,3);imshow(im3);title('Imagem filtro motion');
>> h=fspecial('unsharp');
>> im4 = imfilter(im,h,'replicate');
>> subplot(2,2,4);imshow(im4);title('Imagem mais nitida');
```

No conjunto de instruções de cima, foi utilizada a instrução 'subplot'. Ela permite criar uma grelha para visualização ordenada de várias imagens no mesmo écran. O primeiro argumento indica o número de linhas dessa grelha; o segundo argumento, o número de colunas; o terceiro argumento indica em que posição desejamos que seja apresentada a imagem. Uma grelha de 2 x 2, tem 4 posições para apresentar imagens ou gráficos. A primeira posição corresponde à linha 1, coluna 1; a segunda posição à linha 1 coluna 2; a terceira posição à linha 2, coluna 1; a quarta posição corresponde à linha 2, coluna 2.

Podemos criar uma matriz com valores aleatórios e apresentá-la como sendo uma imagem, usando a função built-in '*image*'; por exemplo, desta forma:

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

Podemos converter uma imagem a cores para uma imagem a preto e branco, nível de cinzas ou imagem a cores indexada com as funções built-in '`image`', '`rgb2gray`', '`rgb2ind`'; por exemplo, desta forma:

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

Podemos criar uma sub-imagem a partir de uma imagem já importada e acabada de visualizar:

```
>>im=imread('bird-blue.jpg');
>>imshow(im);
>>subImagem=imcrop;
% definir com o rato a área desejada, em cima da imagem original; acabar
com "crop image"
>>imshow(subImagem);
```

Quando se executam operações aritméticas em imagens, deve-se ter em atenção que numa qualquer imagem os pixels podem assumir valores numa escala de 256 (tons de cinza numa imagem sem cor, tonalidades de cores numa imagem Gif, ou intensidade de cada um dos

componentes RGB de uma imagem bitmap). Assim, a adição de duas tais imagens, por exemplo, pode resultar numa nova imagem com pixels de valor superior a 255. Ou a subtração resultar em valores inferiores a 0. Para contornar estes problemas, existem basicamente duas alternativas: (1) normalização de valores e 2) truncatura de valores. No primeiro caso efectuam-se os cálculos e no final à normalizam-se todos os valores obtidos (expressão indicada em baixo). No segundo caso, à medida que se vão efectuando os cálculos, todos os valores superiores a 255 são convertidos para 255 e todos os negativos são convertidos para 0. A segunda alternativa é mais simples do que a primeira mas pode conduzir a resultados piores do ponto de vista perceptual.

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

```
>> 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)));
```

Uma das operações que é frequente realizar com imagens é a convolução, a qual consiste num conjunto de sucessivas multiplicações, adições e deslocamentos entre duas matrizes, sendo uma delas a imagem e outra um operador que se deseja aplicar à imagem. A matriz que contem o operador é normalmente de dimensões bastante inferiores às da imagem. Uma operação semelhante é a correlação.

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