Laboratório 01 - Processamento de Imagens
Professora: Mylène Christine Queiroz de Farias
Monitor: André Henrique Macedo da Costa

Introdução ao MATLAB

Sistema de coordenadas

O MATLAB armazena informação no formato de matrizes. Qualquer estrutura dentro do MATLAB pode ser encarada como uma matriz. Observando as estrutura de imagens em tons de cinza, os pixels das imagens são representados por elementos em uma matriz $img(x, y)$, na qual x representa as linha e y representa a coluna. Caso a imagem seja colorida, existe uma terceira dimensão (z) para os canais de cor $img(x, y, z)$. Logo, uma imagem colorida com dimensões 720×480 seria representada por uma matriz img de tamanho 720×480×3. É importante notar que o MATLAB utiliza a posição $(1, 1)$ para o primeiro pixel das imagens, enquanto que em outras linguagens de programação o primeiro pixel é representado pela posição $(0, 0)$.

Comandos Básicos

As funções mais básicas de processamento de imagens que podem ser realizadas são leitura, apresentação e escrita de imagens. A função do MATLAB utilizada para a leitura de uma imagem é a `imread('path')`, onde path pode ser tanto o nome do arquivo mais a sua extensão quanto o caminho completo até o arquivo da imagem.

A função `size(img)` retorna as dimensões de uma matriz. No caso de imagens, ela retorna respectivamente o número de linhas, colunas e canais de cores. É possível acessar cada uma das dimensões através de um segundo argumento de entrada, como pode ser visto no código abaixo.

```
[row, col] = size(img);
```

ou

```
row = size(img, 1);
col = size(img, 2);
```

A apresentação da imagem é feita por meio da função `imshow(img)`, que tem como única variável a imagem. A imagem será mostrada em um janela *pop-up*, sendo possível salvar a imagem. Essa função permite conferir visualmente os resultados de um processamento. A exportação (escrita) de imagens pode ser feito com a função `imwrite(img, filename)`. A função `imwrite` recebe dois elementos de entrada, a variável da imagem e o nome do arquivo no qual ela será salva. O nome do arquivo pode ser apenas o nome com a extensão desejada ou pode incluir o caminho completo da posição do computador onde se deseja salvar a imagem. Caso se utilize apenas o nome do arquivo, o arquivo será salvo na pasta atual do MATLAB . Um exemplo é mostrado abaixo:

```
imwrite(img, 'myImage.ext');
```

ou

```
imwrite(img, 'C:\myFolder\myImage.ext');
```

no qual pode ser qualquer formato de imagens aceito (png, jpg, tiff, etc.)

Classes de dados

Apesar das coordenadas de uma imagem serem números inteiros, o conteúdo de cada coordenada pode variar. Tipos diferentes de imagens apresentam dados com valores diferentes. Os tipos de dados mais comumente encontrados no MATLAB são `uint8` e `double`, porém existem outros 10 tipos, que são listados a seguir.

- double
- single
- uint8
- uint16
- uint32
- int8
- int16
- int32
- char
- logical

O tipo de dado padrão para uma imagem é o `uint8` que corresponde a uma escala inteira que abrange no intervalo de valores positivos entre 0 e 255. Este tipo de dados é utilizado para representar tanto imagens em tons de cinza quanto imagens coloridas. A apresentação de imagens no MATLAB (`imshow(img)`) são realizadas considerando o tipo `uint8`. O tipo `double` corresponde a números em ponto flutuante, no intervalo de -10^{308} até 10^{308}. Este tipo é geralmente utilizado para operações matemáticas de processamento de imagens. Todos os tipos de classes de dados são

Tipos de imagens e classes de dados

Existem 4 tipos de imagens dentro do MATLAB : imagens de intensidade, imagens binárias, imagens RGB e imagens indexadas. Dependendo da aplicação, um ou mais tipos de imagens podem ser utilizados. Além disso, uma imagem pode mudar de tipo múltiplas vezes dentro de um mesmo código. É importante reconhecer os diferentes tipos de imagens e como converter entre eles.

Imagens de intensidade são imagens cujos pixels representam diferentes intensidades. Seus dados são geralmente do tipo `uint8` ou `uint16`, cujo intervalo de intensidades é, respectivamente, [0,255] ou [0,65535]. Em casos onde o tipo é `double`, o intervalo é normalizado para o intervalo [0, 1]. Como regra geral, as imagens são convertidas de `uint8` para `double` e, em seguida, são processadas. Ao final, a imagem é convertida de volta para `uint8`, o que permite que ela possa ser visualizada ou armazenada corretamente.

Imagens binárias são um caso especial dentro do MATLAB . O valor dos pixels de uma imagem binária são 0 ou 1. Imagens binárias podem ser criadas a partir da seguinte função:

```
B = logical(A).
```

A função `logical` transforma os valores dos pixels em valores lógicos 0s e 1s. Caso seja necessário checar se a imagem é binária, utiliza-se a seguinte função:

```
islogical(A).
```

Esta função retorna o valor 1 caso a imagem seja binária e 0 caso contrário.

A conversão entre classes de dados deve ser feita com o intervalo dos valores dos pixels em mente. Para esta conversão, podem ser utilizadas, por exemplo, as funções `uint8()` e `imuint8()`. É importante lembrar que a maioria das classes de dados possuem funções de conversão com o próprio nome da classe de dados. Seguem alguns exemplos:

```
B = double(A)
```

ou

```
C = uint8(A).
```

A conversão de uma matriz com valores da classe `uint8()` (valores no intervalo [0,255]) para `double` pode ser realizada utilizando a função `double`, que mantém o valor dentro do intervalo [0,255]. Na conversão de volta para `uint8()` os valores serão mantidos dentro desse intervalo. Problemas podem ocorrer caso seja feita alguma operação que modifique os valores dos pixels, de forma que valores fora deste intervalo sejam criados. A função `uint8()` não fará o escalonamento dos valores para o intervalo [0,255], diferentemente da função `imuint8()`. A função `imuint8()` transforma qualquer valor inferior a zero para zero e faz o escalonamento dos demais valores. A Tabela apresenta as funções de conversão entre as classes de dados.

Função	**Output**	**Input**
im2uint8	uint8	logical, uint8, uint16 e double
im2uint16	uint16	logical, uint8, uint16 e double
mat2gray	double (intervalo [0,1])	double
im2double	double	logical, uint8, uint16 e double
im2bw	logical	uint8, uint16 e double

Tabela 1: Funções de conversão de imagens.

Índices de vetores e matrizes

O MATLAB possui diversas formas de manipular vetores e matrizes. As indexações utilizadas permitem uma variedade de tipos de manipulações em uma imagem. No MATLAB , vetores são criados utilizando colchetes e os elementos desse vetor são separados por vírgula ou por espaço. Os elementos de um vetor coluna são separados por ponto e vírgula. Por exemplo:

```
linha = [1 2 3 4 5]
coluna = [1;2;3;4;5]
```

É possível transpor um vetor através da notação(.'):

```
coluna = linha.'
```

O acesso de partes do vetor é feito através do par de coordenadas do elemento desejado. É possível também acessar intervalos dentro do vetor por meio do uso de dois pontos. Por exemplo:

```
linha(1,3) = 3
linha(1:3) = [1 2 3]
linha(3:end) = [3 4 5]
```

Outra forma de se transpor vetores é através da indexação. A notação `linha(:)` produzirá um vetor coluna com os elementos do vetor linha. O processo inverso também é possível. A notação `coluna(1:end)` produzirá um vetor linha com o conteúdo do vetor coluna. O acesso da informação de um vetor não é necessariamente sequencial. É possível extrair elementos salteados usando a notação início:passo:fim, onde inicio será o primeiro elemento extraído, passo será o número (inteiro) de posições até o próximo elemento extraído e fim será o último elemento a ser acessado. Por exemplo:

```
linha(1:2:end) = [1 3 5]
linha(end:-1:1) = [5 4 3 2 1]
```

A notação de matrizes é similar a notação de vetores, sendo necessário apenas levar em conta a segunda dimensão. Um exemplo de uma matriz 3×5:

```
M = [ 1 2 3; 4 5 6 ; 7 8 9]
```

que resulta na seguinte saída:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

O acesso dos elementos de uma matriz é feito de forma semelhante ao caso de vetores. Utiliza-se um par de coordenadas no formato linha e coluna, por exemplo:

```
M(2,3) = 6
```

Pode-se retirar intervalos da matriz utilizando intervalos em cada uma das dimensões. Por exemplo:

$$\texttt{M(2:3;1:2)} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$$

Existem outras formas de se selecionar elementos de uma matriz sem utilizar intervalos. É possível realizar a combinação de índices de linhas e índices de colunas seguindo a notação `M([a b],[c d])`. Essa notação irá combinar os valores dos índices das linhas com os valores dos índices das colunas resultando na seguinte matriz:

$$\begin{bmatrix} (a,c) & (a,d) \\ (b,c) & (b,d) \end{bmatrix}.$$

Exercícios

1. Imagem Negativa:
   (a) Realize a leitura de uma imagem em escala de cinza utilizando a função `imread( )`.
   (b) Mostre as dimensões da imagem a função `size( )`.
   (c) Faça o display da imagem utilizando a função `imshow( )`.
   (d) Binarize a imagem transformando os níveis de cinza do intervalo [1,128] para 1 e o intervalo [129, 256] para 256.
   (e) Inverta os tons de cinza dentro do intervalo de 255 tons da imagem original.
   (f) Faça o display do resultado do item anterior.
   (g) Extra - Realizar o item C com os 3 canais de uma imagem RGB.
2. Inversão dos eixos da imagem:
   (a) Inverta as linhas da imagem.
   (b) Inverta as colunas das imagens.
   (c) Reduza o tamanho da imagem pela metade.
   (d) Dica: utilize a notação de arrays do MATLAB.
3. Realce de um canal de cor:
   (a) Utilizando a seguinte equação de transformação, gera uma imagem em tons de cinza a partir de uma imagem colorida (RGB):

$$\text{Gray} = 0,3 \cdot \text{canalRed} + 0,59 \cdot \text{canalGreen} + 0,11 \cdot \text{canalBlue}.$$

   (b) Realize a transformação das cores dos três canais de cores para um canal com níveis de cinza, considerando que o canal vermelho tem valores inferiores a 80, o canal verde tem valores superiores a 100 e o canal azul tem valores superiores a 100.

Dicas

Segue alguns links de materiais utéis sobre o MATLAB:

- `www.mathworks.com`
- `https://www.mathworks.com/company/newsletters/articles/programming-patterns-maximizing-code-performance-by-optimizing-memory-access.html`
- `https://www.youtube.com/playlist?list=PLlPCaGk10CBvkGqgyyMDfKJOLCNbIIRXH` - Curso UFPA.