Basic MATLAB Commands for Handling Images

Nausheen Qaiser
PUCIT
22nd March 2014

You may install any version of MATLAB but try for atleast MATLAB 7.

In MATLAB main interface, there are three panels visible by default: 'Current Directory', just showing the files present in current working directory,'Workspace' which shows the variables created by the program which is being run, and 'Command Window' where you enter commands to run them. But instead of entering commands into the command window, it is convenient to first make a matlab file, write your complete MATLAB function in it and save this file. Then enter the name of this file in command window to run the function in this file.

Let's make one MATLAB function file. From the file menu, goto File – New – Blank M-File. A new MATLAB file will be opened. Here you will be typing MATLAB commands. First, declare this M file as a function by typing:

```
function functionName
```

in this file. If the function takes one parameter and returns one parameter, then the format is:

```
function outputVariable = functionName(inputVariable)
```

'function' is a keyword that defines this M file as a function. For example to convert an RGB image to YCbCr, we need to take an image from user and also return image, so the first line of our M file may be:

```
function myYCbCrImage = myRGBtoYCbCr(myRGBImage)
```

After this line you write the commands or MATLAB code.

To read an image:

```
myRGBImage = imread('baboon.png');
```

This will search for 'baboon.png' in the current directory and load it into memory as a matrix named 'myRGBImage'.
Please open MATLAB Help (press F1) and type 'imread' in Help Navigator. It will give best explanation for all possible ways you can use 'imread' command.
By default, the image type is 'double'. Functions are available in MATLAB for data type conversion.

To write an image into file:

```
imwrite(myRGBImage,'baboon1.png');
```

This will save 'myRGBImage' matrix as an image file with the name baboon1.

To view matrix as an image:

```
imagesc(myRGBImage);
```

This will open a figure window and display 'myRGBImage' as an image. Another 'imagesc()' command will display the image in the same figure window, overlaying the previous one. To display an image in another window, type 'figure' command which will open a new figure window, and then type 'imagesc' command. The figure has a property called 'colormap' which basically defines the colors which are used by that figure. To view grayscale image, colormap is set to 'gray'. To set colormap to gray, type:

```
colormap gray;
```

To find size of a matrix:

```
[nRows nCols] = size(myImage);
```

Here 'nRows' will contain number of rows present in 'myImage' and 'nCols' will contain number of columns in 'myImage'. You can use any variable names besides 'nRows' and 'nCols'.

To access locations of matrices:

For indexing a 2D matrix, the format is:

variableName = *matrixName*(*rowIndex*, *columnIndex*);

In MATLAB indices start from 1, not 0. If we want to index all locations in a dimension, we use ' : ' operator. For example, 'myImage' is a 2D matrix and we want to store its first row in another matrix (rather vector) named 'myRow':

```
myRow = myImage(1,:);
```

Here ' : ' means all columns.

If we want to store only top half of an image and discard bottom half, where size of the image is 100x100 (rows x columns), we'll use indices as:

```
topHalf = myImage(1:50,:);
```

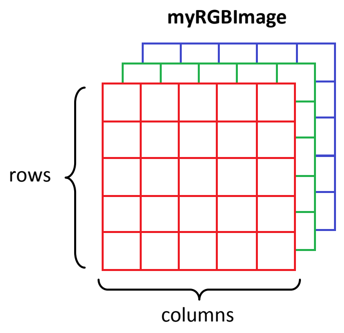
Here '1:50' means row 1 to 50. If we want to select odd numbers from 1 to 100, we would write 1:2:100. It would give 1,3,5,7,...,99.

For indexing 3D matrix, the format is:

variableName = *RGBimage*(*rowIndex*, *columnIndex*, *colorIndex*);

For example, 'myRGBImage' is a 3D matrix and we want to store its red, green and blue components in separate matrices, then we'll write following:

```
myRedColor = myRGBImage(:,:,1);
myGreenColor = myRGBImage(:,:,2);
myBlueColor = myRGBImage(:,:,3);
```



Similarly, we can access N-D matrices.

To access each location of a 2D matrix, we need two nested *for* loops: one iterating its rows and one iterating its columns. For example, we read each pixel of a grayscale image (2D matrix) and check that if the pixel value is 255 (white), change it to say 150 (in other words, make the white pixels gray):

```
for i = 1:nRows
    for j = 1:nCols
        if myImage(i,j) == 255
            myImage(i,j) = 150;
        end
    end
end
```

Each *for* statement, *while* statement and *if* statement has its corresponding *end* statement.

To iterate a matrix such that accessing a row after every 2 rows, use:

```
for i = 1:2:nRows
```

This will give i = 1,3,5,7,...,nRows.

To see format and examples of *if*, *elseif* and *else* commands, please type 'if' in MATLAB Help.

To multiply matrices:

```
C = A * B;
```

Please type 'arithmetic operations' in MATLAB Help and read the first page that opens.

The most convenient way to find whether MATLAB has a function already defined for what you want to perform, is to simply type it in MATLAB Help.

An example:

Below is a MATLAB code for a function called 'makeBinary'. Create new M-file, copy following code to it. To run this file, either enter 'makeBinary' in command window, or use Run button (green colored) seen in the toolbar of Editor window.

```
% This function converts a color image to a binary image.
function makeBinary

grayImage = rgb2gray(imread('baboon.png'));      % Reads an image and converts
                                                   it to grayscale
figure;                    % Opens a new figure
imagesc(grayImage);        % Displays the grayscale image
colormap gray;             % Sets colormap of current figure to grayscale

[nRows nCols] = size(grayImage);     % Gets size of each dimension of matrix.

binaryImage = zeros(nRows, nCols);   % Creates a new matrix of size nRows by
                                       nCols, with all values zero (black)
for i = 1:nRows
    for j = 1:nCols
        if grayImage(i,j) > 130      % If pixel value is greater than 130
            binaryImage(i,j) = 1;    % then make it 1 (white)
        end
    end
end
binaryImage = logical(binaryImage); % Converts the image to boolean image
figure;                    % Opens a new figure
imagesc(binaryImage);      % Displays the binary image
colormap gray;             % Sets colormap to grayscale
imwrite(binaryImage, strcat('binary',int2str(nRows),'x',int2str(nCols),'.png'));
% Saving the image with the name: 'binary' followed by image size
```