Basic Matlab Commands for Digital Image Processing

This document will introduce some of the very basic and essential functions and pseudo code snippets which you will require in all the image processing assignments in Multimedia Communication Systems.

The very first thing we would need to do is to read an image and store it as an array of pixel values (in a 3D matrix). Normally any image is in RGB format and the way it is read in Matlab is as follows

```
rgbImage = imread('C:\image.jpg','jpg');
```

imread supports BMP, GIF and other common formats as well.

Thus **rgbImage** will be a 3D array with dimensions ***rows x cols x component*** in that order; i.e., **rgbImage**(:,:,1) will give you the complete red component of the image.

Now for any image processing application we need to first convert the image into YCbCr format for its manipulation. This is done by the **rgb2ycbcr** command,

```
ycbcr=rgb2ycbcr(rgbImage);
```

This will again read the image into a 3D matrix form where you get the Y Cb and Cr values.

Thus **ycbcr** will be a 3D array with dimensions ***rows x cols x componen***t in that order; i.e., **ycbcr** (:,:,1) will give you the complete Y component of the image.

Once you have read the image into Matlab by doing the above, use the following tricks as a guideline for actually manipulating the image.

Some tips and tricks:

1. Use nested 'for' loops for subsampling and upsampling by accessing the required rows and cols by using a variable 'i' inside the 'for' loop. This is used to parse the whole image component and manipulating only those pixels which are expected in the subsampling type. (The trick is to use for loops like we use in C programming).

```
for cols
    for rows
        ImageComponent(rows, cols + i , component) = 0;
    End
End
```

Here ‘i’ is the fixed jump to the pixels ahead and the ‘component’ will be either 2 or 3 which implies Cb or Cr component, which is manipulated. To do this, use the ‘for’ loop with a fixed jump.

Example: `‘for cols=1:4:640’` increments by 4 every time. Thus we can use this to make the 3 columns zero and retaining every 4th column.

The trick is to make all the non-required components zero, as seen in the pseudo code above.

2. For upsampling especially when we need to average the values between pixels, use this ‘for’ loop method again. However just keep in mind that the averaging and pixel manipulation is done in the ‘for’ loop itself.

3. Use the `subplot()` followed by `subimage()` method to plot the original image and the corresponding 2 subsampled images and the 2 upsampled images for easy comparison. Repeat this for linear interpolation and replication based upsampled images.

To calculate the MSE use the formula.

$$\frac{1}{MN}\sum_{y=1}^{M}\sum_{x=1}^{N}\left[I(x,y) - I'(x,y)\right]^2$$

where M,N are the dimensions of the image
I(x,y) corresponds to the original image
I’(x,y) corresponds to the reconstructed image

Keep in mind that.

- we can directly subtract 2 matrices of same dimensions without using ‘for’ loops (unlike in C)
- we still use the **.^** to raise each of the elements in the matrix by some power ex: X.^2 will square every element in the matrix
- use the ‘sum’ command to sum up all the elements of a matrix eg: sum(X(:)) - no need to use a double for loop to sum up the elements. (The double summation might trick you into using a double for loop, which is not actually required.)