# Image Processing - 67829

# Exercise 1: Image Representations & Intensity Transformations & Quantization

Due date: 8/11/2015

## 1 Overview

The main purpose of this exercise is to get you acquainted with Matlab's basic syntax and some of its image processing facilities. This exercise covers:

- Loading grayscale and RGB image representations.
- Displaying figures and images.
- Transforming RGB color images back and forth from the YIQ color space.
- Performing intensity transformations: histogram equalization.
- Performing optimal quantization

## 2 Background

Before you start working on the exercise it is recommended for those of you using Matlab for the first time to go over the first week's exercise class notes. Those already familiar with Matlab may still find the "Image Processing Toolbox" part of the Matlab documentation useful. **Relevant reading material:** You can read about power law transformations and histogram equalization in Gonzalez and Woods book.

# 3 The Exercise

## 3.1 Reading an image into a given representation

Write a function that reads a given image file and converts it into a given representation. The function should have the following interface:

```
im = imReadAndConvert(filename, representation)
```

having the following input arguments:
`filename` - string containing the image filename to read.
`representation` - representation code, either 1 or 2 defining if the output should be either a grayscale image (1) or an RGB image (2).

Make sure that the output image is represented by a matrix of class `double` with intensities (either grayscale or RGB channel intensities) normalized to the range $[0, 1]$. You will find the function `rgb2gray` useful, as well as `imfinfo` which will allow you to check the representation of the stored image file (using the returned `ColorType` field) prior to loading it with `imread`. We won't ask you to convert a grayscale image to RGB.

## 3.2 Displaying an image

Write a function that utilizes `imReadAndConvert` to display a given image file in a given representation. The function should have the following interface:

```
imDisplay(filename, representation)
```

where `filename` and `representation` are the same as those defined in `imReadAndConvert`'s interface. The function should open a new figure window and display the loaded image in the converted representation. Also use the `impixelinfo` function to turn on the interactive pixel value display in the opened figure window.

## 3.3 Transforming an RGB image to YIQ color space

Write two functions that transform an RGB image into the YIQ color space (mentioned in the lecture) and vice versa. Given the red (R), green (G), and blue (B) pixel components of an RGB color image, the corresponding luminance (Y), and the chromaticity components (I and Q) in the YIQ color space are linearly related as follows:

$$
\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}
$$

The two functions should have the following interfaces:

```
imYIQ = transformRGB2YIQ(imRGB)
imRGB = transformYIQ2RGB(imYIQ)
```

where the inputs and the outputs of the functions are $height \times width \times 3$ `double` matrices with values in $[0, 1]$. In the RGB case the red channel is encoded in `imRGB(:,:,1)`, the green in `imRGB(:,:,2)`, and the blue in `imRGB(:,:,3)`. Similarly for YIQ, `imYIQ(:,:,1)` encodes the luminance channel Y, `imYIQ(:,:,2)` encodes I, and `imYIQ(:,:,3)` encodes Q.

### 3.4 Histogram equalization

Write a function that performs histogram equalization of a given grayscale or RGB image. The function should also display:

- The input image
- The equalized output image

The function should have the following interface:

```
[imEq, histOrig, histEq] = histogramEqualize(imOrig)
```

where:
`imOrig` - is the input grayscale or RGB double image with values in $[0, 1]$.

`imEq` - is the equalized image. grayscale or RGB double image with values in $[0, 1]$.
`histOrig` - is a 256 bin histogram of the original image (256x1 vector).
`histEq` - is a 256 bin histogram of the equalized image (256x1 vector).

If an RGB image is given the following equalization procedure should only operate on the Y channel of the corresponding YIQ image and then convert back from YIQ to RGB. The required intensity transformation is defined that the gray levels should have an approximately uniform gray-level histogram (i.e. equalized histogram) stretched over the entire $[0, 1]$ gray level range. You may use the Matlab functions imhist, cumsum. to perform the equalization **but not histeq**. Also note that although imOrig is of type double, you are required to internally perform the equalization using 256 bin histograms.

### 3.5 Optimal image quantization

Write a function that performs optimal quantization of a given grayscale or RGB image. The function should also display:

- the input image
- the quantize output image
- an error plot representing the total intensities error for each iteration in the quantization procedure.

The function should have the following interface:

```
[imQuant, error] = quantizeImage(imOrig, nQuant, nIter)
```

where:
imOrig - is the input grayscale or RGB image to be quantized.
nQuant - is the number of intensities your output imQuant image should have.
nIter - is the maximum number of iterations of the optimization procedure. (May converge earlier.)

If an RGB image is given, the following quantization procedure should only operate on the Y channel of the corresponding YIQ image and then convert back from YIQ to RGB. Each iteration in the quantization process contains two steps:

- Finding z - the borders which divide the histograms into segments. z is a vector containing nQuant+1 elements. The first and last elements are 0 and 255 respectively.
- Finding q - the values that each of the segments' intensities will map to. q is also a vector, however, containing nQuant elements.

More comments:

- You should perform the two steps above nIter times.
- You should find z and q by minimizing the total intensities error. The close form expressions for z and q can be found in the lecture notes.

- The quantization procedure needs an initial segment division of [0..255] to segments, `z`. If a division will have a gray level segment with no pixels, procedure will crash **(Why?)**. In order to overcome this problem, you are required to set the initial division such that each segment will contain approximately the same number of pixels.

- The output `error` is a vector with `<nIter x 1>` elements (or less in case of converges). Each element is the total intensities error in a current iteration. The exact error calculation is given to you in the lecture notes.

- Convergence is defined as a situation where the `z` values were not changed during the last iteration.

- Please notice that your function should plot the error as a function of the iteration number using the command `figure; plot(error)`. As a sanity check make sure that the error graph is monotonically descending.

- For simplicity, you may assume that we won't ask you to quantize the image to more than half of the total number of gray-shades

- In general we will not check your quantization on extreme cases. For example, you may assume that there is no bin in the histogram that contains more pixels than the initial number of pixels per segment.

### 3.5.1 Bonus - 10 points

If you find this exercise too easy, you are welcome to try and perform quantization for full color images. You will have to find the methods yourself.

## 4 Specific Guidelines

Pay attention to the following guidelines:

1. Test your programs on the provided example images and on other images you may find or create. Try not to use very big images (more than 1000X1000 pixels) for saving your workspace memory and runtime.

2. Document all interfaces, code blocks, and crucial parts of the program (but not every single line).

3. You must write vectorial code where possible (try to avoid using for-loops). Sections 3.1, 3.2, 3.4 should be done without loops at all. In section 3.3 you may loop over the 3 channels, but not on

the pixels of the image. In section 3.5 you will need to use a loop for performing the iterations. You can also loop over `nQuant`, but you should not loop over 256 or over the pixels of the image.

4. In sections $3.3, 3.4, 3.5$ the output images should be doubles in the range $[0, 1]$. You can assume that the input images are doubles in the range $[0, 1]$. .

5. Your program may only crash in case of serious abuse by the user. You can use a try-catch blocks in your code.

6. Write reusable code. This will make solving this and the following exercises much simpler.

7. You may add new functions to your implementation.

8. In sections $3.4, 3.5$ for the transformation from RGB to YIQ color space you can use `rgb2ntsc` Matlab function, and for the transformation from YIQ to RGB color space you can use `ntsc2rgb`. You should not use them in section 3.3.

9. Mention in your README file the matlab version and platform on which you have tested your program on. List the files you are submitting and a short description of each. Also list in the README file the functions you've written with a short description of each. You should also answer the question from section 3.5.

# 5 Submission

Submission instructions may be found in the "Submissions Guidelines" document published on the course web page. Please read and follow them carefully. In this exercise you are also required to submit two test images that you have created (or downloaded), named `testImg1.jpg` and `testImg2.jpg` (different from those that we've supplied). Explain in your README file why you designed them in this way and how they were used to examine your implementation.

Good luck and enjoy!