Lab 1 - Introduction to Image Analysis with Matlab

This assignment will introduce you to some of the basic image operations in Matlab.

Manupulating images

Loading images

The following function is used for loading images of different formats into Matlab.

```
>> [im, map] = imread('filename.type');
```

im is a matrix of the image and map is the corresponding color map (if available). For RGB images map will be empty and im will be a 3D matrix (see ***Image color types***).

Viewing images

No images are much of use, if we can't display them. To view the image, we use the following function

```
>> image(im)
colormap(map)
```

A second possibility is to use the function imagesc(im). This function automatically adapts the contrast of the image. In some cases this could be desirable.

If the aspect ratio of the image is off, you can use

```
>> axis image
```

Saving images

We may at some point in the process want to save the changes and or corrections we have made. This is done by the imwrite function.

```
>> imwrite(im, map, 'filename.type')
```

Image color types

Image data can be either indexed or true color. An indexed image stores colors as an array of indices into the figure colormap. A true color image does not use a colormap

(when you read an image with imread, as discussed before, an empty colormap is returned). Instead, the color values for each pixel are stored directly as RGB triplets. In MATLAB, the CData property of a true color image object is a three-dimensional (m-by-n-by-3) array. This array consists of three m-by-n matrices (representing the red, green, and blue color planes) concatenated along the third dimension.

Image data types

When you read image data into MATLAB using imread, the data is usually stored as an array of 8-bit integers. However, imread also supports reading 16-bit-per-pixel data from TIFF and PNG files. These are more efficient storage methods than the double-precision (64-bit) floating-point numbers that MATLAB typically uses. However, it is necessary for MATLAB to interpret 8-bit and 16-bit image data differently from 64-bit data. This table summarizes these differences.

Image Type	Double-Precision Data (double Array)	8-Bit Data (uint8 Array) 16-Bit Data (uint16 Array)
Indexed (colormap)	Image is stored as a two-dimensional (m-by-n) array of integers in the range [1, `length(colormap)`]; `colormap` is an m-by-3 array of floating-point values in the range [0, 1].	Image is stored as a two-dimensional (m-by-n) array of integers in the range [0, 255] (`uint8`) or [0, 65535] (`uint16`); `colormap` is an m-by-3 array of floating-point values in the range [0, 1].
True color (RGB)	Image is stored as a three-dimensional (m-by-n-by-3) array of floating-point values in the range [0, 1].	Image is stored as a three-dimensional (m-by-n-by-3) array of integers in the range [0, 255] (`uint8`) or [0, 65535] (`uint16`).

From indexed to RGB image

With the function ind2rgb, Matlab converts an indexed image to an RGB image.

```
>> RGBim = ind2rgb(im, map)
```

From RGB to indexed image

Unfortunately the function rgb2ind is only available with the image processing toolbox.

Image details

With the function imfinfo, you can obtain detailed information about your graphics file.

```
>> imfinfo('filename.type')
```

Questions

Question 1.1

Load and display the images *lena_gray.tif and Lena_RGB.tif.* Examine the image properties with *imfinfo* and supply the requested data below.

Image File name	***lena_gray.tif***	***lena_rgb.tif***
Width		
Height		
Image Color Type		
Image Data Type		
Mean Pixel value		

Question 1.2

Load the indexed image mandrill from Matlab *(*load mandrill*)* and convert it to an RGB image. Save the RGB image as a *mandrill.bmp*.
Write a function that converts the RGB values to grayscale values by forming the following standard weighted sum of the R, G, and B components:

```
0.2989 * R + 0.5870 * G + 0.1140 * B
```

The head of the function should look like somewhat like this:

```
function [grayimage_out] = rgbtograyscale(rgbimage_in)
```

Question 1.3

Display the images *lena_gray.tif* and *man.bmp* and their histograms.

a) What can you say about the contrast of the images based on their histograms?
b) Perform histogram equalization. Ensure that you create a function histogramequalization.m to do it. You might need it later. Matlab also provides a histeq function. You can compare things.
c) Compare the histograms, and verify the result.
d) Did it help increasing the image contrast?

Intensity Mappings

The probably most simple form of pixel manipulation is where we access each of the pixels im(r,c) in the image. If the images have been loaded as arrays, the addressing of pixels is done by

```
>> I_out(123,133) = I_in(123,133);
```

This mapping is actually a special case of the linear mapping.

Affine (linear) mapping

The general form of this mapping is

```
I_out(r,c) = a * I_in(r,c) + b
```

This gives us a mapping of the pixel values as seen in Figure 1.

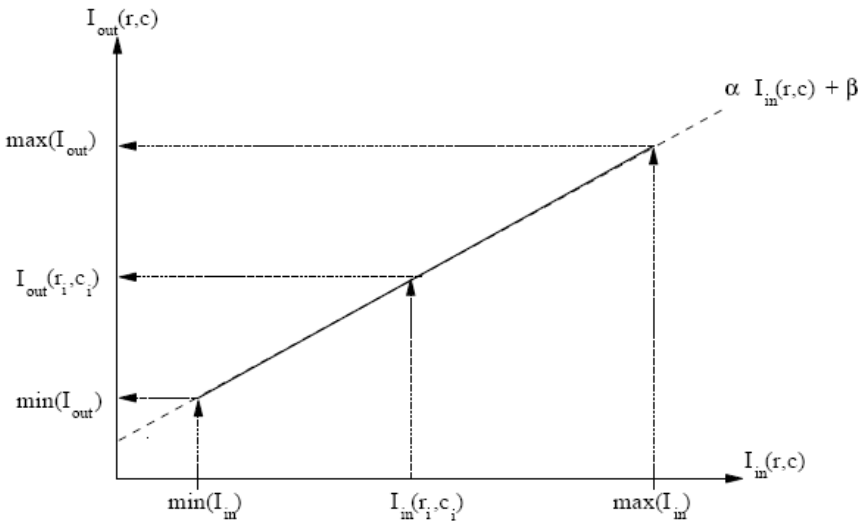

Figure 1: Illustration of the Linear Mapping.

Gamma mapping

The gamma (γ) mapping is defined for positive γ as:

$I\_out(r,c) = I\_out(r,c)^{\gamma}$

For γ=1 we get the intensity mapping. For $0< \gamma<1$ we increase the dynamics in the dark areas by increasing the mid-levels. For $\gamma > 1$ we increase the dynamics in the bright areas by decreasing the mid-levels.

Thresholding and Slicing

Thresholding and Slicing generates a binary image by setting all pixels values in a given range, $[I_{min};I_{max}]$, to 1 and the rest of the pixel values to 0.

$$I_{out}(r,c) = \begin{cases} 0 & \text{if } I_{in}(r,c) < I_{\min} \text{ or } I_{in}(r,c) > I_{\max} \\ 1 & \text{otherwise} \end{cases}$$

When only one of the threshold values is used, it is called Thresholding, and when both values are used, it is called Slicing. In Matlab the logical operators may be used for thresholding images. If we want to threshold an image Im_in at the gray level value 134.2, this could be done by

```
>> image(I_in>134.2)
```

RGB Mapping

Color images are usually represented as three channel images. For simplicity, let us consider all three channels as scaled between 0 and 1. A given color is obtained by selecting the amounts of three primary colors.

From (R,G,B) to (I,v_1,v_2)

$$\begin{pmatrix} I \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \\ \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} & 0 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Note that this (I,v_1,v_2) is slightly different from the (y,Cb,Cr) on page 74 of the book and you can easily construct a matrix that does the same trick for obtaining a (y,Cb,Cr)/

(Y,C_b,C_r) representation. "The nice thing about standards is that you have so many to choose from" – Andrew S. Tanenbaum.
After this conversion, we calculate the Hue and Saturation from v_1 and v_2.

$$S = \sqrt{v_1^2 + v_2^2}$$

From (I,v_1,v_2) to (I,H,S)

$$H=\arctan2(v_2,v_1).$$

Questions

Question 1.4

Write a Matlab function that linearly scales all intensity values in the input images between 0 and 1, and returns the scaled image. The head of the function should look somewhat like this:

```
function [I_out] = linearscale(I_in, I_min_out, I_max_out)
```

Question 1.5

Load and display *eye.png*. Look at the histogram and see if you can find the optimum value that gives the best discrimination between the pupil and the rest of the eye.

a) Use thresholding to set all pixel values of the pupil to one.

As you can see, some parts of the pupil are missed, caused by the white reflections. Look again at the histogram of the original image to determine a value that discriminates between reflections and the rest of the image.

b) Use slicing to select the pixel values of the pupil including the reflections.

Question 1.6

The goal of this exercise is to make a color transformation of RGB colors into HSI (Hue, Saturation, Intensity) values. Load the image *mandrill.bmp* (saved in exercise 2) into the variable Z.

a) What is the dimension of your image?

b) Generate a new variable

```
>> im1 = Z(:,:,1);
```

```
>> im2 = Z(:,:,2);
>> im3 = Z(:,:,3);
>> im = double([im1(:) im2(:) im3(:)]);
```

c) What is the dimension of the new variable im, and what is the difference with this and the first variable?

Try:

```
>> imRED   = reshape(im(:,1), nr, nc);
>> imGREEN = reshape(im(:,2), nr, nc);
>> imBLUE  = reshape(im(:,3), nr, nc);
```

Where nr and nc are the number of rows and columns of your original image.

d) Display the images. What has happened?

e) Write a Malab function that takes im as argument and converts it to HSI values. The head of the function should look somewhat like this:

```
Function [HSI_out]= rgbtohsi(RGB_in)
```

Note that you can also program it to work directly on Z. It requires that you are comfortable with Matlab.

Convolution and filtering

Convolution and filtering can conveniently be handled with the Matlab filter function. The following tasks will help you to gain some familiarity with it

Questions

Question 1.7

Create a signal that looks as follows:
s={2, 4 , 8, …, 64, 63, 31, …, 1}
Use the filter function to convolute {2, 2, -2, -2} with that signal s.
Is the result as you expected? Did some sort of padding take place?

Question 1.8

Repeat question 1.7 by using the Matlab function filtfilt.

What did Matlab do differently now?

Question 1.9

Take the intensity of the Mandrill image from question 1.2 and plot it with a gray colormap using imagesc.
You can create a 20*20 Gaussian window by using the calls:

```
h1=gausswin(20);
h2=kron(h1,h1');
```

Use filter2 to apply this filter to the intensity channel of the mandrill image and display the original picture and the filtered version in two subplots.