Course: DD2427 - Exercise Set 1

Before Starting

You will write the programming exercises for this course in *Matlab*. Besides invoking *Matlab* commands, you will be required to run a few operating system commands. For these commands I will assume your computer's operating system is either linux or unix. If otherwise, you'll have to fend for yourself. But all the non-*Matlab* commands needed are more-or-less trivial.

These notes, and those to follow, give explicit instructions about which *Matlab* commands to use. However, I will not give detailed explanations about their usage. I assume you have some previous experience with *Matlab* and are aware of many of the in-built functions and how to manipulate vectors and matrices. Keep in mind the function `help` can be called to obtain information about particular functions. So for example

```
>> help plot
```

will display information about the *Matlab* command `plot`.

Background 1: *Getting Started*

Set up your environment

Create a new directory to hold all the *Matlab* files you will write for this course:

```
$ mkdir DirName
$ cd DirName
$ mkdir Pics
$ mkdir Result_Pics
```

Download the test images, for the *Matlab* functions you write, from the course website
`http://www.csc.kth.se/utbildning/kth/kurser/DD2427/bik11/`. Move the `StartImages.tar` file to the `Pics` directory you have just created, `untar` the file and then move up to the parent directory. Also download the file `montage.m` and move it to `DirName`.

```
$ mv montage.m DirName/
$ mv StartImages.tar DirName/Pics
$ cd DirName/Pics
$ tar xvf StartImages.tar
$ cd ..
```

Starting *Matlab*

We are now ready to start running *Matlab*

```
$ matlab &
```

Once the matlab interpretor window is open you should see the prompt `>>`. First check which directory you are in.

```
>> pwd
```

If the response is not equal to `DirName` type

```
>> cd DirName
```

Next use the command `addpath` to tell *Matlab* where the images are stored.

```
>> addpath DirName/Pics
```

Background 2: *Useful Display Function*

At the beginning of this document you copied a function called `montage.m`, which is a slightly modified version of the function available at:
http://www.mathworks.com/matlabcentral/fileexchange/22387

This is a useful function as it allows you to efficiently view the images in a directory:

```
>> montage('Pics');
```

Use `help` to find out the different ways `montage` can be called.

Background 3: *Images in Matlab - The Basics*

We are ready to start; to begin, enter the instructions below to load an image and display it

```
>> im = imread('bike_small_gray.jpg');
>> fig = figure;
>> imagesc(im);
>> axis equal
>> colormap(gray)
```

You have just loaded and displayed a grayscale image whose height is larger than its width. The matrix `im` contains the pixel intensity data. Typing the command

```
>> size(im)
```

returns the numbers `717 538`, which are the dimensions of the matrix `im`. It has 717 rows and 538 columns. However, note that the image loaded has dimensions of width 538 pixels and height 717 pixels. The rows correspond to the $y-$ dimension and the columns to the $x-$ dimension. Therefore to obtain the value of the image intensity of pixel (x, y) you need to access the xth column and yth row of the matrix `im`. For example the intensity at pixel $(500, 355)$ is given by

```
>> im(355, 500)
```

which gives the result 123. To plot a cross on the image at the location $(500, 355)$ invoke the following commands:

```
>> figure(fig);
>> hold on
>> plot(500, 355, 'gx', 'LineWidth', 2, 'MarkerSize', 12)
```

To reinforce this point, plot the intensity values along the vertical line from pixel $(500, 355)$ to pixel $(500, 595)$ by calling these commands.

```
>> plot([500, 500], [355, 595], 'r-', 'LineWidth', 2)
>> pix_int = im(355:595, 500);
>> y_values = 355:595;
>> fig1 = figure;
>> figure(fig1);
>> plot(y_values, pix_int, 'b-')
```

Examine the fluctuations of the intensity pattern especially those that correspond to the white stripes of the zebra crossing.

It is possible to turn the matrix `im` into a vector via

```
>> pixs = im(:);
>> disp([size(pixs)])
>> disp(size(im, 1)*size(im, 2))
```

Notice that `pixs` is a column vector whose length is equal to the number of pixels in the image. However, now you may be unsure, depending on your

knowledge of *Matlab* memory structure, how to access the intensity value of pixel (x, y) via the vector `pixs`. *Matlab* stores the elements of `im` column-wise. The command `im(:)` concatenates the columns of `im` starting with the first one. Therefore pixel (x, y)'s intensity can be indexed at the `ii`th entry of `im(:)` where

```
ii = size(im, 1)*(x-1) + y
```

As a concrete example the index for pixel $(500, 355)$ is

```
>> ind = size(im, 1) * 499 + 355
>> disp([im(355, 500), pixs(ind)])
>> disp([pix_int(1:5), pixs(ind:ind+4)])
```

To reassure yourself you can re-plot the intensities along the vertical line from pixel $(500, 355)$ to pixel $(500, 595)$ by accessing the values via `pixs`.

```
>> ind_vals = size(im, 1) * 499 + y_values
>> pix_intA = pixs(ind_vals)
>> fig2 = figure;
>> figure(fig2)
>> plot(y_values, pix_intA, 'b-')
```

As a quick note you can use the function `reshape` to obtain the original image matrix from `pixs`.

```
>> oim = reshape(pixs, size(im, 1), size(im, 2));
>> size(oim)
```

Background 4: *Colour Images*

So far we have only loaded a grayscale image. Now it is time to explore the world of colour! Let's load a colour image:

```
>> col_im = imread('bike_small.jpg');
>> col_fig = figure;
>> imagesc(col_im);
>> axis equal
```

You have loaded and displayed a colour version of our grayscale image. The array `col_im` contains the RGB data for each pixel. How is this data stored in `col_im`? Enter the following commands:

```
>> ndims(col_im)
>> size(col_im)
```

The number `3`, the number of dimensions of the array `col_im`, is returned after the first command and the numbers `717 538 3` after the second. This indicates we have three matrices of size 717×538. The RGB data can be accessed as

```
>> R = double(col_im(:, :, 1));
>> G = double(col_im(:, :, 2));
>> B = double(col_im(:, :, 3));
```

Each of the matrices `R`, `G`, `B` can be manipulated identically to the grayscale image `im`. The red value of pixel $(500, 355)$ is given by

```
>> R(355, 500)
```

and the RGB value of pixel $(500, 355)$ by

```
>> disp([R(355, 500), G(355, 500), B(355, 500)])
```

This latter command will output `125 124 120`. We can also display each component.

```
>> figure(col_fig)
>> montage({R, G, B}, 'Size', [1, 3]);
```

These commands result in a figure in which the colour image and then each of the colour components are displayed. Notice how the lady's red hair is bright in the `R` image and dark in the `G`, `B` images and the blueish jumper is brightest in the `B` image.

Exercise 1: *Digitally simulating dichromat vision*

Background theory

Colour-blindness is hereditary and mainly affects males; upto 8% of all males suffer from it in some form. Therefore, when you are designing web-pages or talks you should choose colour schemes which are discernible to both people with normal colour vision and those with colour blindness. **Dichromacy** is a particular form of colour-blindness. It occurs when one type of the colour cones in the retina is missing and perceived colours can only be generated from mixing the responses from the two remaining cone types.

Luckily though, you are now going to write a function to simulate the appearance of a colour picture to a person with dichromatic vision. In the future you will be able to check if your digital content has been well-designed for all! We will describe a method for computing the transform presented by Viénot et al. [1].

The two most common dichromats are **Protan** (person only possesses green and blue cones on the retina) and **Deutan** (person only possesses red and blue cones on the retina). These are the two types of colour blindness you will simulate. The steps of the algorithm are now described.

The details

Assume that a pixel's colour in the original image is (R, G, B). Then the steps of the transformation to convert it to (R_p, G_p, B_p) and (R_d, G_d, B_d) its representation for a protan dichromat and deutan dichromat respectively are as follows:

1. Transform the original (R, G, B) via

$$r = (R/255)^{2.2}, \quad g = (G/255)^{2.2}, \quad b = (B/255)^{2.2} \tag{1}$$

2. Slightly reduce the colour domain represented. A different transform is used for each type of the dichromat vision:

protanopes	**deuteranopes**
$r_1 = 0.992052\, r + 0.003974$	$r_1 = 0.957237\, r + 0.0213814$
$g_1 = 0.992052\, g + 0.003974$	$g_1 = 0.957237\, g + 0.0213814$
$b_1 = 0.992052\, b + 0.003974$	$b_1 = 0.957237\, b + 0.0213814$

3. Transform the RGB representation to a LMS representation using matrix M.

$$M = \begin{pmatrix} 17.8824 & 43.5161 & 4.11935 \\ 3.45565 & 27.1554 & 3.86714 \\ 0.0299566 & 0.184309 & 1.46709 \end{pmatrix} \tag{2}$$

   In the LMS system colours are specified in terms of the relative excitations of the longwave sensitive (red), the middlewave sensitive (green), and the shortwave sensitive (blue) cones.

4. Reduce the colour domain to the dichromat domain. In the LMS representation it is possible to learn the projection matrix from normal vision to a dichromat's vision, which the authors of [1] have done. These two projection matrices are

$$P = \begin{pmatrix} 0 & 2.02344 & -2.52581 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 & 0 \\ 0.494207 & 0 & 1.24827 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

5. Transform back to RGB space using M^{-1}. The concatenation of the transformations in steps 3-5 is

$$\begin{pmatrix} r_p \\ g_p \\ b_p \end{pmatrix} = M^{-1}PM \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix}, \quad \begin{pmatrix} r_d \\ g_d \\ b_d \end{pmatrix} = M^{-1}DM \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} \tag{4}$$

6. Finally, invert the first step:

$$R_p = 255(r_p^{1/2.2}),\ G_p = 255(g_p^{1/2.2}),\ B_p = 255(b_p^{1/2.2}) \tag{5}$$

$$R_d = 255(r_d^{1/2.2}),\ G_d = 255(g_d^{1/2.2}),\ B_d = 255(b_d^{1/2.2}) \tag{6}$$

Your task

What you have to do now is to write a function `MakeDichromatIms.m` that takes an image as input and outputs two images. One of the images is how a protan dichromat would perceive the input image and the other how a deutan dichromat would. Or in words apply the transformation(s) just described to each pixel in the input image.

```
[pim, dim] = function MakeDichromatIms(im)
```

You can display the results with the function

```
>> montage({im, pim/256, dim/256}, 'Size', [1, 3]);
```

For the image `FireEngine.jpg` you should obtain the images in figure 1.

Original image **protanope sees** **deuteranope sees**

Figure 1: The middle and right picture show what your algorithm should output. (This picture is best viewed on screen).

Deadline for uploading to Bilda: 12pm of 21st of March

- *Upload the function* `MakeDichromatIms.m` *and the pictures you compute from the image* `Flowers.jpg`*, with the commands:*

```
>> imwrite(pim/256, 'Result_Pics/pFlowers.png', 'png');
>> imwrite(dim/256, 'Result_Pics/dFlowers.png', 'png');
```

to bilda.

- *For the lecture on the 21st bring a print out of your function* `MakeDichromatIms.m`.

References

[1] F. Viénot, H. Brettel, and J. Mollon. Digital video colourmaps for checking the legibility of displays by dichromats. *Color Research and Application*, 24:243–252, 1999.