Introduction to Image Processing with MATLAB

1. Introduction

This laboratory session provides a tutorial on working with images in MATLAB. This laboratory is not assessed. However, the methods that you will learn about in this session will be required to complete the subsequent assessed laboratory exercises. If you have little prior experience of MATLAB you should work through a general introductory tutorial. There are many such tutorials available online (see the module's KEATS webpage). You will need to complete these additional tutorials in your own time.

2. MATLAB Fundamentals

You can use MATLAB interactively by typing commands at the prompt. You can also write MATLAB functions by creating m-files (text files containing MATLAB commands). As with built-in MATLAB functions, user-written functions can be executed by typing the name of the m-file at the MATLAB prompt (assuming the directory containing the m-file is on your path). To obtain information on any in-built MATLAB function, just type "`help functionname`" at the MATLAB prompt (replacing "functionname" with the name of the function you want to know about).

Common Pitfalls:

- Be sure to use .* instead of * when you're multiplying two images together element-by-element. Otherwise, MATLAB will do a matrix multiplication of the two, which will take forever and result in total nonsense.
- Be sure not to forget the semicolon at the end of a command. Otherwise, you may sit for a while watching all the pixel values from a large image scroll by on the screen.
- MATLAB is painfully slow at executing loops, so only use them when absolutely necessarily. Instead, write commands that perform operations on whole vectors or matrices in one go, rather than using loops to perform the same operation on each element one at a time.

3. Image Input

A digital image is composed of pixels. An image file or image representation provides instructions as to how to color each pixel. To store information describing all the pixel values in a large image takes a lot of memory. Hence, images are often represented in a compressed format. Most images you find on the Internet are JPEG-images which is the name for one of the most widely used compression standards for images.

Reading an image into MATLAB is performed by the function `imread`. This function supports the a number of image file formats (these can be listed by executing the command `imformats`).

> Copy the following two image files to the directory in which you are running MATLAB: rooster.jpg elephant.png
>
> At the MATLAB prompt execute the following commands:
> ```
> Ia=imread('rooster.jpg');
> Ib=imread('elephant.png');
> ```

4. Image Representation

Once an image file has been read, it is stored internally using one of several formats. These image types are primarily for the purpose of display as they are not always compatible with many mathematical functions available in MATLAB. The most common MATLAB image types are:

1. Intensity images
   This is the equivalent to a "grayscale image". It represents an image as a matrix where every element has a value corresponding to how bright/dark the pixel at the corresponding position should be colored. There are two main ways to store the number that represents the brightness of the pixel. The double data type assigns a floating number between 0 and 1 to each pixel, the value 0 corresponds to black and the value 1 corresponds to white. An alternative is to use a data type called uint8 which assigns an integer between 0 and 255 to represent the brightness of a pixel, the value 0 corresponds to black and 255 to white. The class uint8 only requires roughly 1/8 of the storage compared to the class double. On the other hand, many mathematical functions can only be applied to the double class.
2. Binary images
   This image format also stores an image as a matrix but can only color a pixel black or white (and nothing in between). It assigns a 0 for black and a 1 for white.
3. RGB images
   This is a format for color images. It represents an image using a 3-dimensional matrix. The third dimension

(sometimes called the channel) corresponds to one of the colors red, green or blue and the values of the elements specify how much of each of these colours a certain pixel should use.

4. Indexed images
   An indexed image stores an image as two matrices. The first matrix has the same size as the image and one number for each pixel. The second matrix is called the color map and its size may be different from the image. The numbers in the first matrix is an instruction of what number to use in the color map matrix.

> At the MATLAB prompt execute the following command:
> `whos`
> You will see that Ia (the MATLAB variable representing the rooster image) is a 341x386 pixel RGB image (or equivalently a 3-dimensional matrix), with pixel colour values stored using uint8 format. Ib (the MATLAB variable representing the elephant image) is a 512x512 pixel greyscale image (or equivalently a 2-dimensional matrix), with pixel intensity values stored using uint8 format.

5. Image Display

To display and image we can use a number of commands; `imshow, image,` and `imagesc`.

These commands display images using a colormap which describes the mapping between the numbers in the matrices encoding the image and the colours displayed on the screen. To see how numbers map onto colours use the command `colorbar`. To change the colour map, we can use the command `colormap`.

You can display multiple images by either creating multiple figures with the `figure` command or by putting multiple images in the same figure with the `subplot` command.

You can label each plot or subplot using the `title` command.

> At the MATLAB prompt execute the following command:
> `figure(1), subplot(2,2,1), imagesc(Ia);`
> This displays the original image of the rooster.
>
> Now execute:
> `subplot(2,2,2), imagesc(Ia(:,:,1)); title('red channel'); colorbar`
> `subplot(2,2,3), imagesc(Ia(:,:,2)); title('green channel'); colorbar`
> `subplot(2,2,4), imagesc(Ia(:,:,3)); title('blue channel'); colorbar`
> This displays the intensities of the individual R, G, and B channels that make up the image. You will notice that the rooster's crest and neck feathers have high intensity in the red channel, while the grass has high intensity in the green channel.
>
> At the MATLAB prompt execute the following command:
> `colormap('gray')`
> This will change the colours used to represent the high and low intensity values in the different colour channels. Note also that the top-left image, is unaffected by the choice of colourmap, as it is a colour image displayed in true colour.
>
> At the MATLAB prompt execute the following command:
> `figure(2), imagesc(Ib); colorbar`
> This displays the elephant image. However, the colour map shows dark pixels as blue and light pixels as yellow. To display the image in greyscale, execute the command:
> `colormap('gray');`

Another useful command when displaying images is the axis command, this allows you to control the range of pixels that are displayed, the scaling applied to the x- and y-axes, and the appearance of the axes.

> At the MATLAB prompt execute the following command:
> `figure(2), axis('off')`
> This removes the numbers labelling the axes.
> `axis('equal')`
> This reshapes the image so that both the x- and y-axes are scaled equally (this is a square image, 512x512 pixels, so it is now displayed square).
>
> A similar result can be produced using imshow (without the need to change the colormap or axis attributes):
> `figure(3), imshow(Ib);`

However, in general, imagesc is much better than imshow. I strongly recommend using imagesc for subsequent labs.

6. Image Conversion

There are several MATLAB commands to convert images from one format to another:

Conversion:	**MATLAB command:**
from RGB format to indexed format.	`rgb2ind()`
from RGB format to intensity format.	`rgb2gray()`
from RGB format to binary format.	`im2bw()`
from indexed format to RGB format.	`ind2rgb()`
from indexed format to intensity format.	`ind2gray()`
from indexed format to binary format.	`im2bw()`
from intensity format to indexed format.	`gray2ind()`
from intensity format to binary format.	`im2bw()`
from a matrix to intensity format by scaling.	`mat2gray()`
Converts an image from uint8 to double data type.	`im2double()`
Converts an image from double to uint8 data type.	`im2uint8()`

At the MATLAB prompt execute the following commands:
```
Iad=im2double(Ia);
whos
```
Note that Iad takes up significantly more memory than Ia.

Now execute:
```
Ia(1:6,1:4,1)
Iad(1:6,1:4,1)
```
This displays the numerical values for a small range of elements from the red channel. You can see that Ia has integer values, while Iab has floating point values.

Now execute:
```
figure(3), imagesc(Iad);
```
This displays the rooster image that is stored using double precision floating point numbers. The image looks identical to the original, but the mapping that has been applied between numbers and colours is different.

At the MATLAB prompt execute the following commands:
```
Iag=rgb2gray(Ia);
whos
```
You will see that Iag is a 341x386 pixel image (or a 341x386 element matrix), with only one rather than 3 intensity channels.

Now execute:
```
figure(4), imagesc(Iag); colormap('gray'); colorbar
```
This displays the rooster image converted to greyscale.

7. Image Reshaping, Cropping and Creation

As seen in the previous exercise, we can select parts of an image using MATLAB's standard "colon" operator. We can also replace parts of an image using this technique. Sometimes it is convenient to represent an image as a vector rather than as a matrix. If `I` is an image in matrix form, `Iv = I(:);` will create a vector `Iv` whose elements are the columns of `I` appended end-to-end to form one column vector. To return a vector to matrix form, use `reshape`.

At the MATLAB prompt execute the following commands:
```
figure(1), clf
imagesc(Ib(100:320,80:380)); colormap('gray')
```
This display a part of the elephant image.
```
Ibr=Ib(100:320,80:380);
Ibr(150:200,1:50)=255;
imagesc(Ibr);
```
This makes a rectangular section of the image white.

We can use this technique to create artificial images:
```
Iart=zeros(201,201);
Iart(51:150,51:150)=1;
imagesc(Iart);
Iart(81:120,:)=0.5;
Iart(:,81:120)=0.75;
imagesc(Iart);

Iartv=Iart(:);
```

```
whos
Iartv(1:202:40101)=1;
imagesc(reshape(Iartv,201,201));
```

Cropping can also be done interactively with the command `imcrop`

8. Image Resizing

Scaling an image can be done with the command `imresize`

At the MATLAB prompt execute the following commands:

```
Ibsmall=imresize(Ib,0.5);
Iblarge=imresize(Ib,2);
figure(1), clf
subplot(2,2,1), imagesc(Ibsmall)
subplot(2,2,2), imagesc(Iblarge)
```

Notice the difference in the labels on the axes.

9. An Image Processing Example

At the MATLAB prompt execute the following commands:

```
Ibd=im2double(Ib);
Ibdiffv=Ibd(1:511,:)-Ibd(2:512,:);
figure(3), clf, imagesc(Ibdiffv);colormap('gray');colorbar
Ibdiffh=Ibd(:,1:511)-Ibd(:,2:512);
figure(4), clf, imagesc(Ibdiffh);colormap('gray');colorbar
```

In this example we subtract an image from a copy of the same image shifted one pixel upwards or leftwards. The result is two images in which most pixels have low intensity values. The few high intensity values that are significantly different from zero occur where there was a rapid change in intensity value between the image and the shifted image. This tends to occur at edges. This becomes clearer if you do the following:

```
Ibdiffv=abs(Ibdiffv);
figure(3), clf, imagesc(Ibdiffv); colormap('gray'); colorbar
Ibdiffh=abs(Ibdiffh);
figure(4), clf, imagesc(Ibdiffh); colormap('gray'); colorbar
```

We can combine the vertical and horizontal difference images into a single image, by calculating the L2-norm:

```
Ibdiff=sqrt(Ibdiffh(1:511,:).^2+Ibdiffv(:,1:511).^2);
figure(1), clf, imagesc(Ibdiff); colormap('gray'); colorbar
```

We can convert this to a binary image, and display it, as follows:

```
bw=im2bw(Ibdiff,0.075);
figure(2), clf, imagesc(bw); colormap('gray'); colorbar
```

10. Image Output

Once we are done processing an image, we may want to write it to an image file that can be read by other software. This can be achieved using the `imwrite` command. This command takes three arguments. The first specifies the name of the MATLAB variable that contains the image you wish to store, the second contains the name of the file you wish to store the image in, and the third specifies the file format you wish to use. Alternatively, we may wish to save the image in MATLAB format, this can be achieved using the `save` command.

At the MATLAB prompt execute the following commands:

```
imwrite(Ibdiff,'elephant_diff.png','png')
```

Alternatively, you might want to export a MATLAB figure window for inclusion in a report. This can be done using the MATLAB `print` command. You execute this command having selected the figure you wish to save, either by using the mouse, or by using the `figure` command.

At the MATLAB prompt execute the following commands:

```
figure(2)
print -dpdf bw_diff.pdf
```

This will produce a pdf file called bw_diff.pdf. By typing `help print` you can find out how to produce figures in other formats.