| | |
|---|---|
| **Started on** | Tuesday, 29 September 2020, 7:06 PM |
| **State** | Finished |
| **Completed on** | Monday, 5 October 2020, 4:45 AM |
| **Time taken** | 5 days 9 hours |
| **Marks** | 6.00/6.00 |
| **Grade** | **1.00** out of 1.00 (**100**%) |

**Feedback**

**Well done!**

**You have successfully passed this quiz.**

**Information**

# Introduction

This coursework provides an introduction to working with images in MATLAB. The information provided in this coursework will be required to complete subsequent coursework exercises.

Before starting you are required to have:

1. access to MATLAB (instructions for running MATLAB are available on the module's KEATS webpage).
2. familiarity with the MATLAB interface and programming language (links to general introductory MATLAB tutorials are available on the module's KEATS webpage.).

You will also need access to the module's KEATS webpage to download some images that will be used later in this, and subsequent, courseworks.

Note that commands which you can type at the MATLAB prompt are indicated by text in the following typeface:

```
matlab_function(parameter1, parameter2);
```

**Information**

## 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 user-defined MATLAB functions). You can also write MATLAB scripts which a m-files that just contain a sequence of MATLAB commands. As with built-in MATLAB functions, user-written functions and scripts 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).

For your this and subsequent courseworks it is recommended that you write a MATLAB script file, containing all your work, so that you can easily save your work and return to it at a later time.

To obtain information on any in-built MATLAB function, just type "`help function_name`" at the MATLAB prompt (replacing "function_name" with the name of the function you want to know about).

Common Pitfalls when working with images:

- Be sure not to forget the semicolon at the end of a command. Otherwise, you may sit for a while watching all the values from a large variable (such as an image) scroll by on the screen.
    - For example, create two large matrices containing random values by typing the following at the MATLAB prompt:
      `I1=rand(1000,1000)`
      `I2=rand(1000,1000);`
      The first command (without the semi-colon) prints lots of numbers to the screen, while the second command (with the semi-colon) does not.
- 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.
    - For example, to add together the values in the two random matrices you just created, you might be tempted to use nested for-loops, e.g.:
      ```
      for i=1:1000
         for j=1:1000
      ```

```
      Isum(i,j)=I1(i,j)+I2(i,j);
    end
end
```

However, it is much better (both in terms of code clarity and computation time) to add two matrices as follows:

```
Isum=I1+I2;
```

- Be sure to use .* instead of * when you are multiplying two images together element-by-element. Otherwise, MATLAB will do a matrix multiplication of the two, which will generate a meaningless result.
    - For example,

      ```
      ImultA=I1*I2;
      ```

      produces results that are very different from:

      ```
      ImultB=I1.*I2;
      ```

      Note that addition, as used in the preceding example, is an element-wise operation, so it is not necessary to use .+ (in fact, .+ is not defined).

**Information**

## Image Input

A digital image is composed of pixels. An image file contains information about the intensity and/or colour each pixel. To store information describing all the pixel values in a large image takes a lot of memory. Hence, images are often stored in a compressed format. Most images you find on the internet are JPEGs 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 a number of image file formats (these formats can be listed by executing the command `imformats`).

Copy the rooster.jpg and elephant.png files from the module's KEATS webpage to the directory in which you are running MATLAB. To read these image files into MATLAB, execute the following commands:

```
Ia=imread('rooster.jpg');
Ib=imread('elephant.png');
```

**Information**

## Image Representation

Once an image file has been read, it is stored internally using one of several formats. The most common formats used by MATLAB are:

- **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. There are two main ways to store the number that represents the brightness of the pixel. The double data type assigns a floating-point number between 0.0 and 1.0 to each pixel, the value 0.0 corresponds to black and the value 1.0 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.
- **Binary images**
  This image format also stores an image as a matrix but can only represent pixels as black or white (and nothing in between). It assigns a 0 for black and a 1 for white.
- **RGB images**
  This is a format for color images. It represents an image using a 3-dimensional matrix. The third dimension (sometimes called the colour channels of the image) 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 has.
- **Indexed images**
  An indexed image stores an image using 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. Each of the numbers in the first matrix is an instruction of what number to use from the color map matrix to define the pixel's colour.

If you have been following the instructions so far, you will have a number of variables in memory. You can get information about these variables, by looking at the workspace window, or by typing `whos` at the MATLAB prompt. If you do that you should get:

```
Name           Size                Bytes  Class     Attributes

I1          1000x1000            8000000  double
I2          1000x1000            8000000  double
Ia           341x386x3            394878  uint8
Ib           512x512              262144  uint8
ImultB      1000x1000            8000000  double
Isum        1000x1000            8000000  double
```

You will see that `Ia` (the MATLAB variable representing the rooster image) is a 386-by-341 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 512-by-512 pixel greyscale image (or equivalently a 2-dimensional matrix), with pixel intensity values stored using uint8 format. The random matrices you produced, and the ones other matrices derived from them, (i.e. `I1, I2, Isum, ImultA, ImultB`) are 1000-by-1000 element 2-dimensional matrices, with values stored using the double format. You can think of these as synthetic, grayscale, images.

**Question 1** Complete Not graded

## Image Display

MATLAB provides a number of different commands for displaying an image on the screen; `imshow, image,` and `imagesc`. However, in general, imagesc is nearly always the most appropriate. So we will use this in the current coursework, and you should use it in subsequent courseworks.

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.

To illustrate how these commands work, at the MATLAB prompt execute the following:
`figure(1), subplot(2,2,1), imagesc(Ia);`
This displays the image of the rooster which you read from a file earlier.

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 red, green, and blue channels. Note also that the top-left image, is unaffected by the choice of colormap, 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. The colour map shows dark pixels as blue and light pixels as yellow. To display the image in greyscale change the colormap to gray by executing 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. For example, 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).

To check how you are doing please enter a number between 0 and 100: 98 and then click:

If you have been following the instructions so far, you should have two figures that look like this:

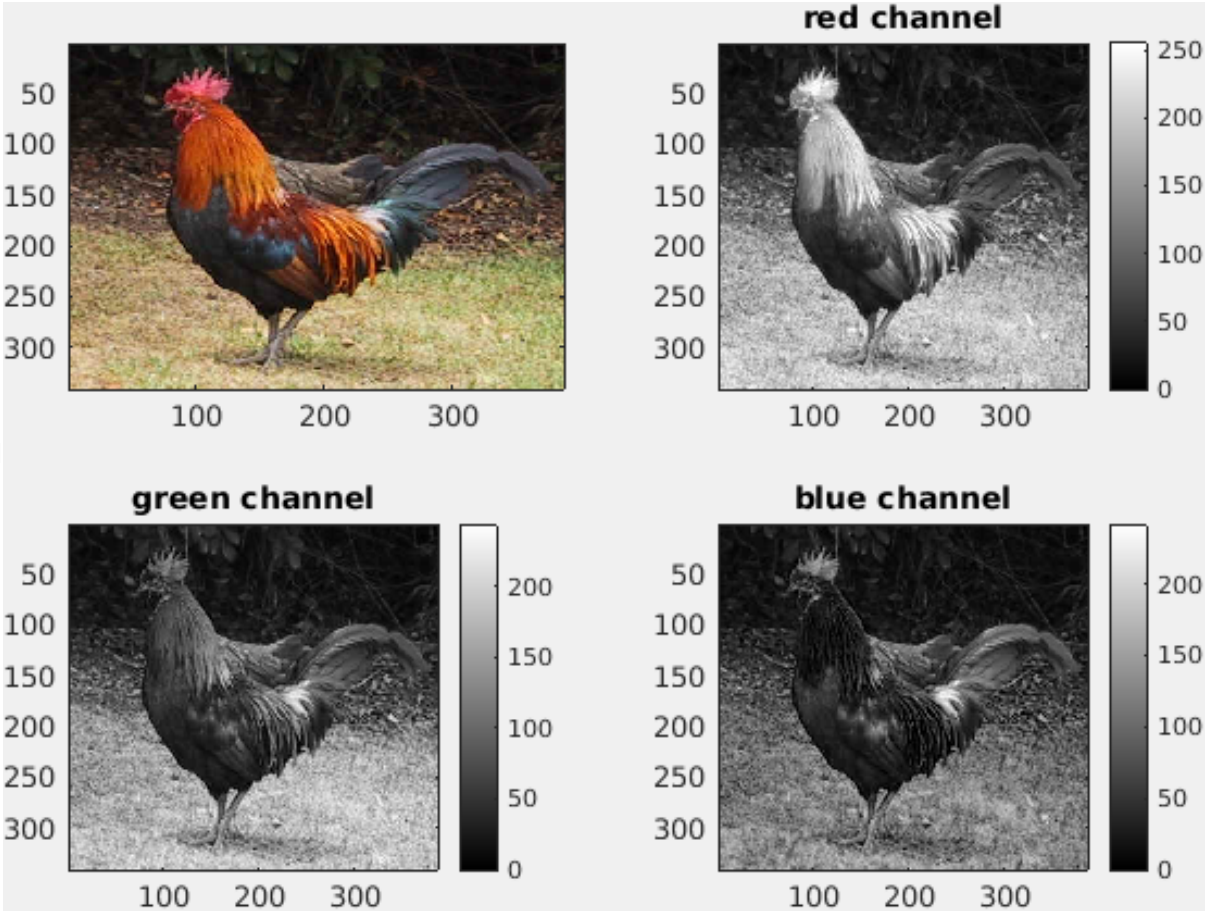


**Figure 1**

**Figure 2**


If not, please go back and follow the instructions until you do.

**Question 2** Correct Mark 3.00 out of 3.00

## Reading Values From An Image

As seen in the previous section, one way to examine the pixels values in an image is to use `imagesc` to show (a single channel of) the image on the screen, and `colorbar` to show a scale that can be used to read-off the approximate values.

To see the raw numerical values, you could just type the name of the variable at the MATLAB prompt (without a trailing semicolon), e.g.:
`Ib`
This will show the contents of the variable, but if the variable is an image (Ib is the elephant image) it is likely to produce many values that will be difficult to interpret.

It is possible to be more selective about what values are shown on the screen. For example, execute:
`Ib(3,:)` This will show all the pixel values in the 3rd row of the elephant image.
`Ib(:,3)` This will show all the pixel values in the 3rd column of the elephant image.
`Ib(1:6,1:4)` This displays the numerical values for a patch of the elephant image in the top-left corner (pixels within the first 6 rows and the 1st 4 columns).
`Ia(1:6,1:4,1)` This displays the numerical values for a patch of the rooster image in the top-left corner (note that these values are from the 1st channel of the rooster image, the red channel). To see the values in the green channel for the same patch, we use:
`Ia(1:6,1:4,2)`

Note that both the elephant image (`Ib`) and the rooster image (`Ia`) have integer pixel values (as discussed earlier in the "Image Representation" section of this coursework). Also, that `Ib` is a 2-dimensional matrix (as the elehant image is an intensity or greyscale image), while `Ia` is a 3-dimensional matrix (as the rooster image is a colour, RGB, image).

MATLAB treats images as matrices. Values in matrices are indexed using (row,column) coordinates. Hence, to obtain the value at row 3 and column 4 of the elephant image, you would use:
`Ib(3,4)`

However, you should be aware that in computer vision (and in the lectures and tutorials) we index images using (x,y) coordinates: this will be discussed in Lecture 2.

## EXERCISE:

For the elephant image, determine the values at the following coordinates:

Pixel intesity at row=72, column=236: 144 ✓

Pixel intesity at row=31, column=227: 144 ✓

Pixel intesity at row=81, column=112: 144 ✓

If you have got the correct answers, well done! If you got the wrong answers, here is what you should have done:

Pixel intesity at row=72, column=236: `Ib(72,236)`

Pixel intesity at row=31, column=227: `Ib(31,227)`

Pixel intesity at row=81, column=112: `Ib(81,112)`

**Question 3** Correct Mark 3.00 out of 3.00

## Image Conversion

As discussed earlier in the "Image Representation" section of this coursework, there are a number of different ways of representing images. MATLAB provides 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()` |

**EXERCISE:**

Using the above information, convert the rooster image from RGB to intensity format. You should see that the converted image is represented by a 341x386 element matrix, rather than a 341x386x3 element matrix. Now convert the rooster image in intensity format (the image you just created) from uint8 to double format. You should see that the image takes up more memory after this conversion.

For the image you have just created via the two conversions, determine the values at the following coordinates:

Pixel intesity at row=154, column=91: 0.1765 ✓

Pixel intesity at row=227, column=230: 0.1569 ✓

Pixel intesity at row=275, column=37: 0.7059 ✓

If you got the wrong answers, here is what you should have done (you could, obviously, have chosen to call your variables by different names):

To convert from RGB to intensity format:

```
Iag=rgb2gray(Ia);
```

To convert from RGB to intensity format:

```
Iagd=im2double(Iag);
```

To get the requested pixel values you should have done the following:

```
Iagd(154,91)
```

```
Iagd(227,230)
```

```
Iagd(275,37)
```

◀ slides01d_tackling_the_problem_of_vision

Jump to...

Tutorial Questions ▶