Computer Vision
Workshop 2
MATLAB Images

Aims of the workshop

The aim of this workshop is to introduce images, their properties, and manipulation within MATLAB. This builds upon Workshop 1. Previously, in lectures, we've looked at Colour Models, resizing, and histogram representation (with equalisation). Today, we will explore the use of histograms on images, as well as the impact of resizing images.

Please see 'Useful Information' below on how to lookup certain MATLAB functionality. The concept behind this workshop is about discovery, and experimentation surrounding topics covered so far.

Feel free to discuss the work with peers, or with any member of the teaching staff.

Useful Information

MATLAB Documentation

MATLAB functionality mentioned during lecture or workshops is fully-documented and available at: https://uk.mathworks.com/help/matlab/

You may find the “Getting Started” section of this documentation to be useful:
https://uk.mathworks.com/help/matlab/getting-started-with-matlab.html

In addition, MATLAB Answers may be useful in finding common solutions to any issues you may encounter:
https://uk.mathworks.com/matlabcentral/answers/help?s_tid=gn_mlc_ans_hlp

Useful functions: imread, size, figure, subplot, imshow, imhist, imresize, histeq, rgb2hsv, rgb2ycbcr, rgb2lab

Reminder

We encourage you to discuss the contents of the workshop with the delivery team, and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

Note: In these workshops, and the ACW, you can work in either .m or a .mlx “live script”. Feel free to experiment with using either, and see which you find most useful.

Loading and Inspecting Images

Exercise1: Use *imread* to read in circles.png, coins.png, cameraman.tif, and lighthouse.png. Using the information below, assign these as variables. E.g

```
CameraManGreyscale = imread(‘cameraman.tif’);
```

Note - These are special in the computer vision toolbox. MATLAB will know what these images are (they are standard in computer vision) even if you don’t have the image in your directory. Any images you wish to load yourself (that you have downloaded) will follow the normal file-paths.

Circles.png - A binary image
Coins.png - Greyscale
Camerman.tif - Greyscale
Lighthouse.png - Colour Image

Exercise 2: Look at the **Workspace** panel. You should see the variables that you created. Double clicking on the variable name will allow you to inspect the raw matrix values.

The value column, in the workspace panel, also shows the dimensions of the matrix as well as the data type. Compare how the binary image, the greyscale image, and the colour image vary in their data type; inspect the values of each variable.

Exercise 3: Some MATLAB function calls can return more than one value. To show this, let’s get the *size* of one of our image variables.

```
[imgHeight, imgWidth] = size( CameraManGreyscale );
```

a) Obtain the height, and width of each of your images. Does this match up with what the Workspace panel tells you? You can use the ***disp*** function to print out imgHeight, or imgWidth in the console.
b) What happens if you forget the **;** ?

Exercise 4: Create a new ***figure*** and use ***subplot*** to make a 2x2 grid to ***imshow*** each of your images you have loaded in.

```
figure
subplot( 2, 2, 1); % A 2x2 arrangement. First item.
imshow( … );
subplot( 2, 2, 2); % Same 2x2 arrangement. Second item.
imshow( … );
…
subplot( 2, 2, 4); % Same 2x2 arrangement. Last item.
imshow( … );
```

You should obtain something similar to the following:

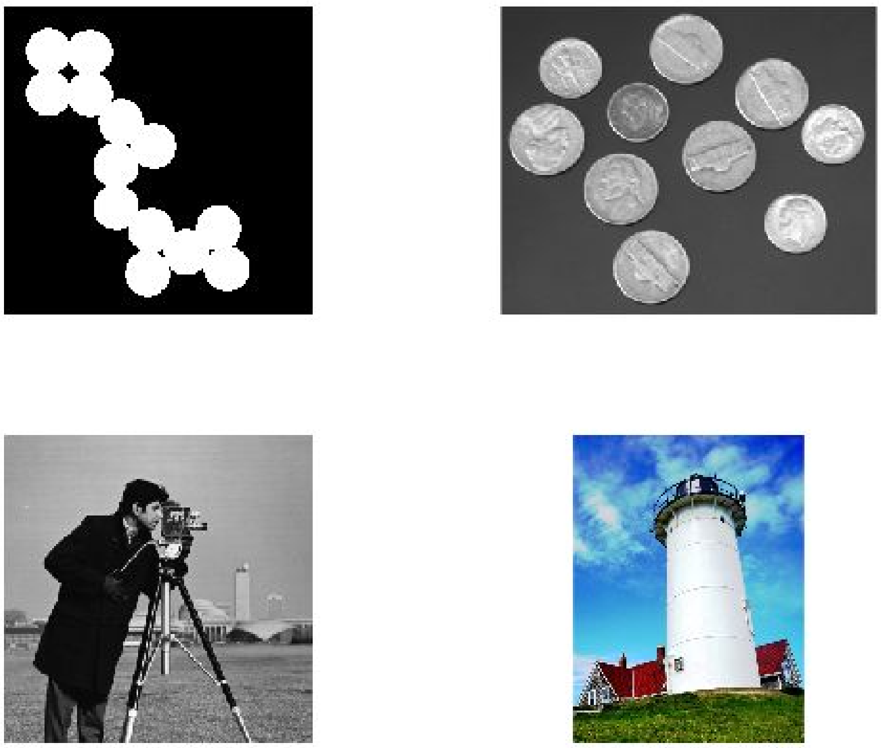


Image Histograms and Resizing

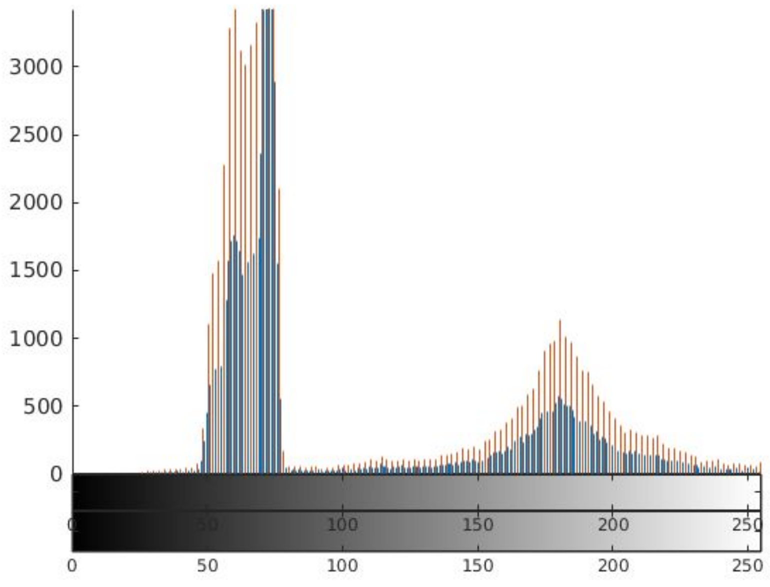
You may delete all variables from your workspace, except coins and lighthouse images.

Exercise 5: *imhist* can be used to calculate the histogram of an input image. It returns the count of each bin (how many items fall into that bin) as an array, as well as binLocations (what values are considered bin 0, bin 1, etc, also as an array.

1. Obtain the [counts, binLocations] from passing *imhist* the coins greyscale image.
2. Imhist allows you to specify the number of bins you want for the histogram. Call imhist(Coins_Image, 128); to obtain 128 bins.

Exercise 6: If we just call *imhist*, without assigning it to our variables, it will display our histogram for us. We can combine this with *figure,* and *hold on/off* to display our histogram, and overlay another on-top.

```
figure;
hold on;
imhist(Coins_Image); % Display histogram of your coins image.
imhist(Coins_Image, 128); % Display histogram with 128 bins.
hold off;
```



What happens as we reduce the number of bins towards 0?

Exercise 7: Using *imresize*, resize your coins image to 0.25 (25% per-axis = 6.25% of original pixels), assigning this to a new variable: coints_resized. Display this using imshow. What differences can you see between the resized and the original?

Exercise 8: Using *imresize* again, resize **coins_resized** by 5. (5x as big per-axis!); assigning this to a new variable (name of your choice).

1. Compare the *size* of this image with your original, it should be almost identical (It may not be as 246 / 5 = 49.2; doesn't divide nicely).
2. Compare the quality of the re-resized image, what do you notice?

Exercise 9: imresize also accepts a string, the type of interpolation to perform when resizing. We briefly covered this during lectures.

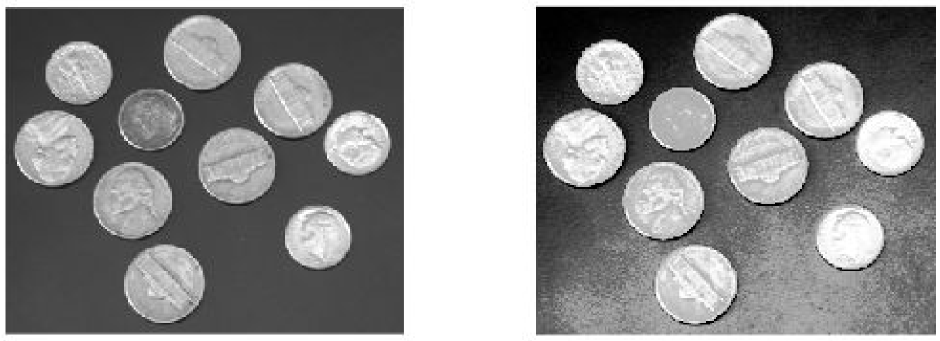
```
    X = imresize(img, 3, 'bilinear');
```

Try resizing your images to be 3x larger using bilinear, and nearest. What do you notice?

Exercise 10: We can perform histogram equalisation easily within MATLAB by utilising the *histeq* function. We can create a new image (variable) from the output. E.g

```
    coinsEq = histeq(Coins_Image);
```

Using *figure* and *subplot*, put the original coins side-by-side with the equalisation. Note: Take care of which way you define subplot. E.g subplot(1, 2, …) vs subplot(2, 1, …).



Colour Models and Thresholding

Exercise 11: Using our colour image we previously loaded in (lighthouse), we can extract the red channel, blue channel, and green channel using slice notation.
Colon (:) simply says to take ALL the values from lowest-highest in that dimension. As we know in our colour model, that the first two dimensions are spatial (X direction, Y direction), we want ALL our pixels, but taking only their 1st component, the red channel. Remember: RGB.

```
redchannel = my_lighthouse_image(:, :, 1);
greenchannel = my_lighthouse_image(:, :, 2);
Bluechannel = my_lighthouse_image(:, :, 3);
```

1. Use *imshow* to display each component channel individually.
2. Use *figure* and *subplot* to plot these all side-by-side.

Exercise 12: MATLAB makes it easy to convert between colour models/representations. Our colour image is currently in RGB. We can use *rgb2hsv* to convert our model to HSV, which might make our job easier. E.g

```
HSV_lighthouse = rgb2hsv( my_lighthouse_image );
```

Once you have converted the image to HSV, create new variables for hue, saturation, and value, and get those channels from the image. Hint: Hue is the 1st channel. Note how the images look entirely different.

Exercise 13: Convert the RGB image into the YCbCR colour space using *rgb2ycbcr* investigating the colour channels. Feel free to reference the lecture slides.

Exercise 14: Convert the RGB image into the L*a*b* colour space using *rgb2lab* investigating the colour channels. Feel free to reference the lecture slides.