Lab 1A: Introduction to MATLAB

CM30080

Dr Matthew Brown

Introduction

This worksheet will guide you through basic usage of MATLAB for image manipulation. When you have finished this worksheet you can move on to Lab 1B: Pyramid Blending.

Those with no previous MATLAB experience might find the following tutorial useful `http://www.mathworks.co.uk/help/pdf_doc/matlab/getstart.pdf`. You should be familiar with how to create and manipulate matrices, and create your own scripts and functions (e.g., sections 1-6, 1-24, 2-12). You should also be familiar with multidimensional arrays (used to store colour images) and cell arrays (used to store multiple differently sized images, for example). See section 2-28 for help on these types of arrays.

Basic Image Manipulation

Open a MATLAB command window and ensure that you have a suitable test image, e.g., `'tst.jpg'` in the current working directory. Type the following commands at the MATLAB command prompt:

```
imc = double(imread('tst.jpg'))/255;
im = mean(imc, 3);
figure; imshow(imc);
figure; imshow(im);
```

The `mean(imc, 3)` command computes the mean of the image on the 3rd dimension, which averages together the R, G, B colour bands giving a grayscale image. The `/255` is there to normalise the original 8-bit (0-255) colour values into the 0-1 range.

You can index pixels one at a time, e.g., `im(5, 10)` refers to the pixel at row 5, column 10 in the grayscale image, and `imc(5, 10, 2)` refers to the "green" band (band 2) at the same location in the colour image. Try the following code, which creates a bilinear ramp in a monochrome image, and then modify it to set different values in each colour band:

```
imr = zeros(10, 10);
for i = 1:10
  for j = 1:10
    imr(i, j) = i * j / 100;
  end;
end;
```

```
imr = imresize(imr, 10, 'nearest');
figure; imshow(imr);
```

Note that the `imresize(imr, 10, 'nearest')` command scales the image by a factor of 10 using nearest neighbour interpolotion, for visualisation purposes. You can also index pixels in groups using colon notation, e.g.,

```
im10 = im(1:10, 1:10);
im10c = im(1:10, 1:10, 1:3);
im10
imc
```

Convolution

To perform convolution of an image in MATLAB you can use the command `conv2`. Try the following, which convolves our test image with a horizontal gradient filter and then writes the result to file:

```
kd = [-1 1];
imx = conv2(im, kd);
imwrite(imx, 'imx.jpg');
```

Check the size of the image before and after the operation using:

```
[rows1, cols1, bands1] = size(im);
[rows2, cols2, bands2] = size(imx);
```

Note that the number of columns has increased by 1. You can ensure that the output image has the same size as the input using:

```
imx = conv2(im, kd, 'same');
```

Modify the above code to compute a colour derivative image. Hint: you can extract a colour band using a single colon to specify all row/column elements, e.g., `im_red = ... imc(:, :, 1);`.

Now try convolving an image with a smoothing filter as follows:

```
ks = [1 2 4 2 1];
ks = ks / sum(ks);
ims = conv2(im, ks, 'same');
```

What is the effect of this filter? Try convolving with the transpose of this filter (`ks'`) instead (the `'` symbol denotes transposition in MATLAB). What do you notice? Now try convolving with both (rows followed by columns). Separable filtering like this can be done in a single command in MATLAB, so the following code has the same effect as convolution by `ks` on the rows and columns of `im`:

```
ims = conv2(ks, ks, im, 'same');
```

Image Normalisation

In the previous section you created a horizontal derivative image that contained negative elements, these could not be viewed and were lost when written to file. Often it is useful to scale an image into the viewable range to see negative or out of range values. In this section you will write a function to do this.

Create a file called `NormaliseImage.m` in the current working directory containing the following:

```
function imn = NormaliseImage(im)
```

This specifies a function that takes an input array `im` and returns a normalised output `imn`. We will now write code to scale the ouptut image to $0-1$. First we need to compute the max and min of the image:

```
mx = max(im(:));
mn = min(im(:));
```

Now write your own code to offset and scale the image pixels so that the new min value is 0 and the max value 1. Note that you can scale a matrix `A` by a scalar `s` simply by multiplying `A * s` or dividing `A / s`. Test your function by normalising and viewing the horizontal gradient image (`conv2(im, [-1 1])` and verifying that you can see positive and negative edges.

Image Resize

Test resampling your image by a factor of 2 as follows:

```
im2 = imresize(im, 0.5);
```

Check the size before and after using the `size` command:

```
[nRows, nCols, nBands] = size(im);
```

What happens if one of the dimensions is odd? Reconstruct a larger size image using `imr = imresize(im, 2.0)`. Compare `imr` and `im`. What do you notice? Note that if `imr` is larger than `im` you may have to discard the last row or column to reconstruct an image of the same size. This can be done using the colon operator in MATLAB, e.g., `imcrop = imr(1:10, 1:10, 1:3)` would select the first 10 rows and columns.

Cell Arrays

Note that you can store differently sized images in a cell array as follows:

```
cell_arr = [];
cell_arr{1} = im;
```

```
cell_arr{2} = imresize(im, 0.5);
cell_arr{3} = imresize(im, 2.0);
```

They can then be accessed using curly bracket notation, i.e., `imt = cell_arr{2}`.

Conclusion

This worksheet has introduced some basic MATLAB commands needed for the subsequent labs. For more info on any MATLAB command you can type `help command` at the command prompt, or `doc command` to see the command in context in the documentation browser.