Signal and Image Processing Course (SIP 2014)

Week 1

Herein you will find both the hands-on exercises to help you learn the week's material (Monday: Exercises) and the week's assignments for which you will need to submit INDIVIDUAL reports (Wednesday: Assignment).

Monday: Exercises

In order to ensure that you have understood the reading material, follow the code examples given below.

THESE EXERCISES ARE NOT TO BE SUBMITTED AND WILL NOT BE ASSESSED - THEY ARE FOR YOUR OWN BENEFIT TO HELP YOU UNDERSTAND THE TOPICS.

NOTE: some exercises are different than those in the book. Type the code into Matlab. You may find it best to enter all the code into a single '.m' file, separating the examples into different 'cells' using the '%%' characters. Use the help function to read about those functions which you have not used previously. Write down or save any results, and save any figures or resultant images on disk - it will help you remember and you can go back to it in the future.

Finally, if you don't understand something, ASK!

Chapter 1 - Representation

Example 1.1

Type in the following code:

```
imfinfo('cameraman.tif')
A=imread('cameraman.tif');
imwrite(A,'cameraman.jpg','jpg');
imfinfo('cameraman.jpg')
```

The output of 'imfinfo' is different for different image file formats, but the first few fields are common and always returned. How many common fields are there?

Where is the JPEG file saved?

What coding method was used to save the JPEG file?

Why is the file size different for the PNG and JPEG files?

Example 1.2

Type in the following code and notice the changes occurring to the image:

```
A=imread('cameraman.tif');      % Read in intensity image
imshow(A);                      % First display image using imshow
imagesc(A);                     % Next display image using imagesc
axis image;                     % Correct aspect ratio of displayed image
axis off;                       % Turn off the axis labelling
colormap(gray);                 % Display intensity image in gray-scale
```

Example 1.3

Type in the following code and notice the changes occurring to the image:

```
B=rand(256).*1000;      % Generate random image array in range 0-1000
imshow(B);              % Poor contrast results using imshow because data
                        % exceeds expected range
imagesc(B);             % imagesc automatically scales colourmap to data range
axis image; axis off;
colormap(gray); colorbar;
imshow(B,[0 1000]);     % But if we specify range of data explicitly then
                        % imshow also displays correct image contrast
```

Example 1.4

Type in the following code and notice the changes occurring to the image:

```
B=imread('cell.tif');       % Read in 8-bit intensity image of cell
C=imread('spine.tif');      % Read in 8-bit intensity image of spine
D=imread('onion.png');      % Read in 8-bit colour image.

subplot(3,1,1); imagesc(B); axis image;     % Creates a 3 by 1 mosaic of plots
axis off; colormap(gray);                   % and display 1st image

subplot(3,1,2); imagesc(C);     % Display 2nd image
axis off; colormap(jet);        % Set colourmap to jet (false colour)

subplot(3,1,3); imshow(D);      % Display 3rd (colour) image
```

Try resizing the figure window to tall-and-thin and short-and-wide. Notice what happens to the different images.

Example 1.5

Type in the following code and notice the changes occurring to the image:

```
B=imread('cell.tif');       % Read in 8-bit intensity image of cell
imtool(B);    % Use the cursor to examine grayscale image in interactive viewer

D=imread('onion.png');      % Read in 8-bit colour image.
imtool(D);    % Use the cursor to examine RGB image in interactive viewer

B(25,50)                  % print pixel value at location (25,50)
B(25,50) = 255;             % set pixel value at (25,50) to white
imshow(B);                 % view resulting changes in image

D(25,50, :)                 % print RGB pixel value at location (25,50)
D(25,50, 1)                  % print only the red value at (25,50)
D(25,50, :) = [255, 255, 255];  % set pixel value to RGB white
imshow(D);                    % view resulting changes in image
```

After changing the pixel value of B or D, does the image in the viewer update or not?

Where is the image origin? Is the order of (X, Y) the same in the imtool window as on the Maltab command line?

Example 1.6

Type in the following code:

```
D=imread('onion.png');      % Read in 8-bit RGB colour image.
Dgray = rgb2gray(D);        % convert it to a grayscale image

subplot(2,1,1); imshow(D); axis image;
subplot(2,1,2); imshow(Dgray);        % Display both images in the same figure
```

Example 1.7

Type in the following code:

```
D=imread('onion.png');      % Read in 8-bit RGB colour image.
```

```
Dred  = D(:,:,1);          % extract red channel   (1st channel)
Dgreen = D(:,:,2);          % extract green channel (2nd channel)
Dblue = D(:,:,3);          % extract blue channel  (3rd channel)

figure;     % Create a new figure window
subplot(2,2,1); imshow(D); axis image;       % display all in 2x2 plot
subplot(2,2,2); imshow(Dred); title('red');
subplot(2,2,3); imshow(Dgreen); title('green');
subplot(2,2,4); imshow(Dblue); title('blue');
```

Exercise 1.1

Using the examples presented for displaying an image in Matlab together with those for accessing pixel locations, investigate adding and subtracting a scalar value from an individual location, i.e. I(i,j) = I(i,j) + 25 or I(i,j) = I(i,j) - 25. Start by using the grey-scale 'cell.tif' example image and pixel location (100, 20). What is the effect on the grey- scale colour of adding and subtracting?
Expand your technique to RGB colour images by adding and subtracting to all three of the colour channels in a suitable example image. Also try just adding to one of the individual colour channels whilst leaving the others unchanged. What is the effect on the pixel colour of each of these operations?

Exercise 1.2

Based on your answer to Exercise 1.1, use the for construct in Matlab (see help for at the Matlab command prompt) to loop over all the pixels in the image and brighten or darken the image.
You will need to ensure that your program does not try to create a pixel value that is larger or smaller than the pixel can hold. For instance, an 8-bit image can only hold the values 0–255 at each pixel location and similarly for each colour channel for a 24-bit RGB colour image.

Exercise 1.3

Using the grey-scale 'cell.tif' example image, investigate using different false colour maps to display the image. The Matlab function colormap can take a range of values to specify different false colour maps: enter help graph3d and look under the Color maps heading to get a full list. What different aspects and details of the image can be seen using these false colourings in place of the conventional grey-scale display?
False colour maps can also be specified numerically as parameters to the colormap command: enter help colormap for further details.

Exercise 1.4

Load an example image into Matlab and using the functions introduced in Example 1.1 save it once as a JPEG format file (e.g. sample.jpg) and once as a PNG format image (e.g. sample.png). Next, reload the images from both of these saved files as new images in Matlab, 'Ijpg' and 'Ipng'. We may expect these two images to be exactly the same, as they started out as the same image and were just saved in different image file formats. If we compare them by subtracting one from the other and taking the absolute difference at each pixel location we can check whether this assumption is correct.

Use the imabsdiff Matlab command to create a difference image between 'Ijpg' and 'Ipng'. Display the resulting image using imagesc.
The difference between these two images is not all zeros as one may expect, but a noise pattern related to the difference in the images introduced by saving in a lossy compression format (i.e. JPEG) and a lossless compression format (i.e. PNG). The difference we see is due to the image information removed in the JPEG version of the file which is not apparent to us when we look at the image. Interestingly, if we view the difference image with imshow all we see is a black image because the differences are so small they have very low (i.e. dark) pixel values. The automatic scaling and false colour mapping of imagesc allows us to visualize these low pixel values.

Chapter 2 - Formation

Example 2.1

Type in the following code and try to understand what is going on:

```
f = ones(64,1); f = f./sum(f);  % Define rectangle signal f and normalize
g = conv(f,f); g = g./sum(g);  % Convolve f with itself to give g and normalize
h = conv(g,g); h = h./sum(h); % Convolve g with itself to give h and normalize
j = conv(h,h); j = j./sum(j);     % Convolve h with itself to give j and normalize

subplot(2,2,1),plot(f,'k-'); axis square; axis off;
subplot(2,2,2),plot(g,'k-'); axis square; axis off;
subplot(2,2,3),plot(h,'k-'); axis square; axis off;
subplot(2,2,4),plot(j,'k-'); axis square; axis off;
```

Matlab function: conv.
This example illustrates the repeated convolution of a 1-D uniform signal with itself. The resulting signal quickly approaches the form of the normal distribution, illustrating the central limit theorem of statistics.

Example 2.2

Type in the following code and try to understand what is going on:

```
A = imread('trui.png');
PSF = fspecial('gaussian',[5 5],2);    % Define Gaussian convolution kernel
h = fspecial('motion',10,45);    % Define motion filter
B = conv2(PSF,double(A));       % Convolve image with convolution kernel
C = imfilter(A,h,'replicate');     % Convolve motion PSF using alternative function
D = conv2(double(A),double(A)); % Self-convolution - motion blurred with original
subplot(2,2,1),imshow(A);       % Display original image
subplot(2,2,2),imshow(B,[]);    % Display filtered image
subplot(2,2,3),imshow(C,[]);    % Display filtered image
subplot(2,2,4),imshow(D,[]);   % Display convolution image with itself (autocorrln)
```

Matlab functions: conv2, imfilter.
This example illustrates 2-D convolution. The first two examples show convolution of the image with a Gaussian kernel using two related Matlab functions. The final image shows the autocorrelation of the image given by the convolution of the function with itself.

Example 2.3

Type in the following code and try to understand what is going on:

```
A=imread('cameraman.tif');   % Read in an image
[rows dims]=size(A);         % Get image dimensions
Abuild=zeros(size(A));       % Construct zero image of equal size

% Randomly sample 1% of points only and convolve with gaussian PSF
sub=rand(rows.*dims,1)<0.01; % Generate a vector of random sample points
Abuild(sub)=A(sub);
h=fspecial('gaussian',[10 10],2);
B10=filter2(h,Abuild);
subplot(1,2,1), imagesc(Abuild);
axis image; axis off;colormap(gray); title('Object points')
subplot(1,2,2), imagesc(B10);
axis image; axis off;colormap(gray); title('Response of LSI system')
```

What does the ‘2’ specify in the ‘fspecial’ command?
Is ‘filter2’ performing convolution or correlation? What is the difference?

Exercise 2.1

Using Example 2.3 we can investigate the construction of a coherent image from a limited number of point samples via the concept of the PSF introduced in Section 2.1.
Here, we randomly select 1% (0.01) of the image pixels and convolve them using a Gaussian-derived PSF (see Section 4.4.4) applied as a 2-D image filter via the filter2() function. Experiment with this example by increasing the percentage of pixels randomly selected for PSF convolution to form the output image. At what point is a coherent image formed? What noise characteristics does this image show? Additionally, investigate the use of the fspecial() function and experiment with changing the parameters of the Gaussian filter used to approximate the PSF. What effect does this have on the convergence of the image towards a coherent image (i.e. shape outlines visible) as we increase the number of image pixels selected for convolution?

Exercise 2.6

High dynamic range (HDR) imaging is a new methodology within image capture (formation) whereby the image is digitized with a much greater range of quantiza- tion levels (Section 2.3.2.1) between the minimum and maximum colour levels. Matlab supports HDR imaging through the use of the hdrread() function. Use this function to load the Matlab example HDR image ‘office.hdr’ and convert it to a conventional RGB colour representation using the Matlab tonemap() function for viewing (with imshow()). Use the Matlab getrangefromclass() function and imtool() to explore the value ranges of these two versions of the HDR image.

Chapter 3 - Pixels

Example 3.1

Type in the following code and notice the changes occurring to the image:

```
A=imread('cameraman.tif');   % Read in image
subplot(1,2,1), imshow(A);    % Display image
B = imadd(A, 100);            % Add 100 pixel values to image A
subplot(1,2,2), imshow(B);    % Display result image B
```

What is the advantage of using ‘imadd’ instead of simple addition?

Example 3.2

Type in the following code and notice the changes occurring to the image. You will need to use the images downloaded from the course homepage (cola1.png and cola2.png):

```
A=imread(‘cola1.png');  % Read in 1st image
B=imread(‘cola2.png'); % Read in 2nd image

subplot(1,3,1), imshow(A); % Display 1st image
subplot(1,3,2), imshow(B); % Display 2nd image

Output = imsubtract(A, B); % subtract images
subplot(1,3,3), imshow(Output); % Display result
```

Example 3.3

Replace the last two lines of Example 3.2 with the following:

```
Output = imabsdiff(A, B); % subtract images
subplot(1,3,3), imshow(Output); % Display result
```

Compare the two subtraction methods - which is best?

Example 3.4

Replace the last two lines of Example 3.2 with the following:

```
Output = immultiply(A,1.5);     % multiple image by 1.5
subplot(1,3,3), imshow(Output); % Display result

Output = imdivide(A,4);        % divide image by 4
subplot(1,3,3), imshow(Output); % Display result
```

Are these results useful?

Example 3.5

Type in the following code and notice the changes occurring to the image.

```
A=imread('cameraman.tif');   % read in image
subplot(1,2,1), imshow(A);    % display image
B = imcomplement(A);          % invert the image
subplot(1,2,2), imshow(B);    % display result image B
```

Is this method useful?

Example 3.6

Type in the following code and notice the changes occurring to the image. You will need to use the images downloaded from the course homepage (toycars1.png and toycars2.png):

```
A=imread('toycars1.png');  % Read in 1st image
B=imread('toycars2.png');  % Read in 2nd image

subplot(2,3,1), imshow(A);
subplot(2,3,2), imshow(B);
subplot(2,3,3), imshow(imabsdiff(A, B));

Abw=im2bw(A);    % Automatically convert to binary
Bbw=im2bw(B);    % Automatically convert to binary

subplot(2,3,4), imshow(Abw);
subplot(2,3,5), imshow(Bbw);

Output = xor(Abw, Bbw);   % Perform XOR of images
subplot(2,3,6), imshow(Output);   % Display result
```

Notice the following:

- the images are first converted to binary using the Matlab im2bw function (with an automatic threshold - Section 3.2.3).
- the resulting images from the im2bw function and the xor logical operation is of Matlab type 'logical'.
- the operators for AND is '&' whilst the operator for OR is '|' and are applied in infix notation form as A & B, A | B.

Is the result as you would have expected?

Example 3.7

Type in the following code and notice the changes occurring to the image.

```
I=imread('trees.tif');     % Read in 1st image
T=im2bw(I, 0.1);           % perform thresholding
subplot(1,2,1), imshow(I);     % Display original image
subplot(1,2,2), imshow(T);     % Display thresholded image
```

Try a few different thresholds and see how the output changes.

Example 3.8

Type in the following code and notice the changes occurring to the image.

```
I=imread('cameraman.tif'); % Read in image
subplot(2,2,1), imshow(I); % Display image

Id=im2double(I);
Output1=2*log(1+Id);
Output2=3*log(1+Id);
Output3=5*log(1+Id);

subplot(2,2,2), imshow(Output1); % Display result images
subplot(2,2,3), imshow(Output2);
subplot(2,2,4), imshow(Output3);
```

Which areas of the image gain more visible detail, and which lose it?

Example 3.9

Type in the following code and notice the changes occurring to the image.

```
I=imread('cameraman.tif');   % Read in image
subplot(2,2,1), imshow(I);   % Display image
```

```
Id=im2double(I);
Output1=4*(((1+0.3).^(Id)) - 1);
Output2=4*(((1+0.4).^(Id)) - 1);
Output3=4*(((1+0.6).^(Id)) - 1);

subplot(2,2,2), imshow(Output1);   % Display result images
subplot(2,2,3), imshow(Output2);
subplot(2,2,4), imshow(Output3);
```

Which areas of the image gain more visible detail, and which lose it?

Example 3.10

Type in the following code and notice the changes occurring to the image.

```
I=imread('cameraman.tif'); % Read in image
subplot(2,2,1), imshow(I); % Display image

Id=im2double(I);
Output1=2*(Id.^0.5);
Output2=2*(Id.^1.5);
Output3=2*(Id.^3.0);

subplot(2,2,2), imshow(Output1); % Display result images
subplot(2,2,3), imshow(Output2);
subplot(2,2,4), imshow(Output3);
```

Which areas of the image gain more visible detail, and which lose it?

Example 3.11

Type in the following code and notice the changes occurring to the image.

```
A=imread('cameraman.tif');    % Read in image
subplot(1,2,1), imshow(A);     % Display image
% Map input grey values of image A in range 0-1 to an
% output range of 0-1 with gamma factor of 1/3 (i.e. r = 3).
B=imadjust(A,[0 1],[0 1],1./3);

subplot(1,2,2), imshow(B);   % Display result.
```

Try different values of gamma and notice how the image details become more or less visible.

Wednesday: Assignment

Complete all sections. Include your own code and resultant images in your report. Make sure you adhere to the report guidelines and deadline as specified on the course homepage.

1.1

Write a program to display a random 20-by-20 pixel image. In the figure, make sure that the axes are labelled 'X' and 'Y', the tick marks are visible and also labelled 1-20. Use the Matlab function ginput to record the selection of a pixel using the mouse, and display that pixel's X, Y co-ordinates, and then change the colour of the selected pixel to black. Include your code and a screenshot of the figure in your report.

1.2

Matlab has several built-in functions to display images (imshow, image, imagesc). Write a few lines of code that generate a figure containing three examples of displaying the same image (a random one, or one from the "Test Images" folder on the course homepage), and which highlights the differences between the functions. Include the code and one or more screenshots of the figure in your report.

1.3

Select one of the greyscale images from the "Test Images" folder on the course homepage. Implement a program to perform the bit-slicing technique as described in Section 1.2.1 of the book, and extract/display the resulting plane images (see Figure 1.3 in the book) as separate Matlab images. Include your Matlab code in your report, along with a figure showing 8 bit-planes (hint: use the subplot function).

1.4

Pick one of your own photos and import it into Matlab (use 'png', 'tif' or 'jpg' format) and check that it is in RGB format. Using the Matlab rgb2hsv function, write a program to display the individual hue, saturation and value channels of your (or any other) RGB colour image. You may wish to refer to book Example 1.6 on the display of individual red, green and blue channels. Include the Matlab code and resultant images in your report.

1.5

The engineering of image formation poses limits on the spatial resolution (see book Section 1.2) of an image through the digitization process. At the limits of spatial resolution certain image details are lost. Using your own image from above, experiment with the use of the imresize() function both in its simplest form (using a single scale parameter) and using a specified image width and height (rows and columns). As you reduce image size, identify details/structures that deteriorate beyond recognition/comprehension. Save 3 examples of resizing and highlight the relevant changes in your report. Then investigate the use of imresize() for enlarging your image, using all three of the available interpolation functions. Save the results for your report, and comment on the differences between them. Also report the time each of the different resizing interpolation options takes (Hint: Use the Matlab tic/toc functions).

1.6

The Matlab function imtool() allows us to display an image and perform a number of different interactive operations upon it. Use this function to load the example image shown in Figure 2.16 of the book (available as ‘railway.png’ in the “Test Images” folder on the course homepage) and measure the length in pixels of the railway sleepers in the foreground (front) and background (rear) of the image. Based on the discussion of Section 2.3.1 in the book, what can we determine about the relative distance of the two sleepers you have measured to the camera? If we assume the standard length of a railway sleeper is 2.5 m, what additional information would be need to determine the absolute distance of these items from the camera? How could this be applied to other areas, such as estimating the distance of people or cars from the camera?

1.7

Using the examples that you tried on Monday on arithmetic operations (Examples 3.1 – 3.4 in the book) we can investigate the use of these operators in a variety of ways. First, load the Matlab example images ‘rice.png’ and ‘cameraman.tif’ (also available in the “Test Images” folder on the course homepage). Investigate the combined use of the Matlab imresize() and size() functions to resize one of the images to be the same size as the other. The Matlab command whos used in the syntax ‘whos v’ will display information about the size and type of a given image or other variable v. Try adding both of these images together using the standard Matlab addition operator ‘+’ and displaying the result. What happens? Now add them together by using the imadd() function but adding a third parameter to the function of ‘uint16’ or ‘double’ with quotes that forces an output in a 16-bit or floating-point double-precision data type. You will need to display the output using the imagesc() function. How do these two addition operator results differ and why do they differ? Repeat this exercise using the standard Matlab subtraction operator ‘-’ and also by using the imsubtract() function.

1.8

Use the sequence of cell images (‘AT3_1m4_01.tif’, ‘AT3_1m4_02.tif, . . . ‘AT3_1m4_09.tif’, ‘AT3_1m4_10.tif’) in combination with the Matlab imabsdiff() function and a Matlab ‘for loop’ construct to display a mosaic of the differences between images in the sequence (9 images in total). Include your code and figure in your report. You may wish to use an additional enhancement approach to improve the dynamic range of difference images that result from the imabsdiff() function. What is result of this differencing operation? How could this be useful?
Hint. You may wish to set up an array containing each of the image file names 01 to 10.

1.9

Using a combination of the Matlab immultiply() and imadd() functions implement a Matlab function (or sequence of commands) for blending two images A and B into a single image with corresponding blending weights w_A and w_B such that output image C is
C = w_A . A + w_B . B
Experiment with different example images and also with blending information from a sequence of images (e.g. the cell images from 1.6). How could such a technique be used in a real application?
