Chapter 2 Exercises

The following exercises are designed to reinforce and develop the concepts and Matlab examples introduced in this chapter. Additional information on all of the Matlab functions represented in this chapter and throughout these exercises is available in Matlab from the function help browser (use *doc <function name>* at the Matlab command prompt, where *<function name>* is the function required)
Matlab functions: **rand(), fspecial(), filter2(), imresize(), tic(), toc(), imapprox(), rgb2ind(), gray2ind(), imnoise(), imtool(), hdrread(), tonemap(), getrangefromclass().**

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:

Example 2.3

Matlab code **What is happening?**

```
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;
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')
```

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 2.2 The engineering of image formation poses limits on the spatial resolution (Section 1.2) of an image through the digitization process. At the limits of spatial resolution certain image details are lost. Using the Matlab example images 'football.jpg' and 'text.png', 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, at what point do certain image details deteriorate beyond recognition/comprehension within the image? You may also wish to investigate the use of *imresize*() for enlarging images. Experiment with the effects of enlarging an image using all three of the available interpolation functions for the Matlab *imresize*() function. What differences do you notice? (Also try timing different resizing interpolation options using the Matlab t*ic()/toc()* functions.)

Exercise 2.3 In addition to spatial resolution, the engineering of image formation also poses limits on the number of quantization (colour or grey-scale) levels in a given image. Using the Matlab function *imapprox*() we can approximate a given image represented in a given colour space (e.g. RGB or grey scale, Section 1.4.1.1) with fewer colours. This function operates on an indexed image (essentially an image with a reference look-up table for each of its values known as a colour map) – see 'indexed images' in Matlab Help.

A conventional image (e.g. Matlab example 'peppers.png') can be converted to an indexed image and associated look-up table colour map using the Matlab *rgb2ind*() function (or gray2ind() for grey-scale images). Investigate the use of *imapprox*() and *rgb2ind*()/*gray2ind*() to reduce the number of colours in a colour image example and a greyscale image example. Experiment with different levels of colour reduction. How do your findings tally with the discussion of human colour perception in Section 2.3.2.1)?

Exercise 2.4 The Matlab *imnoise*() function can be used to add noise to images. Refer to Example 4.3 for details of how to use this function and experiment with its use as suggested in the associated Exercise 4.1.

Exercise 2.5 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 (available as 'railway.png') 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, 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?

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 quantization 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.