Chapter 4 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**: imnoise , plot , tic , toc , min , max , mean , function , imresize , for , colfilt , roipoly , roifilt , imfilter , fspecial .**

Exercise 4.1 Based on Example 4.3, experiment with the Matlab *imnoise*() function for adding different levels of 'salt and pepper' and Gaussian noise to images. Use the 'peppers.png' and 'eight.tif' example images as both a colour and grey-scale example to construct a series of image variables in Matlab with varying levels and types of noise (you may also wish to investigate the other noise types available from this function, type *doc imnoise* at the Matlab prompt). Based on the filtering topics presented in Section 4.4, investigate the usefulness of each of the mean, median and Gaussian filtering for removing different types and levels of image noise (see Examples 4.4, 4.5 and 4.7 for help).

Exercise 4.2 Building on Example 4.2, experiment with the use of different neighbourhood dimensions and the effect on the resulting image output. Using the plotting functions of Matlab (function *plot*()) and the Matlab timing functions tic and toc , create a plot of neighbourhood dimension N against operation run-time for applying the *min*() ,*max*() and *mean*() functions over an example image. Does the timing increase linearly or not? Why is this?

Exercise 4.3 Building on the local pixel neighbourhood definition described Section 4.2, write a Matlab function (see Matlab help for details on functions or type doc function at the Matlab function) to extract the NxN pixel neighbourhood of a given specified pixel location (x,y) and copy the pixel values to a new smaller NxN image. You may want to take a look at the syntax for the Matlab for loop function (doc for) in the first instance. Combine your extraction program with the Matlab *imresize*() function to create a region extraction and magnification program.

Exercise 4.4 The Matlab *colfilt*() function utilizes the highly efficient matrix operations of Matlab to perform image filtering operations on pixel neighbourhoods. Using the Matlab timing functions *tic*() /*toc*() , time the operation performed in Example 4.2 with and without the use of this optimizing parameter. Vary the size of the neighbourhood over which the operation is performed (and the operation: *min*() /*max*() /*mean*()). What do you notice? Is the increase in performance consistent for very small neighbourhoods and very large neighbourhoods?

Exercise 4.5 Based on Example 4.4 for performing mean filtering on a given image, record the execution time for this operation (using Matlab timing functions *tic*() /*toc*()) over a range of different neighbourhood sizes in the range 0–25. Use Matlab' s plotting facilities to present your results as a graph. What do you notice? How does the execution time scale with the increase in size of the neighbourhood? (Hint . To automate this task you may want to consider investigating the Matlab for loop constructor). As an extension you could repeat the exercise for differently sized images (or a range of image sizes obtained from *imresize*()) and plot the results. What trends do you notice?

Exercise 4.6 Repeat the first part of Exercise 4.5, but compare the differences between mean filtering (Example 4.4) and median filtering (Example 4.5). How do the trends compare? How can any differences be explained?

Exercise 4.7 A region of interest (ROI) within an image is an image sub-region (commonly rectangular in nature) over which localized image-processing operations can be performed. In Matlab an ROI can be selected interactively by first displaying an image (using *imshow*()) and then using the *roipoly*() function that returns an image sub-region defined as a binary image the same size as the original (zero outside the

ROI and one outside the ROI). Investigate the use of the *roifilt*() function for selectively applying both Gaussian filtering (Example 4.7) and mean filtering (Example 4.4) to an isolated region of interest within one of the available example images.
You may also wish to investigate combining the ROI selection function with your answer to Exercise 4.3 to extract a given ROI for histogram equalization (Section 3.4.4) or edge detection processing (Section 4.5) in isolation from the rest of the image.

Exercise 4.8 Several of the filtering examples in this chapter make use of the Matlab *imfilter*() function (Example 4.2), which itself has a parameter of how to deal with image boundaries for a given filtering operation. Select one of the filtering operations from this chapter and experiment with the effect of selecting different boundary options with the *imfilter*() function. Does it make a difference to the result? Does it make a difference to the amount of time the filtering operation takes to execute? (Use *tic*() /*toc*() for timing.)

Exercise 4.9 Implement a Matlab function (*doc function*) to perform the conservative smoothing filter operation described towards the end of Section 4.4.3. (Hint . You may wish to investigate the Matlab f*or loop* constructor.) Test this filter operation on the noise examples generated in Example 4.3. How does it compare to mean or median filtering for the different types of noise? Is it slower or faster to execute? (Use *tic*() /*toc*() for timing.)

Exercise 4.10 Considering the Roberts and Sobel edge detectors from Example 4.8, apply edge detection to three-channel RGB images and display the results (e.g. the 'peppers.png' and 'football.jpg' images). Display the results as a three-channel colour image and as individual colour channels (one per figure). Note how some of the edge responses relate to distinct colour channels or colours within the image. When an edge is visible in white in the edge-detected three-channel image, what does this mean? You may also wish to consider repeating this task for the HSV colour space we encountered in Chapter 1. How do the results differ in this case?

Exercise 4.11 Building upon the unsharp operator example (Example 4.12), extend the use of this operator to colour images (e.g. the 'peppers.png' and 'football.jpg' images). How do the areas of image sharpening compare with the areas of edge-detection intensity in Exercise 4.10? The unsharp operator can also be constructed by the use of an 'unsharp' parameter to the Matlab *fspecial*() function for use with *imfilter*() . Compare the use of this implementation with that of Example 4.12. Are there any differences?