Sample Projects

for

Digital Image Processing Using MATLAB®

2nd edition

Rafael C. Gonzalez
Richard E. Woods
Steven L. Eddins

©2009
Gatesmark Publishing
a division of Gatesmark, LLC
ISBN: 9780982085400

Book web site: www.imageprocessingplace.com

Version 1.0

May 1, 2009

Introduction

This document contains sample project statements dealing with the first five chapters in the book. These projects are intended to serve as guidelines for formulating projects dealing with other topics in the book.

The projects range in complexity from straightforward extensions of the material in the book to more comprehensive undertakings that require several hours to solve. All the material required for the projects is contained in the book web site. Normally, the solutions are based on material that has been covered up to the chapter in question, but we often use material from later chapters in order to arrive at a well-formulated solution. In these instances, we indicate where the needed material is located in the book and, on rare occasions, we point to online resources.

One of the most interesting aspects of a course in digital image processing in an academic environment is the pictorial nature of the subject. It has been our experience that students truly enjoy and benefit from judicious use of computer projects to complement the material covered in class. Since computer projects are in addition to course work and homework assignments, we try to keep the formal project reporting as brief as possible. In order to facilitate grading, we try to achieve uniformity in the way project reports are prepared. A useful report format is as follows:

Page 1: Cover page.

- Project title
- Project number
- Course number
- Student's name
- Date due
- Date handed in

Abstract (not to exceed 1/2 page)

Page 2: One to two pages (max) of technical discussion.

Page 3 (or 4): Discussion of results. One to two pages (max).

Results: Image results (printed typically on a laser or inkjet printer). All images must contain a number and title referred to in the discussion of results.

Appendix: Program listings, focused on any original code prepared by the student. For brevity, functions and routines provided to the student are referred to by name, but the code is not included.

Layout: The entire report must be on a standard sheet size (e.g., 8.5 by 11 inches), stapled with three or more staples on the left margin to form a booklet, or bound using a clear cover, standard binding product.

Although formal project reporting of the nature just discussed is not typical in an independent study or industrial/research environment, the projects outline in this document offer valuable extensions to the material in the book, where lengthy examples are kept to a minimum for the sake of space and continuity in the

discussion. Image processing is a highly experimental field and MATLAB, along with the Image Processing Toolbox, offers an unparalleled software development environment. By taking advantage of MATLAB's vectorization capabilities solutions can be made to run fast and efficiently, a feature that is especially important when working with large image data bases. The use of DIPUM Toolbox functions is encouraged in order to save time in arriving at project solutions and also as exercises for becoming more familiar with these functions.

The reader will find the projects in this document to be useful for both learning purposes and as a reference source for ideas on how to attack problems of practical interest. As a rule, we have made an effort to match the complexity of projects to the material that has been covered up to the chapter in which the project is assigned. When one or more functions that have not been covered are important in arriving at a well-formulated solution, we generally suggest in the problem statement that the reader should become familiar with the needed function(s).

There usually isn't a single "best" solution to a given project. The following criteria can be used as a guide when a project calls for new code to be developed:

How quickly can I implement this solution?
How robust is the solution for "rare" cases?
How fast does the solution run?
How much memory does the solution require?
How easy is the code to read, understand, and modify?

Also, it is important to keep in mind that solutions can change over time, as new MATLAB or Image Processing Toolbox functions become available.

Chapter 2

Sample Projects

PROJECT 2.1

MATLAB does not have a function to determine which elements of an array are integers (i.e., . . ., 2, 1, 0, 1, 2 , . . .). Write a function for this purpose, with the following specifications:

```
function I = isinteger(A)
%ISINTEGER Determines which elements of an array are integers.
%   I = ISINTEGER(A) returns a logical array, I, of the same size
%   as A, with 1s (TRUE) in the locations corresponding to integers
%   (i.e., . . . -2 -1 0 1 2 . . . )in A, and 0s (FALSE) elsewhere.
%   A must be a numeric array.
```

Use of `while` or `for` loops is not allowed. Note: Integer and double-precision arrays with real or complex values are numeric, while strings, cell arrays, and structure arrays are not. *Hints:* Become familiar with function `floor`. If you include the capability to handle complex numbers, become familiar with functions `real` and `imag`.

PROJECT 2.2

MATLAB does not have a function to determine which elements of an array are even numbers (i.e., . . . 4, 2, 0, 2, 4 , . . .). Write a function for this purpose, with the following specifications:

```
function E = iseven(A)
%ISEVEN Determines which elements of an array are even numbers.
%   E = ISEVEN(A) returns a logical array, E, of the same size as A,
%   with 1s (TRUE) in the locations corresponding to even numbers
%   (i.e., . . . -3, -1, 0, 2, 4, . . . )in A, and 0s (FALSE) elsewhere.
%   A must be a numeric array.
```

Use of `while` or `for` loops is not allowed. See Project 2.1 regarding numeric arrays. *Hint*: Become familiar with function `floor`.

PROJECT 2.3

MATLAB does not have a function to determine which elements of an array are odd numbers (i.e., . . ., 3, 1, 1, 3 , . . .). Write a function for this purpose, with the following specifications:

```
function D = isodd(A)
function D = isodd(A)
%ISODD Determines which elements of an array are odd numbers.
%   E = ISODD(A) returns a logical array, D, of the same size as A,
%   with 1s (TRUE) in the locations corresponding to odd numbers
%   (i.e., . . . -3, -1, 1, 3, . . . )in A, and 0s (FALSE) elsewhere.
%   A must be a numeric array.
```

Use of `while` or `for` loops is not allowed. See Project 2.1 regarding numeric arrays. *Hint*: Become familiar with function `floor`.

PROJECT 2.4

Write an M-function with the following specifications:

```
function H = imcircle(R, M, N)
%IMCIRCLE Generates a circle inside a rectangle.
%   H = IMCIRCLE(R, M, N) generates a circle of radius R centered
%   on a rectangle of height M and width N. H is a binary image with
%   1s on the circle and 0s elsewhere. R must be an integer >= 1.
```

Your program must check the validity of `R` and also it should check to make sure that the specified circle fits in the given rectangle dimensions. Use of `for` or `while` loops is not permitted. *Hint:* Review function `meshgrid` and become familiar with function `floor`.

PROJECT 2.5

The main purposes of this project are to learn how work with displaying and changing directories and how to use that information to read an image from a specified directory. Write an M-function with the following specifications:

```
function [I, map] = imagein(path)
%IMAGEIN Read image in from current-working or specified directory.
%   I = IMAGEIN displays a window containing all the files in the
%   current directory, and saves in I the image selected from the
%   current directory.
%   [I, MAP] = IMAGEIN variable MAP is required to be an output
%   argument when the image being read is an indexed image.
%   [ . . .] = IMAGEIN('PATH') is used when the image to be read
%   resides in a specified directory. For example, the input
%   argument 'C:\MY_WORK\MY_IMAGES' opens a window showing
%   the contents of directory MY_IMAGES. An image selected from
%   that directory is read in as image I.
```

Hint: Use MATLAB help to become familiar with functions `cd, pwd,` and `uigetfile`. For an alternative solution, consider using function `fullfile` instead of function `cd`.

Chapter 3

Sample Projects

PROJECT 3.1 Intensity Transformation (Mapping) Function

Write an M-function for performing general intensity transformations of gray-scale images. The specifications for the function are as follows:

```
function z = intxform (s, map)
%INTXFORM Intensity transformation.
%   Z = INTXFORM(S, MAP) maps the intensities of input
%   image S using mapping function, MAP, whose values are assumed to
%   be in the range [0 1].  MAP specifies an intensity transformation
%   function as described in Section 3.2. For example, to create a map
%   function that squares the input intensities of an input image of
%   class uint8 and then use function INTXFORM to perform the mapping
%   we write:
%
%       t = linspace(0, 1, 256);
%       map = t.^2;
%       z = intxform(s, map);
%
%   The output image is of the same class as the input.
```

Hint: Your code will be simplified if you use functions `tofloat` and `interp1`.

PROJECT 3.2 Histogram Processing

Image Processing Toolbox histogram function, `histeq`, attempts to produce as flat a histogram. Write a histogram equalization function that implements the equations discussed in Section 3.3.2 directly:

$$\begin{aligned} s_k &= T(r_k) \\ &= \sum_{j=1}^{k} p_r(r_j) \\ &= \sum_{j=1}^{k} \frac{n_j}{n} \end{aligned}$$

The function for this project has the following specifications:

```
function h = histeq2(f)
%HISTEQ2 Histogram equalization.
%   H = HISTEQ2(F) histogram-equalizes F and outputs the result
%   in H. Unlike Toolbox function histeq, HISTEQ2 implements the direct,
%   "standard" histogram equalization approach explained in Section
%   3.3.2. F can be of class uint8, uint16, or double. If F is double,
%   then its values are assumed to be in the range [0 1]. The intensity
%   range of the input image (regardless of class) is divided into 256
%   increments for computing the histogram and the corresponding cumulative
%   distribution function (CDF). Recall that the CDF is actually the mapping
```

```
%   function used for histogram equalization.
%
%   The output is of the same class as the input.
```

Hint: Your code will be simplified if you use the function developed in Project 3.1 and function `cumsum`.

PROJECT 3.3 Local Histogram Equalization

The global histogram equalization technique is easily adaptable to local histogram equalization. The procedure is to define a square or rectangular window (neighborhood) and move the center of the window from pixel to pixel. At each location, the histogram of the points inside the window is computed and a histogram equalization transformation function is obtained. This function is finally used to map the intensity level of the pixel centered in the neighborhood to create a corresponding (processed) pixel in the output image. The center of the neighborhood region is then moved to an adjacent pixel location and the procedure is repeated. Since only one new row or column of the neighborhood changes during a pixel-to-pixel translation of the region, updating the histogram obtained in the previous location with the new data introduced at each motion step is possible. This approach has obvious advantages over repeatedly computing the histogram over all pixels in the neighborhood region each time the region is moved one pixel location.

Write an M-function for performing local histogram equalization. Your function should have the following specifications.

```
function g = localhisteq(f, m, n)
%LOCALHISTEQ Local histogram equalization.
%   G = LOCALHISTEQ(F, M, N) performs local histogram equalization
%   on input image F using a window of (odd) size M-by-N to produce
%   the processed image, G. To handle border effects, image F is
%   extended by using the symmetric option in function padarray.
%   The amount of extension is determined by the dimensions of the
%   local window. If M and N are omitted, they default to
%   3. If N is omitted, it defaults to M. Both must be odd.
%
%   This function accepts input images of class uint8, uint16, or
%   double. However, all computations are done using 8-bit intensity
%   values to speed-up computations. If F is of class double its
%   values should be in the range [0 1]. The class of the output
%   image is the same as the class of the input.
```

Hint: One approach is to write a function that uses the results of Projects 3.1 and 3.2. This will result in a simpler solution. However, the function will be on the order of five times slower than a function that computes the local histogram once and then updates it as the local window moves throughout the image, one pixel displacement at a time. The idea is that the new histogram, *h_new*, is equal to the to the old histogram, *h_old*, plus the histogram of data added as the window is moved, *h_datain*, minus the histogram of the data that moved out of the window as a result of moving the window, *h_dataout*. That is: *h_new = h_old + h_datain h_dataout*. In addition to being faster important advantage of this implementation is that it is self-contained. Note that the histograms *h_datain* and *h_dataout* are normalized by the factor equal to the total number of pixels, *mn*, in order to be consistent with the fact that the area of histograms *h_new* and *h_old* must be 1.0. Your code will be simplified if you use functions `cumsum` and `tofloat`. Keep in mind that only the intensity level of the center of the neighborhood needs to be mapped at each location of the window.

PROJECT 3.4 Comparing Global and Local Histogram Equalization.

(a) Download image `FigP0304(a)(embedded_objects_noisy).tif` and process it with function `localhisteq` using neighborhoods of sizes 3 x 3 and 7 x 7. Explain the differences in your results.

(b) Histogram-equalize it using the global function `histeq2` from Project 3.2. If you did not do that project, then use Image Processing Toolbox function `histeq`. Although the background tonality will be different

between the two functions, the net result is that global histogram equalization will be ineffective in bringing out the details embedded in the black squares within the image. Explain why. Note in both (a) and (b) how noise is enhanced by both methods of histogram equalization, with local enhancement producing noisier results than global enhancement.

PROJECT 3.5 Experimenting with Larger "Laplacian" Masks

It is shown in Example 3.10 that the Laplacian mask `w8 = [ 1, 1, 1; 1 – 8 1; 1, 1, 1]` yields a result sharper than the result with a similar mask with a 4 in the center. **(a)** Write an M-function that generates a "Laplacian" mask of arbitrary odd size. For example, the mask of size 5 x 5 would consist of all 1s with a –24 in the center location. **(b)** Download the image `FigP0305(blurry_moon).tif` and compare the results between Fig. 3.18(c) in the book and the results obtained with masks of size n x n for n = 5, 9, 15, and 25. **(c)** Explain the differences in the resulting images.

Chapter 4

Sample Projects

PROJECT 4.1 Working with the Power Spectrum and Using Functions `varargout` and `varargin`

The main objectives of this project are to investigate the effect that filtering has on average image power and to practice using functions `varargout` and `varargin`.

As discussed in Section 4.1, the power spectrum is defined as

$$P(u,v) = |F(u,v)|^2$$

where $F(u,v)$ is the Fourier transform of an image, $f(x,y)$. The *average image power* is defined as

$$f_A = \frac{1}{MN}\sum_u\sum_v |F(u,v)|^2$$

(a) Write an M-function with the following specifications:

```
function [varargout] = impower(f, varargin)
%IMPOWER Filters an image and computes its average power.
%   fA = impower(f) Computes the image power, fA.
%   [fA, fR, g] = impower(f, TYPE, SIG) computes the average image power,
%   fA, of input image f. It also filters f using a Gaussian highpass
%   filter if TYPE = 'high' or a Gaussian lowpass
%   filter if TYPE = 'low'. Either filter has sigma = SIG. The resulting
%   filtered image and its power are output as g and fR.
```

As discussed in Section 3.2.4, `varargout` is one way to write a function whose number of outputs is variable. Although in this case we could just as easily have used the syntax `function [fA, fR, g]` to handle both instances of outputs, the purpose of using `varargout` here simply as an exercise. In general, `varargout` is quite useful when the number of outputs is large and variable. Similar comments apply for `varargin` regarding input variables.

(b) Download `FigP0401(test_pattern).tif`. Obtain its average image power, `fA`, using function `impower`.

(c) Then, use `SIG = 16` and generate `fRL` and `gL` for a lowpass filter and `fRH` and `gH` for a highpass filter. Show the original image and the two filtered images.

(d) You will find that `fRL` and `fA` are close in value, but `fRH` is approximately one order of magnitude smaller than these two quantities. Explain the main reason for the difference.

PROJECT 4.2 Working with Phase Angles

As indicated in Section 4.1 of the book, the Fourier transform is complex, so it can be expressed in polar form as

$$F(u,v) = |F(u,v)|e^{j\phi(u,v)}$$

[Note that the exponent is positive, not negative as it is shown in the initial printing of the book]. Here,

$$|F(u,v)| = \left[R^2(u,v) + I^2(u,v)\right]^{1/2}$$

is the spectrum and

$$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right]$$

is the phase angle. The purposes of this projects are (1) to show that the phase angle carries information about the location of image elements, (2) that the spectrum carries information regarding contrast and intensity transitions, and (3) that phase information is dominant over the spectrum regarding visual appearance of an image. Download `FigP0402(a)(woman).tif` and `FigP0402(b)(test_pattern).tif` (call them `f` and `g`). It is recommended that you look up functions `complex` and `angle` before starting the project.

(a) Compute the spectrum and the phase angle of image `f` and show the phase image. Then compute the inverse transform using only the phase term (i.e., `ifft2` of $e^{j\phi(u,v)}$] and show the result. Note how the image is void of contrast, but clearly shows the woman's face and other features.

(b) Compute the inverse using only the magnitude term (i.e., `ifft2` of $|F(u,v)|$). Display the result and note how little structural information it contains.

(c) Compute the spectrum of `g`. Then computer the inverse FFT using the spectrum of `g` for the real part and the phase of `f` for the imaginary part. Display the result and note how the details from the phase component dominate the image.

PROJECT 4.3 Fun Project: Distorting Signs and Symmetries in the Fourier Transform

Interesting and sometimes nonsensical results are obtained when the symmetry and signs in the Fourier transform are not preserved, as the following projects show.

Read image `FigP0402(woman).tif`, and obtain its spectrum, `S`, and phase, `P`, as in Project 4.2. Then,

(a) Let `S1 = S` and `S1(1:M/2, 1:N/2) = 0` where `[M, N] = size(S)`. Recover the image using `S1` and `P`. Show your result.

(b) Let `P2 = cong(P)`, where `conj` is the complex conjugate. Recover the image using `S` and `P2`. Show your result.

(c) In Chapter 4, all filters used are zero-phase-shift filters, which have no effect on the phase because they multiply the real and imaginary components of the transform by the same quantity. Here, we want to show the effects of changing the real and imaginary components of the Fourier transform differently. Let `F2 = complex(0.25*real(F), imag(F))`. Note that this affects both the spectrum and the phase. However, based on the results from Project 4.2, we expect that the effect on the phase will dominate in the recovered image. Obtain the inverse transform and show the resulting image.

PROJECT 4.4 Bandreject and Bandpass Filtering

In the book we discuss lowpass and highpass filtering in detail. Although used less often, bandreject and bandpass filters are an important category of frequency domain filters, especially because they are much more

difficult to implement using spatial techniques. The transfer function of a Butterworth bandreject filter (BBRF) of order *n* is

$$H_{br}(u,v) = \frac{1}{1 + \left[\frac{D(u,v)W}{D^2(u,v) - D_0^2}\right]^{2n}}$$

where W is the width of the band, and D_0 is its center. The corresponding bandpass filter is given by

$$H_{bp}(u,v) = 1 - H_{br}(u,v) .$$

(a) Write a function with the following specifications to implement these filters:

```
>> function H = bandfilter(type, M, N, D0, W, n)
%BANDFILTER Generate a bendreject or bandpass filter.
%   H = BANDFILTER(M, N, D0, W, TYPE) generates a bandreject
%   filter (if TYPE is specified as 'reject') of size M-by-N,
%   order n, width W and center at D0. To generate a bandpass
%   filter specify TYPE as 'pass'.
```

(b) Generate and show 3-D perspective plots of the two filters for `M = N = 600`, `D0 = 50`, `W = 8`, and `n = 0.1`. Don't forget to use function `fftshift` to center the filter and thus produce nicer looking plots.

(c) Write the plots to disk in `tif` format using 300 DPI resolution.

PROJECT 4.5 Using Bandreject Filtering for Image Denoising

Image `FigP0405(HeadCT_corrupted).tif` is a tomography image of a human head, heavily corrupted by sinusoidal noise in at least two directions.

(a) Clean up the image using bandreject filtering. To simplify your project you may ignore padding in this case. *Hint:* Since the noise is sinusoidal, it will show in the spectrum as impulses. Display the spectrum as a guide to where to set up the band of your filter. If you use function `impixelinfo` to determine the coordinates of the impulses interactively, keep in mind that this function lists the column coordinates (`v`) first and the row coordinates (`u`) second.

(b) Use bandpass filtering to extract the noise pattern.

PROJECT 4.6 Unsharp Masking

Unsharp masking has been a staple in photography darkroom for countless years. The technique consists of generating a sharp image by subtracting from the image a blurred version of itself. In frequency domain terminology this means generating a highpass filtered image by subtracting from the image a lowpass filtered version of itself. To obtain a little more flexibility, the technique often is modified to subtract a blurred and dimmed (multiplied by a positive constant between 0 and 1) version of the blurred image. Download image `FigP0406(blurry_moon).tif` and sharpen it using the modified version of unsharp masking.

PROJECT 4.7 Laplacian in the Frequency Domain

It is not difficult to show that the Laplacian in the frequency domain can be implemented using the filter $H(u,v) = -4\pi^2(u^2 + v^2)$; that is,

$$\Im\left[\nabla^2 f(x,y)\right] = -4\pi^2(u^2 + v^2)F(u,v)$$

(a) Download image `FigP0406(blurry_moon).tif` and implement in the frequency domain the steps leading to the images in Fig. 3.16(d) in the book. You will note that your image looks more like Fig. 3.18(c), indicating that results obtained in the frequency domain more closely resemble the spatial results obtained using a Laplacian mask with a 8, rather than a 4, in the center.

(b) Generate a frequency domain filter from `h = [1 1 1; 1 -8 1; 1 1 1]` using the approach discussed in Section 4.4. Use this filter to process the image in the frequency domain and compare your result with (a).

PROJECT 4.8 Homomorphic Filtering

(a) Homomorphic filtering attempts to achieve simultaneous reduction of the dynamic range of an image and coupled with an increase in sharpness (see Gonzalez and Woods [2008] for details of this technique). A frequency domain filter used frequently for homomorphic filtering has the form

$$H(u,v) = A + \frac{C}{1+\left[D0/D(u,v)\right]^B}$$

where $D(u,v)$ is the distance from the origin of the (centered) Fourier transform, and $A < B$. Write a function for performing homomorphic using this filter as the default:

```
function g = homofilt(f, varargin)
%HOMOFILT Homomorphic filtering.
%   G = HOMOFILT(F, H) performs homomorphic filtering on the input
%   image, F, using the specified filter, H.
%   G = HOMOFILT(F, A, B, C, D0) performs homomorphic filtering on
%   F using the default filter:
%
%        H = A + C/{1 + [D0/D(U, V)]^B}
%
%   For this filter to make sense, it is required that A < B. A
%   good set of starting values: A = 0.25, B = 2, C = 2, and
%   D0 = min(M, N)/8.
```

(b) Read image `FigP0408(PET_image).tif` and perform homomorphic filtering on it using the default filter.

Chapter 5
Sample Projects

PROJECT 5.1 Estimating Noise PDFs and Their Parameters

The image in `FigP0501(noisy_superconductor_image).tif` is corrupted by noise.

(a) Download this image and extract its *noise* histogram. Display the histogram (using function `bar`) and indicate (by name) what you think the noise PDF is. Determine the relevant noise parameter(s) using the histogram you extracted. (*Hint*: Use function `roipoly` to extract the data you think will help you identify the noise.)

(b) Use function `imnoise` or `imnoise2`, as appropriate, to generate `X` samples of the noise type and parameter(s) you determined in (a). Generate the histogram of the samples using function `hist`, and display the histogram. Here, `X` is the number of pixels in the ROI in (a). Compare with the corresponding histogram from (a).

PROJECT 5.2 Spatial Noise Reduction

(a) Use function `spfilt` to denoise image `FigP0502(a)(salt_only).tif`. This is an optical microscope image of a nickel-oxide thin film specimen magnified 600X. The image is heavily corrupted by salt noise.

(b) Use function `spfilt` to denoise image `FigP0502(b)(pepper_only).tif`. This is the same nickel oxide image, but corrupted this time by pepper noise only.

The tradeoff in your selection of a filter in (a) and (b) should be to produce the cleanest image possible with as little image distortion (e.g., blurring) as possible. (*Hint:* If you need to become more familiar with the details of the filters available in function `spfilt`, consult Chapter 5 of Gonzalez and Woods [2008].)

PROJECT 5.3 Adaptive Median Filtering

Repeat Project 5.2 using adaptive median filtering and compare the results by generating difference images of the results. Select the size of filter window to give the best (in your opinion) visual results.

PROJECT 5.4 Notch Filtering

Use function `cnotch` from Section 4.7.2 to denoise `FigP0405(HeadCT_corrupted).tif`. Show the original image, an image of your specified filter, and the restored result. If you completed Project 4.5, compare your result with the image denoised using bandreject filtering in that project. If you use function `impixelinfo` to determine the coordinates of the impulses interactively, keep in mind that this function lists the column coordinates (`v`) first and the row coordinates (`u`) second.

NOTE: When attempting to remove sinusoidal spatial interference, it generally is advisable to do all DFT filtering without padding. Even if the pattern is a pure sine wave, padding introduces frequencies in the sine

wave that often are quite visible and difficult to remove in the filtered image. Thus, in this case, the effect of wraparound error introduced by not using padding usually is negligible by comparison.