Lab Assignments Image Processing

M. A. Westenberg, J.B.T.M. Roerdink

Version 1.8 (2014)

1 General information

1.1 Getting image files

A collection of images related to the Image Processing Labs can be found in Nestor in an archive `ImageProcessing.zip` under "Assignments".

1.2 Setting the Matlab path

After extracting the files from `ImageProcessing.zip` and storing the resulting subdirectory `ImageProcessing` in your home directory, you can instruct Matlab to find the images by setting the Matlab path on your unix prompt:

```
export MATLABPATH=$HOME/ImageProcessing/images
```

(or include this line in your .profile). You can also set the path on the Matlab command prompt (type "help path" in Matlab to find out how).

1.3 Matlab's image processing toolbox

One of the goals of the lab exercises is to learn to develop your own image processing functions. Therefore, you *cannot* make use of the functions provided by the Image Processing Toolbox in Matlab, unless stated otherwise *explicitly* in an exercise.

1.4 Image I/O

Images can be read into Matlab by the function `imread`. For example:

```
x = imread('blurrymoon.tif');
```

reads the `blurrymoon` image from either the current directory or the search path.

The function `imwrite` can be used to write images to disk. For example:

```
imwrite(x, 'moon.tif');
```

would write the image in `x` in TIFF format to the current directory.

Both `imread` and `imwrite` support a number of optional parameters, see Matlab's help for more details.

1.5 Data types

Most images will be 8-bits grey scale, corresponding to the data type `uint8` in Matlab. Color images are also of type `uint8`, but they have three color planes (RGB), see `imread` for details. In many cases, you will need to convert an input image to double precision before performing calculations, e.g.

```
y = im2double(x);
```

converts the image `x` to double precision and stores the result in `y`.

Matlab's image processing and display functions assume that images of type `uint8` contains pixel values in the range 0 to 255. A double precision image is expected to have pixel values in the range 0 to 1.

1.6 Displaying images

The function `imshow` can be used to display in image in a figure. For example, to display image `x`,

```
imshow(x)
```

will draw the image in the current figure or open a new window.

This function assumes certain pixel value ranges. An alternative function for displaying images is `imagesc`, which will scale pixel values in such a way that the full grey scale range is used.

1.7 Reporting

For all labs, a concise report is expected as a single PDF file (using the LaTeX template provided in Nestor) in which:

- **a.** you describe, for each part of every exercise, how you arrived at the solution;
- **b.** you include all the relevant input and output images produced in each exercise;
- **c.** you describe your observations about the effect of the various image operations using the input and output images;
- **d.** you answer all the questions posed in each exercise, with a clear motivation based on both theoretical concepts and experimental observations;
- **e.** you provide the Matlab source code of all the required implementations.

Make sure the report contains your name(s), and the number of the lab.
Since you work in pairs, you need to include information in your report about your individual contributions. For each assignment, indicate the contribution (mention percentages) for each of you in terms of tasks performed (program design, program implementation, answering questions posed, writing the report).

2 Labs

A number in brackets in the exercise title indicates the maximal number of points you can earn.

LAB 1.

Exercise1 — Zooming and shrinking by bilinear interpolation (40)

(a) Write a function `IPresizebl` capable of zooming and shrinking an image by bilinear interpolation. The input to the function is the resizing factors in the x- and y-dimensions.

(b) Load `chronometer1250dpi.tif` and shrink this by a factor of 12.5 in both dimensions.

(c) Zoom the image in (b) back to its original size. Explain the reasons for their differences.

Exercise2 — Enhancement (20)

(a) Write a function `IPlogenhance` to enhance an image by the log transformation (3.2-2):

$$s = c\log(1 + r)$$

(b) Load `fracturedspine.tif` and try to obtain the best visual enhancement possible using the transformation in (a). Experimentation is a must.

Exercise3 — Spatial filtering (40)

(a) Write a function `IPfilter` to perform spatial filtering of an image (see Section 3.4) with a 3×3 mask. Do *not* use `filter`, `filter2`, `conv`, or `conv2` .

(b) Implement the gradient magnitude computation of an image (3.6-11) in a function `IPgradient`, and apply it to `blurrymoon.tif`.

(c) In a given application, an averaging mask is applied to input images to reduce noise, and then a sharpening mask is applied to enhance small details. Would the result be the same if the order of these operations were reversed?

LAB 2.

Exercise1 — Highboost filtering (25)

(a) Implement high boost filtering (Eq. (3.6-9) from the book) in a function `IPhighboost`. The averaging part should be done using the mask in Fig. 3.32(a). Enhance the image `dipxetext.tif` and try to approximate the result in Fig. 3.40(e).

(b) Assume the input image $f(x,y)$ has only positive values. Can the output image $g(x,y)$ have negative values? If so, does the value of the parameter k play a role? Explain your answers.

Exercise2 — Fourier spectrum (25)

(a) Load `characters.tif` and compute its centered Fourier spectrum. You may use Matlab's `fft2` and `fftshift` functions.

(b) Display the spectrum.

(c) Use your result in (a) to compute the average value of the image.

Exercise3 — Filtering (50)

(a) Implement the Gaussian lowpass filter in Eq. (4.8-7) in a function `IPgaussian`. You must be able to specify the size of the resulting 2D function.

(b) Write a function `IPftfilter` to perform filtering in the frequency domain.

(c) Load `characters.tif` and lowpass filter it to duplicate the results in Fig. 4.48.

(d) Implement the Gaussian highpass filter of Eq. (4.9-4).

(e) Load `characters.tif` and highpass filter it to duplicate the results in Fig. 4.56.

LAB 3.

Exercise1 — 1-D wavelet transforms (40)

The purpose of this exercise is to build a rudimentary wavelet transformation package using Haar wavelets. You will use an averaging-and-differencing approach that is unique to Haar basis functions. As an introduction to the method, consider the function in Example 7.8. The necessary operations are:

Step 1 Compute two-point sums and differences across the function vector and divide the results by the square root of 2. Since $f(x) = \{1, 4, -3, 0\}$, we get

$$\{1+4, -3+0, 1-4, -3-0\}/1.414$$
$$\{5, -3, -3, -3\}/1.414$$

Note that the sums are positioned consecutively at the beginning of the intermediate result and followed by the corresponding differences.

Step 2 Repeat the process over the sums computed in the first step to get

$$\{[5+(-3)]/1.414, [5-(-3)]/1.414, -3, -3\}/1.414$$
$$\{1, 4, -2.121, -2.121\}$$

The coefficients of the final vector match those in Example 7.8. The two-step computation generates a two-scale DWT with respect to Haar wavelets. It can be generalized to higher scales and functions with more than 4 points. Moreover, an inverse DWT can be computed by reversing the process.

(a) Write a function `IPdwt` to compute J-scale DWTs with respect to Haar wavelets. Let scale be an input parameter and assume a discrete 1-D input function with a length equal to a power of two.

(b) Write a function `IPidwt` to compute the inverse of a J-scale DWT.

Exercise2 — 2-D wavelet transforms (30)

(a) Write a function `IPdwt2` to compute J-scale two-dimensional wavelet transforms with Haar wavelets. Base your implementation on the discussion of separable wavelets and 2-D wavelet transforms in Section 7.5.

(b) Write a function `IPdwt2scale` to contrast-stretch the wavelet coefficients so that the underlying structure is more visible when displaying wavelet transformed data.

(c) Load `vase.tif` and use your function to generate the three-scale DWT shown in Fig. 7.10(a). Label the various detail and approximation coefficients that make up the transform and indicate their scales.

(d) Write a function `IPidwt2` to compute the inverse two-dimensional DWT with respect to Haar wavelets, and use it to reconstruct the original image from the wavelet decomposition in (c).

Exercise3 — Wavelet denoising (30)

(a) Write a function `IPwaveletdenoise` that implements denoising using a Haar-based DWT. Let the number of scales and a threshold be input parameters.

(b) Load `noisymri.tif` and denoise it using a Haar-based DWT. Experiment with the threshold parameter.

LAB 4.

Exercise1 — Morphological operations (25)

(a) Write a function `IPdilate` that performs a dilation with an arbitrary 3×3 structuring element. Assume that the origin of the structuring element lies in the center, and that binary images have type `logical` in Matlab.

(b) Write a function `IPerode` that performs an erosion with an arbitrary 3×3 structuring element.

(c) Test your functions on the image `wirebondmask.tif`.

Exercise2 — Classification (25)

Write a function that reads the image `nutsbolts.tif`, and produces an output image in which the nuts have grey value 64 and the bolts the grey value 128. The background should have value 0. Use only morphological and set operations.

Exercise3 — Opening by reconstruction (50)

(a) The *geodesic dilation* of the marker image F by the structuring element B with mask image G is defined by by the iteration:

$$X_0 = F, \qquad X_k = (X_{k-1} \oplus B) \cap G, \quad k = 1, 2, 3, \ldots$$

When run until stability ($X_{m+1} = X_m$), X_m is the *morphological reconstruction by dilation of G from F*.

(b) Implement morphological reconstruction by dilation in a function `IPrecon_by_dilation(f, mask, se)`, where `f` is the marker image F, `mask` is the mask image G, and `se` is the structuring element B.

(c) The image `angio.tif` is a black-and-white picture of an angiogram showing blood vessels in a 2D cross section the brain. The image `angio_noise.tif` was obtained by adding shot noise to the background of `angio.tif`. Compute the *opening* of `angio_noise.tif` with a 3×3 structuring element (with all 9 pixels 'on'). Also compute a binary 'difference' image `diff_opening.tif` in which a pixel is 'on' if and only if the opening of `angio_noise.tif` differs from the original image `angio.tif` at that pixel. Count the number of 'on' pixels in the 'difference' image.

(d) As an alternative, compute the *opening by reconstruction* of `angio_noise.tif` by using your function `IPrecon_by_dilation(f, mask, se)`, where the marker image `f` is the erosion of the input image `angio_noise.tif`, the `mask` is the input image itself, and `se` is the same structuring element as in exercise (c). Also compute a binary 'difference' image `diff_opening_by_recon.tif` and count the number of 'on' pixels in this image in the same way as above.

(e) Describe the effect of (i) the opening, and (ii) the opening by reconstruction on the background noise and the angiogram itself. Which of the two operations does a better job of removing the noise while affecting the angiogram as little as possible? Give an explanation for the difference between the results of the two operations. Use both visual observations and the pixel counts in the 'difference' images to support your conclusions.

LAB 5.

Exercise1 — Global thresholding (20)

(a) Write a function `IPautothresh` that performs global thresholding. The threshold should be estimated automatically by the procedure described in Section 10.3.2. The output of the function should be a binary image.

(b) Apply your function to `fingerprint.tif`, and report the estimated threshold.

Exercise2 — Texture (40)

(a) Write a function `IPtexturemeasures` that computes the statistical texture measures in Eqs. (11.3-4)–(11.3-9). Normalize the variance before use in Eq. (11.3-6), and also normalize the third moment.

(b) Extract a 100×100 segment from the lower, right quadrant of each of the following images: `bubbles.tif`, `cktboard.tif`, `cereal.tif`.

(c) Use your function to compute the statistical texture measures of the subimages, and present the results in a table. Discuss your results.

Exercise3 — Principal components (40)

(a) Write a function `IPprincipalcomponents` that implements the principal components transform described in Section 11.4. The input to your function should be a matrix, in which the columns represent the input vectors. For example, the matrix corresponding to the vectors from Example 11.14 is

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The second input to your function should be the number, k, of largest eigenvectors to retain. You can use the Matlab function `eig` to compute eigenvalues and eigenvectors.

(b) The files `WashingtonDC_Band1_564.tif` to `WashingtonDC_Band6_564.tif` correspond to six multispectral satellite images taken in the following spectral bands: visible blue, visible green, visible red, near infrared, middle infrared, and thermal infrared. Arrange these images into a matrix containing 6 rows and a number of columns equal to the number of pixels.

(c) Use your function to perform the principal components transform on the matrix from (b). Report the six eigenvalues and include the corresponding principal component images in your report.

(d) Perform a reconstruction using the eigenvectors that correspond to the three largest eigenvalues. Compute a difference image between each reconstructed spectral image and its original. Include the reconstructed images and difference images in your report. Discuss the results briefly.