university of groningen / faculty of mathematics and natural sciences / computing science

Practicals Image Processing

November 2015

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.

Exercise 1 — Zooming and shrinking by pixel replication (20)

(a) Write a function `IPresize` capable of zooming and shrinking an image by pixel replication. Assume that the desired zoom/shrink factors are integers.

(b) Load `chronometer1250dpi.tif` and shrink the image by a factor of 10.

(c) Zoom the shrunk image in (b) back to the resolution of the original. Explain the reasons for their differences.

Exercise 2 — Histogram equalization (40)

(a) Write a function `IPhistogram` for computing the histogram of an 8-bit image.

(b) Implement the histogram equalization technique discussed in Section 3.3.1 in a function `IPhisteq`.

(c) Load `fracturedspine.tif` and perform histogram equalization on it. Include the original image, a plot of its histogram, the enhanced image, and a plot of its histogram in your report. Use this information to explain why the resulting image was enhanced as it was.

(d) Explain why a second pass of histogram equalization will produce exactly the same result as the first pass.

Exercise 3 — 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) Write a function `IPSobel` to compute the gradient magnitude $M(x, y)$ of an image, Eq. (3.6-12), where g_x and g_y are computed according to Eq. (3.6.16)-(3.6.17) using the Sobel kernels in Fig. 3.41(d)-(e). Apply your function to `blurrymoon.tif`.

LAB 2.

Exercise 1 — 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.

Exercise 2 — Highpass filtering in the frequency domain (50)

(a) Implement the Gaussian highpass transfer function `H` of size $M \times N$ with cutoff frequency D_0, see Eq. (4.9-4), in a function `H=IPgaussian(D0,M,N)`.

(b) Write a function `IPftfilter(x,H)` to perform convolution filtering of an input image `x` where the filter kernel is given in the frequency domain as a transfer function `H`.

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

Exercise 3 — Median filtering (25)

(a) Write a function `IPmedian` to perform median filtering in a $(2k+1) \times (2k+1)$ window, with $k = 1, 2, 3, \ldots$.

(b) Load `circuitboard.tif` and add salt-and-pepper noise to it, with $\text{Pa} = \text{Pb} = 0.2$. You can use the `imnoise` function.

(c) Apply the median filter of size 3×3 to the image in (b). Explain any major differences between your result and Fig. 5.10(b).

LAB 3.

Exercise 1 — 1-D wavelet transforms (30)

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.

Exercise 2 — 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).

Exercise 3 — Image compression (40)

(a) Write a function `IPwaveletcompress` that achieves image compression by truncating the detail coefficients of the DWT of the image. Let scale and a threshold be input parameters. The function should report the compression rate achieved, the root-mean-square error, and the mean-square signal-to-noise ratio. It should also return both the reconstructed image and the absolute difference of the original image and compressed image.
It is not necessary to implement a complete compression algorithm. The idea is to report an estimate of how much compression could be achieved after wavelet thresholding (think about a way to estimate this yourself). Feel free to use a built-in Matlab function to compute this estimate, and motivate your choice in the report.

(b) Load `tracy.tif` and use various scales and thresholds to compress it. Report the compression rates and errors in a table, and visually assess the quality of the compressed images and the difference with the original images.

LAB 4.

Exercise 1 — Skeletons (40)

Let $SK(X)$ be the morphological skeleton of an image X with structuring element B. Assume that the structuring element contains the origin, i.e., $\mathbf{0} = (0,0) \in B$. The skeleton of an image X with structuring element B is defined as

$$SK(X) = \bigcup_{k=0}^{K} S_k(X)$$

with K the largest integer such that

$$(X \ominus KB) \neq \emptyset$$

and

$$S_k(X) = (X \ominus kB) - (X \ominus kB) \circ B$$

see also Section 9.5.7. We define $(X \ominus kB) = X$ when $k = 0$.

(a) Prove that $SK(X) = \emptyset$ if $B = \{\mathbf{0}\}$.

(b) Prove that $SK(X) = X$ if $X \ominus B = \emptyset$.

(c) A compact way to encode all skeleton sets $S_k(X)$ is to introduce the *skeleton function* $skf(X)$ as

$$[skf(X)](i,j) = \begin{cases} k+1, & (i,j) \in S_k(X) \\ 0, & (i,j) \notin S_k(X) \end{cases}$$

Write a function `IPskeletondecomp` that constructs the skeleton of a binary input image, and encodes the skeleton sets according to the above skeleton function.

(d) The reconstruction of an image X from its skeleton sets $S_k(X)$ is given by

$$X = \bigcup_{k=0}^{K} S_k(X) \oplus kB$$

Write a function `IPskeletonrecon` that performs the above reconstruction. Assume that the skeleton sets are encoded by the skeleton function.

(e) Compute the skeleton of `nutsbolts.tif`. Demonstrate (experimentally) that the reconstruction equals the input image.

Exercise 2 — Grey-scale morphology (30)

(a) Implement grey scale dilation and erosion with arbitrary 3×3 flat structuring elements in functions `IPgdilate` and `IPgerode`, respectively.

(b) Load `vase.tif` and perform grey scale dilation, erosion, opening, and closing on this image. Use a box (center pixel and its 8-connected neighbors) as a structuring element. Describe what you observe in the output images.

Exercise 3 — Classification (30)

Describe and implement a morphological method to suppress the small disks in the image `blobs.tif`. Note that because of image noise, complete removal is not possible. However, your approach should set their grey value close to the average background value, see Fig. 1 for a possible (and acceptable) result image. Your approach should not alter the large disks.

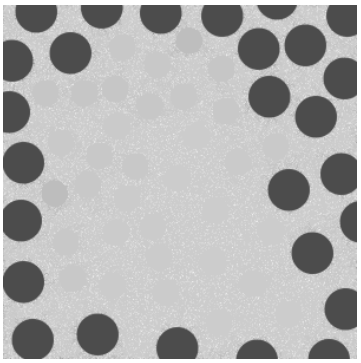

Figure 1: Blobs image with small disks suppressed.

LAB 5.

Exercise 1 — Edge detection (30)

(a) Write a function `IPMarrHildreth(x,sigma)` that implements the Marr-Hildreth edge detector according to Section 10.2.6 of the book. Here `x` is the input image and `sigma` is the standard deviation of the Gaussian function. The output of the function should be a binary image.

(b) Apply your function to `house.tif`, and comment on the result.

Exercise 2 — Region splitting and merging (35)

The function `splitmerge` implements the region splitting and merging algorithm described in Section 10.4.2. It outputs an image in which each connected component is labelled with a different integer. Furthermore, it outputs an image which shows the decomposition of the input image into quadtree regions.

(a) Load `tiger.tif` and define a region predicate that can distinguish 'tiger' region from 'sky' region. You can determine parameters manually, for instance, by inspecting certain regions of the image. Hint: consider defining a predicate for the sky rather than for the tiger.

(b) Implement the region predicate in a Matlab function `IPpredicate`, and apply the split-and-merge method to `tiger.tif`. Experiment with various minimum block sizes, and report which block size gives the best result. Explain what you mean by "best".

Exercise 3 — Fourier descriptors (35)

(a) The Matlab function `bwtraceboundary` implements the boundary following algorithm described in Section 11.1.1 (please consult the erratum on Fourier descriptors, see under Course Documents). It takes several parameters, see the documentation, amongst which the starting point of the trace. Implement a function `IPcontour` that takes a binary image containing a single object as input, determines the starting point, and uses `bwtraceboundary` to trace the contour.

(b) Apply your function to `lincoln.tif` and report the starting point.

(c) Write a function `IPfourierdescr` that implements the Fourier descriptor scheme described in Section 11.2.3. Your function should accept an array of boundary points and the number of Fourier descriptors, P, to retain. For testing purposes during development, you can use `chromosome_boundary.tif`, and try to reproduce the results in Fig. 11.20 (do not put this into your report).

(d) Apply your algorithm to the boundary from (b). Experiment with the parameter P, and report also the lowest number of descriptors required to keep the silhouette recognizable.