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

Practicals Image Processing

November 2018

1 General information

1.1 Getting image and auxiliary files

A collection of images related to the Image Processing Labs can be found in Nestor in an archive `ImageProcessing.zip` under "Assignments". The `ImageProcessing` folder also contains an example Matlab script `IPtest.m` that you can adapt when testing your Matlab implementations.

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 *should not* make use of the functions provided by the Image Processing Toolbox in Matlab, unless *explicitly* stated otherwise 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. 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.

In most cases, you will need to convert an input image to double precision before performing calculations, e.g.,

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

scales the integer image `x` to the range [0,1], converts it to double precision and stores the result in `y`.

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.

See the test script `IPtest.m` for some examples of image I/O.

1.7 Reporting

For all labs, a 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 (follow the a,b,c-structure of the exercises);
- **b.** you explain what your Matlab function achieves and if you have a non-trivial implementation how it achieves it (we are not only interested in your code, but in your explanations as well); if you have written a Matlab function create a small test program in Matlab to test your function, or small parts of it (see `IPtest.m` for an example test script); in most cases you can manually determine the output of a small input, or see whether the course book has an example; use this to check if your function works as expected;
- **c.** you include all the relevant input and output images and plots produced in each exercise; refer to the figures containing your results;
- **d.** you describe your observations about the effect of the various image operations using the input and output images; discuss your results, don't just show some images, show that you have spent time thinking about your results;
- **e.** you answer all the questions posed in each exercise, with a clear motivation based on both theoretical concepts and experimental observations;
- **f.** you structure your answers; generally a well-structured answer has the following components:
    1. introduce the problem theoretically, with formulas, explain why the formulas make sense if that seems applicable;
    2. refer to the implementation, in the following terms: "Listing *x* presents *IP....(image, x, y)* that implements the process discussed above", or something similar;
    3. explain the implementation if it is non trivial; for example, if you literally implemented the formulas you presented earlier then that is sufficient; however, if you are performing convolution by shifting the image under your filter explain what you are doing;

4. refer to the results, like: "Applying *IP.....* on Figure q with $x = ?$ and $y = ?$ we find Figure z";
5. discuss your results and explain them if applicable: "In Figure 6 we see that .... this is caused by ....".

g. you provide the Matlab source code of all the required implementations as an archive. The submission must be completely self-contained, that is, it should include all the source code that is necessary to run your programs. Even if you have submitted some code as part of a previous assignment, you should resubmit it.

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.

Sections, equations and figures mentioned in the assignments refer to the course book: R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4 / E. - Global Edition (Pearson, 2018). ISBN: 9781292223049.
Website: `http://www.imageprocessingplace.com`.

LAB 1. Start by reading Section 1.7 of this manual.

Exercise 1 — Reducing the number of intensity levels (40)

(a) Write a function `IPreduce` to reduce the number of intensity levels in an image from 256 to 2 in integer powers of 2. The desired number of intensity levels needs to be a variable input to your program.

(b) Load `ctskull-256.tif` and duplicate the results shown in Fig. 2.24 of the book.

Exercise 2 — Enhancement (60)

(a) Write a function `IPcontraststretch` to perform linear contrast stretching of an N-bit image by formula:

$$f_{out}(x,y) = \frac{2^N}{M-m}\left(f_{in}(x,y) - m\right),$$

where the input image f_{in} has minimum and maximum grey levels m and M ($M > m \geq 0$), respectively.

(b) Load `trui.tif` and apply `IPcontraststretch` to obtain the contrast-stretched version `trui_stretched.tif`. Also make plots of the histograms of both the input and output images (you can use the Matlab function `hist` for this purpose).

(c) Describe the effect that contrast stretching has on the input image and its histogram.

LAB 2.

Exercise 1 — Spatial filtering (50)

(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 (Equation 3-58) 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?

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

(a) Implement the Gaussian lowpass transfer function `H` of size $M \times N$ with cutoff frequency D_0, see Eq. (4-116), in a function `H=IPgaussian(D0,M,N)`.

(b) Write a function `IPftfilter(x,H)` to perform frequency-domain 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 lowpass filter it to duplicate the results in Fig. 4.44.

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 7.19. 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 7.19. 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 (40)

(a) Write a function `IPdwt2` to compute J-scale two-dimensional wavelet transforms with Haar wavelets. Base your implementation on the discussion of Wavelet transform in two dimensions in Section 7.10, pp. 520-524.

(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 two-scale DWT shown in Fig. 7.30(c). 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 — 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.

Exercise 1 — Binary morphological operations (30)

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

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

Exercise 2 — Boundary extraction (20)

(a) Write a function `IPboundary` that performs morphological boundary extraction of a binary image.

(b) Extract the boundary from `lincoln.tif`.

Exercise 3 — Opening by reconstruction (50)

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*.

(a) 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.

(b) 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.

(c) 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.

(d) 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.

Exercise 1 — Region splitting and merging (50)

The function `splitmerge` implements the region splitting and merging algorithm described in Section 10.4, pp. 768-770. 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 2 — Texture (50)

(a) Write a function `IPtexturemeasures` that computes the statistical texture measures in Eqs. (11-27)–(11-31). Normalize the variance before use in Eq. (11-29), 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.