university of groningen

faculty of science and engineering

Practicals Image Processing

November 2020

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 functions (not) to use

One of the goals of the lab exercises is to learn to develop your own image processing functions. Therefore, the following rules apply.

a. Built-in functions from the standard MATLAB library can be used, unless they have the same name (excluding the prefix, "IP") or essentially do the same as the function you are requested to implement. E.g. Histogram() cannot be used; Histc() or Repelem() can be used.

b. Also, you are not allowed to use any functions from the signal processing and image processing toolboxes, unless explicitly indicated in the exercise.

You can use the *which* command to see which function belongs to which toolbox/library.
Examples:

```
>> which histc
built-in (/Applications/Matlab/MATLAB_R2016b.app/toolbox/matlab/datafun/histc)

>> which histogram
/Applications/Matlab/MATLAB_R2016b.app/toolbox/matlab/specgraph/histogram.m

>> which downsample
Applications/Matlab/MATLAB_R2016b.app/toolbox/signal/signal/downsample.m

>> which histeq
/Applications/Matlab/MATLAB_R2016b.app/toolbox/images/images/histeq.m
```

NOTE: "signal" indicates the signal processing toolbox, "images" indicates the image processing toolbox.

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.
Note: use `imshow`, not `imagesc`, when comparing images before and after contrast enhancement.

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 **include listings of the Matlab source code** of all the required implementations in an appendix.

You also have to 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), student number 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 specific contributions (percentages) for each of you in terms of tasks performed (program design, program implementation, answering questions posed, writing the report). Your contributions will be assessed individually.

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`.

2.1 Lab 1

Start by reading Section 1.7 of this manual.

Exercise1 — Downsampling, upsampling, and zooming (30)

(a) (5pt) Write a function `IPdownsample` capable of downsampling (shrinking) an image. Assume that the desired downsampling (shrink) factor is an integer.

(b) (3pt) Load `cktboard.tif` and downsample the image by a factor of 4. Report the result.

(c) (5pt) Write a function `IPupsample` capable of upsampling an image by introducing zeroes. Assume that the desired upsampling factor is an integer.

(d) (3pt) Load `cktboard.tif` and upsample the image by a factor of 4. Report the result.

(e) (5pt) Write a function `IPzoom` capable of zooming an image by pixel replication. Assume that the desired zoom factor is an integer.

(f) (4pt) Load `cktboard.tif` and zoom the image by a factor of 4. Explain the difference between the output with that of (d).

(g) (5pt) Zoom the downsampled image in (b) back to the resolution of the original. Compare the result with the original image and explain the reasons for the differences you observe.

Exercise2 — Histogram equalization (35)

(a) (10pt) Write a function `IPhistogram` for computing the histogram of an 8-bit image.

(b) (15pt) Implement the histogram equalization technique discussed in Section 3.3 in a function `IPhisteq`.

(c) (10pt) Load `blurrymoon.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.

Exercise3 — Spatial filtering (35)

(a) (15pt) Write a function `IPfilter` to perform spatial filtering of an image (see Section 3.4 of the book) with a 3×3 mask. Do *not* use the standard Matlab functions `filter`, `filter2`, `conv`, or `conv2` .

(b) (10pt) Implement the Laplacian enhancement technique (3-54) in a function `IPlaplacian`.

(c) (10pt) Apply your function function `IPlaplacian` to `bubbles.tif`. Discuss the effects the Laplacian filter has on the image.

2.2 Lab 2

Exercise1 — Fourier spectrum (25)

(a) (10pt) Load `characters.tif` and compute its centered Fourier spectrum (that is, the magnitude of the Fourier transform, which is a complex number for each pixel, see Section 4.2 of the book). You may use Matlab's `fft2` and `fftshift` functions.

(b) (10pt) Display the spectrum.

(c) (5pt) Use your result in (a) to compute the average value of the image.

Exercise2 — Highpass filtering in the frequency domain (50)

(a) (20pt) Implement the Butterworth highpass filter (BHPF) transfer function `H` of size $M \times N$ with cutoff frequency D_0 and order n, see Eq. (4-121), in a function `H=IPbhpf(D0,n,M,N)`.

(b) (20pt) 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) (10pt) Load `characters.tif` and highpass filter it to duplicate the BHPF results in Fig. 4.53, right column. Discuss the results.

Exercise3 — Median filtering (25)

(a) (10pt) Write a function `IPmedian` to perform median filtering in a $(2k+1) \times (2k+1)$ window, with $k = 1, 2, 3, \ldots$.

(b) (5pt) Load `circuitboard.tif` and add salt-and-pepper noise to it, with $P_s = P_p = 0.2$. You can use the `imnoise` function.

(c) (10pt) 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).

2.3 Lab 3

Exercise1 — Laplacian pyramid: decomposition (50)

Let f be a grey scale image. The Laplacian pyramid of f is defined by the iteration:

$$\begin{aligned} f_1 &= f \\ f_j &= \text{REDUCE}\,(f_{j-1}), \qquad j = 2, \ldots J \\ d_j &= f_j - \text{EXPAND}\,(f_{j+1}) \qquad j = 1, 2, \ldots J-1. \end{aligned}$$

Here J is the number of levels of the decomposition; $d_1, d_2, \ldots, d_{J-1}$ are the detail signals (residuals) and f_J the approximation signal at the coarsest level. Furthermore, the reduce and expand operators are defined by:

$$\begin{aligned} \text{REDUCE}\,(f) &= \downarrow_2 \,(h_\sigma * f) \qquad \text{(Gaussian filtering followed by shrinking)} \\ \text{EXPAND}\,(f) &= h_\sigma * (\uparrow_2 \,(f)) \qquad \text{(zooming followed by Gaussian filtering))} \end{aligned}$$

where $\downarrow_2$ and $\uparrow_2$ denote shrinking/zooming by a factor of 2 in each direction, and h_σ the Gaussian filter kernel with standard deviation σ.

(a) (25pt) Write a Matlab function `g=IPpyr_decomp(f,J,sigma)` that implements the Laplacian pyramid decomposition. Parameters of this function are the input image `f`, the number of levels `J`, and the standard deviation `sigma` of the Gaussian filter. You may assume that the input is a square image of size $M \times M$, where M is a power of 2. For zooming, use the functon `IPzoom` implemented in Lab 1.
Hint: Be sure to convert `f` first to double by using the `im2double` function before starting the computations.
The output `g` should be a rectangular array `g` (of type double) of size $P \times M$. Start by initializing `g` with constant values (zeroes or ones).
Then insert the computed arrays $d_1, d_2, \ldots, d_{J-1}, f_J$ in `g` by vertically stacking them upon one another (first d_0, then centered below it d_1 which is half in size, then centered below it d_2 which is again half in size, etc.; and finally f_J; see Figure 1, right picture). This means that the vertical size P of the array will be defined by the formula

$$P = M \times \left(1 + \frac{1}{2} + (\frac{1}{2})^2 + \cdots + (\frac{1}{2})^{J-1}\right)$$

Sum this geometric series to obtain an explicit formula for P in terms of M and J.

(b) (15pt) Write a Matlab test script `IPpyr_decomp_test.m` that applies your pyramid decomposition function `g=IPpyr_decomp(f,J,sigma)` to the image `plant.tif` as input, with `J=3` and `sigma`=1.0. Your script should visualize both the input image `f` and the output `g` in a single figure (the output should look like Figure 1). See also the script `IPtest.m` provided in the `ImageProcessing` folder.

(c) (10pt) Save the figure produced by your script `IPpyr_decomp_test.m` to a file by the `saveas` command. Also save the decomposition data `g` to a `.mat` file by the `save` command. Include the name of the input image and the values of `J` and `sigma` in the filenames.

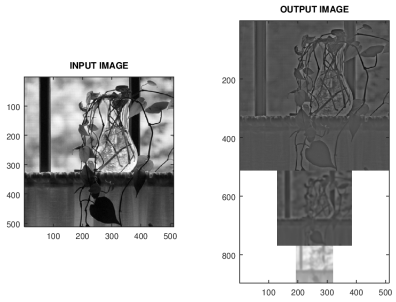

Figure 1: Left: input image. Right: result of the pyramid decomposition ($J = 3, \sigma = 5.0$).

Exercise2 — Laplacian pyramid: reconstruction (50)

Let f be a grey value image with Laplacian pyramid decomposition of J levels given by detail signals $d_1, d_2, \ldots, d_{J-1}$ and coarsest approximation f_J. As before you may assume that the image f is a square image of size $M \times M$, where M is a power of 2.
Reconstruction from the pyramid decomposition is defined by the formuala:

$$f_j = \text{EXPAND}\,(f_{j+1}) + d_j, \quad j = J-1, \ldots, 1,$$

where f_1 will be equal to the original image f. The EXPAND operator is defined in Exercise 1.

(a) (25pt) Write a Matlab function `g2=IPpyr_recon(g,J,sigma)` that implements the Laplacian pyramid reconstruction. Parameters of this function are the $P \times M$ pyramid decomposition array `g` (as defined in Exercise 1), the number of levels `J`, and the standard deviation `sigma` of the Gaussian filter.

(b) (10pt) Write a Matlab test script `IPpyr_recon_test.m` that applies your reconstruction function `g2=IPpyr_recon(g,J,sigma)` with the pyramid decomposition `g` of the image `plant.tif` as input, with `J=3` and `sigma`=1.0 (start by loading the decomposition array `g` from the file you saved in Exercise 1). Visualize both the original plant image `f` and the reconstructed image `g2` in a single figure.

(c) (5pt) Extend your Matlab test script `IPpyr_recon_test.m` by computing the error between the reconstructed image `g2` and the input image `f` (make sure that the files you compare have the same data type). Use the following formula for the mean absolute error between two $M \times N$ images f and g:

$$\text{error}(f,g) = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} |f(i,j) - g(i,j)|$$

Also, your Matlab script should compute the difference image `d` between `g2` (first converted to `uint8`) and the input image `f`.

(d) (10pt) Apply your extended script to the image `plant.tif`. Visualize both the original plant image `f`, the reconstructed image `g2`, and the difference image `d` in a single figure. Save the figure produced by your script to a file by the `saveas` command. This figure and the error value should be put in your report.

2.4 Lab 4

Exercise1 — Basic morphology (20)

Solve Problem 9.9 from the book. Sketch the solutions by hand, or use your favorite vector drawing program.

Exercise2 — Binary morphological operations (30pt)

(a) (10pt) 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) (10pt) Write a function `IPerode` that performs a binary erosion with an arbitrary 3×3 structuring element.

(c) (10pt) Test your functions on the image `wirebondmask.tif`.

Exercise3 — Skeletons (50)

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 of the course book. We define $(X \ominus kB) = X$ when $k = 0$.

(a) (20pt) 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. Use the recursive implementation discussed in the lecture sheets.

(b) (20pt) 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. Use the recursive implementation discussed in the lecture sheets.

(c) (10pt) Compute the skeleton of `nutsbolts.tif`. Demonstrate (experimentally) that the reconstruction equals the input image.