university of groningen

faculty of mathematics and natural sciences

computing science

# Practicals Image Processing

**November 2017**

# 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 *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. 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 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.

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.** Start by reading Section 1.7 of this manual.

**Exercise 1 — Downsampling, upsampling, and zooming (30)**

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

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

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

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

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

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

(g) 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.

**Exercise 2 — Histogram equalization (35)**

(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.

**Exercise 3 — Spatial filtering (35)**

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

(b) Implement the Laplacian enhancement technique (3.6-7) in a function `IPlaplacian`, and apply it to `tiger.tif`. Discuss the effects the Laplacian filter has on the image.

**LAB 2.**

**Exercise 1 — Fourier spectrum (30)**

(a) 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 4.2.4 of the book). 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 — Highboost filtering (70)**

Highboost filtering of an input image $f(x,y)$ produces an output image $g(x,y)$ by

$$g(x,y) = f(x,y) + k \cdot g_{\mathrm{m}ask}(x,y), \tag{1}$$

where $k$ is a constant with $k \geq 1$, and $g_{\mathrm{m}ask}(x,y)$ is a 'mask' image defined by

$$g_{\mathrm{m}ask}(x,y) = f(x,y) - f_{\text{smooth}}(x,y),$$

with $f_{\text{smooth}}$ a smoothing of the input image $f(x,y)$.
Assume that the input image has size $M \times N$, and that the smoothing operation which produces $f_{\text{smooth}}$ is implemented by the following $3 \times 3$ mask:

$$h_{\text{smooth}} = \frac{1}{5} \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 1 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

(a) Implement highboost filtering (Eq. (1) above, or Eq. (3.6-9) from the book) in a function `IPhighboost`. The averaging part should be done using the mask $h_{\text{smooth}}$ above (Fig. 3.32(a) from the book). Make use of the function `IPfilter` you developed in Lab 1.
Enhance `dipxetext.tif` and try to approximate the result in Fig. 3.40(e).

(b) The frequency domain transfer function of the smoothing filter (centered around $(0,0)$) is given by

$$H_{\text{smooth}}(u,v) = \frac{1}{5}\Big[1 + 2\cos(2\pi u/M) + 2\cos(2\pi v/N)\Big]$$

This frequency domain transfer function satisfies $H_{\text{smooth}}(0,0) = 1$. What property in the spatial domain does this formula correspond to?

(c) In the frequency domain, equation (1) has the representation

$$G(u,v) = H(u,v)\,F(u,v),$$

where the transfer function $H(u,v)$ has the form

$$H(u,v) = k + 1 - k\,H_{\text{smooth}}(u,v). \tag{2}$$

Give a derivation of formula (2). Clearly indicate what property of the Fourier transform you are using in the proof.

(d) Develop a *frequency-domain* implementation of highboost filtering (Eq. (1) above, or Eq. (3.6-9) from the book) in a function `IPhighboost_freq`. The averaging part should be done using the same mask as in (a), but now in the frequency domain. *Hint*: Consult the sheets of Lecture 2 for examples of frequency domain implementations.
Again enhance `dipxetext.tif` and compare the result with the spatial domain implementation you developed in (a). Are the results identical?

## LAB 3.

**Exercise 1 — Laplacian pyramid: decomposition (50)**

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

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

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:

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

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

(a) 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) 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`. 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) 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.

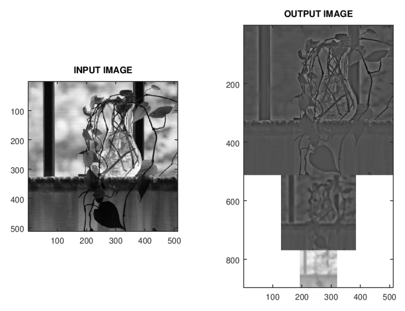

Figure 1: Left: input image. Right: result of the pyramid decomposition ($J = 3, \sigma = 5.0$).

**Exercise 2 — 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) 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) 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) 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, compute the difference image `d` between `g2` (first converted to `uint8`) and the input image `f`.
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.

**LAB 4.**

**Exercise 1 — Laplacian pyramid: partial reconstruction (40)**

This exercise is a continuation of Exercise 2 of Lab 3.
As we have seen, reconstruction from the Laplacian pyramid decomposition is defined by the formula:

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

where $f_1$ will be equal to the original image $f$.
Instead of this *perfect reconstruction* one may ignore some of the detail signals $d_j$ by setting them to zero before applying the reconstruction formula. This is called *partial reconstruction,* which may be useful for image compression.
Let $K$ be an integer with $0 \leq K \leq J-1$; $K$ is called the partial reconstruction level. In partial reconstruction of level $K$ all details $d_j$ for level $1 \leq j \leq K$ are set to zero. We can define the case $K = 0$ to correspond to perfect reconstruction.

(a) Adapt your Matlab function `g2=IPpyr_recon(g,J,sigma)` written in Lab 3 to a Matlab function `g2=IPpyr_recon(g,J,K,sigma)` that implements partial Laplacian pyramid reconstruction. Parameters of this function are the $P \times M$ pyramid decomposition aray `g`, the number of decomposition levels `J`, the partial reconstruction level `K`, and the standard deviation `sigma` of the Gaussian filter.

(b) Adapt the Matlab test script `IPpyr_recon_test.m` you wrote in Lab 3 so that it applies your function `g2=IPpyr_recon(g,J,K,sigma)` to the pyramid decomposition `g` of the image `plant.tif` as input, with `J=3`, `sigma=1.0`, and `K=2`. Visualize both the original plant image `f`, the reconstructed image `g2`, and the difference image `d` between `g2` and the input image `f` in a single figure. Also compute the mean absolute error between `f` and `g2`. This figure and the error value should be put in your report.

(c) Repeat (b) for `K=1` and add the results to your report. Compare and discuss the results obtained for the cases `K=2` and `K=1`.

**Exercise 2 — Binary morphological operations (30)**

(a) Write a function `IPdilate` that performs a binary dilation with an arbitrary $3 \times 3$ 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 $3 \times 3$ structuring element.

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

**Exercise 3 — Grey-scale morphology (30)**

(a) Implement grey scale dilation and erosion with arbitrary $3 \times 3$ 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.

**LAB 5.**

**Exercise 1 — Gradient with thresholding (30)**

(a) Write a function `IPgradientthresh(f,p)` that computes the gradient image $g = |f_x| + |f_y|$ of an input image `f`, and then thresholds the gradient image `g` at `p`% of the maximum value of `g`, with $0 \leq$ `p` $\leq 100$. The output of the function should be a binary image.

(b) Apply your function to `fingerprint.tif` for a number of values of `p` between 10 and 50 and report the results with your observations.

(c) Repeat the experiment, but now with the input image first uniformly smoothed.

**Exercise 2 — Edge detection (35)**

In this assignment you will implement the Marr-Hildreth edge detector according to Section 10.2.6 of the book. The essential step of this method is a convolution filter where the kernel is the *Laplacian-of-Gaussian* (LoG), with $\sigma$ the width of the Gaussian:

$$K_\sigma(x,y) = -\frac{2}{\pi\sigma^4}\left(1 - \frac{r^2}{2\sigma^2}\right) e^{-r^2/2\sigma^2} \qquad (r^2 = x^2 + y^2). \tag{3}$$

(a) Write a function `IPMarrHildreth(f,sigma,p)` that applies the LoG filter with standard deviation `sigma` to an input image `f`, and then thresholds the resulting image `g` at `p`% of the maximum value of `g`, with $0 \leq$ `p` $\leq 100$. The output of the function should be a binary image.
*Hint*. Compute a discrete approximation (floating point values) of the LoG kernel in Eq. (3) as an $n \times n$ matrix where $n$ has a value of around $6\sigma$. You may use the Matlab function `filter2` in your implementation.

(b) Apply your function to `house.tif` for `sigma`=1.0 and a number of values of `p` between 10 and 50. Repeat the experiment with `sigma`=5.0. Report the results with your observations.

**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.