1 General information

The labs are made in pairs. For each lab, you have the choice to make the exercises in either `Matlab`, `C` (using the provided imaging framework on Nestor), or `C++` (using the Cimg library, see `https://cimg.eu/`). Only students that have experience in programming in `C++` are advised to use the `Cimg` option, because this library is a so-called template library (template libraries are not suitable for novice C++ programmers). Nevertheless, `Cimg` is a very powerful library, which is highly recommended for real-world image processing.

1.1 Reporting

For all labs, a report is expected as a single PDF file 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 program code does 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); 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 and include it in the report.
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 something more advanced, then 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 if applicable: "In Figure 6 we see that .... this is caused by ....".

g. you include **listings of the source code** of all the required implementations in an appendix.
You also have to provide the source codes 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 session. **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.**

1.2 Getting images and auxiliary files

A collection of images related to the Image Processing Labs can be found on Nestor in the `Labs` tab (see `Course Documents/Labs`). Moreover, for those who want to do the lab exercises in `C` (which is highly recommended) a simple framework for basic image processing is available.
For each programming environment (`Matlab/C/C++`), a short document is available to set the stage. Read the document for the programming environment of your choice before you start the lab exercises.

2 Lab Session 1

Exercise 1 — Image Interpolation (20 pt)

(a) (5 pt) Write a function `resizeGreyImageNN(imageIn, sX, sY)` which returns a new image `imageOut` which is a scaled version of the input `imageIn`. Here `sX` and `sY` are positive real values (so, not necessarily integers!). If the image domain of `imageIn` is `[minX,maxX]`×`[minY,maxY]` then the output image has domain `[sX*minX,sX*maxX]`×`[sY*minY,sY*maxY]`. Use *nearest neighbour* interpolation.

(b) (5 pt) Write a function `resizeGreyImageBL(imageIn, sX, sY)`, with the same functionality which uses *bilinear interpolation*.

(c) (10 pt) Define some distance metric on images such that two images (that have the same domains) can be compared. Use this distance metric to quantify the quality of of the two interpolation techniques. In which situation would you use which interpolation technique?

Exercise 2 — Histogram stretching/equalization (20 pt)
(a) (5 pt) Write a function `contrastStrech(imageIn, min, max)` that returns a new image which is the *contrast stretched* version of `imageIn`. The dynamic range of the input image should be stretched to the new dynamic range `[min,max]`.

(b) (5pt) Write a function `histogramEqualization(imageIn, min, max)` that returns a new image that is obtained by *histogram equalization*. The dynamic range of the output should be `[min,max]`.

(c) (10pt) Perform contrast stretching and histogram equalization on the images (downloadable via Nestor) `gradient.pgm`, `gradient2.pgm`, `blurrymoon.pgm`, and `peppers.pgm`. Include the original images, a plot of their histograms, the output images, and a plot of their histograms in your report.
Use this information to explain why the resulting images were 'enhanced' as they were.

Exercise 3 — Quantization (10pt)
Write a function `quantization(imageIn, k)` that returns a new image which is a *grey level quantization* of the input image. The returned image has (at most) `k` grey levels. If the input image has fewer than `k` grey levels, then the output image should simply be equal to the input image. Otherwise, the output image has exactly `k` grey levels. Use the `k`-means algorithm to implement this function.
Apply the function to the `peppers.gm` image for several choices of `k` and discuss the results in your

report.

Exercise 4 — Connected Components (25pt)
(a) (10 pt) Write a function `deltaComponents` that computes the delta-components of an image: `deltaComponents(imageIn, connectivity, delta)` returns a new grey scale image that consist of a labeling of the *delta-components* of the input image. The parameter `connectivity` can be 4 or 8 for respectively 4-connected or 8-connected adjacency. Two pixels p and q should have the same labeling if and only if they are connected via a path that corresponds with the supplied connectivity, and in each step of the path the absolute difference in grey level is at most `delta`. In the output image, the pixels p and q should have the same grey level (label). Once you have completed the code, apply it to the `emmius.pgm` image for several values of `connectivity` and `delta`, and write your observations in your report.

(b) (5 pt) Write a function `components(imageIn, connectivity)` that returns a new grey scale image that consist of a labeling of the *connected components* of the input image. The parameter `connectivity` can be 4 or 8 for respectively 4-connected or 8-connected adjacency. Note that two pixels p and q are connected if they have the same grey value, and they are connected via a path that corresponds with the supplied connectivity. In the output image, the pixels p and q should have the same grey level (label).

(c) (10 pt) Write a program that reads the `dice.pgm` image and outputs the number of dice, the histogram of the number of pips, and the total number of pips. Explain in your report which steps your program makes. You are not allowed to used morphological operations (you will do that in the next exercise).

Exercise 5 — Basic Mathematical Morphology (25pt)
(a) (10 pt) Write a function that, given a binary image and a structuring element (which is itself a small binary image), returns a new binary image which is the *dilation* of the image with the structuring element. Note that, dependent of which frame work you are using (C/Cimg/Matlab), you may have to pass the image domain and the origin locations as extra parameters to the function. Explain in your report the implementation details. You are not allowed to use built-in library functions that already implement morphological operations.

(b) (5 pt) Similar as in part (a), write a binary erosion function.

(c) Repeat exercise 4(c), this time using morphological operators.