Introduction to Image Analysis

Compulsory Assignment 1

Version 1.0

Allan Rasmusson

2023.09.27

1 Assignment Description

This document (8 pages in total) describes the first compulsory assignment for the course "Introduction to Image Analysis" taking place at Faculty of Mathematics and Informatics, Vilnius University, Lithuania, Autumn 2023. This report is the first of a total of three assignments; it will be scored between 0 and 100 points and weighted by 30% in the overall grade for the course.

In short, the assignment concerns the practical exercises of the first five weeks of the course, and, as such, can be interpreted as giving a summary of the material covered in the course so far and the practical implementation of it.

Engineering and Computer Science students must use C++ as implementation language while Data Scientists and pure Mathematics students are allowed to choose between C++ and Python. The choice of implementation language used in Compulsory Assignment 1 must be used for upcoming compulsory assignments 2 and 3 also.

The student must include a written report with a short description of each topic, describe how it was implemented and discuss any choices made in the implementation and applications. Both the report *and* the implemented code (C++ or Python) must be handed in. You must include a description of how each example can be run on a command line in Ubuntu Linux. For C++ code you must also describe how to compile it using a combination of `make` and `qmake`.

Although the assignment covers the practical exercises already handed out, some clarification and minor adjustments are described for this summarizing assignment in the following sections. The handed out data is available in a zip file accompanying this document and under "Files" in the "General" channel in the MIF-IA Teams Channel (link).

The report and code must be handed in a single zip-file uploaded in the VU Emokymai system.

Deadline is Sunday 2023.10.22, 23:59, one minute to midnight.

Some Advice:

- Read the assignment description *at least* twice!
- Ask questions as early as possible.
- If you are having problems with some parts of the exercises, continue to the next and make it work with what you've got.
- Make sure you have access to the handed out files.
- Think about which part of the material is relevant for each weekly topic and describe this in the report.

2 Assignment Details

2.1 Image Loading

Purpose: Get practical understanding of different image formats, their memory layout and how to combine/manipulate these. Visualizing the images is done in as either RGB or grayscale, but always using 8 bits per channel.

Implementations and Tasks:

- Image investigations:
  Use the commandline tool `tiffinfo` to investigate the handed out images and describe the information you encounter. Comment on any observations you make.
- Image Loading
  - Loading images using libtiff. For C++ load tiled and striped formats. For Python load whole images and sub-rectangles using `read_one_tile`.
    Load and visualize the images:
    `Kidney1.tif`
    `Region_001_FOV_00041_Acridine_Or_Gray.tif`
    `Region_001_FOV_00041_DAPI_Gray.tif`
    `Region_001_FOV_00041_FITC_Gray.tif`
    `SkinOverview.tif`
    `TMA2-V2.tif`
    If you encounter any anomalies investigate potential reasons for this using the information you extracted with the `tiffinfo` command. Describe the anomalies and reasons you find for them (you do *not* need to correct any anomalies as long the image is otherwise loaded correctly).
  - Load and visualize different sub-images of the svs file: `Kidney2_RGB2_20x.svs`. Describe the internal tiff file structure and comment on why it could have been designed like this.
- Image combination:
  - Combine the three fluorescence images list below into one contiguous RGB image and save it to disk.
    `Region_001_FOV_00041_Acridine_Or_Gray.tif`
    `Region_001_FOV_00041_DAPI_Gray.tif`
    `Region_001_FOV_00041_FITC_Gray.tif`

Data: Images in `ImgSet1.zip`

2.2 Intensity Transformations

Purpose: Gain practical experience with:

- Image intensity histograms.
- Image intensity transformations and how they affect the image by looking at the intensity histograms.
- Precision for intermediate calculations.

Implementations and Tasks: Only 8 bit grayscale images will be used, and so it's natural to use an `unsigned char` or equivalent 8-bit type to hold pixel intensities. You must yourself consider if intermediate calculations should be done in decimal numbers, e.g. `floats`. Describe your choice.

- Implement the power law for intensities (you may disregard the constant c)
  Describe your implementation.
  Visualize the effect of γ smaller and larger than 1 in the power law.
- Implement piece-wise linear intensity transformations.
  Give an example (and visualize the results of applying) one histogram stretching transformation and one thresholding.
- Implement histogram calculation for grayscale images.
- Implement histogram normalization.
  Compare the piece-wise linear transformation applied above to the histogram normalization function.
  Explain how/if the number of intensity values changes when performing a histogram normalization.
- Implement lookup table intensity transformations and use it to perform the intensity transformations above.
  Discuss the differences between evaluating the intensity transform for each pixel and using a look-up table.
  Are there any image/pixel formats that are not well suited to application of a look-up table?

Data: Images in `ImgSet2.zip`

2.3 Multistep Image Processing and Spatial Filtering

Purpose: Gain practical experience with:

- Linear and non-linear filtering in the spatial domain.
- Multi-step image processing.

Implementations and Tasks: Only 8 bit grayscale images will be loaded and visualized. Intermediate results will be calculated using images that holds pixel intensities in *floats*. The conversion back to the original 8 bit image format is needed as viewing of only 8 bits images will be allowed.

- Implement separate separate image arrays (or image classes) for with pixel intensities stored in `8-bit/unsigned char` and `float`, and methods to convert between the two formats. The conversion steps must be described in the report.
- Illustrate working image blurring by spatial filtering.
  Use any appropriate image.
- Illustrate image sharpening using intermediate linear filtering steps unsharp mask and Laplacian.
  Use any appropriate image.
- Calculate magnitude of gradient using Sobel operators and show an application.
  Use any appropriate image.
- Redo example 3-43 using multistep image processing. Describe handling of intermediate results.

Data: Images in `ImgSet3.zip`, which include example 3-43.

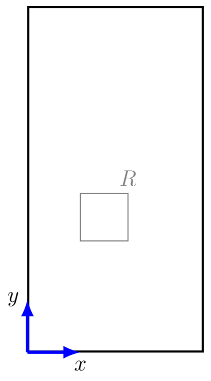
2.4 Image Transformations

Purpose: Gain practical experience with:

- Different image coordinate system (both 2D world and local coordinates, 2D pixels indices).
- Composite transformations for transforming between image spaces and to apply arbitrary affine transformations.
- Application of affine transformations to 2D images.
- Image interpolation schemes for constructing result of affine image transformations.

Implementations and Tasks:

- Assigning coordinate systems to images and calculation of index-to-local and local-to-world transformations taking pixel size into account.
- Composite affine transformations.
- Nearest Neighbor and Bi-linear Interpolation for visualization of transformed images.
- Visualize different affine transformations using any appropriate image. Render images with all implemented interpolation types.
- Extraction of specific image regions from `Kidney1-Res.tiff`.
  In the world coordinate system using µm units, the image is placed as follows:



  - Extract the tissue in the region R (gray box) given by (world) coordinates:

$$(134, 218) \times (204, 330) \text{ [µm]}$$

  - Extract an image region around the same tissue piece in R by manually defining an image grid located at the center of R and rotated so that the amount of white background pixels are minimized (You determine width, height and angle).
  - Visualize the extracted regions with the *same* pixel size as the `Kidney1-Res.tiff` and a pixel size 3 times smaller. Use both implemented interpolation schemes.

Data: Images in `ImgSet4.zip`, which include revised `Kidney1-Res.tiff`.

A Handing in code

A.1 Python

- You must hand in your code and report in a single zip-file.
- The archive will be extracted to a directory, e.g. `student-xx`.
- All handed out image data will be copied here in a directory named “data”.
- The code will then be run by:
  passing a single file to the interpreter on the commandline
- For different examples, you may:
  - have different files, e.g. `imageIO.py`, `intensitytransform.py`, etc.
  - You may use different input args to run separate examples.
- You *must* describe in the report how to run each example.
  - **NB**: Do *not* rely on reviewers editing your code to run different examples!
    Editing your code may introduce unintended behaviours and will *not* be done.
    You risk not being scored for such examples.
- The python process could look like this on the Ubuntu command line:

```
> mkdir student-7
> unzip student-7.zip student-7
> cd student-7
> cd ``CodeSubdirectory'' #... or similar, describe in report
> cp -r /from/somewhere/data .
> python3 imagefilters.py ex1
> python3 imagefilters.py ex2
...
> python3 imagetransforms.py ex3 #... or similar, describe in report
...
```

A.2 C++

- Students using C++ will also hand in code and report in a single zip-file.
- The archive will be extracted to a directory, e.g. `student-xx`.
- All image data will be copied here in a directory named “data”.
- The code will then be compiled by running `make` and `qmake`.
- For different examples, you may:
  - have different files, e.g. `imageIO.cpp`, `intensitytransform.cpp`, etc.
  - You may use different input args to run separate examples.
- You *must* describe in the report how to run each example.
  - **NB**: Do *not* rely on reviewers editing your code to run different examples! Editing your code may introduce unintended behaviours and will *not* be done. You risk not being scored for such examples.
- The c++ process could look like this on the Ubuntu command line:

```
> mkdir student-3
> unzip student-3.zip student-3
> cd student-3
> cd ''CodeSubdirectory'' #... or similar, describe in report
> cp -r /from/somewhere/data .
> cd imagelib; make
> cd ../viewer; qmake
...
> ./imageIO ex1        #... or similar, describe in report
...
```