Introduction to Image Analysis

HT 2024

Computer Exercise 1: Histogram analysis. Segmentation by thresholding. Colour

In this exercise you will explore some fundamental image processing techniques, such as histogram analysis and segmentation. You will work with not only intensity images, but also images composed of many spectral bands. You will learn how to use several existing MATLAB functions, but will also write your own scripts to solve the image analysis tasks.

Image segmentation is the task of partitioning an image into regions of pixels, so that the pixels in the same region fulfil a certain homogeneity criterion, while the different regions are separated by a certain discontinuity criterion. The regions may contain, for example, pixels of the same greylevel intensity, or the same colour, but the criteria are most often much more sophisticated, application dependant and can be numerous. Because of this diversity and complexity of the possible segmentation criteria, a large variety of different methods have been proposed to solve the segmentation task. We will start with the simplest one: thresholding of the pixel intensity values based on histogram analysis. We will also try to apply some additional operations to preprocess the image, with an aim to improve performance of the subsequent segmentation.

Every part of the exercise includes instructions that will guide you through the tasks, some additional explanations and pointers to further reading, and finally some concrete tasks you should complete and questions you should answer. Read them **before** *actually performing the task. When answering, focus on understanding and explaining* **why***, rather than* **how***, you got a particular result. If you get stuck, do not hesitate to contact the Teaching Assistants, preferably during the scheduled lab session. Try to prepare for the scheduled session, to get the most of it!*

Formalities

- **Team up!** You will work in teams of two, in a different team for every computer exercise. The teams are formed in Studium - start with assigning yourself into one of the teams for this exercise, under People → Computer Exercise 1. Initiate communication as soon as possible! We believe that discussions within the team will have positive impact on your learning experience.
- You are requested to **prepare a written lab report** in pdf format – one per group. Please, include answers to all the questions, include explanatory illustrations and, when suitable, your (pseudo-)code. Your report should follow the template provided in Studium, in the Module 'Labs-General'. In addition to the Lab report template, you can find there some useful notes on scientific writing, as well as general instructions related to the computer exercises.
- Please, upload your report to Studium, to 'Assignments' → 'Assignment Lab 1'. If you submit the report by the indicated deadline, the TA will take a look at it, give feedback, and grade it rather quickly (within a week). Assessment of late submissions may take much longer time.

- Follow the status of your reports in Studium, under 'Assignments' and 'Grades'.
- **Deadline: September 16, 2024**

Getting started

Download the instructions and the lab material from Studium, 'Assignment Lab 1'.

1 Histogram Equalization

You will start with writing your own procedure to perform histogram equalization. Histogram equalization spreads the used greylevels in an image as evenly as possible. In the ideal case, this would make the histogram of an equalized image completely flat. However, due to quantization we can only approximate this. For a flat histogram, the grey-value of each pixel should be proportional to the relative number of pixels in the image which are darker than the observed pixel. The proportion of pixels darker than a given value is expressed by the *Cumulative distribution function* (CDF).

Read [Szeliski 2022], Section 3.1.4, for more details on histogram equalization.

- Load the image `images/greytoys.png`.
- Perform histogram equalization on this image using your own algorithm, without using `histeq`. Efficiency is not so important, you may use for-loops or other ways to compute the transformation and apply it to the image.
- Make your own histogram equalization into a callable function, i.e., so that you can execute it as `Inew = myhisteq(I)`.

Some hints:

- A histogram of an image can be produced by `h = imhist(I)`
- `cumsum` is a useful function
- `Inew = T(I)` will replace values in I using T as a *lookup table*, e.g. `T = [1 4 9 16]` can be used to map 1 to 1, 2 to 4, 3 to 9 and 4 to 16. Try: `I = [1 2; 3 4]; T(I)`. Observe that all values in I have to be integers larger than 0.

1. ***Utilising the*** `subplot` ***command, present together in a Figure the original and equalized images, the original and equalized histogram, as well as the Empirical cumulative distribution function for the image equalized by using your own code. Compare with MATLAB's native*** `histeq` ***function. (Note that*** `histeq` ***defaults to 64 bins.)***
   Include your code for `myhisteq` ***in the Lab Report.***

2 Global Thresholding

In this task you will explore methods to perform segmentation using (global) thresholding.

2. ***Open the Matlab-included image*** `rice.png` ***and inspect the intensity histogram. Given this histogram, what do you expect a good threshold to be if you want to segment the image into grains of rice and the background?***

The following is a simple and intuitive iterative thresholding algorithm:

1. Select an initial estimate for a threshold T;
2. Segment the image into "dark" (intensities $\leq T$) and "bright" (intensities $> T$) regions;
3. Calculate the averages m_1 and m_2 of the pixel intensities in "dark" and "bright";
4. As a next estimate for T choose the average of those two means, $T = \frac{m_1+m_2}{2}$;
5. Repeat from 2, until convergence.

This approach can be thought of as iteratively grouping pixels to its nearest class mean m_i. Actually the method is equivalent to a clustering method known as *k-means clustering* with the number of clusters $k = 2$. You will learn more about that method in a few weeks.

In the provided script `IterativeThresh.m` you can find code for implementing the above algorithm in a fast way using the image histogram. Read through and test the script.

3. ***Using iterative global thresholding, does your initial guess for a threshold matter? If so, in what way?***

Next, you will implement Otsu's method for finding a threshold. Otsu's thresholding aims to find the optimal threshold between two groups of pixel intensities in the histogram of an image. The idea is to interpret the image histogram as a mixture of two probability distributions and to try to find the threshold which separates them as well as possible. Although you do not know the probability distribution of your two classes beforehand, it is possible to minimize the variance within the classes, by minimizing

$$\sigma_w^2 = P_1\sigma_1^2 + P_2\sigma_2^2 \,,$$

where P_1 and P_2 are the relative numbers of pixels in the first class and second class, respectively (i.e., the empirical probability for a random pixel to belong to the resp. class).

Otsu showed that minimization of the above expression is equivalent to maximization of the variance between the two classes, which is the same as maximization of

$$\sigma_b^2 = P_1(m_1 - m_g)^2 + P_2(m_2 - m_g)^2 \,,$$

where m_g is the global mean and m_1 and m_2 the means of the two classes.

Instead of starting from an initial estimate, Otsu's method simply tries all threshold levels and picks the one which maximizes σ_b^2, as expressed above.

4. ***Write a function which performs Otsu's method to threshold a given greyscale image. Do plot the value of σ_b^2 for the different thresholds to get a better feeling for how the method works.***

5. ***Include pseudo-code of your implementation in your report.***

6. ***Compare the result of your own implementation of Otsu's method to MATLAB's built-in function* `graythresh` *on a provided* `images/hand2BW.png`*. Do you find the same threshold? Note: Make sure that your threshold is in the same range as your image data format when you compare them, i.e. for* uint8 *a threshold at 128 corresponds to a threshold at 0.5 if you are using* double *and your intensities are in the range of [0,1].***

3 Colour Spaces

So far, we have experimented with intensity (i.e. greyscale) images, in which each pixel is assigned a single (scalar) value. Let us see what we can do with images in which each pixel is assigned a vector (i.e., the image function is a vector-valued function). Each component of this vector is called a *channel* (greyscale images are single-channel images).

Most often appearing, and probably most important, are colour images; they are represented by three channels (to match the sensitivity of the human visual system). There are several ways to define these three channels, and the corresponding *colour spaces*. Most commonly, colour is represented in the RGB colour space, with a red, a green, and a blue channel. Each channel separately (each component of the vector) can be regarded as one separate intensity image (of the same spatial dimension as the observed colour image), and can often be processed as such, using the tools and methods applicable to greyscale images. However, it is often beneficial to utilize the combined information from the multiple channels. As an example, conversion of an RGB image to a greyscale image is commonly performed by computing a linear combination of the R,G, and B channels.

$$0.2989 * R + 0.5870 * G + 0.1140 * B$$

is the linear combination used in the MATLAB command `rgb2gray` which approximates the perceptual brightness of each colour. The image `greytoys.png` used above (when you tried histogram equilization) was created using `rgb2gray`. There are many other ways to convert a colour image into greyscale.

RGB is not the only colour space used to represent colours and colour images in image processing. Other commonly used colour spaces are HSV (Hue-Saturation-Value), YCbCr and L*a*b*. You learned about them in Lecture 2. To learn more about the colour spaces and conversions between them, see Color Spaces. You may also take a look at [Szeliski 2022], Section 2.3.2.

The following steps will guide you through some of the mentioned operations with colour images:

- Open the image `images/toysflash.png`. Your aim is to separate the white ball in the foreground of the image from the rest of the scene using Otsu's thresholding.
- MATLAB provides a useful function `colorcloud`, to visualize the pixels in the image as a 3D point cloud in the specified colour space. Try the function `colorcloud`, and look at how the pixel values of the image are located in RGB, HSV, YCbCr and L*a*b* colour spaces.
- Write a function which splits a 3-channel image into its three individual components (channels) and displays all of the three next to each other using `subplot`.

- Perform Otsu's thresholding, separately on each of the channels in the RGB space.
- Convert the RGB image into HSV colour spaces by using MATLAB's functions for that. Try again to segment the ball by Otsu's thresholding, now on each channel in the HSV space.

7. ***Do you manage to segment well the white ball in any of the channels? Include into your report a figure of the channel where segmentation worked best.***

Let us use more channels at the same time. One way to do so could be to mark regions in the 3D scatterplot given by `colorcloud` function. Marking regions in 3D is, however, not trivial with a 2D interface and a mouse. Therefore you will try a 2D version instead.

- Try the `colorseg_RB.m` function, which allows you to interactively mark a polygonal region in a 2D histogram of the R and B channels. Double-click to place the last corner point of your marked region. You can edit vertices and move the polygon over the 2D histogram. Do you manage to better segment the ball now? What about the Blue cup?
- Try the `colorseg_SV.m` function as well. It does the same as `colorseg_RB.m`, but using Saturation and Value channels instead of R and B.

8. ***Include the images with your best segmentation results for the R+B and the S+V combinations in the report.***

4 Multispectral Images

Some imaging devices and techniques generate images with more than three channels. Such images are referred to as multispectral images, in which each channel corresponds to a particular spectral band, i.e., a specific wavelength range across the electromagnetic spectrum. Multispectral images typically have 4 to 15 channels and can capture information which human eye fails to capture with its three receptors (for visible light). Hyperspectral imaging extends this further, often utilizing hundreds, or even thousands of spectral bands (channels); pixels in hyperspectral images are represented by close-to-continuous spectra.

Let us see what is visible in a multispectral image. You will use an image showing parts of Uppsala, acquired by the Landsat 7 Instrument. This instrument produces images with 7 spectral bands:

```
Band 1 Visible (0.45 - 0.52 µm) 30 m.
Band 2 Visible (0.52 - 0.60 µm) 30 m.
Band 3 Visible (0.63 - 0.69 µm) 30 m.
Band 4 Near-Infrared (0.77 - 0.90 µm) 30 m.
Band 5 Near-Infrared (1.55 - 1.75 µm) 30 m.
Band 6 Thermal (10.40 - 12.50 µm) 60 m Low Gain / High Gain.
Band 7 Mid-Infrared (2.08 - 2.35 µm) 30 m.
```

Load the provided multispectral satellite image into MATLAB by using the command `load landsat_data`.

This image has seven channels. Out of these, humans can see only the first three, which belong to the visible spectrum: the spectral bands for blue (Band 1), green (Band 2), and red

(Band 3). Nevertheless, each of the seven bands can be presented as an intensity (greyscale) image and can be shown on the screen. We can visualise them using the R, G, and B colour channels. This means that we can visualize any combination of three (but not more) spectral channels by treating them as if they corresponded to R, G and B colour channels. In this case R, G and B as so-called pseudocolours.

To visualize, e.g., the 4th, the 1st, and the 3rd channel of the multispectral image, you can use the following command:

```
imshow(landsat_data(:,:,[4,1,3]));
```

To display all bands as a montage of greyscale images, use the command:

```
montage(landsat_data);
```

Open all the seven channels of the `landsat_data` as separate intensity images. Do you recognize the landscape? The multispectral image contains regions covered by forest, water, urban areas and agricultural areas around Uppsala. Can you identify them in the seven bands? Make different combinations of the seven channels and try to interpret the result. Which areas are best visible?

9. ***Discuss the visualisations of different combinations of spectral bands. Do you notice difference in resolution of the channels?***
10. ***Include in your report the combination which best visualises the urban area, as well as the one that highlights the agricultural area.***

5 Pixel Neighbourhoods – Mean Filter

After exploring individual pixel intensities and point-wise operators (in the tasks above), we will now try to consider groups of neighbouring pixels. Having on mind geometric organization and structure of image data, it seems natural to include some contextual information when processing particular pixels. A simple approach is to compute average intensity values from the neighbourhood around each pixel - you will notice that it may sometimes be beneficial to replace the original pixel value with this computed average. Your task now is to automate this averaging process.

Make a script which includes the following steps:

- Load the image `images/toysflash.png`.
- Convert the image to greyscale using e.g. `rgb2gray`, and into a floating-point (double) data format (e.g. by using `im2double`).
- Crop the image to make it square, and then resize it to 128×128 pixel size.
- Make your own code for mean filtering: position a window of a size of 5×5 pixels at every pixel location in the resulting (128×128) image, compute the mean value of the pixels covered by the window, and store the obtained value at the corresponding pixel position in a new image. Ensure that the result has the same size in pixels as the original (128×128) image by treating the borders of the input image in some controlled way. Do not use the built in filtering functions in Matlab, eg. *imfilter*.

The described procedure is refereed to as image filtering. We will get back to it in Lecture 3. The averaging window is called Mean (or average) filter.

- Subtract the newly generated (filtered) image from the original (128×128) image.

11. ***Present the original, filtered, and difference images in a figure, using e.g.*** `tiledlayout` ***(you can also use*** `subplot`***), and*** `imagesc`***. Comment the difference image - what is the effect of the performed subtraction?***

12. ***Explain the approach you took when dealing with the image borders. Discuss the effects of your choice on the filtering result. You may want to take a look at Szeliski, Section 3.2 (Padding, Border effects) for some ideas and discussion.***

6 Preprocessing and Segmentation

Now it is time to combine the steps you learned and see if a small resulting image analysis pipeline can lead to some improved solutions.

Revisit segmentation by Otsu's thresholding.

- Open the `images/hand2BW.png` image and use Otsu's thresholding to create a binary image, by segmenting the hand from the background. How does Otsu's thresholding perform in this task? What is the reason for the poor performance of the Otsu's method - why does it fail to separate well the hand from the background? Inspect the image histogram. Where in the histogram do you expect the pixels which belong to the hand?
- Apply your 5×5 mean filter (the one you implemented in the previous task) on the `images/hand2BW.png` image. Inspect the histogram of the filtered image. Apply Otsu's thresholding on the filtered image and report your findings.

13. ***Try to explain the observed performance of Otsu's thresholding on the two images and give arguments for your conclusions.***

14. ***Include both the histograms (of the greyscale images) as well as the thresholded binary images in your report.***