Elementary image operations

The goal of the labs is to gain practice with some of the fundamental techniques in image processing on grey-level images, such as stretching of grey-level values, histogram equalization, sampling and quantization. You will visualize image information in a number of different ways. The lab should be completed by simultaneously study the related sections in the course literature.

Examination: For a more efficient examination, prepare script files, such that the results can easily be reproduced. Questions that are explicitly stated in the text, should be answered in written. If results take too much time to compute, you should complement the script files with print outs.

Before you begin read the instructions in "General laboratory guidelines" and study the related sections on grey-level transformation and histogram equalization in the course literature and seminar notes.

Reading images: Image formats

Images used in this lab are stored in the file system either as

1. MATLAB-variables stored using the command "`save`" in files under the directory `$DSGKHOME/Images`
2. function definitions stored in files under the directory `$DSGKHOME/Images-m`
3. jpeg- and tiff-files under the directory `$DSGKHOME/Images`

If your search paths are correctly set, you can load the files using:

1. Reading MATLAB-variables [1]

   ```
   load canoe256
   ```

2. Reading images defined as function definitions

   ```
   nallo = nallo256;
   ```

3. Reading jpeg- or tiff-images

   ```
   hund1 = double(imread('dog.jpg'));
   hund2 = double(imread('dog.tiff'));
   ```

The size of an image can be found with "`whos`" or

```
size(variable)
```

If you want to know the largest and smallest value in an image, you can use:

```
max(max(Canoe))
max(Canoe(:))
```

The reason for the double calls is that the function `max` applied to a matrix returns a vector of max-values, one value for each column. The alternative notation `Canoe(:)` means that the image matrix is regarded as one long column vector.

[1] If you are not already familiar with the `save`-command, read the documentation with `help save`. From now on it is taken for granted that you use "`help command`" to learn more about commands mentioned in these instructions. Image `Canoe` is stored in a MATLAB-variable. To see the defined variables, as well as their sizes, write "`whos`".

1 Displaying images: Colour tables and quantization

MATLAB has an embedded function `image` to visualize matrices. Since there are many alternatives to represent colors, the translation from numbers in the matrix to colors on the screen is not fixed. Instead a colour table is used. This colour table is represented by a matrix of three columns, each of which assumes values in the interval $[0, 1]$. These columns respectively represent the red, green and blue colour components. When the command

```
image(Canoe)
```

is called, the value of an image element of the matrix `Canoe` is used as an index to this table. All floating-point values are rounded to its nearest integer and values outside the table's definition area are rounded to the nearest boundary value. [2] Let us take a look at an image. Load the canoe image following the instructions in the beginning of this lab. In order not to rescale the pixel values to the interval $[0.5, 64.5]$ we simply use a colour table of 256 grey-levels

```
colormap(gray(256))
```

Question 1: **What happens in the image apart from the color change?**

You may then look at the image with

```
image(Canoe)
```

When displaying an image, we want the image elements to be square. This can be achieved with

```
axis equal
```

When working with images of different types, you may need to make sure that the definition area of the colour table corresponds to the interval in which pixel values exist. A MATLAB-function `showgrey` is provided in the function library of the course. If nothing else is stated, this function first computes the largest and smallest values, and then transforms this interval of grey-levels to an interval of a grey-level colour table of 64 levels. You may also explicitly state the length of the colour table, as well as the interval of values to be shown. For more information on this write

```
help showgrey
```

Now try to look at the canoe image with

```
showgrey(Canoe)
```

[2] You can determine the length of the current colour table by "`size(colormap, 1)`", which returns the number of rows of the matrix. For the display mechanism to work, all values in the matrix to be displayed should be in the interval $[0.5, M + 0.5]$, where M is the length of the colour table.

You may show the elements of the colour table by writing "`colormap`". To graphically visualize the colour table, you write "`plot(colormap)`", which results in three curves being displayed–one for each component. If only one curve is shown, the three curves probably overlap, i.e. the colour table is grey-level.

MATLAB provides a number of predefined colour tables. You may see all defined tables by writing "`help graph3D`". Please, test some other colour tables using commands such as "`plot(hsv)`" and "`plot(copper(16))`". Note that the length of the colour table can be adjusted.

and change the number of colour levels by varying the display function's second argument in even powers of two in the interval $[2, 256]$

```
showgrey(Canoe, 256)
⋮
showgrey(Canoe, 2)
```

This technique simulates the results of quantizing the image using different numbers of bits. Try to apply the method to the following image

```
phone = phonecalc256;
```

Question 2: **Why does a pattern appear in the background of the telephone image? How many grey-levels are needed to get a (subjectively) acceptable result in this case?**

To exclusively base the display of grey-level images on the largest and smallest values has limitations for images in which only a few image elements have values significantly different from remaining values. If you display such an image, the result may typically be very dark (light) with a few light (dark) areas. Try this by loading the following image

```
vad = whatisthis256;
```

and look at it using the default settings with

```
showgrey(vad)
```

Then from the largest and smallest image values (found as described in the first section) enhance different parts of the grey-level interval by applying different interval limits `zmin` and `zmax` to `showgrey`

```
showgrey(vad, 64, zmin, zmax)
```

Question 3: **What does the image show? Why is it difficult to interpret the information in the original image?**

Look at the image "`nallo256`" with the visualization routine "image" and colour tables

```
colormap(gray(256))
colormap(cool)
colormap(hot)
```

Question 4: **Interpret the plots in terms of image content. Which colormap makes the visualization more clear and why?**

---

2 Subsampling

The density of image elements used to sample an image affects, in a fundamental way, the information contained in the image. To simulate the effect of this we shall in this exercise take a given image and reduce the resolution at which image information is represented. In MATLAB the resolution of an image matrix may e.g. be reduced as follows:

```
function pixels = rawsubsample(inpic)
[m, n] = size(inpic);
pixels = inpic(1:2:m, 1:2:n);
```

(This function already exists in the function library of the course). To illustrate the effect of this function, first apply it to a simple test image, such as the 9×9-pixel image defined by the following function call:

```
ninepic = indexpic9
rawsubsample(ninepic)
```

Question 5: **Repeatedly apply the subsampling operator to some of the images mentioned above. What are the results and your conclusions?**

As a comparison you should also study the effect of the subsampling operation

```
function pixels = binsubsample(inpic)
prefilterrow = [1 2 1]/4;
prefilter = prefilterrow' * prefilterrow;
presmoothpic = filter2(prefilter, inpic);
pixels = rawsubsample(presmoothpic);
```

in which the image is filtered with a binomial kernel coefficients

$$\begin{pmatrix} 1/4 \\ 1/2 \\ 1/4 \end{pmatrix} (1/4, 1/2, 1/4) = \begin{pmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{pmatrix}$$

before the pure subsampling stage. If you are unaware of the connections between linear filtering and subsampling you should not worry. We will talk about this later on in this course. Here you only

have to observe the qualitative effects and try to draw conclusions on how and why the methods are different. Apply functions `rawsubsample` and `binsubsample` twice to the image `phonecalc256`.

Question 6: **Describe in which ways the results are similar and different. Explain the reasons behind the differences.**

Question 7: **What will the results be if you repeatedly apply these two types of operators to a textured image?**

---

3 Grey-level transformations and look-up tables

A common way to transform an image is via grey-level transformations. If the purpose is to show an image on screen in MATLAB we may perform these using one of these methods

1. apply embedded point-wise operations on image data,
2. transform grey-levels via a look-up table,
3. create a new colour table.

3.1 Point-wise operations

In MATLAB there is a number of embedded functions within the first category (see e.g. "`help elfun`") that can easily be applied using the image's variable name as argument:

```
neg1 = - Canoe;
showgrey(neg1);
neg2 = 255 - Canoe;
showgrey(neg2);


nallo = nallo256;
showgrey(nallo.^(1/3));
showgrey(cos(nallo/10));
```

Investigate the histograms of the first two image with

```
hist(neg1(:))
hist(neg2(:))
```

Question 8: **Related to neg1 and neg2 - why are the histograms different but images similar? Related to nallo - explain how the above transformation functions affect the image histograms.**

3.2 Look-up tables

For some types of grey-level transformations it may be necessary to represent transformations in terms of look-up tables. Typical examples are transformation that imply extensive computations for each image element, or if a closed expression for a transformation cannot be found.

For methods based on look-up tables, the image value of each point is used as an index to a precomputed table. The grey-level transformation is then implemented as a simple look-up of this table. Understandably, this techniques is suitable only for images of integer values or when rounding to integers is a reasonable approximation. [3] In the course library there is a routine `compose` that performs a composition of an input image `inpic` and a precomputed table `lookuptable`:

```
outpic = compose(lookuptable, inpic)
```

Since techniques based on look-up tables will show to be useful for later exercises, that involve stretching of grey-levels and histogram equalization, we will here illustrate how it be used for reverse traversion. For the grey-level interval $[0, 255]$ it is possible to create a look-up table in MATLAB using

```
negtransf = (255 : -1 : 0)';
```

Then the grey-level transformation is computed using

```
neg3 = compose(negtransf, Canoe + 1);
```

Look at this image on screen and compare it to the earlier negation operation by creating a difference image

```
diff = neg3 - neg2;
```

and compute the min- and max-values of this image and/or histogram.

Question 9: **Why was value 1 added to the image before the look-up?**

---

[3] Naturally, it might be possible to combine table look-ups using linear interpolation and thus improve the accuracy.

3.3 Manipulation of colour tables

As we saw in exercise 1, every colour table also works as a grey-level transformation. Thus it is possible to create a contrast reversed image simply by manipulating the colour table. After you have shown the canoe image with

```
image(Canoe + 1)
```

you may create an affine function that decreases from the value 1.0 to 0.0 in 256 steps by writing

```
negcolormapcol = linspace(1, 0, 256)';
```

and use this to set up a colour table with

```
colormap([negcolormapcol negcolormapcol negcolormapcol])
```

Using a more compact notation, you may let `showgrey` perform the same kind of operations by writing

```
showgrey(Canoe, linspace(1, 0, 256), 0, 255)
```

The two last arguments tell `showgrey` that the image has values in the interval $[0, 255]$, which means that we no longer have to add a value 1. The obvious drawback of just manipulating the colour table is that the result is not available for further processing or analysis.

4 Stretching of grey-levels

Load image `nallo256float` [4] and dislay it using `showgrey`. This image has been scanned with high dynamics so that both grey-level variations in the light part of the image as well as in the lower dark areas have been dissolved. Unlike the integer value image `nallo256` it has been stored as a floating-point image with a finer quantization of grey-levels.

As you can see, the lower part of the image has poor contrast. Compute a histogram of the image and use this as a hint to create a set of different look-up tables, with which to transform the grey-level values. For this image, you may for example begin by stretching each third of the grey-level scale so that the whole dynamics can be viewed on screen. Preferably, use the techniques we covered in previous exercise, based on grey-level specifications to `showgrey` and/or look-up tables. Visualize with `subplot` the following properties on screen:

- the original image
- the histogram of the original image
- the transformation function
- the transformed image
- the histogram of the transformed image.

Question 10: **What would be the best transformation function and why?**

---

[4] The name Nallo refers to an place in the Kebnekaise area where the image was taken.

5 Logarithmic compression

Apply the logarithmic transformation function

$$T(z) = \log(\alpha + z)$$

to `nallo256float`.

Question 11: **What are the effects of the operation? Explain in which cases it is a good technique. Why do you need the parameter α and how does it affect the result? What is a suitable value for α?**

---

6 Histogram equalization

Combine the command `hist` and the command `cumsum` to generate a tabulated grey-level transformation `eqtransf` that equalizes the histogram of the image `nallo` based on some reasonable discrete transformation equivalent to

$$T(z) = \int_{\zeta=0}^{z} p(\zeta)\, d\zeta$$

where $p(\zeta)$ is the normalized frequency function (normalized histogram) of the image. [5] Collect all commands you have used and write a MATLAB-function

```
function pixels = histeq(image, nacc, verbose)
```

The function must accept any image as input together with the number of accumulators `nacc` used for histogramming. Naturally, the output is the histogram equalized image. If the argument `verbose` has a positive value the histograms before and after equalization shall be displayed, as text as well as graphically. If you need advice on how to write functions in MATLAB, you may study the source code of for example the function `showgrey` by typing "`type showgrey`".
Display the generated transformation function, create the histogram equalized image, and compute the histograms for this image based on 16, 64 and 256 accumulators. (Remember: store results either as print outs or scripts, such that they can easily be presented.)

Question 12: **How do you expect the resulting image and histogram will look like? What did you actually get and why is it different? How does the histogram equalized image and its histogram depend on the number of accumulators?**

---

[5] Normalization of histograms is preferably done by dividing it with the MATLAB-expression `prod(size(nallo))`, that returns the product of the elements of the vector `size(nallo)`, i.e. the number of image elements. Remember to normalize after the call to `cumsum`. Why?

Question 13: How are the results compared to the transformation functions used in previous sections? How will the results be for different kinds of images? What happens if you apply `histeq` to the similar image `nallo256what`? (If you do not get similar results, you have probably done something wrong.)

Question 14: What happens if you histogram equalize the image `phonecalc256`? Think of what results to expect, before you perform the actual operation. What kinds of images are suitable for histogram equalization?

Question 15: Would it be possible to improve the results by exploiting the fact that `nallo256float` contains image data stored as floating-point values, unlike `nallo256` that contains integers?
