

Lab 4

Basic Image Processing Algorithms
Fall 2016

Lab 4: histogram-operations, median filter

- implementing
 - histogram viewer
 - histogram stretcher
 - histogram equalizer
 - median filter
- deadline (if you do not finish until the end of the lab): **23:59, 12/10/2016**;
please send it to: bipa2016fall@gmail.com
with the subject: **LAB4**

The Histogram of an Image

- **Histogram:**

$h(k)$ = the number of pixels on the image with value k .



Original Image*

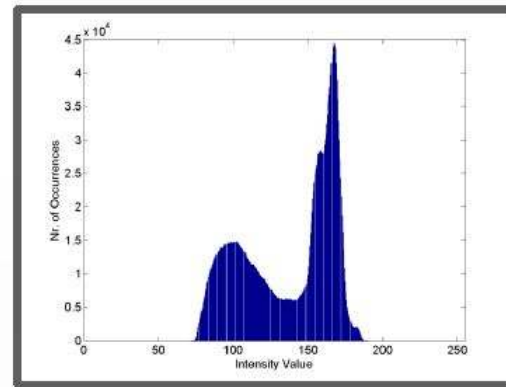


Image Histogram

- The histogram normalized with the total number of pixels gives us the ***probability density function*** of the intensity values.

* Modified version of Riverscape with Ferry by Salomon van Ruysdael (1639)

Exercise 1

Create a function that:

- realizes a **histogram viewer**,
- parameters of this function should be as follows:
 - *input*: image matrix,
 - *no output*,
- it should work with grayscale images (check if this holds),
- it should raise a new figure (please use **figure** command) showing the histogram of the input image, use the built-in **bar** command to plot the column-diagram.

Histogram Transformations

● Histogram Stretching:

- Based on the histogram we can see that the image does not use the whole range of possible intensities:
 - Minimum intensity level: 72
 - Maximum intensity level: 190
- With the following transformation we can stretch the intensity values so they use the whole available range:

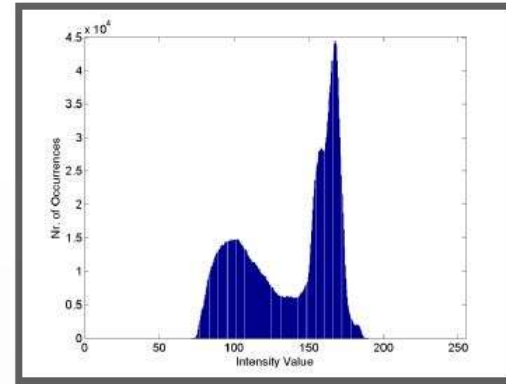


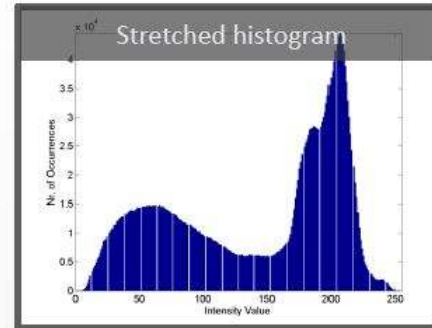
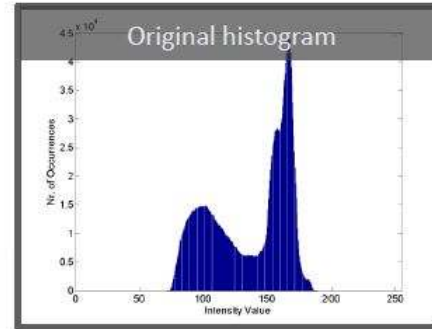
Image Histogram

$$y(n_1, n_2) = \frac{255}{x_{\max} - x_{\min}} \cdot (x(n_1, n_2) - x_{\min})$$

$$x_{\max} = \max_{n_1, n_2} (x(n_1, n_2)) \quad x_{\min} = \min_{n_1, n_2} (x(n_1, n_2))$$

Histogram Transformations

● Histogram Stretching:



Exercise 2

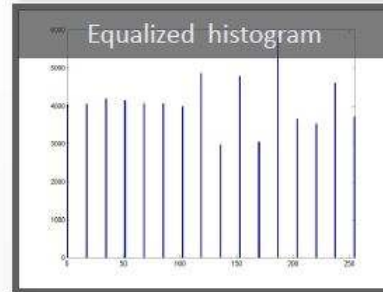
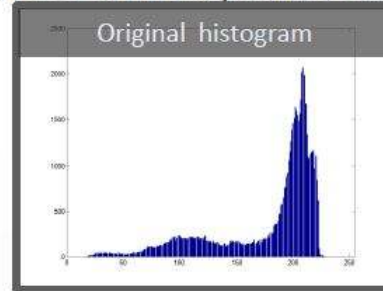
Create a function that:

- realizes a **histogram stretcher**,
- parameters of this function should be as follows:
 - *input*: image matrix,
 - *output*: the matrix of the stretched image,
- it should work with grayscale images (check if this holds).

Histogram Transformations

● Histogram Equalization:

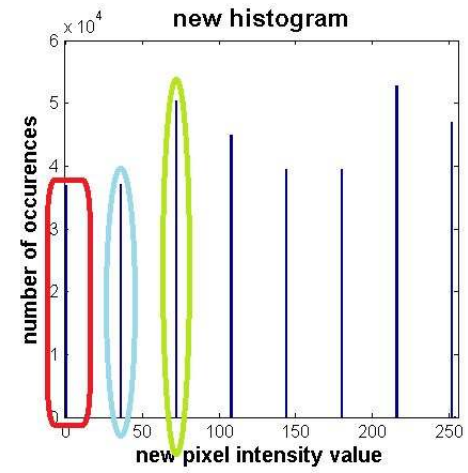
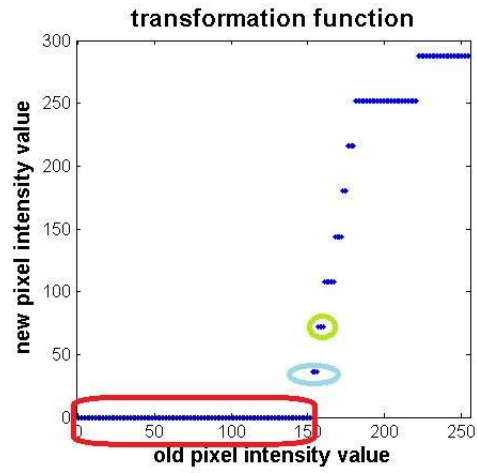
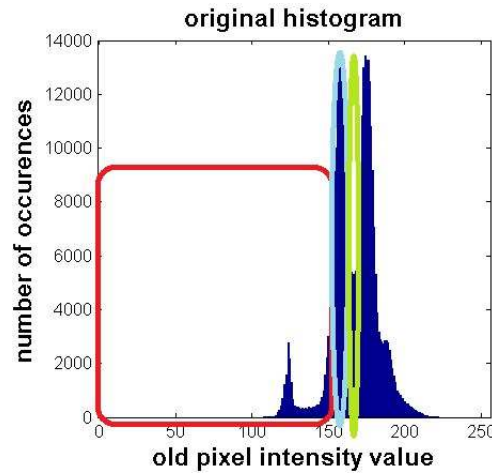
- The goal is to increase the contrast, by distributing the occurrences of the intensity values evenly through the entire dynamic range.



Exercise 3

With the histogram equalizer

- you would like to get an equalized histogram, where the number of the columns is a parameter of the equalization (this is called here *num_of_bins*), and
- the **desired average height** of the new columns can be computed from the **total number of pixels** *image_size* **over number of columns** called *num_of_bins*;
- for the sake of simplicity, you have to implement a **transformation function** (it should be rather called *transformation_vector*), which will **translate the old pixel intensity values to the new intensity values** (like a look-up table; on **axis X** you can see the old pixel intensity values, on **axis Y** you can see the new pixel intensity values).



Exercise 3 - continued

Step-by-step pseudo-code:

```
column_threshold = image_size / num_of_bins % this will define the desired
height of every column on the new histogram
step_size = 256 / (num_of_bins-1) % the stairway-like
transformation_vector will be increased with this, every time you overflow
the column_threshold
sum_var = 0; % cumulate the actually filled column of the new histogram
col_level = 0; % the actual level of the transformation_vector
for idx=1:256
    sum_var += original_histogram(idx);
    while (sum_var >= column_threshold)
        col_level = floor(col_level + step_size);
        sum_var -= column_threshold;
    end
    transformation_vector(idx) = col_level;
end
new_img = transformation_vector(original_img + 1);
```

Exercise 3 - continued

Create a function that:

- realizes a **histogram equalizer**,
- parameters of this function should be as follows:
 - *input1*: image matrix,
 - *input2*: number of bins,
 - *output*: the matrix of the equalized image,
- it should work with grayscale images (check if this holds).

Further comments:

- as you see, the mapping vector contains 9 levels on my image, although I have 8 bins in the new histogram. The reason is: there is no pixel on the original image with intensity value bigger than 221 - so the last step of the mapping vector will not be used.
- at lower bin counts (eg 8 or 16), you will not get back the same new histogram, what the built-in *histeq* would produce in MATLAB, but this time do not care with it.

Spatial Filtering

⦿ Median Filter:

- Replaces each pixel with the *median value* of its analyzed neighborhood. (Median value: the center element of sorted values)
- Non-Linear
- Very effective against impulse noise:



- Not so effective against Gaussian noise.

Exercise 4

Create a function that:

- realizes a **median filter**,
- parameters of this function should be as follows:
 - *input1*: image matrix,
 - *input2*: neighborhood size (eg 3 in case of 3x3 neighborhood),
 - *output*: the matrix of the filtered image,
- it should work with grayscale images (check if this holds),
- rough workaround:
 - go through every pixel of the image (use `padarray` to create virtual cells),
 - take the *input2* x *input2* sized neighborhood of it,
 - vectorize this neighborhood (built-in `reshape`),
 - sort the values (built-in `sort`),
 - take the middle intensity as the new value of the processed pixel-location.

Exercise 5

Create a script that:

- reads in this image: AndreKertesz_Paris_ManOnBicycle_modified.jpg,
- converts it to grayscale,
- calls your histogram viewer on the input image,
- calls you histogram stretcher on the input image,
- calls your histogram viewer on the output of your histogram stretcher,
- calls your histogram equalizer on the input image with 16 bins,
- calls your histogram viewer on the output of your histogram equalizer,
- shows the input image, the stretched and the equalized images as well, see images below:

original input



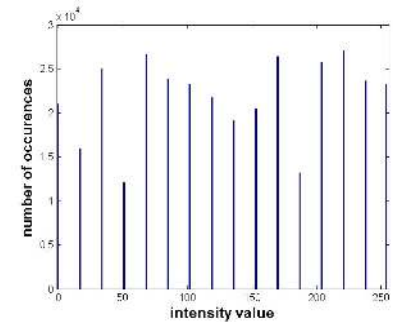
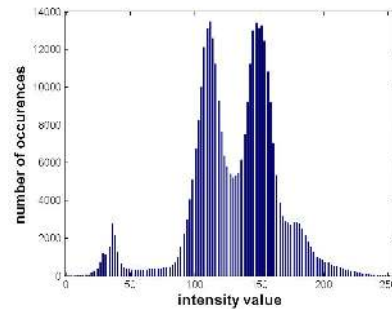
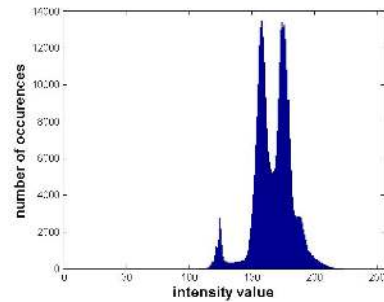
stretched image



equalized image



your histograms should look similarly (order as above):



Exercise 5 - continued

Continue your script:

- load another input image (AndreKertesz_Paris_ManOnBicycle_part.jpg),
- create the noisy-variant of your input image (use the built-in `imnoise` with 'salt & pepper'-type noise),
- call your median filter on the noisy image,
- show the input image, the noisy and the filtered versions as well, see below:

original input



image + noise



median filtered image

