RRY025-- Image Processing

Exercises 1 – Lecture 1 – Introduction and Image Enhancement I

EX 1. Histograms and Contrast Stretching

Inside MATLAB type **imadjdemo**. Start by loading the *pout* image from the dropdown menu. The left side shows the original image and the right the enhanced image. The histograms below each image show the fraction of pixels of each grey level, while the top right panel shows the ’transfer function’ indicating how a grey level in the input image is mapped to a grey level in the output.

The shape of the transfer function can be changed using the the +/- Brightness and +/- Contrast buttons (experiment with them). The transfer function can also be adjusted manually by clicking on the top and bottom yellow circles.

Experiment and try to develop an understanding of how changing the transformation function changes the image and its histogram.

Questions to consider:

Can you understand why contrast and brightness affect the transfer function and the histogram the way they do?

What kind of transform function produces the best image? What does the image histogram look like in this case?

Apply the ’Histogram Equalisation’ function by clicking on the ’Operations’ box, does this produce a nice-looking image?

Try other images. Which images are improved most and why? Which image is improved least and why?

SOME ANSWERS: Often (but not always), the histogram that produces “best” image fills the entire x-axis (i.e., the zeros are trimmed out) and it is relatively flat. However, there are clear cases, such as ‘coin’, that do not follow this.

Why? Even though histogram equalization is best as a “rule-of-thumb”, the properties of the image ultimately dictate what kind of transfer is best. In particular, when the initial histogram has a lot of structure (e.g., is bimodal), it can be that the features of interest appear only in one part of the histogram. In this case, global equalization does not work. Also, “the best” can be subjective.

EX 2. Implementing histogram equalisation algorithmically

Write a matlab script to implement histogram equalization **manually** (i.e., not using the built-in Matlab function **histeq**). You can follow the roadmap below.

- Load the ’pout’ image: **pout = imread(’pout.tif’)**. Display with **imshow(pout,[])**. This image has 8 bits/pixel or 256 levels (gray level 0 to 255).

- Create a 256-element vector containing the input image histogram with command **imhist**. Normalise this histogram by the number of pixels in the image and from this form the transform function **P_x** (see lecture notes for how the transform function is formed). Some useful matlab commands are **numel** and **cumsum**.

- You now have the transfer function in hand. How do you apply it to create an equalized image?

Solution: the transformed image can be made via **pout_equalized = P_x(pout+1)**. If you did not come up with this line yourself, try to make sure you understand what the command does.

- Inspect the histogram of the **pout_equalized**. Why is the histogram not exactly flat, even though you just equalized it?

Some common failure modes: Note the data types (**uint8, double)**. Some functions require integers and some doubles to work properly.

Note the difference between the vector indices, which must be non-zero in Matlab (1-256), and the gray levels that have slightly different indices (0 - 255).

ANSWER: An example given as an .mlx file.

EX 3. Implementing histogram specification algorithmically

Write a Matlab script to implement histogram specification manually for the 'coin' image.

Context: During EX 1, one could notice that the coin image does not look very good even when its histogram is equalized. Why is this the case? The solution for better image quality is to isolate the grayscale range of the coin and equalize only that range. Technically, this equals to histogram specification, i.e., modifying the image to have a specified histogram.

To that goal, prepare a script that tries to take the grayscale range of the coin and equalize only that range. In more detail, the sript should 1) take the pixels in the range [0, upperlimit_in] and distribute them approximately uniformly over the range [0, upperlimit_out]; and 2) take the remaining pixels in the range [upperlimit_in, 255] and distribute them approximately uniformly over the range [upperlimit_out, 255]. You can follow the roadmap below.

- It can be helpful to first sketch the target histogram (and CDF), to gain understanding of how the gray levels are going to be mapped.

- Use the pre-saved image of the coin: **load('quarter.mat')**. The image is saved in a variable called **quarter**.

- Determine a suitable value for the **upperlimit_in** by inspecting the histogram of the coin image. What could be a suitable value for the **upperlimit_out**?

- Compute the original pdf and cdf using the original image. Compute the target pdf and cdf using the determined **upperlimit_in** and **upperlimit_out** values.

- Calculate the transfer function by comparing the cdfs as described in the lecture notes. Useful commands might be **abs, min, find** and **for...end** loops.

- With the transfer function in hand, apply it to the original image. Display and inspect the resulting image and its histogram. Is the result better compared to histogram equalizing the entire grayscale range? Did you choose the upperlimit values well?

ANSWER: An example given as an .mlx file.