COMPUTER VISION 2023 - LAB 2

P. Zanuttigh, G. Rizzoli, D. Shenaj
Topics: Image Equalization, Histograms, Image Filtering

Part 1: Histogram Equalization

Write a program that:

1. Loads an image (e.g., one of the provided images like *"barbecue.jpg"* or *"countryside.jpg", or any custom image you would like to use*)

2. Prints the histograms of the image. You must compute 3 histograms, one for each channel (i.e., R, G and B) with 256 bins and [0, 255] as range. *Notice that you need to use the* `calcHist()` *function separately on the 3 channels.* You can use the provided function (in the "show_histogram_function.cpp" file) to visualize the data.

3. Equalizes separately the R, G and B channels by using `cv::equalizeHist()`.

4. Shows the equalized image and the histogram of its channels.

5. Notice the artifacts produced by this approach. To obtain a better equalization than the one of point 4, convert the image to a different color space, e.g. Lab (use `cv::cvtColor()` with `COLOR_BGR2Lab` as color space conversion code), and equalize only the luminance (L) channel.

Original Image *RGB equalization* *Luminance Equalization*

Figure 1: Example of the results of the first part

Part 2: Image Filtering

Generate a denoised version of the image. You should try different filters and parameter values.

- Write a program that performs the filtering and shows the result.
- Table 1 specifies the requested filters to test and the parameters to be set for each filter.
- You can simply pass the filter parameters from the command line or (advanced solution, optional*) *use some trackbars as in the example in the figure.*
- (Optional, for skilled C++ programmers*) *If you would like to understand how classes inheritance works in C++ try to create a base filter class using the provided source code and extend it creating subclasses for the various filters. See the slides on inheritance in the provided material.*

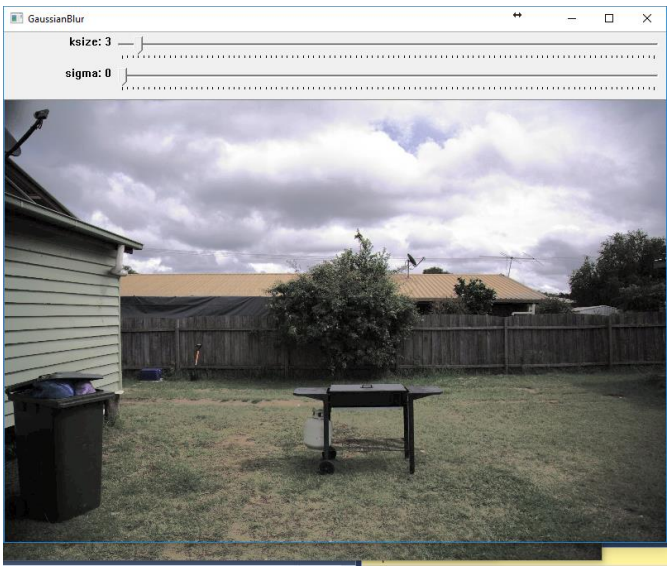

Figure 2: Example of the window with (optional) trackbars

`cv::medianBlur()`	• kernel_size
`cv::GaussianBlur()`	• kernel_size (keep it square) • σ : assume $\sigma_x = \sigma_y$
`cv::bilateralFilter()`	• kernel_size (you can use a fixed value or use the $6\sigma_s$ rule) • sigma_range σ_r • sigma_space σ_s

Table 1: Filters to be implemented and their parameters

Suggestions for who would like to use trackbars:

- In order to generate the trackbars you can use the `cv::createTrackbar()` function.
- In order to pass the image and the parameters to the callback of the trackbar you can create a class containing the image and the filter parameters.

**Note: optional steps are not required, you will get the full mark also without them*