Computer vision - Homework 1 on Lab. 1

Giulio Conca - 2028844

May 8, 2021

1 Introduction

Laboratory 1 was about the computation of the histogram of an image, its equalization and the filtering of the original image, using different types of filters. To this aim, 3 different source files were produced: lab2.cpp, with the main function, filter.h and filter.cpp with the declaration and definition of the class Filter, and relative subclasses.

2 Histogram equalization

The computation and equalization of the histogram has been carried out both for the RGB color space and the LAB color space. For the first case, the function *computeAndEqualizeRGB* was used. In particular, the image was split in the 3 RGB channels, and the *calcHist* function was used for each of them. In order to visualize the images of the histograms in the three color spaces, the function *showHistogram* was used. Subsequently, the equalization was carried out: each channel was equalized indipendently of the others, using a for cycle and the function *equalizeHist*. After that, the image was reconstructed by merging the equalized channels.

In order to obtain a better equalization, the same procedure has been carried out for the LAB color space. To this aim, the function *computeAndEqualizeLAB* was used, which is very similar to the one that was used for the RGB color space. In particular, the original image was converted to the LAB color space, then it was divided in the 3 channels as before. This time, only the plane corresponding to the luminance was equalized, while the other two were not modified. The final image was reconstructed by using, once again, the *merge* function. After that, the image was reconverted to the RGB color space, and the computation of the histogram was done as before. The final result looks better with respect to the RGB equalization.

3 Image filtering

For this part, the class Filter was used, together with the subclasses for each specific type of filter, such as the median filter, the gaussian filter and the bilateral one. For such subclasses, the function *doFilter* was overridden (it was defined as virtual in the base Filter class), in order to use the correct function for each case. In particular, the specific functions to perform the filtering operation

are *medianBlur*, *GaussianBlur* and *bilateralFilter*, provided by OpenCV, and placed in the *doFilter* method of the corresponding subclass.

In order to choose the filtering parameters, some trackbars were defined, using the function *createTrackbars*. Every time the slider of a trackbar was moved, the function *onChange* was called.
The signature of this function must be (int, void*). For this reason, one possible way to pass some parameters to this function is to create an object with all the parameters and then to pass the object to the function after a cast to void*. To this aim, a struct called *TrackbarParameters* was created. The struct contains 3 different constructors (one for each type of filter, because different parameters must be used for each case), all the possible parameters (such as kernel size, different standard deviations and the original image), and an enum class, used to identify the type of filtering that must be performed.

In particular, for each type of filter, a new object of type TrackbarParameters was created, calling the corresponding constructor (for example, for the median filter case, only the image to be filtered, the kernel size and the type of the filter were initialized, because the other parameters are not needed for this case). The object was then passed to the *onChange* function (after a cast to void*), where all the parameters were recovered using a cast to TrackbarParameters*. Once the parameters were recovered, an object of the corresponding filter class was created (passing to the constructor those parameters) and the method *doFilter* was called for each case.

Regarding the trackbars, for the gaussian case two of them were created, one for the kernel size and the other for the standard deviation. For the median case only one trackbar was introduced, to set the kernel size. Finally, for the bilateral case, three trackbars were inserted, one for the kernel size and two for the standard deviations. Since it was not possible to set float values through the trackbars, for the float parameters (for example the standard deviation) a trick was used: the value set by the trackbar is 10 times the parameter. So the value that is finally passed to the *doFilter* function is the one selected by the trackbar divided by 10 and casted to float.

4 Results

These are some results. They were obtained starting from the original image shown in figure 1. From the Figure 2, it's clear that the luminance equalization provides a better result (colors are more similar to the original image), with respect to the rgb equalization.

The other images are about the filtering part. Figure 3 shows the result of a gaussian filter applied to the original image, while figure 4 and figure 5 show the output of a median and bilateral filter, respectively. On the top of the images, the trackbars are visible.
All these images (with other examples) are provided inside the final zip file uploaded on moodle, inside the folder named "Results".

Figure 1: Original image with histogram

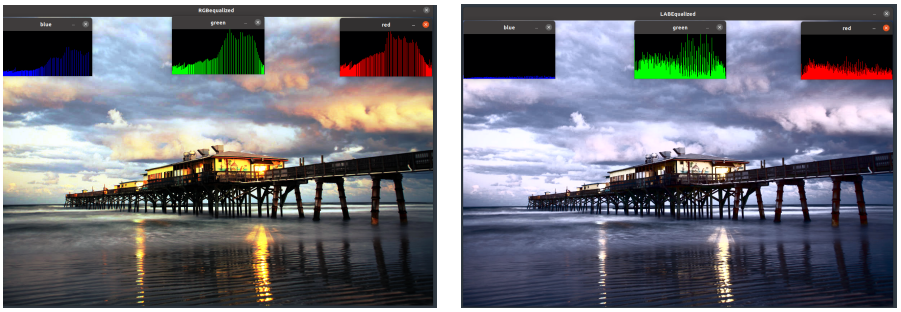

Figure 2: RGB (left) and LAB (right) equalization

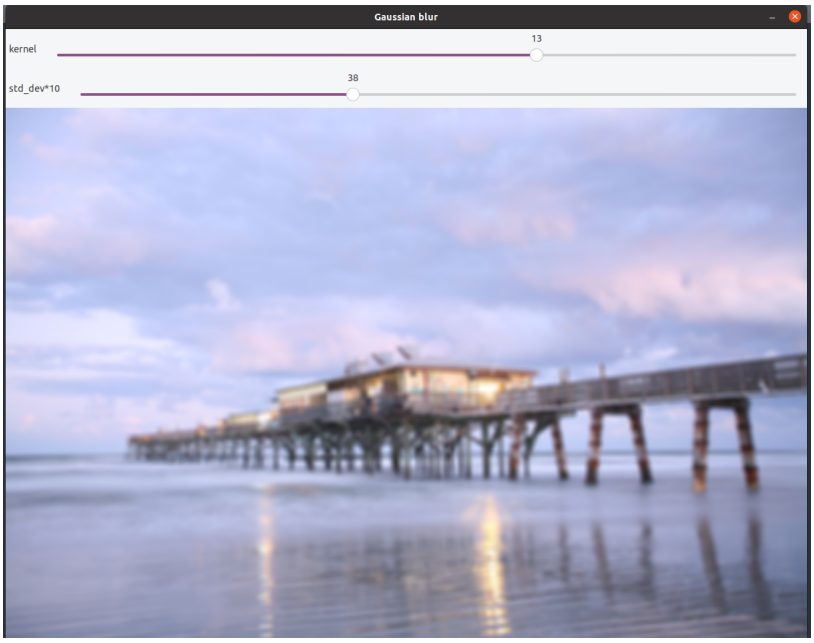

Figure 3: Original image with a gaussian filter

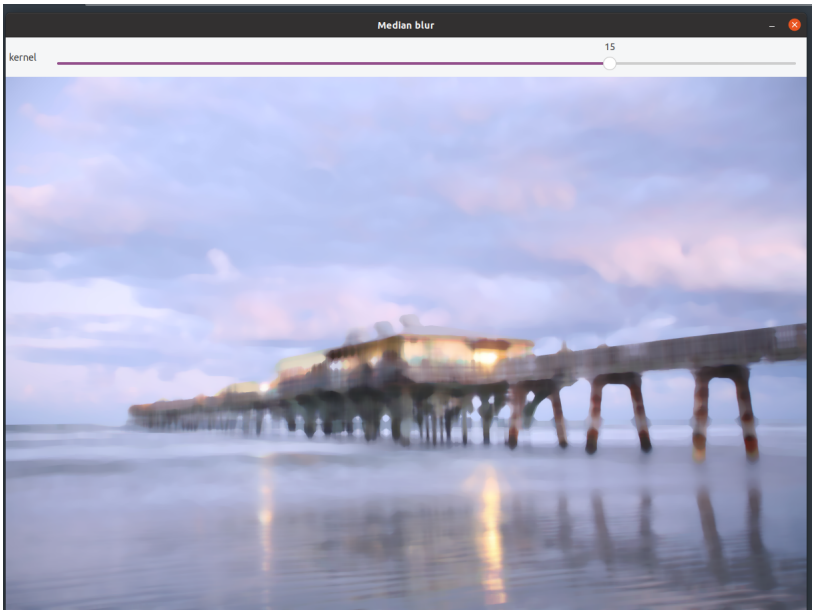

Figure 4: Original image with a median filter

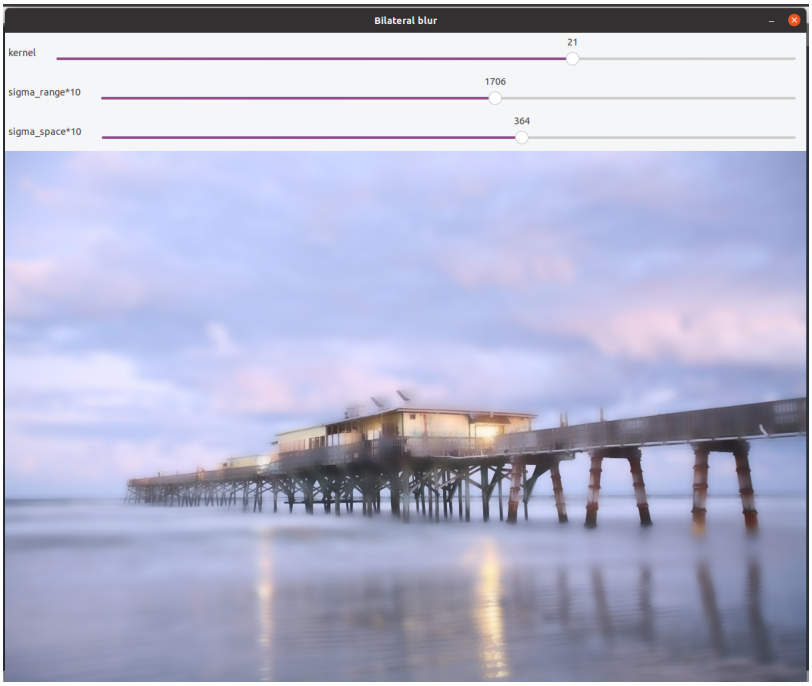

Figure 5: Original image with a bilateral filter