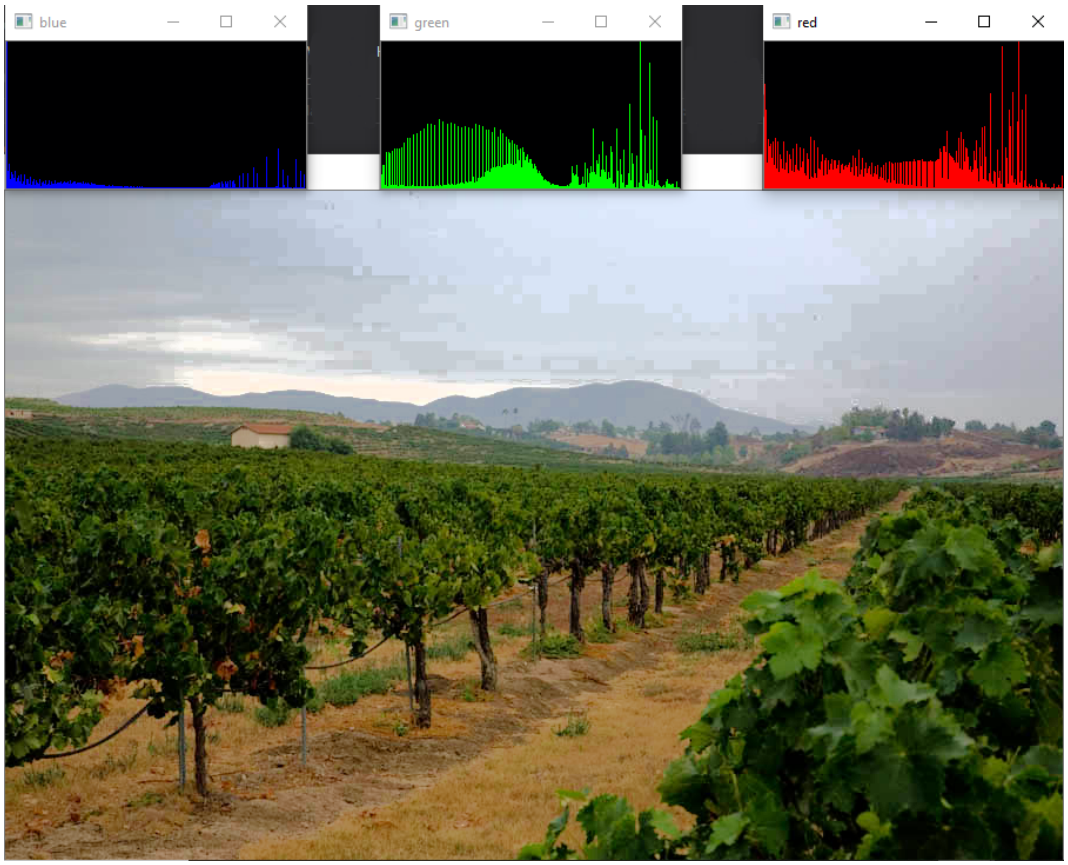
Histogram equalization and image filtering

Mautone Alberto

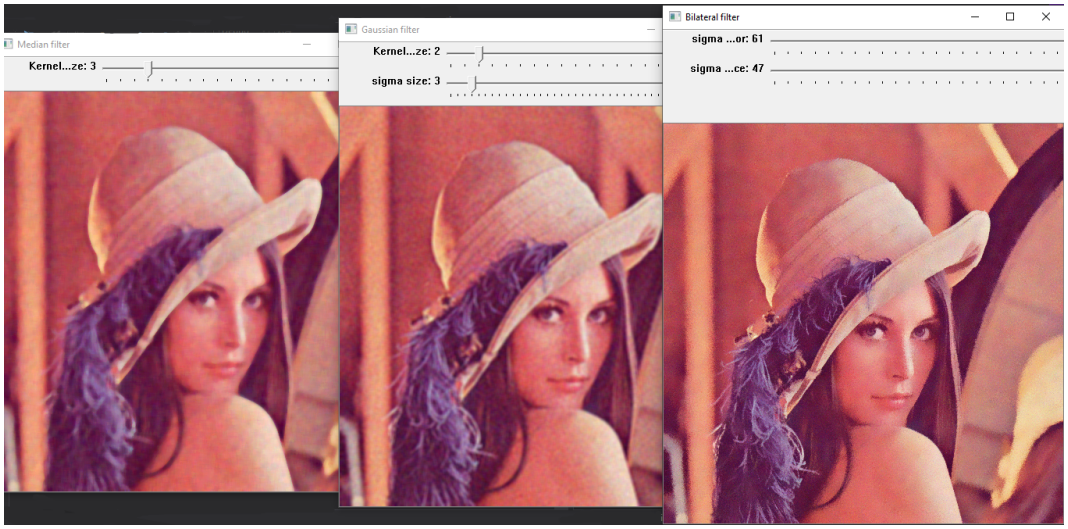
The code starts with the definitions of 2 functions that are going to be helpful for the first part of the homework: **histograms equalization**. The first one is a function that converts the original pic from **BGR** space to the **HSV** space, and goes on by equalizing only **1 channel** as requested, by using the *equalizeHist(...)* function. Of course before doing that, I need to **separate** each channel of the image, easly done by the *split(...)* function. Finally I need to **merge** everything back, so I've used the *merge(...)* function. The second function is used to **show histograms**.
Going on, I defined some **global variables** useful for the second part of the lab, **image filtering**, defining then the **struct** containing all the useful variables for filtering. Follows a list of 3 functions used as soon as the **slider** of the various trackbars **moves** to the right or to the left: every function is linked to a **single trackbar** which refers only to a single filtering method. Trackbars are put **upon** every image, so since I have **3 different filtering methods** I have 3 images and so 3 different trackbars. Each one of this 3 **callback-functions** accepts an int value and a void* pointer. The int value is referred to the slider, while the void pointer refers to the struct in which I put all the necessary parameters and it needs to be **converted back** to the struct type. So for instance, median filtering only requires **one parameters** (and of course the source and destination Mat), so I just pass as parameters of the struct, the 2 Mat and only one parameters (that is the **kernel size**).
Inside the *main()* function basically I just apply these functions: after reading and showing the input image and its histograms I first **equalize** it in the BGR space by separating, as before, each channel thanks to the *split(...)* function, defining some useful variables, and then calculating **histograms** for each channel with the *calcHist(...)* function and showing them all. Finally I equalize them all thank to the *equalizeHist(...)* function and put them all together with the *merge(...)* function. I do the same in **HSV** space equalizing only **one channel** (**testing** the output image for each one of the 3 channels) as defined ad the beginning of the code. Every image and related histograms are showed one per time, by closing the prevous window on user input (I've used *waitKey(0)* followed by *destroyAllWindows()*). My final comment is that equalizing in the **V channel** (also compared to the S and H) gives a **better result** than the BGR one: colors are more vivid and clear.

BGR equalized image with related histograms.



HSV equalized image (only on the V channel) with related histograms.

In the second part of the code I basically apply the 3 **callback-functions** defined before for image filtering: what I do is creating a struct for each one of the 3 filters, and create a trackbar with the *createTrackbar(...)* function and **calling then the specific callback function** for that filter. So for instance, after that I've created the struct for the median filtering containing only 1 parameter (**kernel size**) and the 2 Mat (source image and destination image), first I use *createTrackbar(...)* that of course cointains, among the other parameters, a **referring to the struct and a referring to the callback-function for median filtering** that I called *on trackbar mf* (where mf stands for median filtering). Finally I just **show** all of them in a window with the related trackbar.



On the left: image filtered with median filter.
On the center: image filtered with gaussian filter.
On the right (better result): image filtered with bilateral filter.