Homework reports

Course: Computer Vision 2020

Francesco Pham (1234004)

Lab 3 - Image Equalization, Histograms, Filters, Morphological Operators

My Lab 3 code is inside source file homework_3.cpp, the filters classes are found in *filters.h/filters.cpp* and some utility functions for the histogram visualization inside *utils.cpp*. During program execution you will be prompted to type in the input image path.

Part 1: Histogram Equalization

In this part of the homework I successfully saw the result of the histogram equalization in both RGB space and HSV space. The OpenCV instruction that I used for the equalization is `equalizeHist`, to show the histogram I added a function named `generate_show_histograms` which separates the three channels of the image and generates the histogram images using the function provided by the prof.

After the equalization I noticed how the histogram is more evenly distributed across the intensity values (figure 1), consequently underexposed images are brightened up and overexposed ones are darkened in both RGB and HSV space (figure 2).

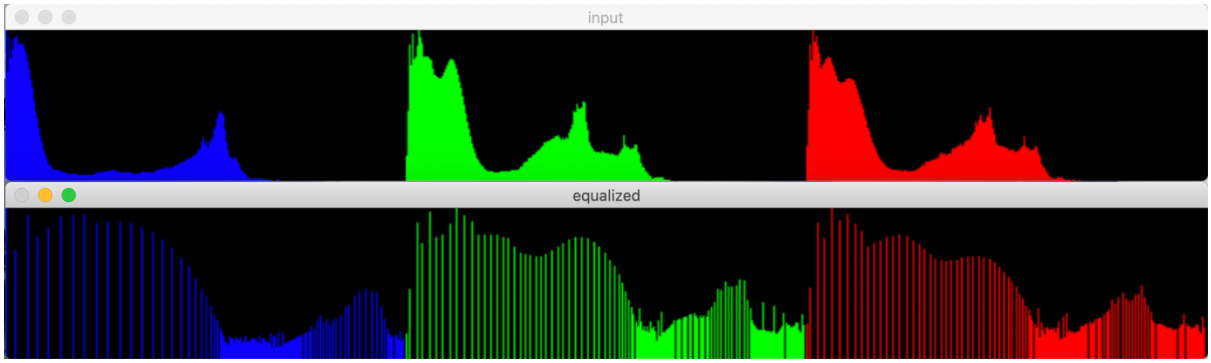


Figure 1

Figure 2 - Equalization in RGB space

In the equalization in HSV space the best results are obtained by equalizing the value channel whereas, when only hue or saturation channels are equalized, underexposed or overexposed images are not corrected since the intensity values remains the same so brightness remains unchanged.

Figure 3 - Equalized value channel

Figure 4 - Equalized saturation channel

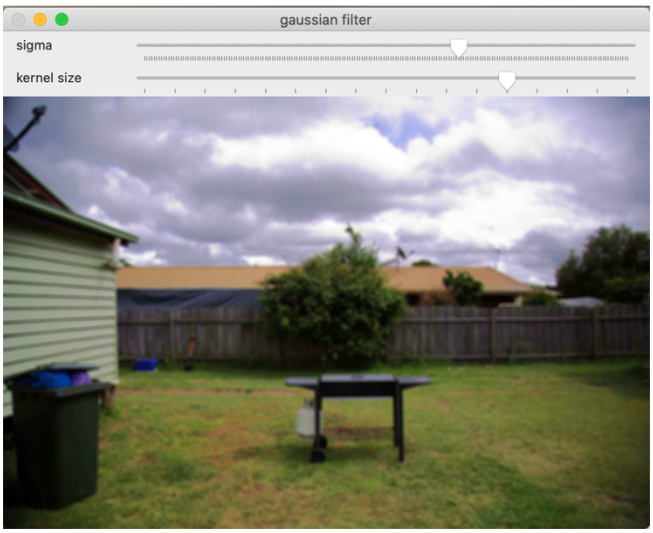
Figure 5 - Equalized hue channel

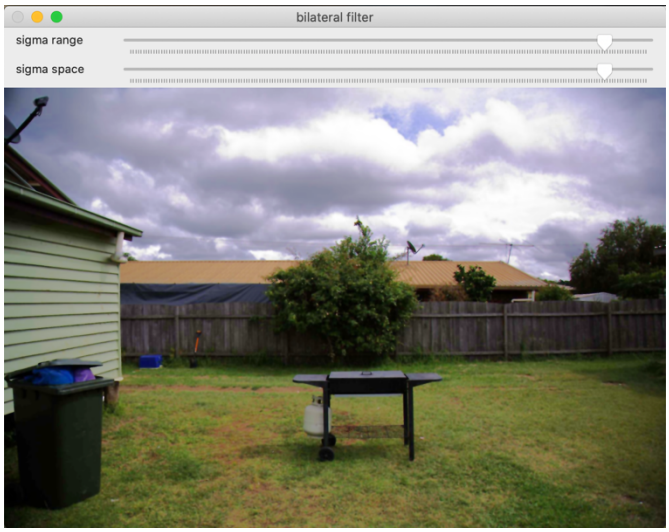
By comparing the results of the equalization in RGB space with respect to the results in HSV space, I noticed how the latter has a better color fidelity and closer to natural colors, whereas the RGB equalized one is more greyish.

Part 2: Image Filtering

In this part of the homework I successfully obtained the filtering of the equalized input image. Three filtering methods are applied on the image to see the differences between different types of filtering with the different parameters configurable with a set of trackbars. The filters parameters passed to the trackbar callbacks are stored inside a structure `FilterParams` which stores the object of a subclass of `Filter` such as `MedianFilter`, `GaussianFilter` or `BilateralFilter` together with their parameters. Each subclass of the class `Filter` implements the method `doFilter` which applies the corresponding filtering such as `medianBlur`, `GaussianBlur` and `bilateralFilter`.

Regarding the result of the filtering with different methods I noticed that the Gaussian filtering just blurs the whole image, in Median blur some edges are preserved whereas in Bilateral filtering unwanted noise are reduced very well while keeping edges very sharp. In Median and Gaussian filtering the increase in blur is very noticeable as we increase the kernel size. In Bilateral filtering as we increase sigma color range the neighboring pixels are more mixed together.





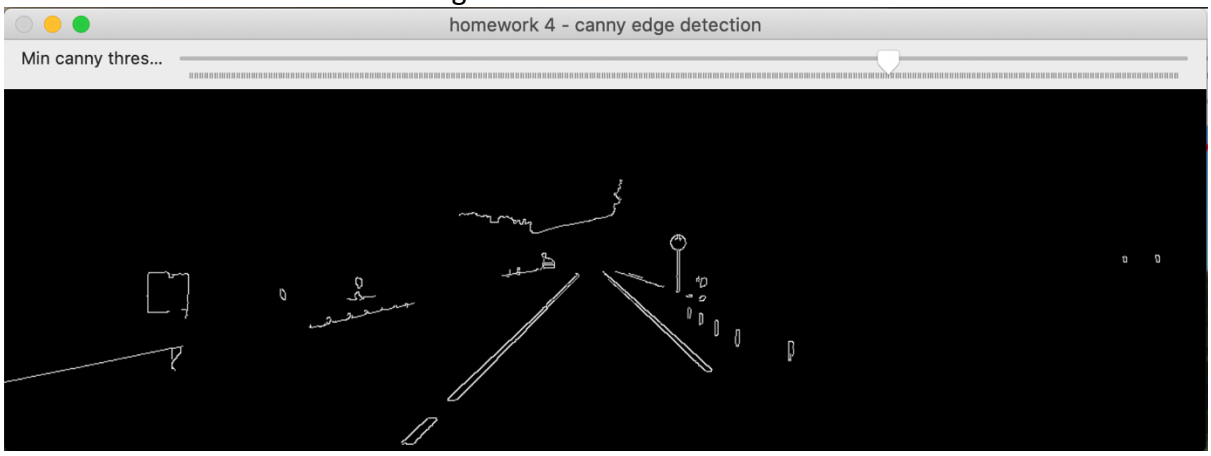
Lab 4 - Hough transform and Edge detection

My lab 4 code is inside source file homework_4.cpp.
During program execution you will be prompted to type in the input image path.

Canny edge detection

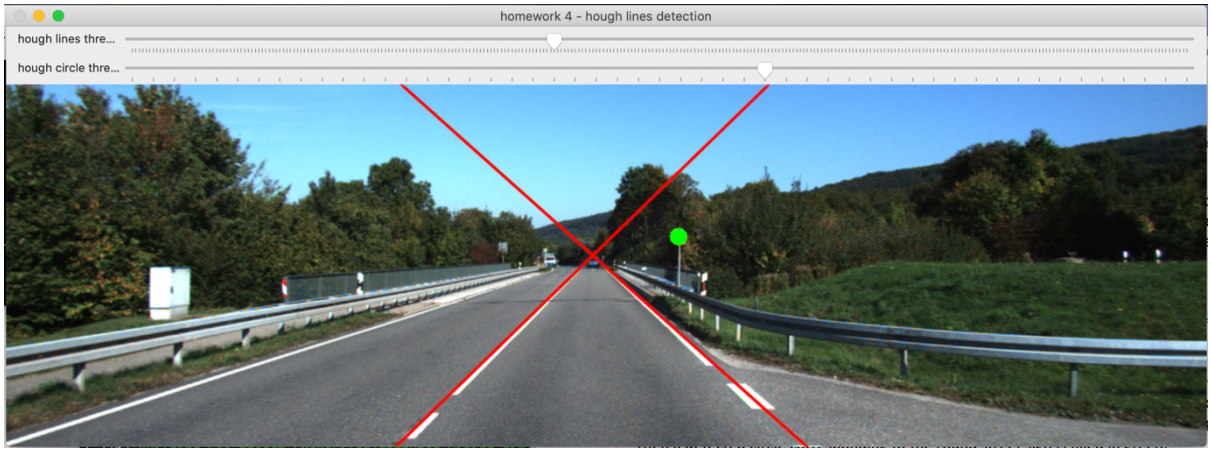
In this lab an input image showing a street is segmented to find the street lanes and a round street sign. First of all, a canny edge detection is performed inside the function called `canny_threshold` which is called as a callback of a trackbar to find the best low threshold value in the "hysteresis thresholding" phase. The high threshold is set 3 times higher than the lower threshold as suggested in the OpenCV documentation and the kernel size is set to 3. The parameters are contained in the struct `CannyParams` and passed to `canny_threshold` which in turn runs the OpenCV function `Canny` on the given parameters.

As we can see in the figure below the edges are correctly detected and in particular the street lines and the round street sign.



Hough transform

The next phase is to run the Hough transform to detect lines and circles. This is performed inside the function `hough_transform` which runs `HoughLines` and `HoughCircles` on the edges detected by Canny. Then the detected lines and circles are drawn on the original image and shown in a window using the OpenCV functions `line` and `circle`. The `hough_transform` function is given as a callback to two trackbars to select the best threshold of the accumulator in the Hough space for both lines detection and circle detection. The parameters are contained in the struct `HoughParams` and passed to the `hough_transform` function.
As we can see in the figure below the street lines and the round street sign are correctly detected and with the correct parameters all the other lines and circles are filtered out.



Final result

Finally, if at least two lines and a circle are detected, the final result is shown in `show_result`. To find the space in between the two street lines I calculated the intersection point of the two lines and filled a red triangle between the two lines and below the intersection point. Then a detected circle corresponding to the round street sign is filled in green.