Audio and Video Coding (2018/19)

Pedro Santos, *Universidade de Aveiro, 76532* Silvério Pereira, *Universidade de Aveiro,*

***Index Terms*—OpenCv,c++,assigment$_3$**

I. Introduction

THIS report has a description and explications about the experiences done in the lesson number three.

II. Exercises

A. Exercise 1

the grayscale and the black and white versions of the image can be seen in Figures To do this we make use of 2 functions:

Fig. 1. Grayscale image

Fig. 2. Black and White image, adaptive threshold

- `cv::cvtColor()` converts our image (a `cv::Mat`) into a different colorspace, even one with a different number of channels, like grayscale images, as such its used to convert our video to grayscale. Specifically, this by passing it a code, in this case `CV_BGR2GRAY`, the full color conversion codes can be seen here.
- `cv::adaptiveThreshold()`, unlike `cv::adaptiveThreshold()`, which we take a better look at in II-C, it thresholds the image (converts a pixel to black or white according to some function), according to local features surrounding the pixel, instead of just an arbitrary value. It provides 2 ways of doing this:
  - `ADAPTIVE_THRESH_MEAN_C`, which calculates the mean of the neighboorhood of the pixel
  - `ADAPTIVE_THRESH_GAUSSIAN_C`, which is a weighted sum according to a gaussian distribuition centered on the pixel

  both of these settings can be thought of as kernels

B. Exercise 2

To save the previous video transformations we use the `cv::VideoWriter` class, it's constructor accepts the output file name, the fps of the output video, the size of the output video and a fourcc code to specify the codec to be used, the list of available codecs can be seen here, we use the MJPG codec due to its wide use. Also the image must be converted to BGR (this doesn't change the colors) before using `VideoWriter::write()` and consequentially saving it.

C. Exercise 3

In this section the `cv::threshold()` function is used, it thresholds the image according to a user specified value, to make this process simpler we first attach a trackbar to the window, this is done with `cv::createTrackbar()` where we have to specify the window name and the address of the value to be changed, it's results are less robust than those gotten from `cv::adaptiveThreshold()`, Figure 3 shows an example of both using the `cv::threshold()` and the `cv::createTrackbar()` functions.

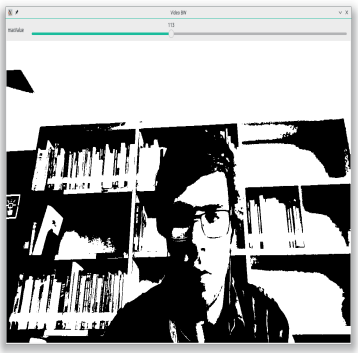

Fig. 3. Black and White image, threshold

D. Exercise 4

Here we use the `cv::inRange()` function, which segments the image according to a range of the channels, specified for each channel (white means the pixel channels are all in range), we explore the HSV and the LAB colorspace

The HSV colorspace separates the colors into Hue, Saturation, Value. the chromaticity (as opposed to luminance) lies in the Hue and Saturation channels, being the color specified individually by the hue channel. The Lab colorspace was made to be perceptually uniform, with the L channel specifying the Lightness so, again, chromaticity is only in two channels. Our approach is to have various trackbars to specify the low and high ranges of each channel. The results of this on a certain setting can be seen in Figure 4 and achieves decent results. But, by changing the location (and therefor the lightning) and

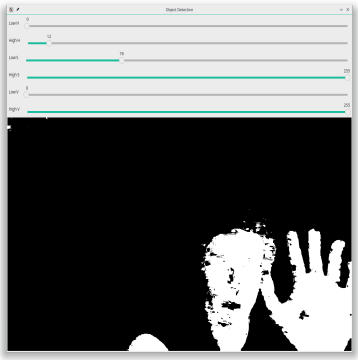

Fig. 4. HSV skin color segmentation, location 1

keeping the settings, we achieve the results seen in Figure 5 We conclude our algorithm is very sensitive to lightning, note

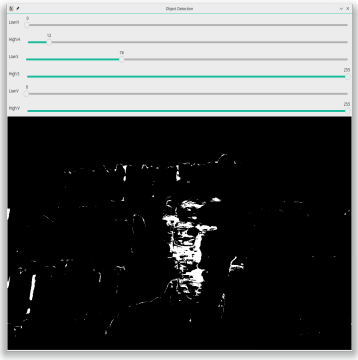

Fig. 5. HSV skin color segmentation with settings form location 1, location 2

though that the chosen setting is particularly difficult, the best settings we could find gave the results seen in Figure 7 The LAB colorspace was a bit more difficult to tune and the results weren't much better as seen in Figure **??**.

E. Exercise 5

The goal of this exercise was to do a program to capture images from the digital camera and explore the use of filters. The filters used are:

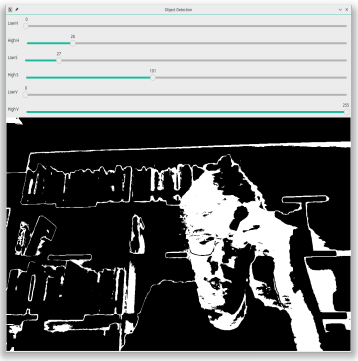

Fig. 6. HSV skin color segmentation, location 2

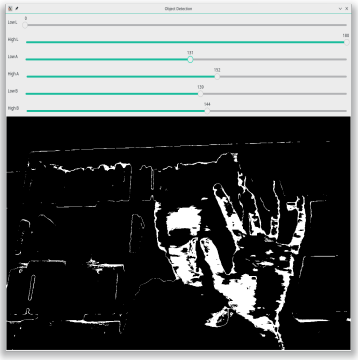

Fig. 7. LAB skin color segmentation, location 2

- blur
- GaussianBlur
- medianBlur
- bilateralFilter
- boxFilter

In the next part it will described the results of each filter. It was implemented a track bar to manipulate some parameters of each function in real time. One thing that should be considered is that the camera has poor quality having already in itself a lot of granularity. In the figure 8 it is presented the figure without filters.

Fig. 8. Image with no filters

1) Blur: The first function that it will be tested is the function blur. This function blurs an image using the normalized box filter. The parameter manipulated was the ksize. This parameter corresponds to the blurring kernel size. It is used the trackbar to choose the values in real time of the ksize, so it is easier to analyze the difference between different values. For show the difference between different values it was screenshot 3 different values. The first value was using the ksize equal to 11 9. The intensity of the effect is not very big, however it is clearly that has some blur. Then, the value that it was screenshot was 31. Here, the effect has some impact in the image and it is lost quite amount of information. For example, the guitar support presented in the right of the image almost disappeared. Lastly, it was captured the image using the value 61 and basically it is increased the intensity of the effect like was expected.

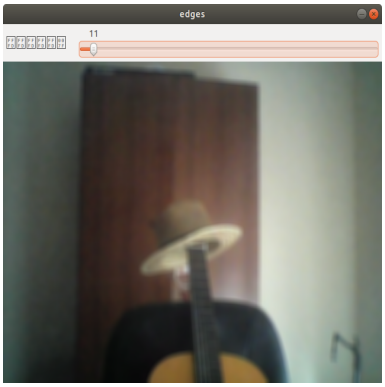

Fig. 9. Blur with ksize=11

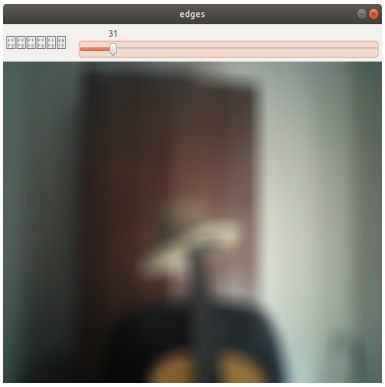

Fig. 10. Blur with ksize=31

2) GaussianBlur: This function blurs the image by using a Gaussian filter. The parameter tested was the Ksize. The value of this parameter must be odd. To compare the different tests it was screenshot the program 3 times. The first time the ksize had the value 11 12, the second time the value was 31 13 and the last time the value was 14. In this camera using the value 11 has not a large impact in the image. However, by using the parameters 31 is clear the effect on the image. It is possible to see that this effect is associated how people see things if they suffer from myopia. Even using the value 63 is possible to recognize the objects, however like it was expected a lot of

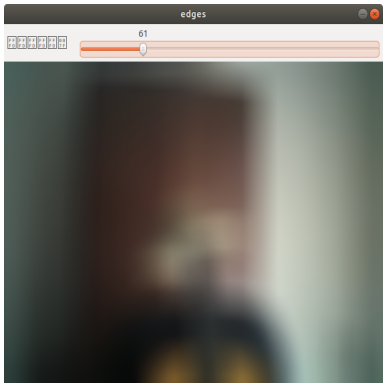

Fig. 11. Blur with ksize=61

information is lost. Comparing this effect to the blur effect, the big difference is that blur effect expands in some way the objects much more and for the same ksize the effect has a bigger impact on the image.

Fig. 12. Gaussian Blur with ksize=11

Fig. 13. Gaussian Blur with ksize=31

3) MedianBlur: In this situation this effect blurs an image by using the median filter. The parameters manipulated was ksize and it must be odd and greater than one. During the tests it was made a screenshot three times. The first screenshot was with the value 11 15 , the second one with 31 16 and the last one with 63 17. By using the value 11 the effect is already very strong. This effect resembles the watercolor painting. The big difference from using the value 11 or the 63 is that by using

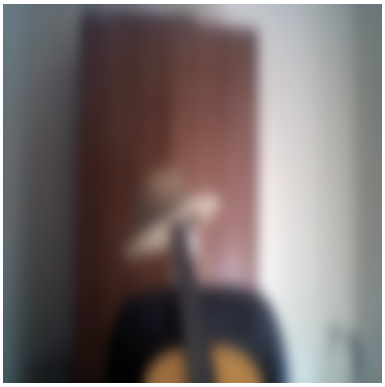

Fig. 14. Gaussian Blur with ksize=63

the value 63 it is lost a lot of information. For example, It is possible to see that the corners of the wardrobe are curved and also the hat almost disappeared.

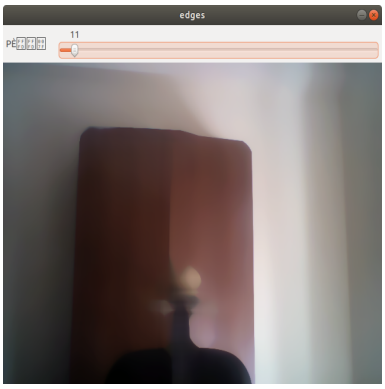

Fig. 15. Median Blur with ksize=11

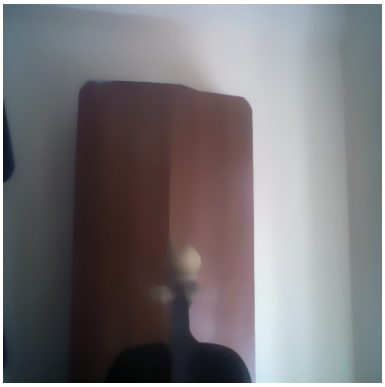

Fig. 16. Median Blur with ksize=31

4) BilateralFilter: This function applies bilateral filtering to the input image. Here it was made four different tests. In this test it was changed three parameters. One of the parameters was d. This parameters corresponds to filter size. The other two parameters were sigmaColor and sigmaSpace. A larger value of the sigmaColor parameter means that farther colors within the pixel neighborhood will be mixed together In the figure. A larger value of the sigmaSpace parameter means that farther pixels will influence each other as long as their colors

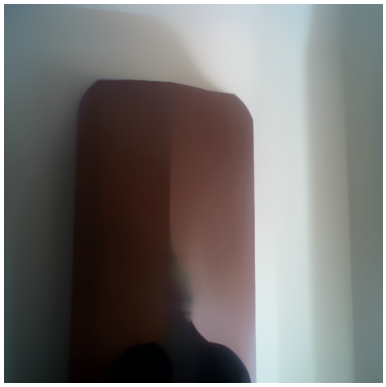

Fig. 17. Median Blur with ksize=63

are close enough. In the figure 18 it is possible to see the figure with a filter size of 10 and both sigmas with the value 300. Then, it was changed the value of the filter size to 5 19. It is possible to see that the effect is much less. This values were tested by reading the documentation. For example, it is advised to use a filter size around 5 if it is an application in real time, however if it is an offline application it can be used a filter size of 9. And, in our application it is possible to see that for bigger filter sizes the application crashes or it is very slow. The next test was to see the difference between the sigmas. In this figure 20 the sigma color was 0 and the sigma space equal to 300 and in this figure 21 it is possible to see the sigma color equal to 300 and the sigma space equal to 0. Here, it is not possible to see much difference. However, we have to remember the poor quality of the camera.

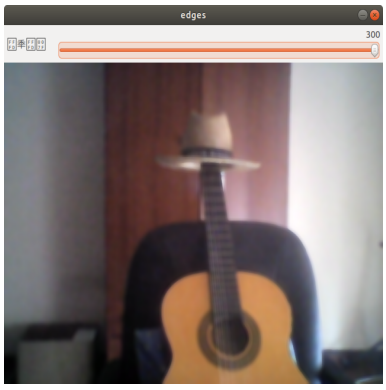

Fig. 18. bilaterail- size filter=10, sigmas=300

5) BoxFilter: This function blurs an image using the box filter. Here it was used the track bar to manipulate the ksize parameter that corresponds to the blurring kernel size. In this function it was screen shot 3 different values to compare them. It is possible to see in the figure 22 that using only a ksize of 10 the effect is clear and the blur is quite a lot. Then, in the figure 23 it is used the value 50 and in the figure 24 the value used is 110. With the value of 110 the effect is very strong and it is impossible to recognize the objects and it is only possible to have an idea of the main colors presented in the image.

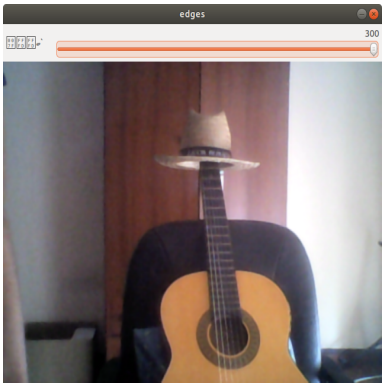

Fig. 19. bilaterail- size filter=5, sigmas=300

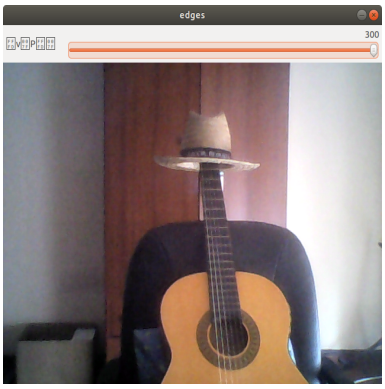

Fig. 20. bilaterail- size filter=5, sigmaColor=0, sigmaSpace=300

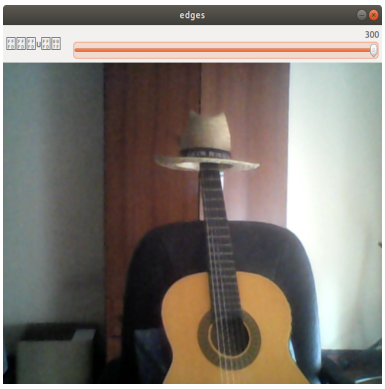

Fig. 21. bilaterail- size filter=5, sigmaColor=300, sigmaSpace=0

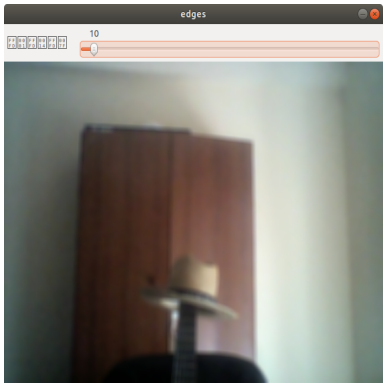

Fig. 22. BoxFilter- 10

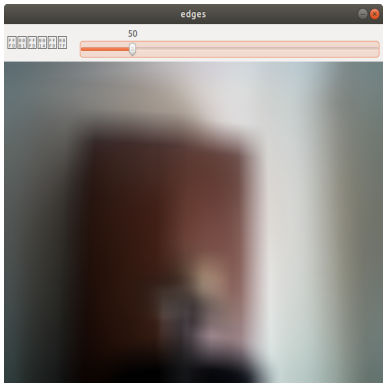

Fig. 23. BoxFilter- 50

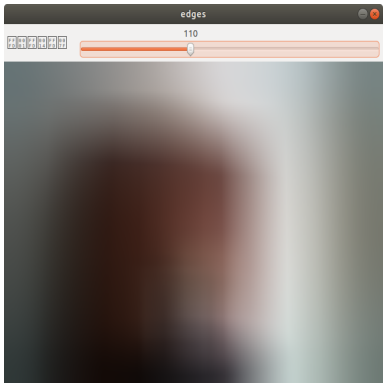

Fig. 24. BoxFilter- 110

F. Exercise 6

The goal of this exercise is to calculate the histogram of each one of the channels (R, G and B) of the captured image and also, to calculate the histogram of the grayscale version of the captured image. In the figure 25 it is possible to see the correspondent image of the histogram of the figure 26. By analyzing the histogram of each channel it is possible to see that each channel has similar intensities, because the histogram of each channel is similar.

By analyzing the histogram of the grayscale image is possible to see that the image has more intensities in the range of the dark gray and higher is the intensity less is the occurrence of that intensity and eventually above one value it is possible to see that there is no pixels with that intensity.

G. Exercise 7

In this exercise the goal is to apply histogram normalization. Comparing the normalized histogram of the rgb image with the histogram without normalization it is possible to conclude that the difference of each channel is much less in the normalized histogram and each channel is much more predictable, stable, having less peaks and being more constant(regular). In the grayscale images it is possible to see the same thing, however in this situation it is not so big the difference.

Fig. 25. RGB Image

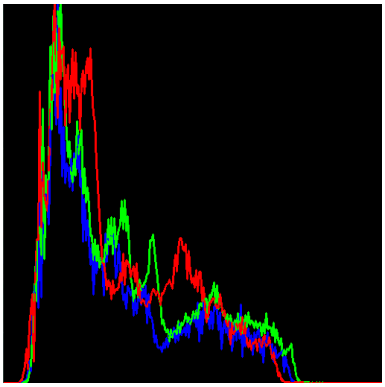

Fig. 26. Histogram of the RGB image

Fig. 27. GrayScale Image

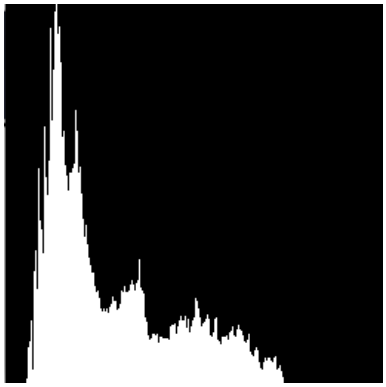

Fig. 28. Histogram of GrayScale Image

Fig. 29. RGB Image of normalized histogram

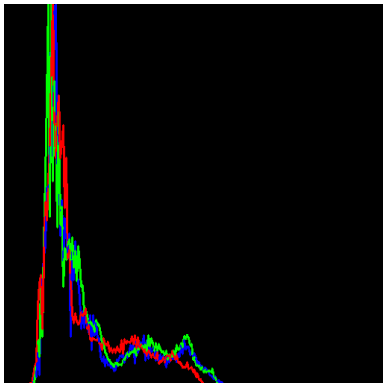

Fig. 30. Histogram Normalization of rgb image

H. Exercise 8

for histogram comparison we use `cv::compareHist()` which accepts two histograms, and a way to compute the distance between them, 4 distance function can be supplied.

1) Correlation
2) Chi-Square
3) Intersection
4) Bhattacharyya

We decide to use the correlation distance whose formula is as follows:

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}}$$

where

$$\bar{H_k} = \frac{1}{N} \sum_J H_k(J)$$

To note that the correlation distance has values in the range of $[-1, 1]$ with 1 being a perfect match, to better think of it as a distance measure we actually use $1 - correlation_distance$, so our range is now $[0, 2]$ with 0 being a perfact match. To test out histogram comparison program we use the MMU Iris Database. The closest image according to our histogram distance definition of Figure 33 was Figure 34

Fig. 31. Grayscale image to calculate histogram with normalization

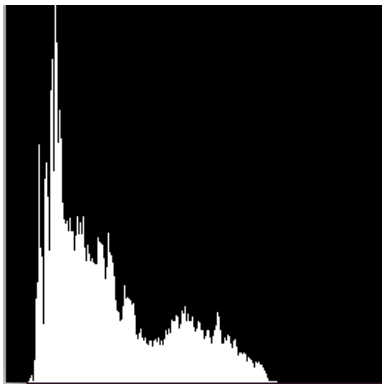

Fig. 32. Histogram Normalization of grayscale image

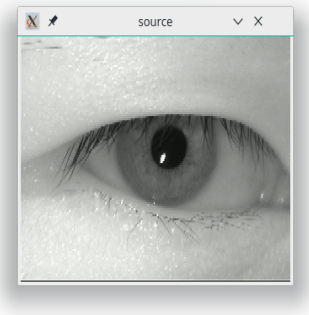

Fig. 33. Loker1 image of the MMU Iris dataset

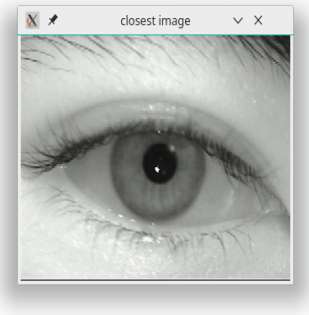

Fig. 34. Eugenehol4 image of the MMU Iris dataset