Computer Vision (2018/19)

Pedro Santos, *Universidade de Aveiro, 76532* Silvério Pereira, *Universidade de Aveiro, 76505*

***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

In the first exercise the goal is to capture images from the digital camera and explore the use of the function Sobel to calculate the image of gradients. Also, it is necessary to add the capability of calculate the Laplacian of an image. The sobel function has eight arguments. Each argument is described in the next list.

- **src-** input image.
- **dst-** output image of the same size and the same number of channels as src .
- **ddepth-** output image depth, see combinations; in the case of 8-bit input images it will result in truncated derivatives.
- **dx-** order of the derivative x.
- **dy-** order of the derivative y.
- **ksize-** size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
- **scale-** optional scale factor for the computed derivative values; by default, no scaling is applied.
- **delta-** optional delta value that is added to the results prior to storing them in dst.
- **borderType-** pixel extrapolation method

The arguments that will be tested in this exercise are the ksize, scale and delta.

In the figure 1 it is possible to see the result of using the value 1 for ksize and scale and value 0 for delta value. The next step is to test with another values for the Ksize. As the Kernel size increases, more pixels are now a part of the convolution process, so it is expected that the edges will be more noticeable. The result of ksize equal to 3 is possible to see in the figure 2 and comparing this to the result of using ksize equal to 1, it is possible to see that the edges are more visible and they have a greater prominence. Mostly likely because of the quality of the camera, when the ksize is equal to 5 it detects edge all over the image. This happens because the quality of the camera is bad and the image is very pixelated, so it will detect changes of intensities easily. One solution that could increase the results would be applying a Gaussian blur to remove some of the noise.

The next step was to test different values for the scale. In the figure 1 it is visible the result of using scale equal to 1. This argument is a scale factor for the computed derivative values, so it is expected also that the edges will be more noticeable, so the result will be similar of using a bigger ksize. In the figure 4 the value of the scale is 5. And it is possible to conclude that the result of using scale 5 or ksize equal to 3 is similar. The advantage of adjust the scale is that is much less sensitive and it is possible to increase on by one. On the other hand, ksize is much more volatile and the change of the value will produce much more different results. In the figure 5 is presented the result of using scale equal to 10. This is a mid term between using ksize equal to 3 ou 5. The last step to conclude this Comparation was to try to manipulate the value of the scale to produce a similar result of using ksize equal to 5. The conclusion was that by using a value around the 74 for the scale it will be very similar of using ksize equalt to 5. As it is possible to see the argument scale is much more controllable and it is possible to produce similar results. One little difference that it is possible to detect it is that the manipulation of the ksize produces a result with a little of blur.

The last step was to test the manipulation of the argument delta. the manipulation of this argument is not comparable to any of the others in terms of results. The manipulation of this value will give a perception of depth in the image. As the delta increase the perception of depth will increase. By other words, this effects increases the perception of the objects that are closer or more far away. The results of using the delta of 0, 10, 20 and 65 are in the respective figures 1, 7, 8 and 9.

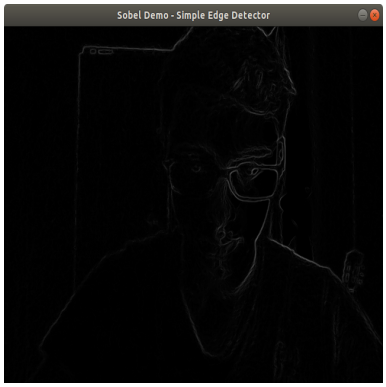

Fig. 1. Sobel Function- ksize=1,scale=1,delta=0

The second part of the exercise was to use the laplace function to detect edges. Comparing the result of using the laplace function, over the sobel function it is possible to see that for the same arguments the function sobel it is much accurate. Also, the laplace function seems to be much more sensitive to the noise of the image. Furthermore, the sobel function produces a distention much more precise of the edges. One thing that it is necessary to have in consideration is that

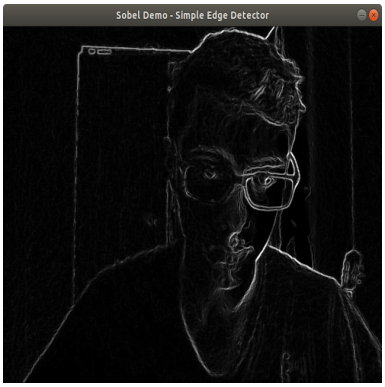

Fig. 2. Sobel Function- ksize=3,scale=1,delta=0

Fig. 3. Sobel Function- ksize=5,scale=1,delta=0

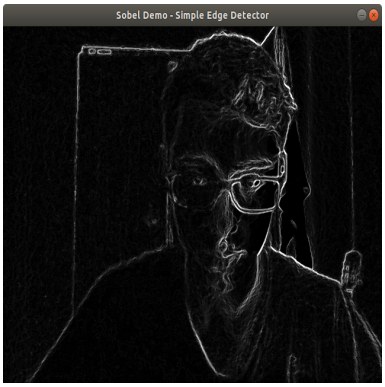

Fig. 4. Sobel Function- ksize=1,scale=5,delta=0

the laplace function computationally is much less expensive because uses only one kernel, in the other hand sobel uses two kernels. The resul of using the laplace with ksize and scale equal to 1 and delta equal to 0 is in the figure 10. The other tests of changing the kernel, scale and the delta are in the next figure 11, 12, 13, 14, 15, 17 and 16. The differences of manipulating the arguments are very similar of the results of manipulating this arguments in sobel. The biggest difference is that the manipulation of the delta value is much less sensitive. In other words, for the same value in laplace it will be seen a much less effect.

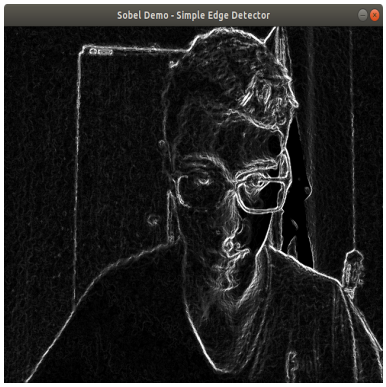

Fig. 5. Sobel Function- ksize=1,scale=10,delta=0

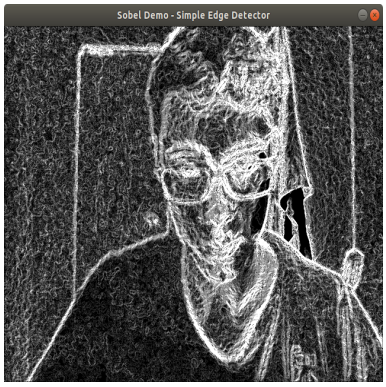

Fig. 6. Sobel Function- ksize=1,scale=74,delta=0

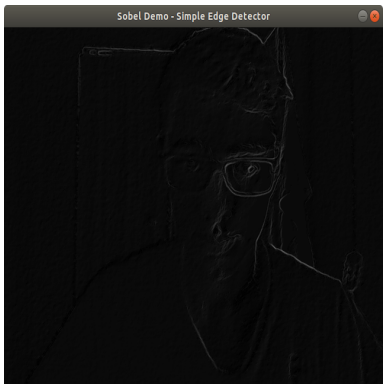

Fig. 7. Sobel Function- ksize=1,scale=1,delta=10

B. Exercise 2

In the second exercise was implemented a program to capture images from the digital camera and perform edge detection with Canny's algorithm. The function canny has the arguments presented in the next list.

- **image-** 8-bit input image.
- **edges-** output edge map; single channels 8-bit image, which has the same size as image.
- **threshold1-** first threshold for the hysteresis procedure.
- **threshold2-** second threshold for the hysteresis procedure.
- **apertureSize-** aperture size for the Sobel operator.
- **L2gradient-** a flag, indicating whether a more accurate

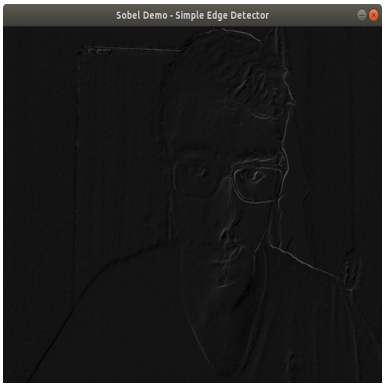

Fig. 8. Sobel Function- ksize=1,scale=1,delta=20

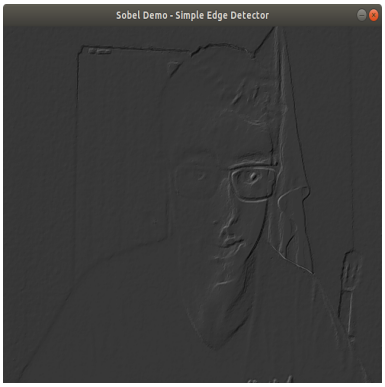

Fig. 9. Sobel Function- ksize=1,scale=1,delta=65

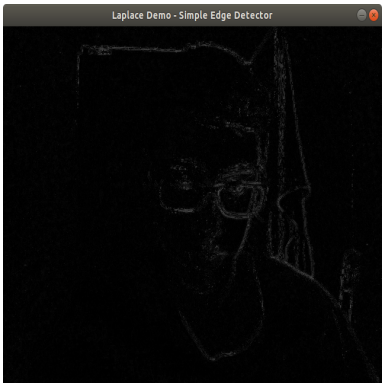

Fig. 10. Laplace Function- ksize=1,scale=1,delta=0

Fig. 11. Laplace Function- ksize=3,scale=1,delta=0

Fig. 12. Laplace Function- ksize=5,scale=1,delta=0

Fig. 13. Laplace Function- ksize=1,scale=5,delta=0

norm should be used to calculate the image gradient magnitude.

In the tests the values that were manipulated were both thresholds and the second one is always 3 times bigger than the first one and the apertureSize. When it is used a treshhold1 of 0 and aperture size of 3, it is possible to see the result in the figure 18. After analyzing the result it is possible to conclude that the threshold is very low and for that reason all the noise of the image is detect like an egde. By analyzing the figures 19,20 and 21 where the threshold is increasing, it is possible to see clearly that each time that the threshold increases, the edges detected are less. So, it is necessary to find the right value of the threshold to not detect the noise and detect the ”real” edges of the frame, because in the other hand if it is used the threshold of 100 a lot of edges are lost due to the fact that the threshold is to much bigger. In the figure 22 is used a aperture size of 5 and a threshold of 0. Comparing this result with the result of using aperture size of 3 and threshold of 0, it is possible to see that the distance between the detected edges is bigger, therefore there are also less detected edges. If it is compared the result of aperture size of 5 and a threshold of 100 it is possible to see that has much more edges detected comparing with the aperture size of 3 and also a threshold of 100. The result of this is in the figure 23.

Fig. 14. Laplace Function- ksize=1,scale=10,delta=0

Fig. 15. Laplace Function- ksize=1,scale=15,delta=0

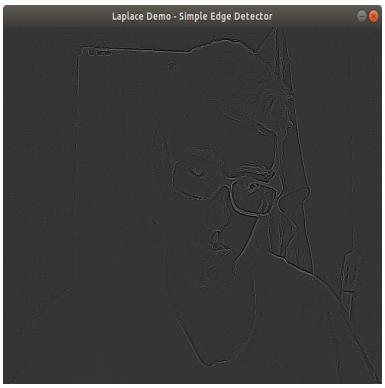

Fig. 16. Laplace Function- ksize=1,scale=1,delta=54

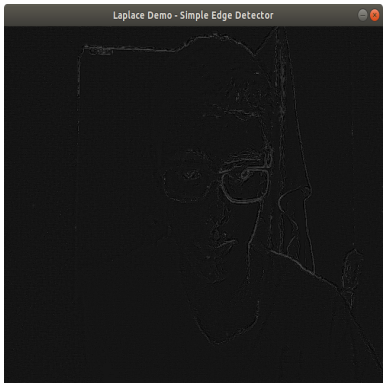

Fig. 17. Laplace Function- ksize=1,scale=1,delta=21

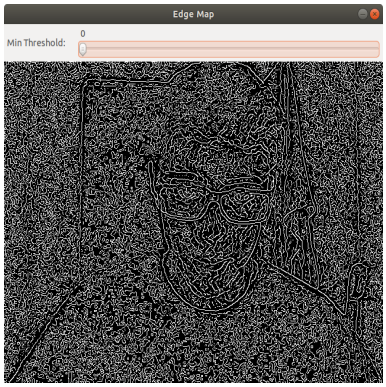

Fig. 18. Canny Function- apertureSize=3,threshold1=0,threshhold2=0

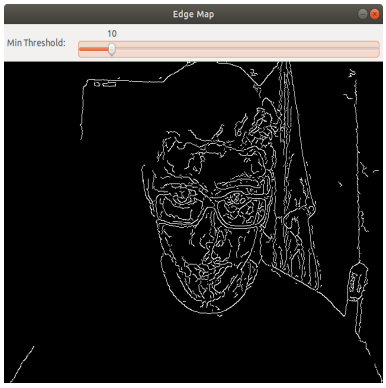

Fig. 19. Canny Function- apertureSize=3,threshold1=10,threshhold2=30

C. Exercise 3

The goal of this exercise is to implement a program to capture images from the digital camera and perform corner detection. The algorithm used to detect the corner was the Harris's algorithm. And then, it was used the function Circle of Open CV to draw a circle in the detected corners. The cornerHarris function has six arguments that are explain in the next list.

- **src-** Input single-channel 8-bit or floating-point image.
- **dst-** Image to store the Harris detector responses.
- **blockSize-** Neighborhood size.
- **ksize-** Aperture parameter for the Sobel() operator.
- **k-** Harris detector free parameter.
- **borderType-** Pixel extrapolation method.

The result of this in in the figure 24. It is necessary to have in mind that this function and then draw each circle in each corner is quite expensive computationally, so it is not viable to do this real time.

D. Exercise 4

In this exercise, we try to detect lines and circles on an image, we will use Figure 25 as an example. To detect lines and circles, we first calculate a binary image with the edges of the original image, for this we try using `cv::Canny`, described in detail in II-B. The best threshold found was 40 and the results can be seen in Figure 26 We then use

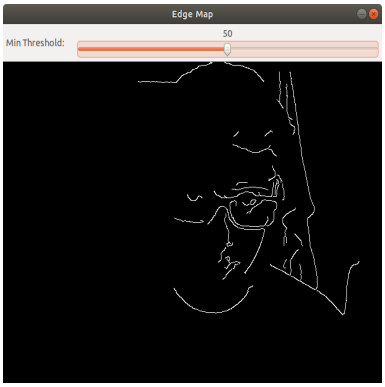

Fig. 20. Canny Function- apertureSize=3,threshold1=50,threshhold2=150

Fig. 23. Canny Function- apertureSize=5,threshold1=100,threshhold2=300

Fig. 21. Canny Function- apertureSize=3,threshold1=100,threshhold2=300

Fig. 24. cornerHarris

Fig. 22. Canny Function- apertureSize=5,threshold1=0,threshhold2=0

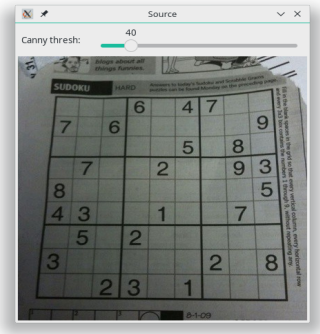

Fig. 25. Original sudoku image

`cv::findContours` to find the regions of interest, which gives us Figure 27 `cv::findContours` Finds countours in a binary image, hence the preprocessing step of detecting edges with `cv::Canny`, it needs:

- An image
- A mode, that specifies which and how to return the contours it finds, it can output: the external contours, a list of contours, or an hierarchy of contours. The logic behind the hierarchy return is that some contours are inside other contours and that information may be useful.
- A method, that specifies how to represent the contours found, it can return all the contour points, or compress horizontal, vertical, and diagonal segments, in this last method, an up-right rectangular contour would be encoded using only 4 points
- An offset, in case we want to find countours in an image ROI

and returns:

- The detected contours
- An hierarchy, optional, which contains information about the contour topology, it's dependant on the specified mode

E. Exercise 5

To use `cv::HoughLines`, we again detect the edges first, the result is the same as in II-D and can be seen in 26, The

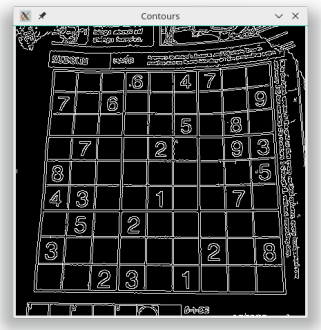

Fig. 26. Sudoku edges

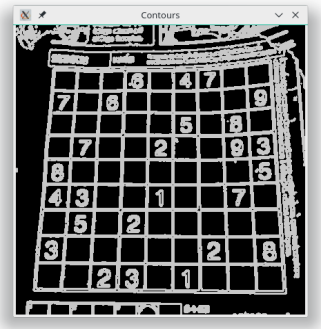

Fig. 27. Sudoku contours

sudoku image is moslty composed of lines so that is what we will be exploring. Using the `cv::HoughLines` we get Figure To do this transformation, `cv::HoughLines` needs:

- A binary source image
- The lines found, it uses a polar coordinate system so each line is represented by a two element vector composed by the distance to origin and the rotation angle (in radians), i.e, $y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$
- rho (density of line in pixels)
- threshold, to filter the resulting lines to the best ones

The parameters used were 1 for rho (equivalent to 1 pixel) and $\pi/180$ for the threshold

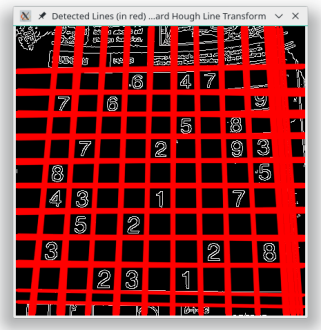

Fig. 28. Sudoku hough lines transform

and returns:

- The detected contours
- An hierarchy, optional, which contains information about the contour topology, it's dependant on the specified mode