LAB 3

EDGE DETECTION AND HOUGH TRANSFORM

Matteo Dicenzi

Marco Demutti

1. Introduction

In this experiment we are going to perform the **edge detection**, that is an image processing technique that can be used to find the boundaries of the objects within an image. Edges occur at boundaries between regions of different intensity. Specifically, we are going to implement the **Marr and Hildreth** edge detector: this algorithm allows to find the edges in an image by looking for the **zero crossings** after applying the **Laplacian of Gaussian Operator** to it. In fact, since the convolution operation is associative, this is equivalent to smoothing the image with a Gaussian Filter to limit the effect of the noise at first, and then computing its Laplacian to highlight the regions of rapid intensity change.
We will then use our algorithm on an image, and evaluate it by comparing the results to the ones obtained using the equivalent MATLAB function ***edge***.
Finally, as a further processing stage, we are going to use the **Hough transform**, that allows to detect the **straight lines** in an image, by grouping edges using a voting technique: each edge point votes for all possible lines passing through it, and then the most voted lines are chosen.

2. Procedure

In the first part of the experiment, the first step is the computation of the **Laplacian of Gaussian Operator (LoG)**: ***laplacianOfGaussian*** is the function we wrote for this purpose. It takes as input the standard deviation, a parameter which is related to the spatial support of the Gaussian filter, and it returns as output the LoG, computed with the following formula:

$$\Delta^2 G_\sigma = \frac{1}{2\pi\sigma^2}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figure 3.1: Laplacian of Gaussian

The function is used to obtain the Laplacian of a Gaussian Filter with two different values of standard deviation: $sd = 1$, $sd = 3.5$. Then, we convolve the test image with the two LoG filters by using MATLAB function *conv2*; each convolved image becomes the input of the implemented function ***detectZeroCrossings***, which detects their zero crossings by scanning along each row and recording an edge point at the location of the zero-crossing (by internally calling ***detectZeroCrossings_rows***), then repeating the same procedure along each column (by internally calling ***detectZeroCrossings_rows*** again but passing the transpose of the same image). In order to avoid weak zero crossing due to noise, a threshold can also be passed to the

function: if so, given for example two subsequent values {a,-b}, the function records a new edge only if $|a + b| > threshold$.
As output, ***detectZeroCrossings*** returns the binary image containing the edge map.
The function is used with the two convolved images and with different values of threshold. All the results are shown with MATLAB function ***imagesc***, and compared with the images obtained by using MATLAB function ***edge('log',…)***.

In the second part of the experiment, edge detection is carried out with the Sobel edge detector on the two inputs images *highway1* and *highway2*, by using MATLAB function ***edge***. Then, through the MATLAB function ***hough,*** we obtain the Standard Hough Transform **H** and the arrays of **ρ** **ϑ** values over which the Hough transform matrix was generated.
In order to locate the peaks in the Hough transform matrix, we use MATLAB function ***houghpeaks***, specifying the maximum number of peaks to identify.
Finally, we can find the straight lines and the line segments within the function ***computeStraightLines***. This function takes as inputs all the variables obtained so far (the original image, the binary image, the hough transform, **ρ** and **ϑ**), finds and plots the straight lines, along with the line segments on the image. This last values are internally obtained with the MATLAB function *houghlines*.
The procedure is repeated several times on both images *highway1* and *highway2*, using different parameters (i.e. number of peaks in function *hough*, 'fillgap' and 'minlength' in function *houghlines*) in order to try to find the best solution, and the results are displayed.

3. Results

Figures 3.1-3.2 show Boccadasse image, once convolved with the LoG filters of $sd = 1$ and $sd = 3.5$. We can see how, in both cases, the Laplacian allows to highlight the regions of rapid intensity change, but at the same time how the smoothing effect of the Gaussian filter is bigger in the case of a bigger standard deviation. So, on the whole, the LoG filter with a smaller standard deviation ($sd = 1$) allows to highlight sharper transitions with respect to the one with bigger standard deviation ($sd = 3.5$).

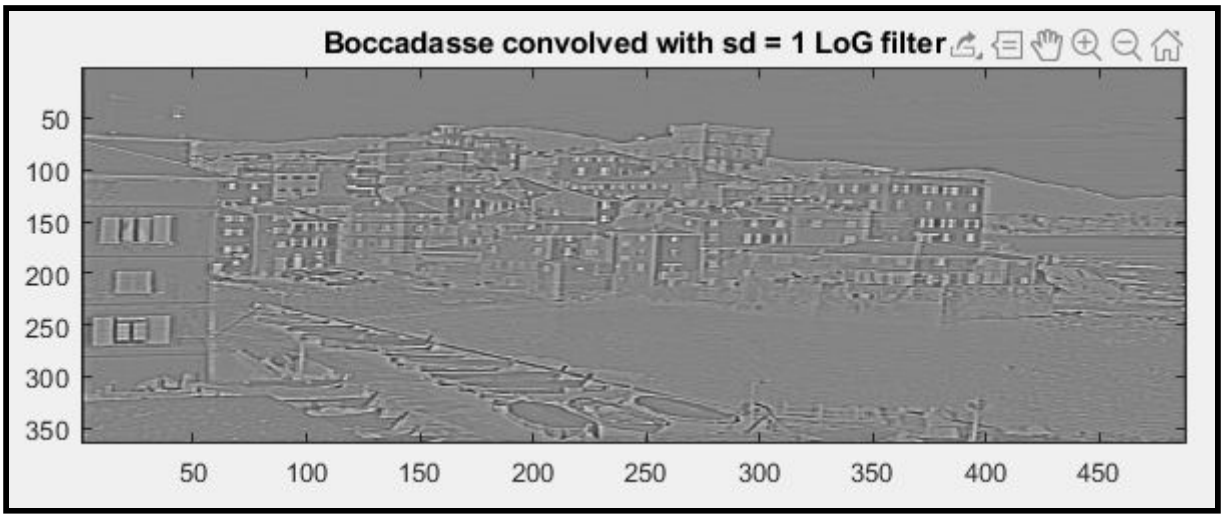

Figure 3.1 - Boccadasse image convolved with LoG filter with sd = 1

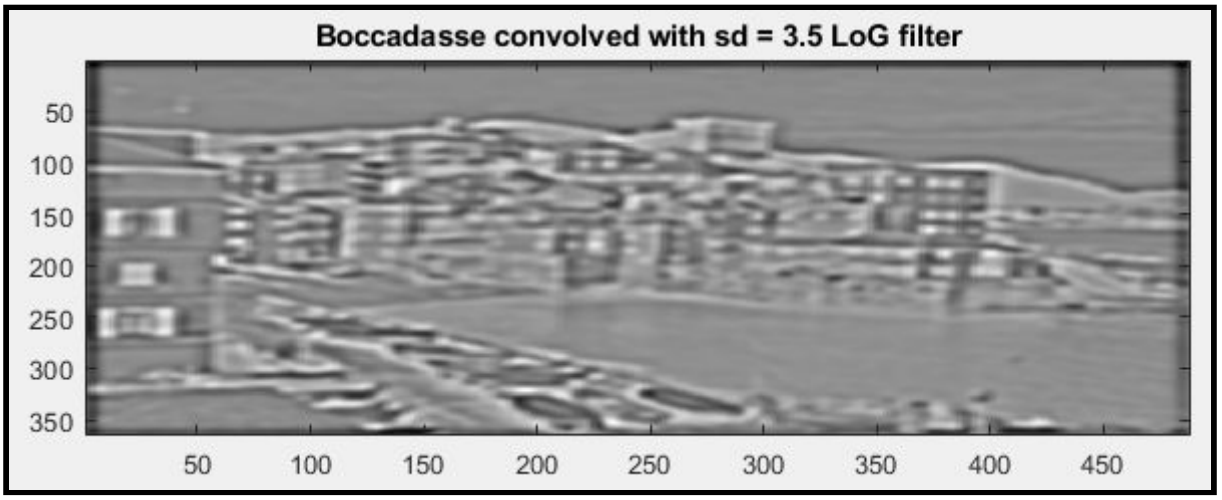

Figure 3.2 - Boccadasse image convolved with LoG filter with sd = 3.5

If we focus only on the case with $sd = 1$ and use function ***detectZeroCrossings***, we obtain the binary image corresponding to the edge map of the original image: each pixel of the map is set to '1' in correspondence of an edge. The results are shown in *Figure 3.3*, where we can also compare them with the ones obtained using *edge* MATLAB function: the two maps are quite different, since the one obtained using *edge* function detects a bigger number of edges.

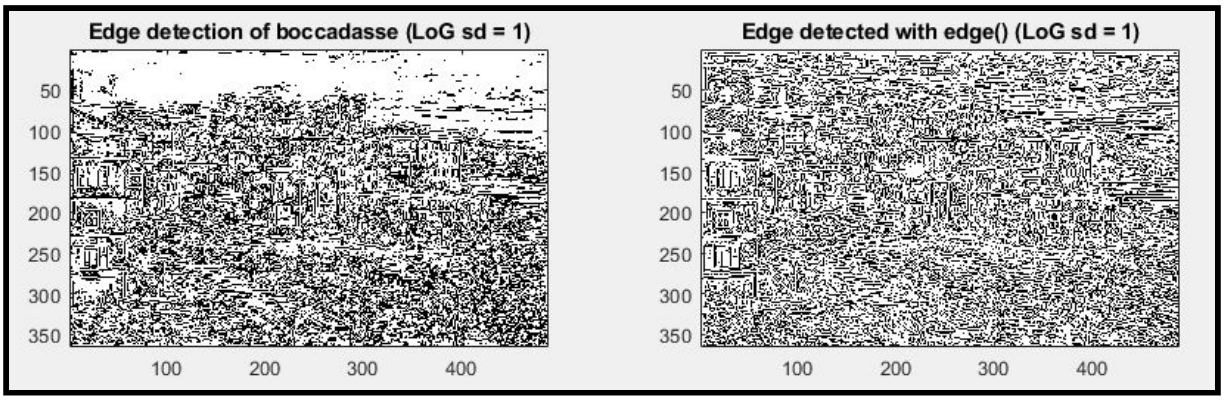

Figure 3.3 - Edge map with the two different functions (sd=1)

In *Figures 3.4 - 3.6* we can see the results obtained by applying the same two functions, but this time also specifying a threshold. In *Figure 3.4* we decided to apply a quite small value of threshold, that nevertheless allows to see a difference with respect to the previous case: some edges, that may be due to noise, are now discarded.
In *Figure 3.5* we can notice what happens when we apply a bigger threshold: more values are discarded, and some details of the original image are no longer visible.
Finally, in *Figure 3.6*, for an even bigger value of threshold, all the details are lost.
In all the three cases, the results are very similar, regardless the function we use (***detectZeroCrossings*** or ***edge***).

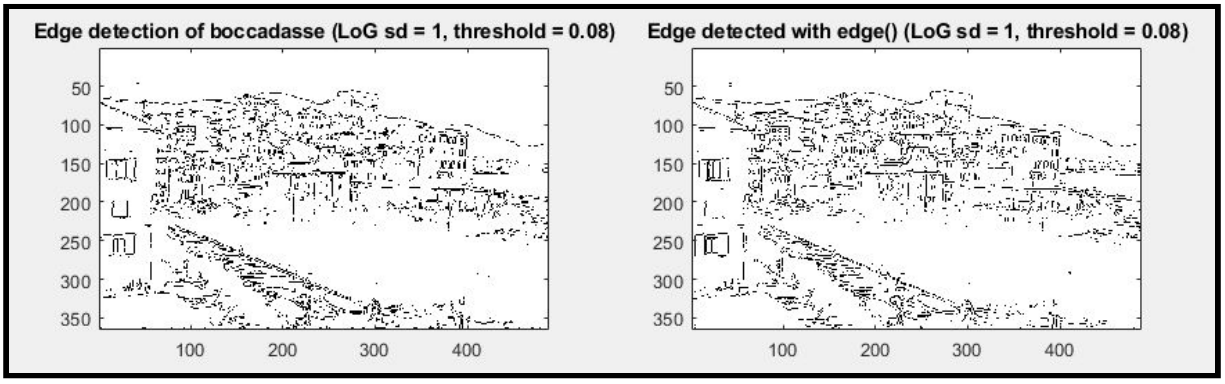

Figure 3.4 - Edge map with the two different functions (sd=1, threshold = 0.08)

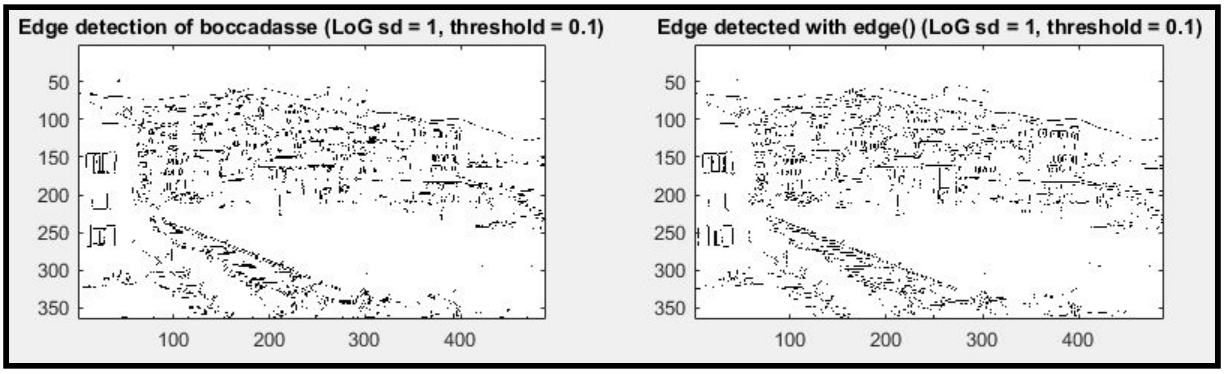

Figure 3.5 - Edge map with the two different functions (sd=1, threshold = 0.1)

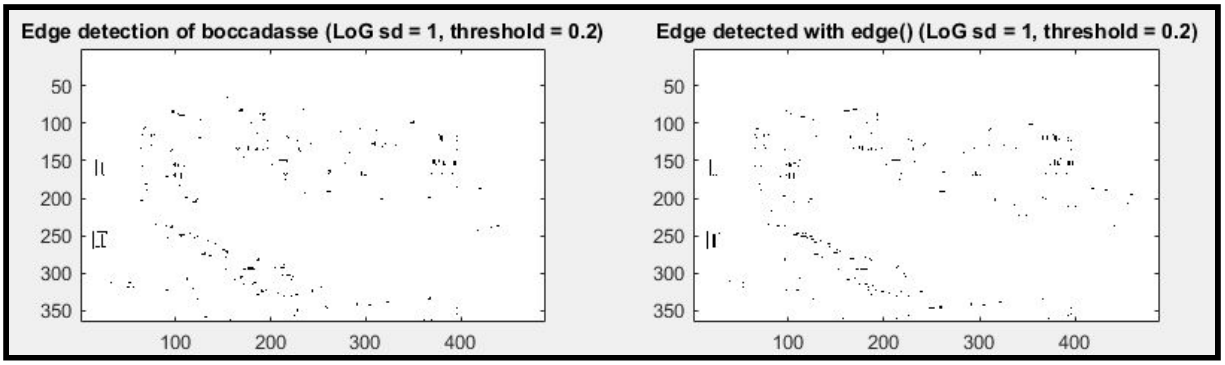

Figure 3.6 - Edge map with the two different functions (sd=1, threshold = 0.2)

In *Figures 3.7-3.8*, after applying the same procedure to the case of $sd = 3.5$, we can notice how the results are similar to the previous case: in the case of no threshold, all the edges are shown, and the outputs of the two functions are quite different; when adding a threshold, instead, some edges are discarded, and the two outputs become very similar.
Of course, as anticipated, the filter with $sd = 3.5$ is not able to detect transitions that are as sharp as the previous filter.

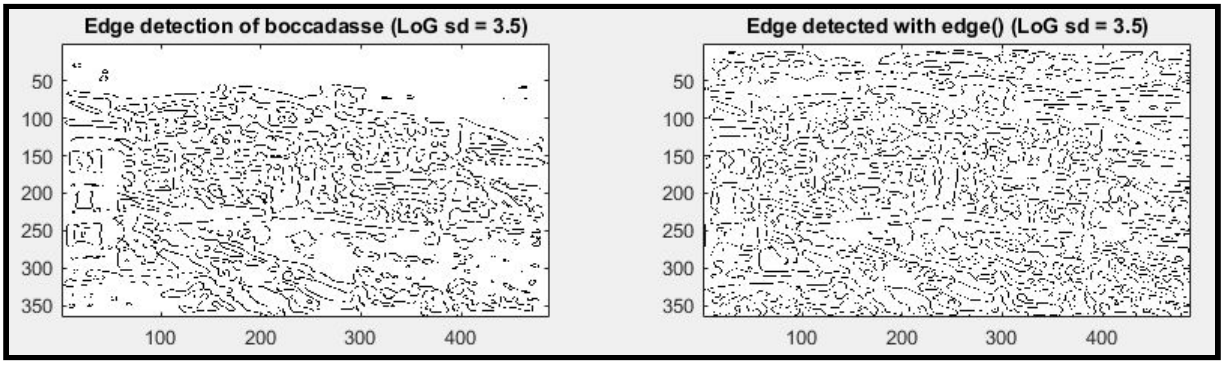

Figure 3.7 - Edge map with the two different functions (sd=3.5)

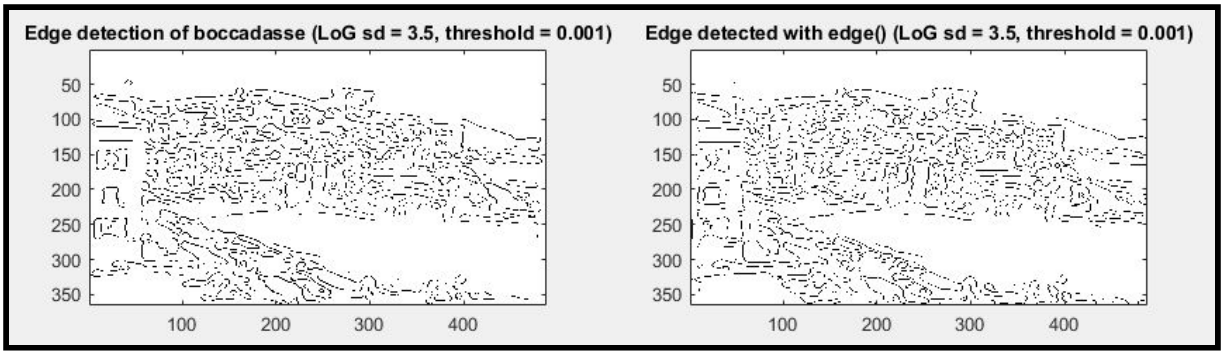

Figure 3.8 - Edge map with the two different functions (sd=3.5)

Considering now the second part of the experiment, in *Figure 3.9* we can see the results of the detection of the straight lines in image *highway1*, when using the following suggested parameters: 4 *maximum peaks* (*houghpeaks* function), *FillGap = 10*,

MinLength = 30 (*houghlines* function). In particular, *FillGap* represents the maximum distance that two generic lines segments can have in order for the *houghlines* function to merge them into a single line segment; *MinLength*, on the contrary, is the minimum length that a segment must have for the function to include it.
By slightly changing these parameters, we can obtain a better result, as shown in *Figure 3.10* (in this case, the *maximum number of peaks* was also changed and set to 10). This can be considered a satisfactory result, even though some imperfections are still visible.

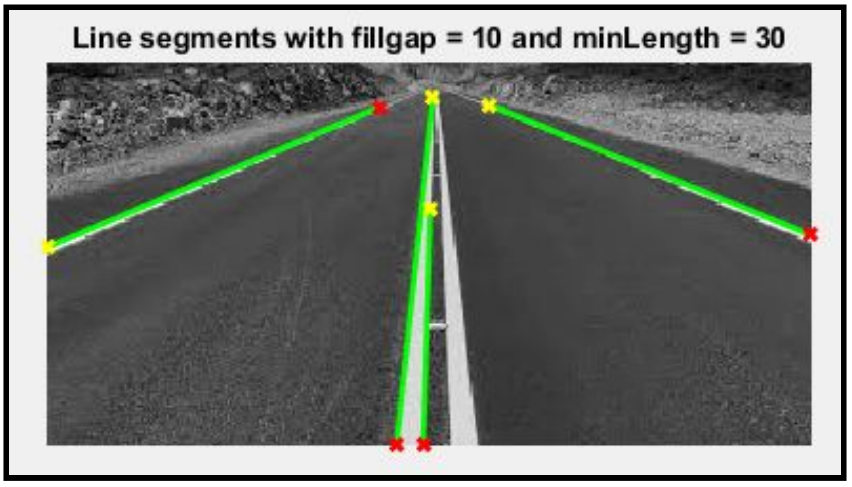

Figure 3.9 - Highway1 with line segments, first version

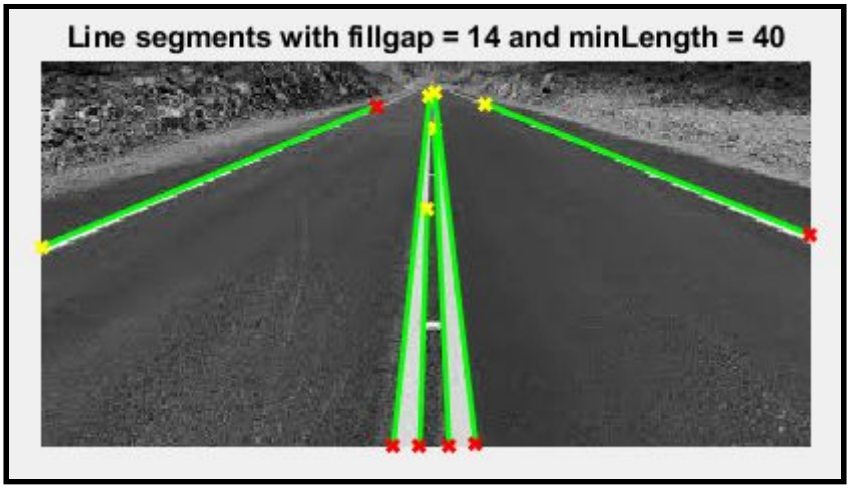

Figure 3.10 - Highway1 with line segments, second version

Finally, *Figure 3.11* show the lines detected in image *highway2* by using the suggested parameters (*maximum number of peaks = 5*, along with *NHoodSize* parameter properly set, in *houghpeaks* function; *FillGap = 10*, *MinLength = 30* in *houghlines* function).
In this case, the bigger road stripes are correctly detected, but we can see some undesired lines as well.
As for the previous image, the result can be slightly improved by changing some parameters (*Figure 3.12*), but some undesired lines remain visible.

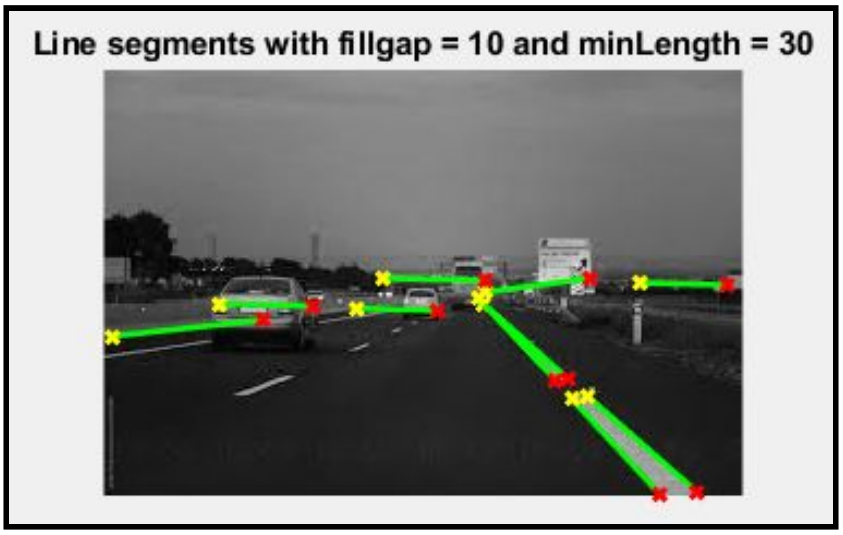

Figure 3.11 - Highway2 with line segments, first version

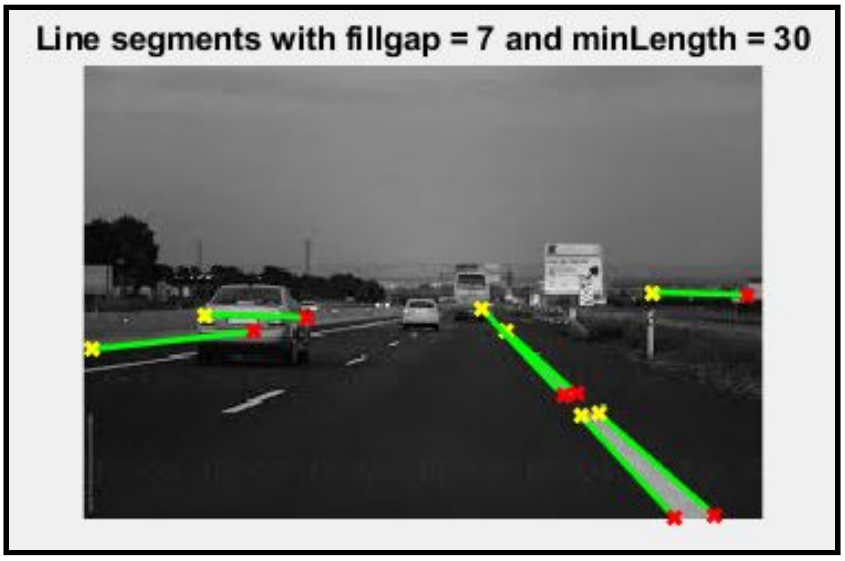

Figure 3.12 - Highway2 with line segments, second version

4. Conclusions

The results in the first part of the experiment were coherent with the ones expected: the Marr and Hildreth edge detector succeeded in detecting the edges in the image. It was also quite evident that applying a threshold is the only choice that allows to obtain a satisfactory image, but at the same time this value must not exceed, otherwise too many edges are discarded and it is no longer possible to distinguish all the details of the image (**trade-off** between smoothing and localization).
As for the second part, in both cases the main straight lines of the two images were correctly detected by using the Hough transform, but we were able to notice how many parameters must be properly chosen in order to obtain a good result. At the same time, the correctness of the result also depends on the type of image chosen.