Computer Vision Laboratory Report n.5

NCC-BASED SEGMENTATION HARRIS CORNER DETECTION

April 26, 2020

Chiara Terrile ID: 4337786
Giulia Scorza Azzarà ID: 4376318
University of Genoa

Contents

Chapter 1

Introduction

1.1 Defining the objective

The main goal of this laboratory is to perform the object detection in some images, as in the previous laboratory, but this time using the so called NCC-based segmentation.
This technique aims to extract a template from an image in order to perform the Normalized Cross Correlation, that is evaluating the correlation between the extracted template and the image with some similar images.
The results will be compared with the ones obtained in the previous laboratory, in terms of computational time and accuracy of detection.
The second part of this laboratory deals with the Harris Corner Detection, a technique that has the purpose to detect and display all the corners of an image.

Chapter 2

Theoretical Background

2.1 NCC-Based Segmentation

The normalized cross-correlation (NCC), usually its 2D version, is often encountered in template matching algorithms, such as in facial recognition, motion-tracking and others. In this laboratory the normalized cross-correlation is used to perform the detection of a car in an image and it is obtained by subtracting the mean value of both the image and the template to themselves and by dividing them for their standard deviation.
In MATLAB this happens by using the function *normxcorr2*.

2.2 Harris Corner Detection

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and deduce features of an image.
Harris Corner Detector takes into account the differential of the corner score with respect to direction and it is very accurate in distinguishing between edges and corners.

Chapter 3

Implementation and Results

The code is composed of two **main** functions, one for the six images representing the black and red cars crossing the road (**main_cars**) and the other one for the image 'i235.png' (**main_i235**). All the functions called in the main scripts are explained below.

In **main_cars** are used the following functions:

- **NCC.m:** which performs the normalized 2-D cross-correlation in order to determine the position of a selected template
- **compTime.m:** which computes the execution time for the function NCC.m
- **accuracyDet.m:** which shows the accuracy of detection of the object in the template

In **main_i235** is called just the following function:

- **largestBlob.m:** which computes the first n largest blobs according to the number desired

3.1 main_cars

In the first main function we have loaded the six images of the cars and then we have selected the two templates corresponding to the position of the two cars we wanted to find in the images (T_blackS and T_redS).
After that we have applied the function **NCC** to select in all six images the two templates.
The second part of this main is based on a comparison of three different size of the window chosen for the templates.
We have chosen other two templates for the black car (T_blackM and T_blackL), both bigger than the initial one. Then, after plotting the results of the car detection with these two new templates, we have applied the function **compTime** to compare the computational times according to the three different sizes of template.
At the end we have also compared the accuracy of the detection in the three windows using the function **accuracyDet**. In the following pages are explained all the functions that we have used.

3.1.1 NCC

This function takes as input the original image (img), the template (T), a boolean variable (plotImg), which is set to true or false according to our will to show or not the results, and a string used for the title of the plotted images.
The function returns as output the position of the center of the template, which corresponds to the highest score (mY and mX). To compute the highest score, we have first used the MATLAB function **normxcorr2**, passing as input the template T and the original image and getting as output a new image C.
In this new image we have checked the maximum value using the MATLAB function **find** to detect the position of the highest score.
Then, after adjusting the coordinates with respect to the sizes of the template, we have computed the size of the rectangle (minX and minY) to box the detected object.
Finally we have subplotted the original image, the template, the result of the normalized cross-correlation (C) and the detected object for both the cars (black and red) in all the six images loaded at the beginning and also for the other two sizes of the window for the black car.
All the results are shown in the next pages from Figure 3.1 to Figure 3.6 (black car), from Figure 3.7 to Figure 3.12 (red car), Figure 3.13 (medium size) and Figure 3.14 (large size).

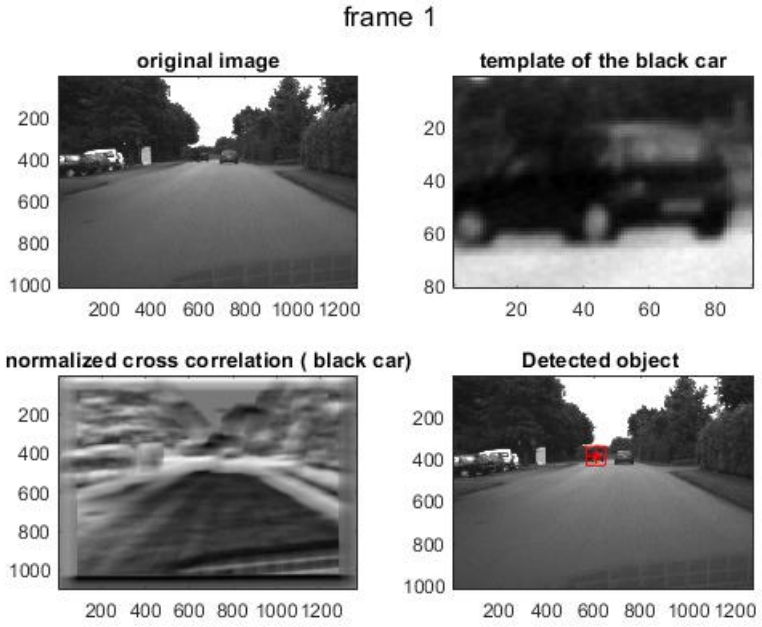

Figure 3.1

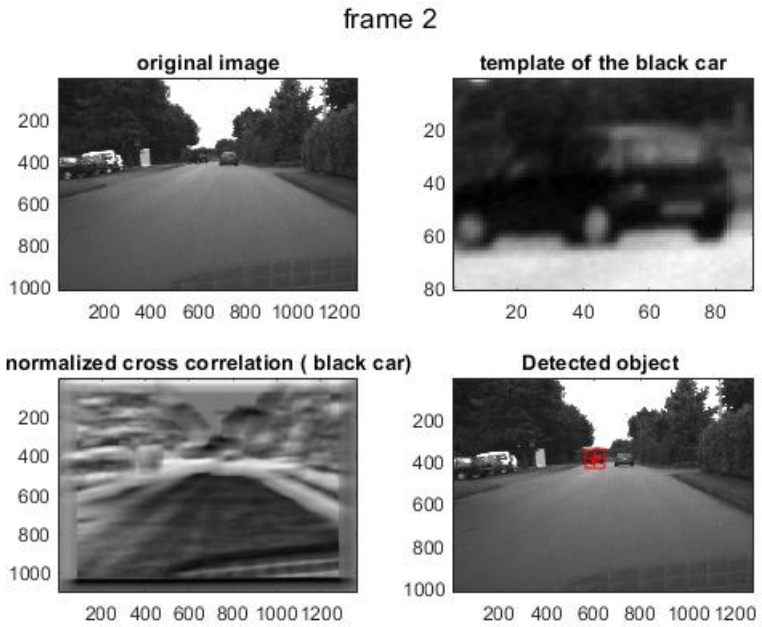

Figure 3.2

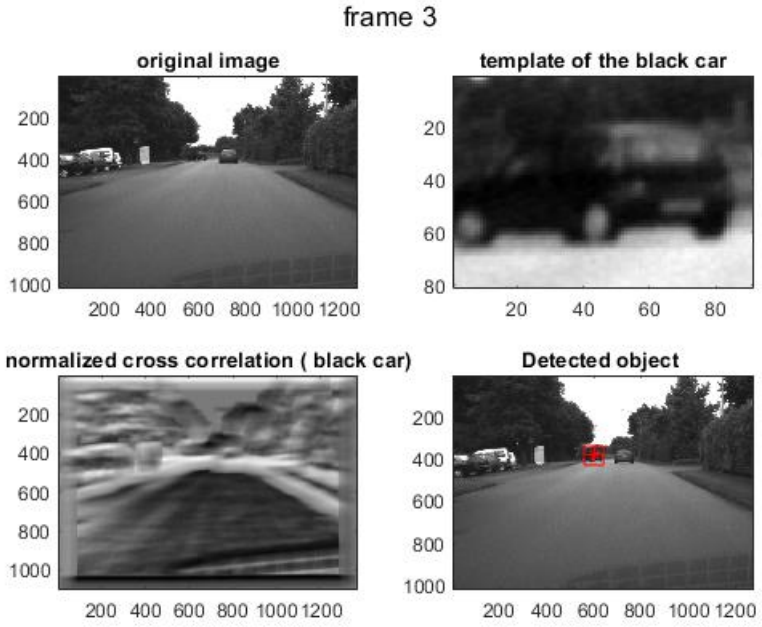

Figure 3.3

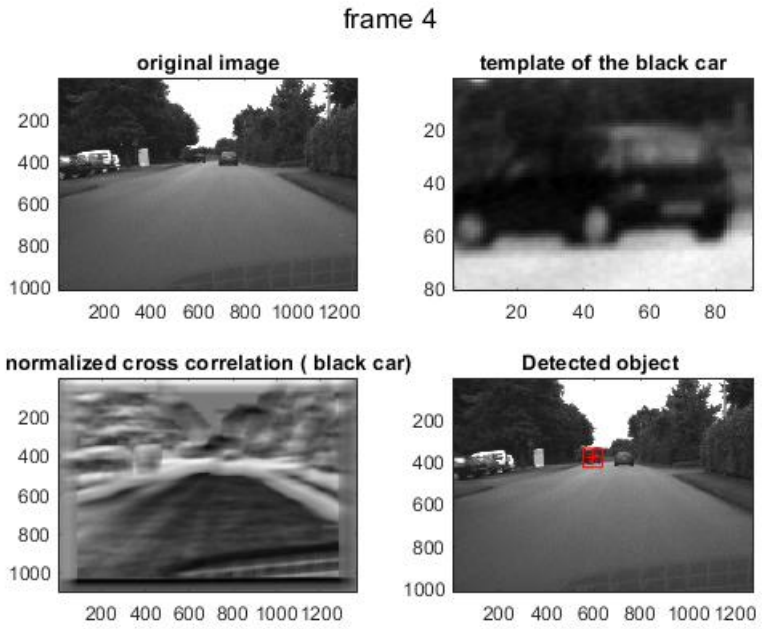

Figure 3.4

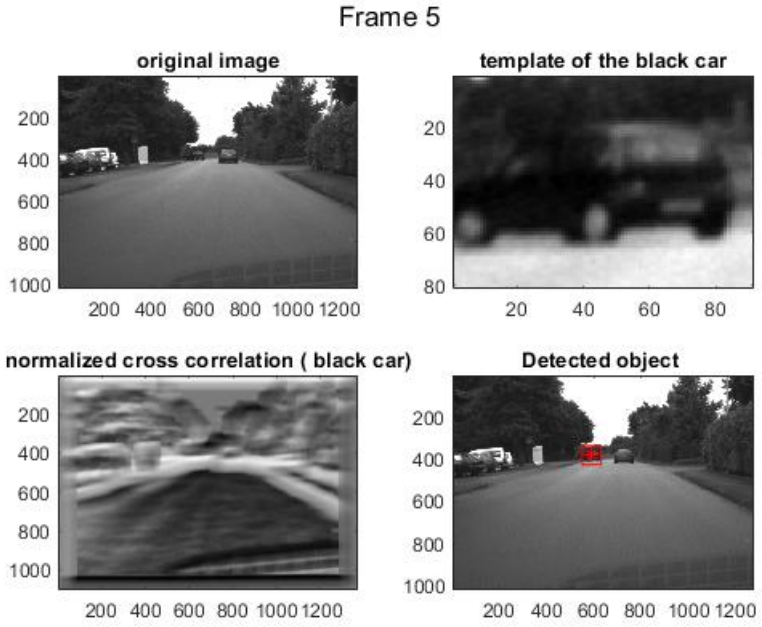

Figure 3.5

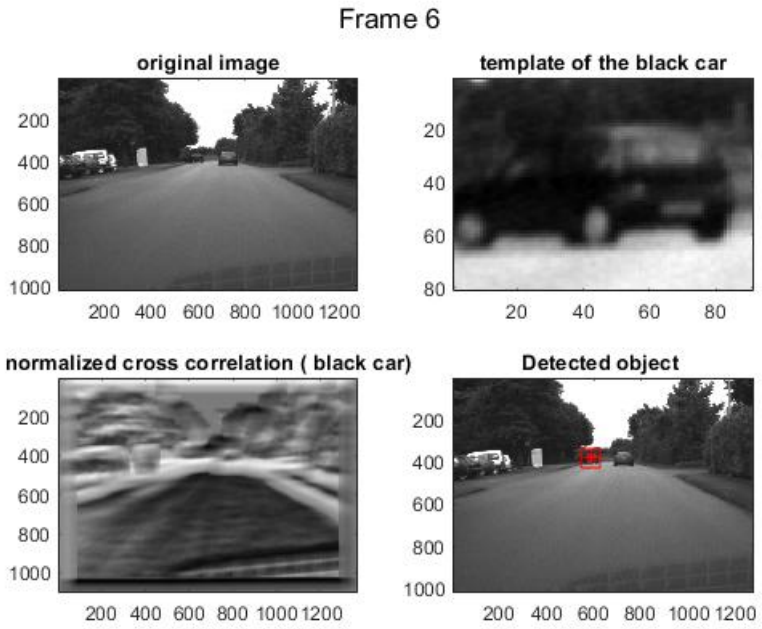

Figure 3.6

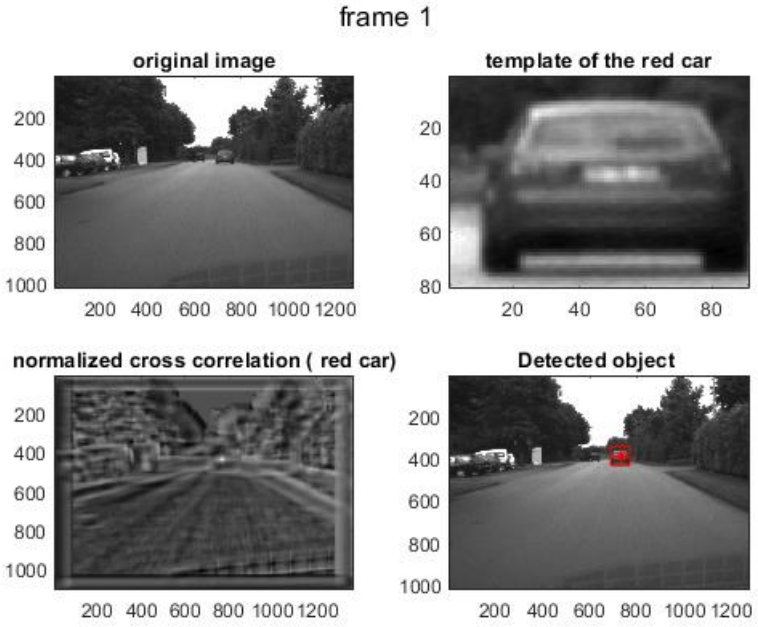

Figure 3.7

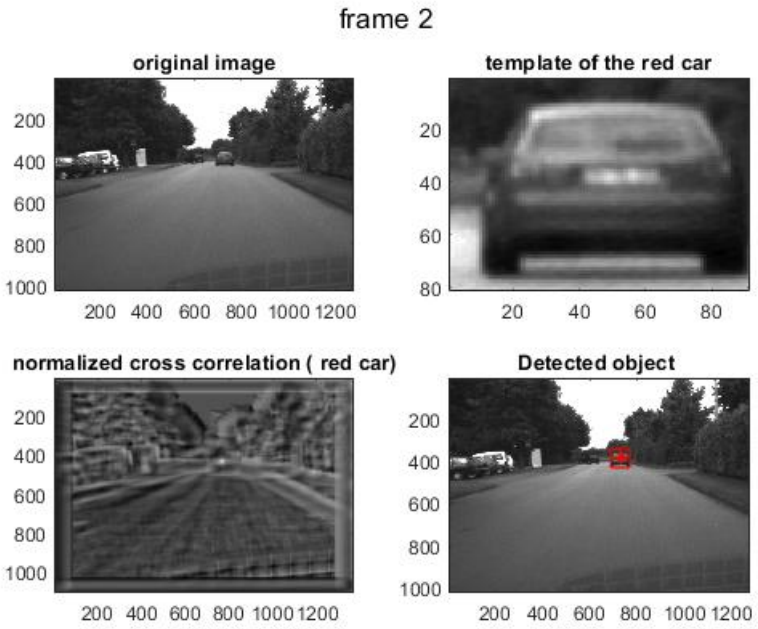

Figure 3.8

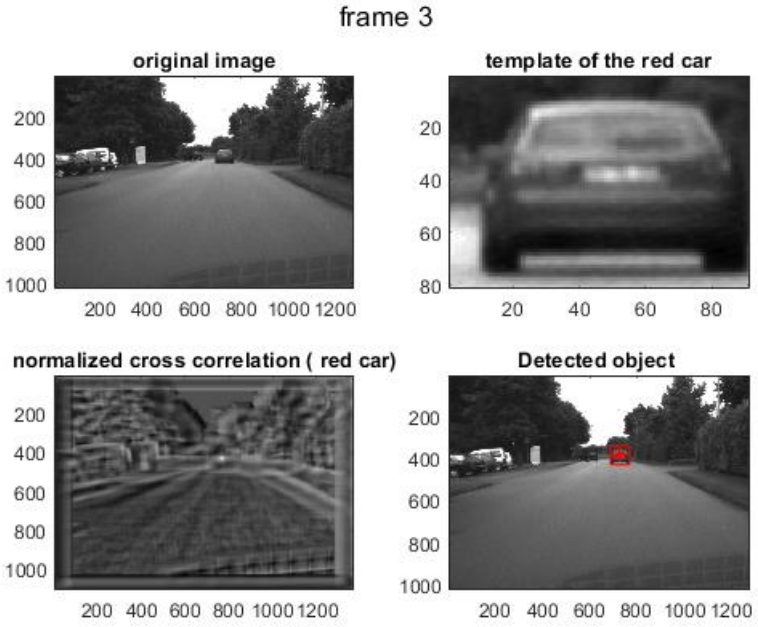

Figure 3.9

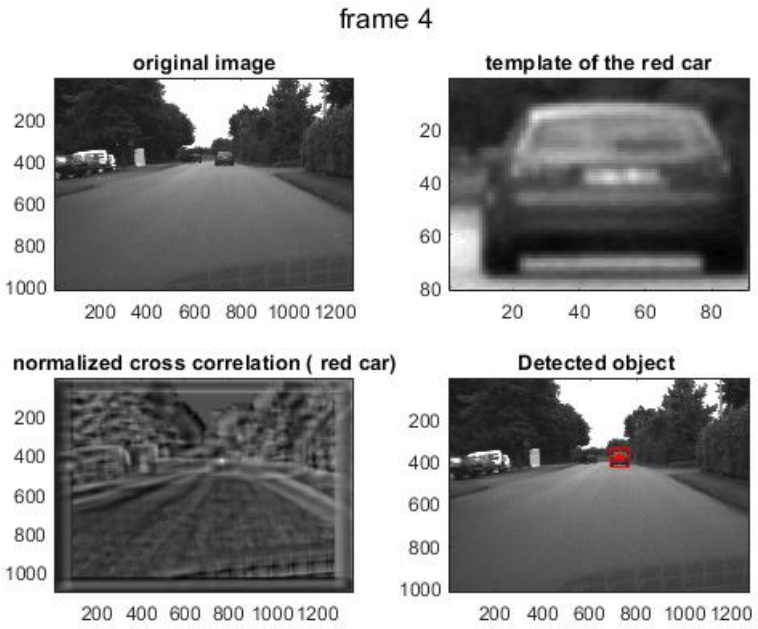

Figure 3.10

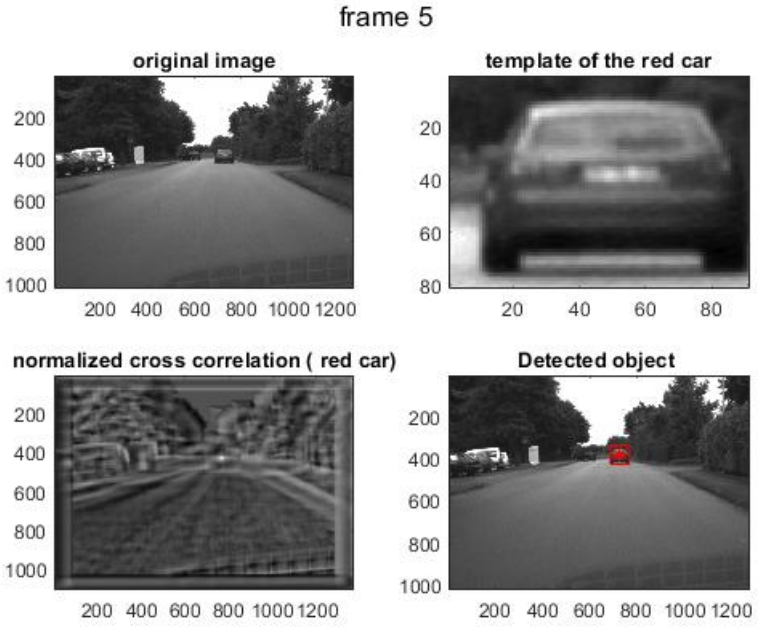

Figure 3.11

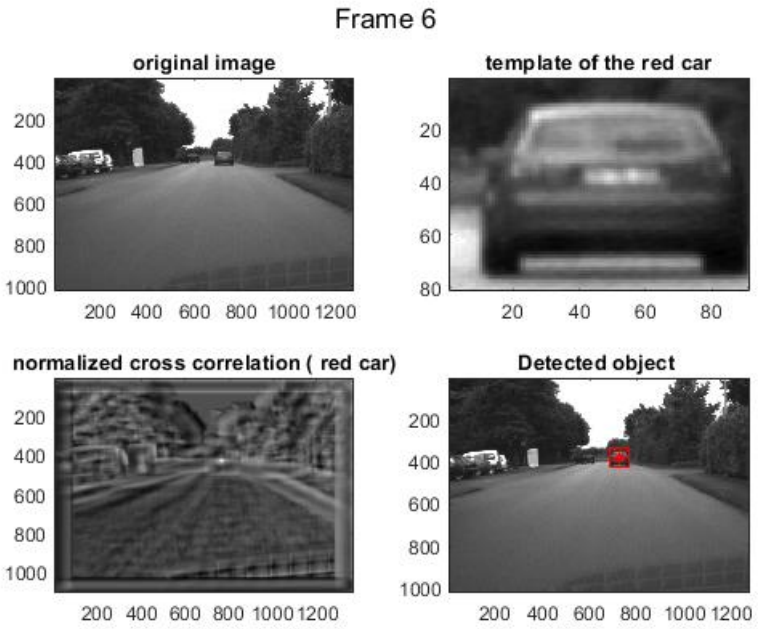

Figure 3.12

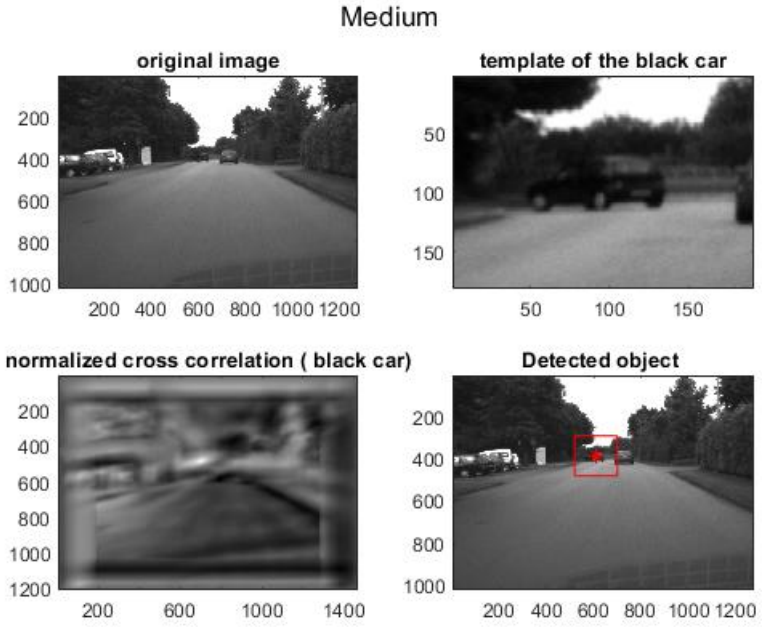

Figure 3.13

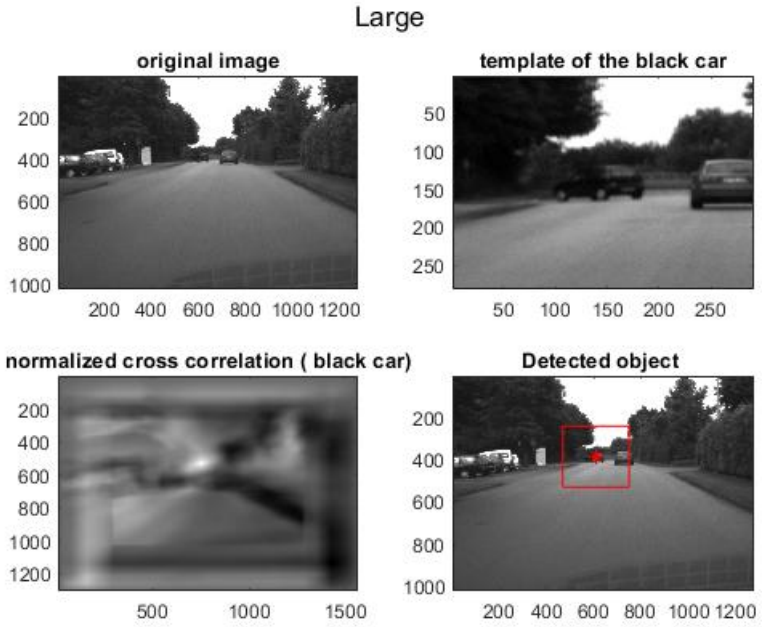

Figure 3.14

3.1.2 compTime

This function takes as input the original image (img) and the template (T). It is obtained the computation of the execution time of the function NCC, passing as input the original image, the template T and the boolean variable notShow initialized to false in order to not show the results.
In the function is done a cycle of 10 and the result is averaged in order to have more precision in computing the time.

3.1.3 accuracyDet

This function takes as input the original image (img) and the template (T). It computes the position of the highest score mX and mY using the function NCC, passing as input the original image, the template T and the boolean variable notShow initialized to false in order to not show the results.
Once it got mX and mY, the function prints them with the size of the template T, that will be then used to make a comparison.

3.1.4 Comparison of the results

Concerning computational times for different template sizes the expected results are that the larger the size, the longer it takes to calculate the result. This is caused by the higher number of operations required for the cross-correlation and also, to find the maximum of the result of cross-correlation, more pixels are checked.
Concerning the accuracy of the detection, we have noticed that increasing the size of the window, centered with respect to the black car, the coordinates of the center don't change.
Hereafter the results are shown:

- The computational time for a template 91x81 is 1.425000e+00
- The computational time for a template 191x181 is 1.284375e+00
- The computational time for a template 291x281 is 1.843750e+00
- The pixel found by a window 91x81 is [609.5,384.5]
- The pixel found by a window 191x181 is [609.5,384.5]
- The pixel found by a window 291x281 is [609.5,384.5]

3.1.5 Comparison with color-based segmentation

By using the NCC-based segmentation for object detection only one object is detected, instead, using the color-based segmentation the result is composed by all the objects in the image with a similar value of hue.

Therefore to detect a particular object, a specific environment is needed (without other objects with similar color and geometric shape in the background).
The color-based segmentation has revealed to be faster than the NCC-based one.
Below the results of the two techniques applied to the same window of pixels:

- The computational time for a template 21x11 (NCC-based) is 9.031250e-01
- The computational time for a template 21x11 (color-based) is 3.156250e-01

3.2 main_i235

In the second main function we have loaded the image 'i235.png' and plotted it.
The first part of this main is related to the computation of the derivatives used for the Harris Detector. To perform this step, we have firstly written the two matrices dx and dy and then, using the MATLAB function **conv2**, we have computed the partial derivatives of the original image I with respect to x and y, obtaining I_x and I_y, as it is shown in Figure 3.15 and Figure 3.16 below.

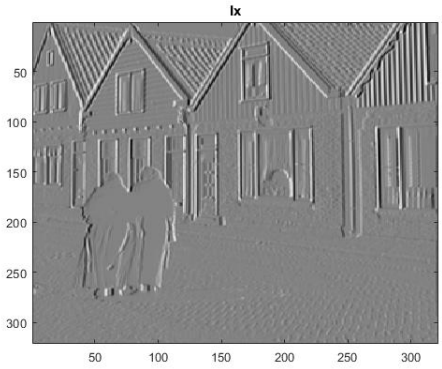

Figure 3.15: Partial derivative with respect to x

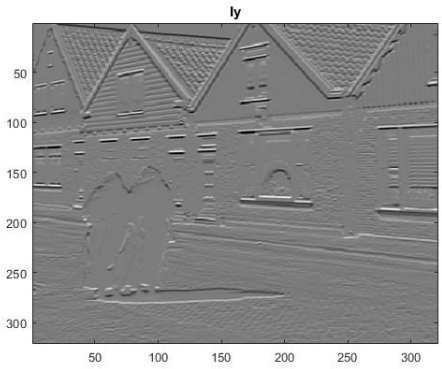

Figure 3.16: Partial derivative with respect to y

After that, we have computed I_x^2, I_y^2, $I_x I_y$ and their convolution with the Gaussian filter (Sx2, Sy2, Sxy). The Gaussian filter is shown in the Figure 3.17 below.

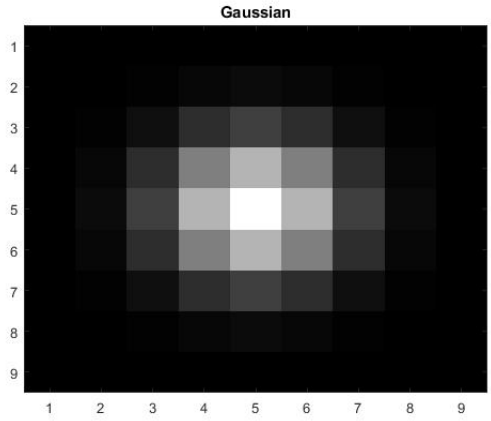

Figure 3.17: Gaussian Filter

The second part deals with the detection of edge regions, corner regions and flat regions. This is done with the second moment matrix M computed from image derivatives. We have used a double for loop to define the M matrix for every single pixel.
Using the determinant and the trace of the M matrix (obtained with the MATLAB functions **det** and **trace**) we have computed the value R for a single pixel and then we have put it into the matrix R_map.
From this value R, putting a certain threshold, we have selected our regions of interest: edge regions, corner regions and flat regions. In our particular case, the interest was in founding the corner regions, so we have applied a threshold equal to $0.3M$, where M was the maximum value of R_map. In the end of this part the R_map and all the three regions have been plotted, as it is shown from Figure 3.18 to Figure 3.21 below.

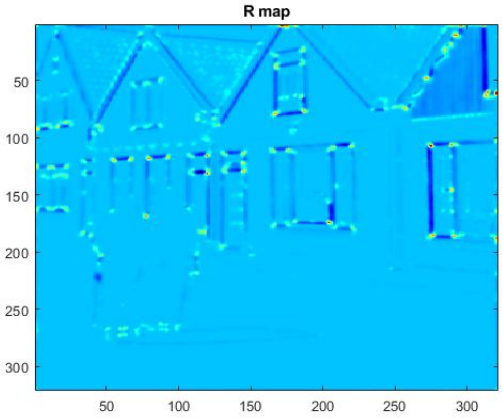

Figure 3.18

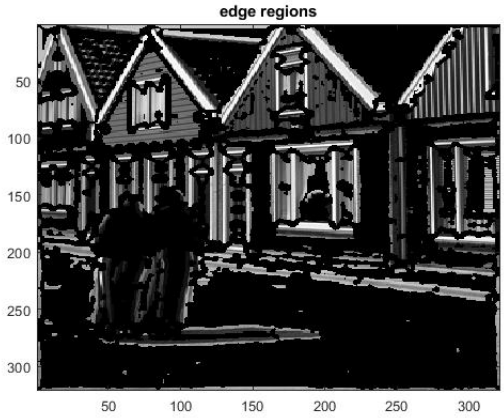

Figure 3.19

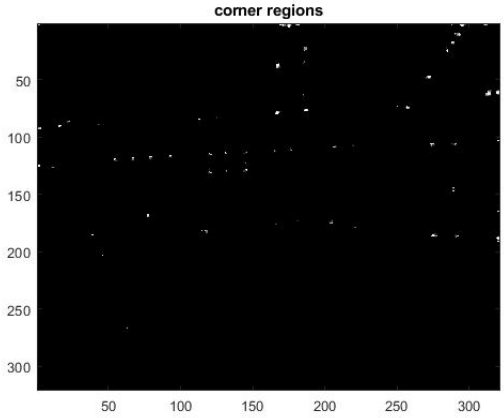

Figure 3.20

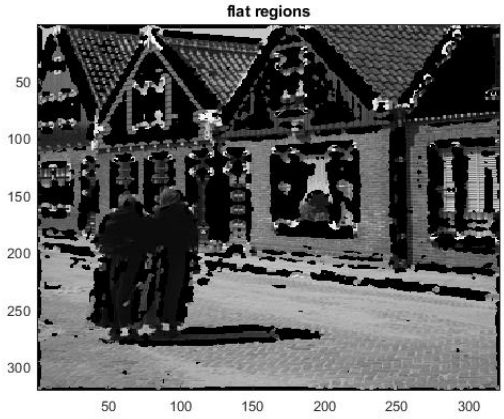

Figure 3.21

The last part is the one related to the detection of the corners in the original image. Since all the corners in the corner regions are blobs, we have used the function **largestBlob** (explained in subsection 3.2.1) to select all the 65 blobs present in the segmented image. Then with the MATLAB function **regionprops** we have extract their coordinates.
Once obtained them we have plotted the original image with all the corners detected. The result is shown in the following Figure 3.22.

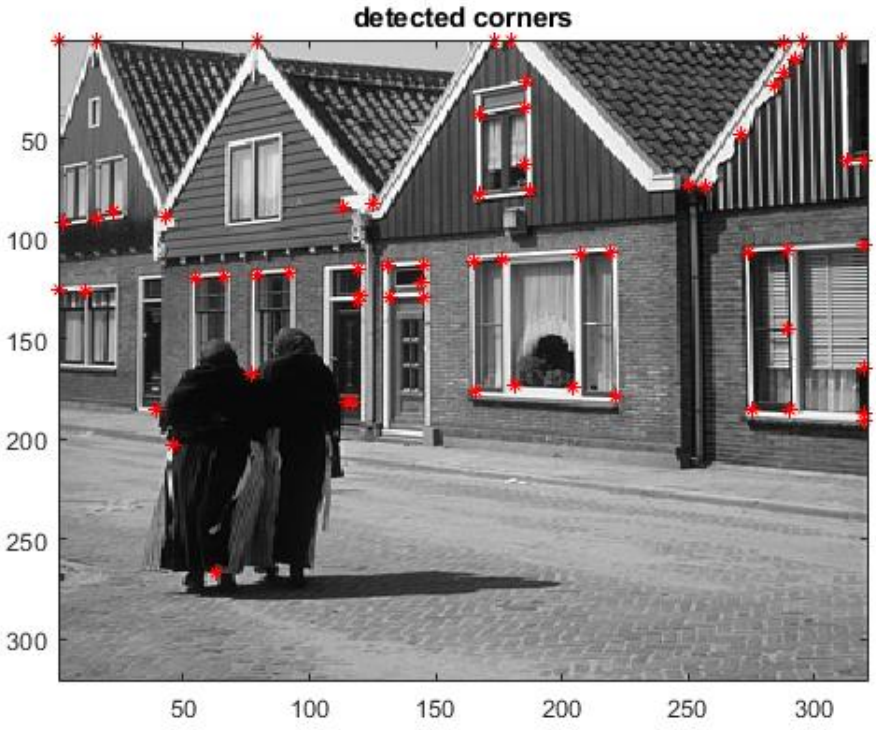

Figure 3.22

3.2.1 largestBlob

This function takes as input the segmented image and the number of blobs that we want to extract (numberToExtract) and returns as output the biggest blob (BinaryImage).
For the implementation we have applied the MATLAB function **regionprops** to the segmented image in order to obtain information about the areas of the blobs.
Then we've ordered all the areas in a descendent order using the MATLAB function **sort**.
Finally, according to the number of blobs to obtain, the function selects the first n-blobs.