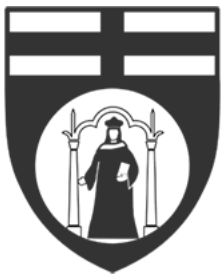


Università degli studi di Genova

Computer Vision

Assignment 4: Color-based segmentation and Blob detection .

Aurora Bertino, Chiara Saporetti

April 16,2020

Introduction

The different perception of colors is caused by the human vision system responding differently to different wavelengths of light. Reflected color is the result of interaction of light source spectrum with surface reflectance. Color spaces (color model) are normally invented for practical reasons and are very useful in image processing:

- *RGB model, which is related to color monitors.*
- *CMY model, which is related to color printers*
- *HSI (HSV) model that corresponds closely with the way humans describe and interpret color.*

In the lab we deal with RGB and HSV models, they are mostly used in Computer Vision applications. The RGB color space is a linear color space infact it's described as a unit cube, usually called the RGB cube, whose edges represent the R, G, and B. But the coordinates of this space may not necessarily encode properties that are important in applications. For this reason, a non-linear transformation to the RGB space is applied starting from the concepts of Hue, Saturation and Intensity (properties of the colors). Respectively, Hue is the property of a color that varies in passing from red to green; Saturation, the property of a color that varies in passing from red to pink; Intensity the property that varies in passing from black to white. In the Lab we load six images and we split them in their different components RGB and HSV in order to notice the different colors that prevail in the images. Then we enter in the field of Objects' description through the color-based segmentation (complementary to the edge detection). The concept is to detect the objects through an image's segmentation into several homogeneous region. Infact, image regions are expected to have homogenous characteristics (e.g. intensity, color) that are different in each region. In the Lab we segment different objects in all the provided images. This computation starts with the segmentation of the image based on color, it's required to represente the color in the hue channel although it is more natural to think of the HSV/HSI space. Subsequently, we classify each pixel in a given image as having a color in the specified range or not and we code these two sets of pixels in an image with black and white produces a binary segmented image. As second step, we recognize the geometrical properties of regions (blob analysis) that are a black car and a red car presented in the middle of the scene. We assume that the blob is represented by a M x N binary sub-image. After that, the last requests are to compute a centroid (center of mass) and the perimeter, which in a region is the sum of its border pixels. A pixel which has at least one pixel in its neighborhood from the background is called a border pixel (Bounding box).

0.1 Matlab Resolution

0.2 Color-based segmentation

Firstly, we load the 6 images and we show them in grayscale, using the function rgb2gray(). As second step, we split them in the three RGB channels and in the three HSV channels through the rgb2hsv() function. Then, we

select in the image the area corresponding to the dark car that turns on the left using the following function:

$$[outputImg3\_black, med\_black, std\_black] = Select\_area\_black(Img3) \tag{1}$$

This function is described by these steps:

- *We take the Hue component.*
- *Given the dark's car position, we select the area using the function rectangle: ('Position',[555,360,90,60],'EdgeColor', 'r').*
- *Considering only the selected area, we compute the average with the function mean2() and the standard deviation through std2().*

Subsequently, we segment the dark car in the 6 images by thresholding the Hue component (e.g. in the range between m-s and m+s). The computation is described by the following function:

$$[outputImg] = Segment\_dark\_car(inputImg, m, s) \tag{2}$$

- *As the previous algorithm, we take the Hue component.*
- *We build a zero matrix with the same dimensions of the image.*
- *We create a mask to compute the thresholding, only considering the pixel included in the range between (m-s) and (m+s).*

Then, we display the images corresponding to the segmentation, and the related centroid and bounding box. We compute the following function:

$$[outputImg] = Segment\_with\_centroid\_black\_car(inputImg, m, s) \tag{3}$$

This functions is similar to the previous one, we implement it adding the following steps:

- *We use the function BW2 = bwareaopen(seg,550) in order to deal with a labeled image. In this way, we remove small objects from the image that have less than 550 pixels.*
- *We compute the bounding box using the function rectangle () and we plot the centroid.*

At this point, we repeat the steps 2-5 for the red car on the street using the same functions.
The following function:

$$[outputImg, m, s] = Select\_area\_red(inputImg) \tag{4}$$

change the values in [690,360,80,65] to select the area with the function rectangle(). Then, we change the values of thresholding using an hue range >0.97 and <1. As extreme higher we round to 1 beacuse we are dealing with a binary image. Lastly, as the previous step, we display the images with the bounding box and the centroid.

0.3 Blob Detection - Sunflower find

This code was used to find the two desired sunflowers circumference centers in the most precise way. It uses the blobDetection function, which will be described in another paragraph of our report, and prints the resulting circles on the input image, by drawing in red the little flowers and in blue the bigger ones. Since the desired flowers were two of the bigger ones, we also write the coordinates of the center of the flower next to it, in order to see and save them for the next steps. In the end, the code shows the image with the two sunflowers framed by a yellow rectangle.

0.4 Blob Detection - Sunflowers

This is the main code for the blob detection part of the laboratory. It uses the function responseOfSingleFlower, which will be described in another paragraph of our report, and then plots the Laplacian response by using its maxima values computed by the function and the sigma values. It then shows the flower by drawing a circle on it with the characteristic scale. It repeats the steps for the second flower.

0.5 Blob Detection - responseOfSingleFlower

$$[radii, maxVal, characteristicScale] = responseOfSingleFlower(flower, sigma, n, sigma_incr) \quad (5)$$

This function is used to find the Laplacian response of a sunflower and its characteristic scale. It has, as input, the flower image, the number of scales desired, the first value of sigma and the sigma incrementation. For each sigma:

- The filter size is set to the flower image size, so that the cross correlation happens once, since the two are equally dimensioned.
- The Laplacian of gaussian is computed with the function fspecial and is normalized.
- The image is filtered.
- The resulting surface is plotted.
- The maximum value of each surface is saved.
- The sigma is saved in a vector and then incremented.

Then it's computed the sigma that gives the maximum response. The function sends in output the set of sigmas used, the maximum value of each filtered image and the characteristic scale of the sunflower.

0.6 BlobDetection

$$[row, col, val] = blobDetection(img, sigma, n, sigma_incr) \quad (6)$$

This function finds the circles inside the input image. It receives, as input, the entire image, the number of scales desired, the first value of sigma and the sigma incrementation. It has an initial structure which is very similar to the previous function's one. For each sigma:

- *The filter size is set to 6 times sigma + 1.*
- *The Laplacian of gaussian is computed with the function fspecial and is normalized.*
- *The resulting surface is saved in a 3d matrix: the final result will be a set of 2d matrices at a fixed scale.*
- *The sigma is saved in a vector and then incremented.*

Then the non-maxima suppression is done for every level of the scale space and the center of each blob is saved in a 2d matrix. The function sends in output the row and column coordinates at which the maxima of the responses are and the maxima values.

Results

0.7 Results for sunflower find and Sunflower

We tested our blob detection code on the "sunflower.jpg" image. First we found the desired sunflowers by plotting the circles on the image and the coordinates of the center of the circle next to the bigger sunflowers. We also showed the image with the two sunflowers framed by a yellow rectangle. In order to find the sunflower we used the following values:
sigma=2; n=10;
sigma_ incrementation=1.5;
Which differ from the given ones for the sigma value. By using sigma=1, in fact, we couldn't detect the first sunflower. We saved the results (namely, the coordinates of our flowers), and used them in the main code. We can see these results in the figure 1.
*Subsequently, we can see the results for Sunflower, infact by using an incrementation of sigma of *1.5, the surface representing the laplacian response varies a lot and is not very precise. However, by using sigmas in a restricted range around the characteristic scale (which means by incrementing the sigma of +0.5 at each round) the Laplacian responses are precise and have a gaussian shape. In the code we left the given incrementation, but below are represented also the results around the characteristic scales. In this report we show just half of the results, since by showing more some details are obviously lost. In the code, however, all 10 results are shown. After plotting the surfaces, we also plotted the maximum response with reference to the scale. In the pictures is indicated with an "x" the maximum height to the curve, and in the plot below there is the detail of the plot around the characteristic scale.*
In the last images there is the complete image again, with the circle drawn using the value of the characteristic scale. We can see these results in Figure 2,3,4,5.

0.8 Results for Color-Based Segmentation

We plotted all the images in grayscale and then we split them in all the RGB and HSV components. From the comparization between RGB and HSV channel of each image, we can notice that the components are very similar for both models. All the images seem to be frames of the same video, so the scene does not change a lot (Figure 6). Subsequently, we select through a rectangle the red car on the right and the dark car on the left in the Hue channel of the image "ur_ c_ s_ 03a_ 01_ L_ 0376.png" (Figure 7). Then we computed a segmentation by thresholding of the dark car in all the provided images and we display the result in Figure 8. The segmentation works putting to 1 the pixel of the segmented car and to 0 the other . In the Figure 8 is also present the red car segmentation by thresholding, we can notice that in this image the segmentation is better and clearer than the other image. The segmentation of the red car is better than the other because the range of the thresholding is smaller, so the result is more precise. Also in this case we showed the results only for the image "ur_ c_ s_ 03a_ 01_ L_ 0376.png". In the Figure 9, we display the result only for the image "ur_ c_ s_ 03a_ 01_ L_ 0376.png" because between all the six images there is not much difference. We display a centroid and a bounding box that are overlapped on the red and dark car.

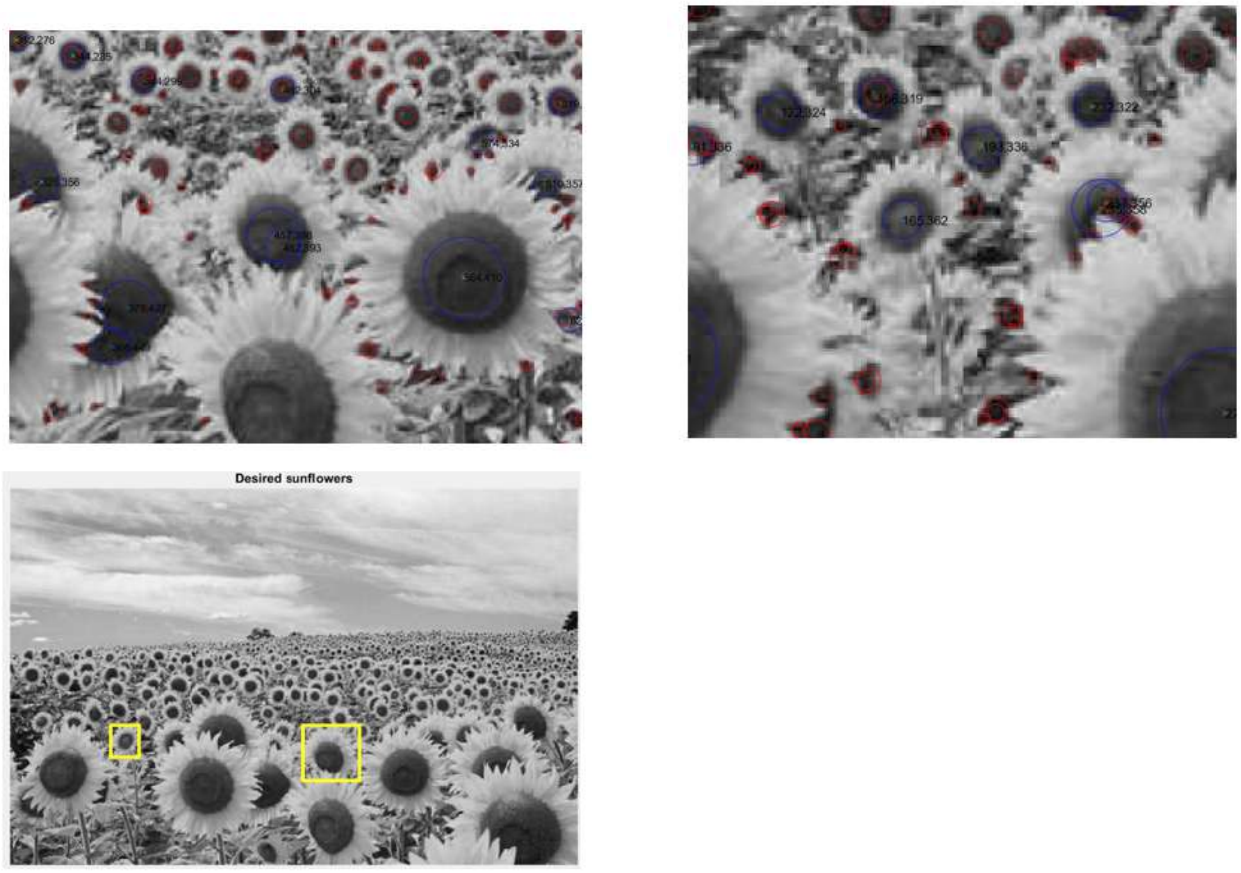


Figure 1

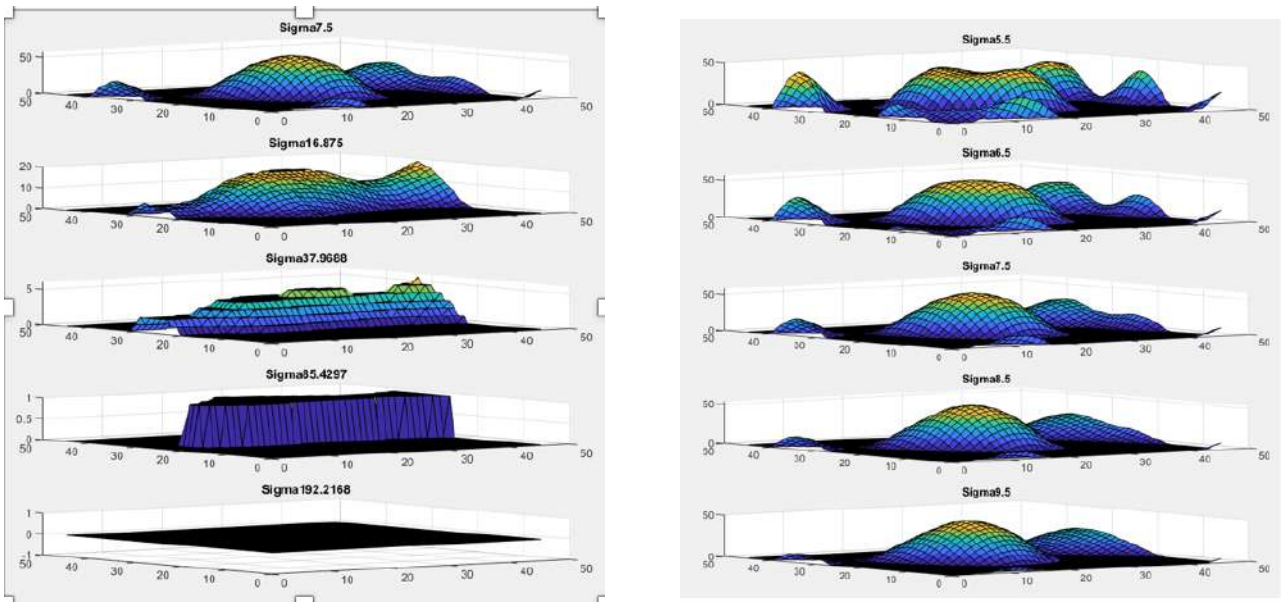


Figure 2

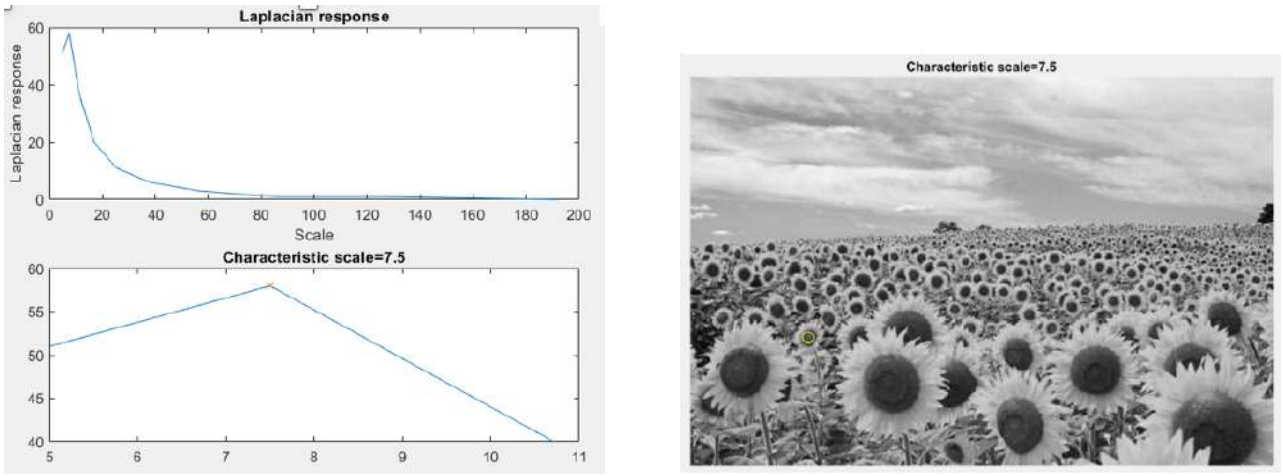
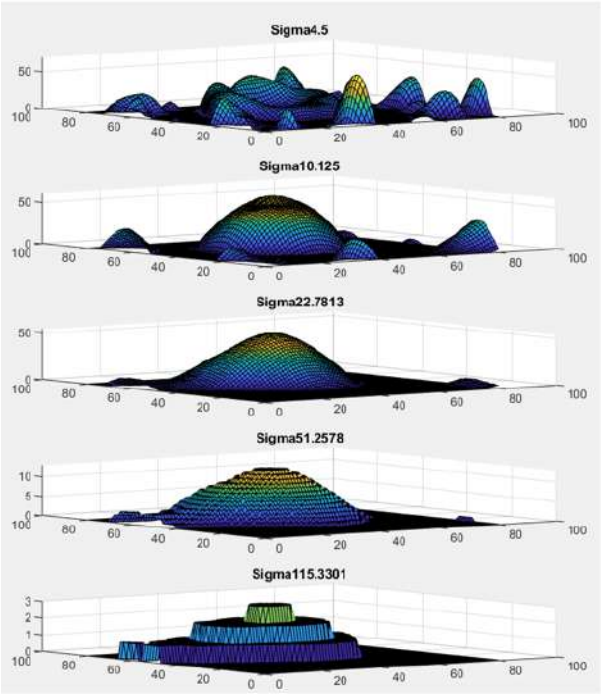


Figure 3



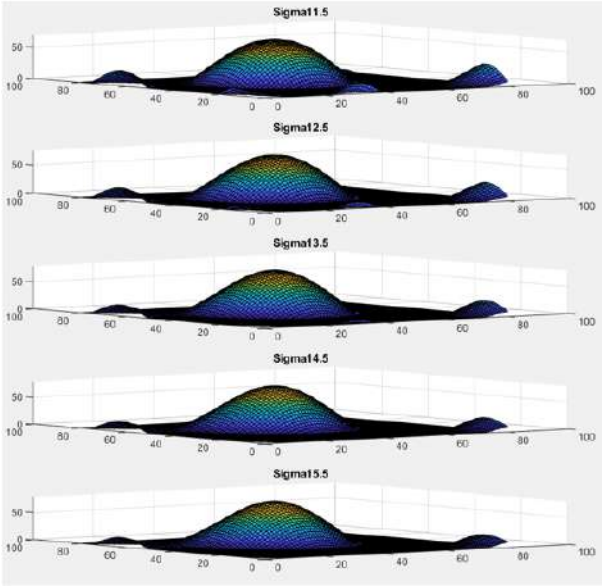


Figure 4

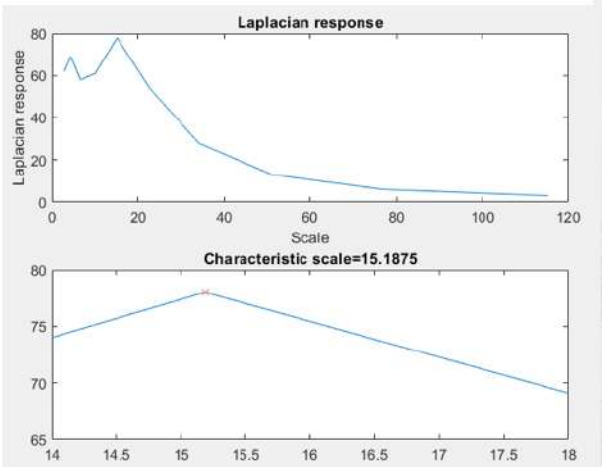


Figure 5

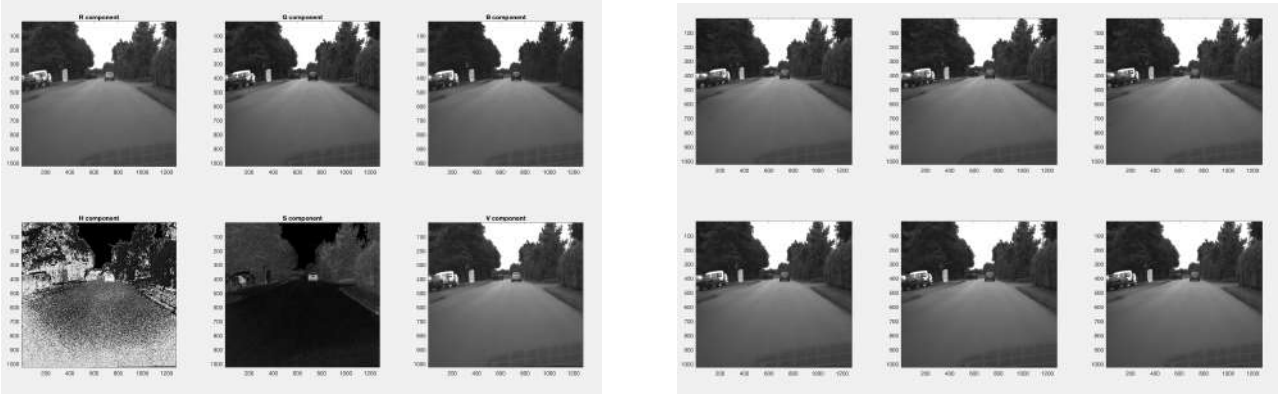


Figure 6

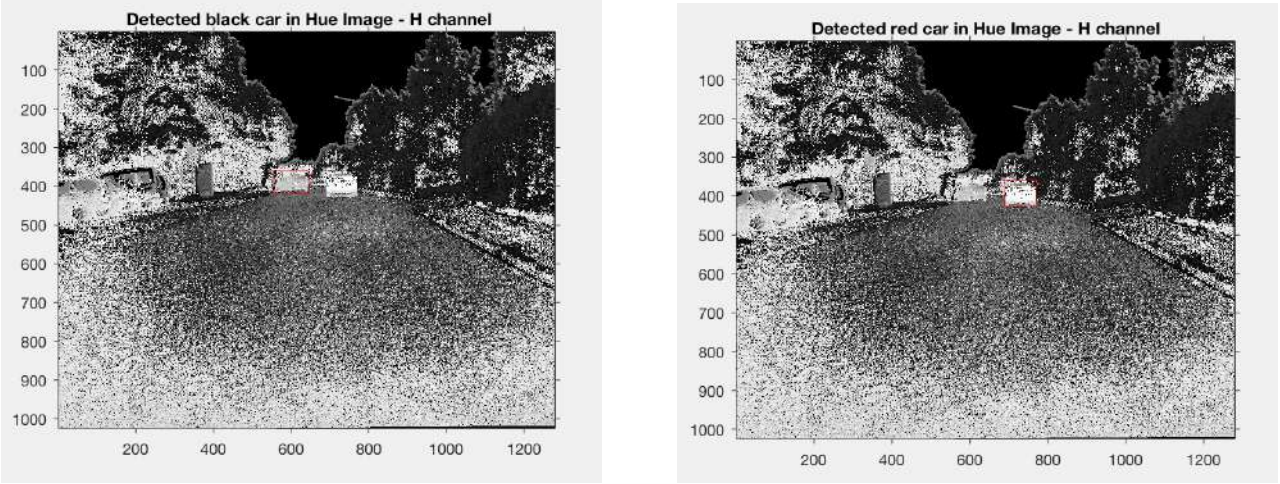
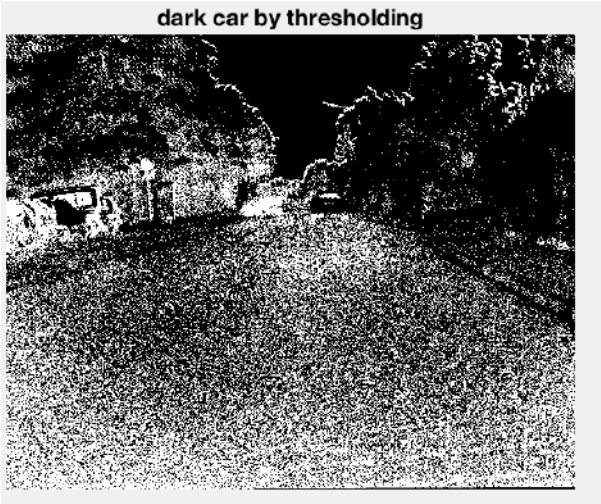


Figure 7



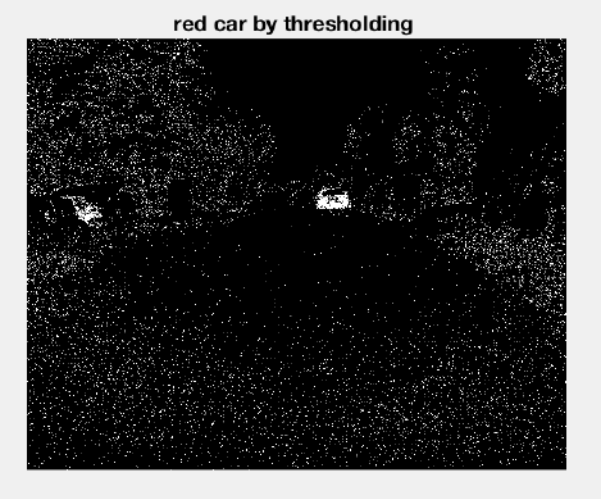
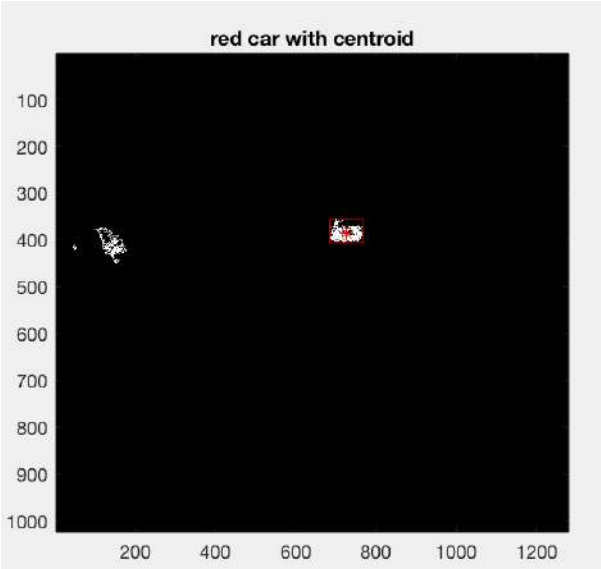


Figure 8



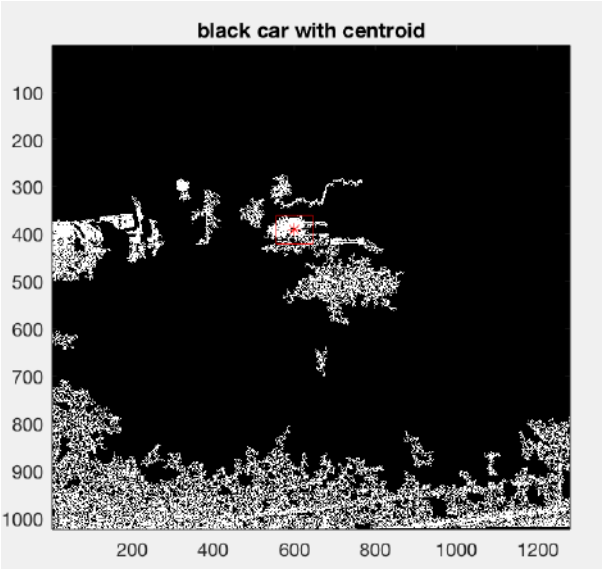


Figure 9