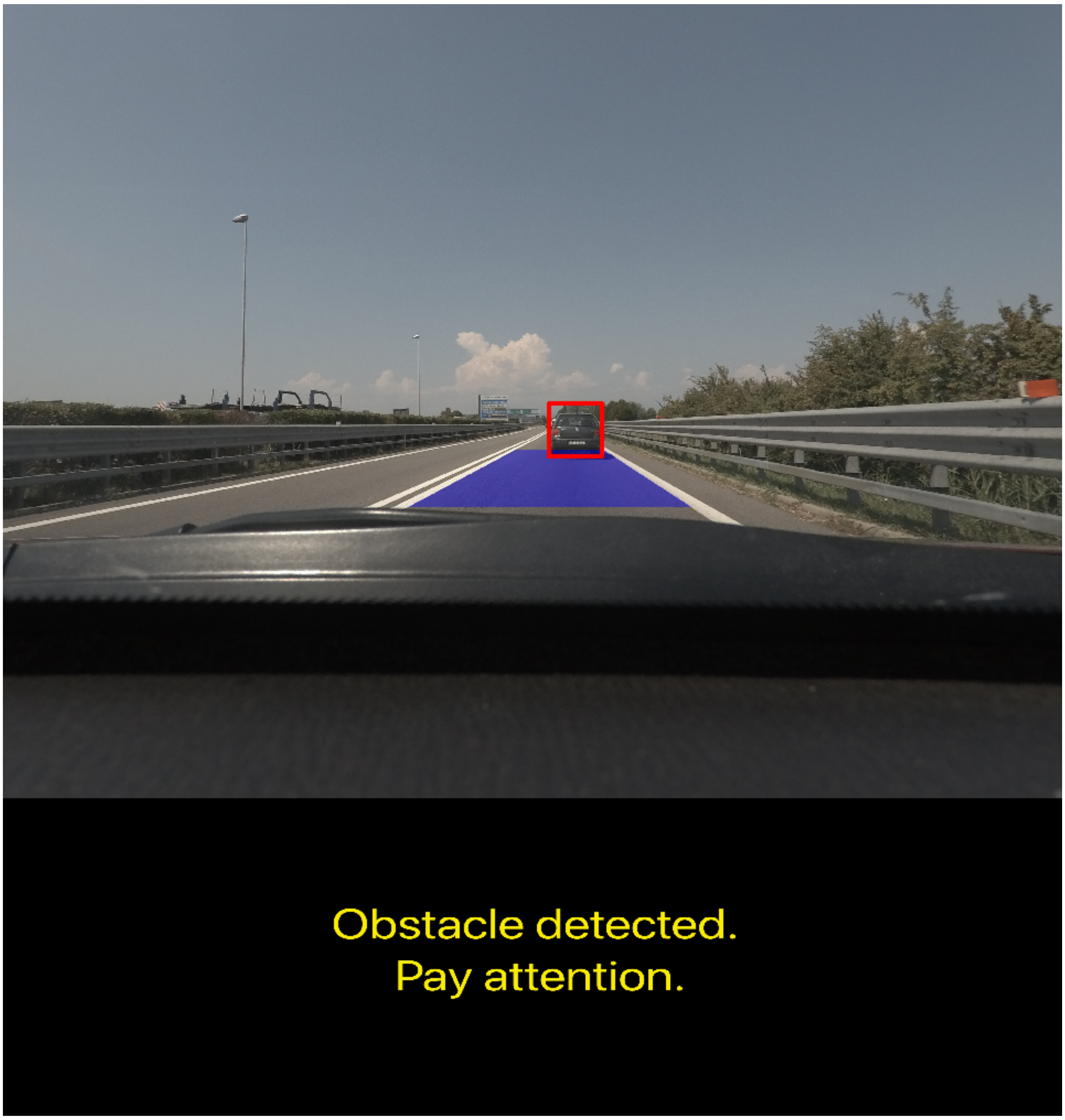
Autonomous driving: Automatic Breaking System

Computer Vision

Alberto Padoan - 16 settembre 2018



Introduction

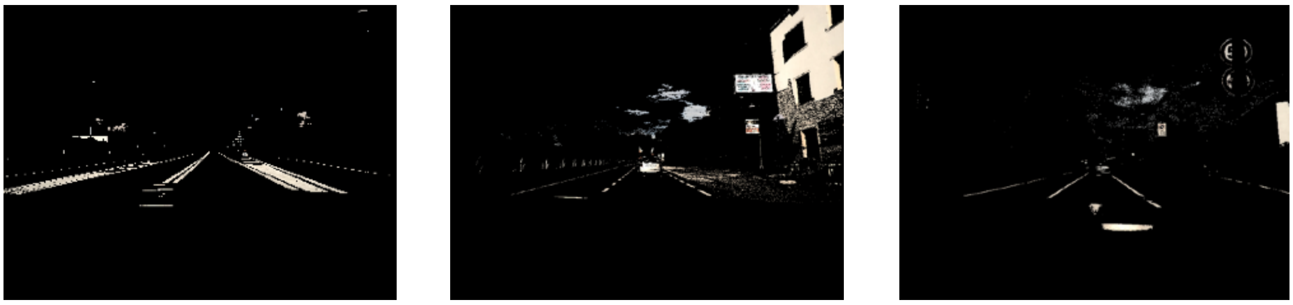
The aim of this project is to develop a computer vision application for autonomous driving. Fist of all, given some pictures taken from a car, the application must be able to detect the lines of the road and recognise the lane that the car is using. Then, if there is a car along the road, the application must be able to detect it and to trigger an alarm if the car is too close.

I assumed that in an autonomous driving system the camera is placed in a fixed and well known place and the application is built on these characteristics. So all the photos used in this application are taken by me always from the same same place of the same car using a GoPro Hero 5 Black with the following settings:

GoPro Hero 5 Black	
FOV	Linear
WB	Native
ISO min/max	100/200
Shutter	Auto
Sharpness	High
Image dimensions	4000x3000

Lane Detection

The application must be able to detect the lane lines. First of all, since all of the lines are white or yellow (in case of working areas), I have developed a method that selects the pixels of the image that are white or yellow. The white and yellow ranges are computed in the HLS color space. I made this choice after several tests using other color spaces; the one that performs better, especially for the yellow range, is the HLS color space. The range values for white pixels are between (0, 190, 0) and (255, 255, 255), for yellow pixels are between (20, 120, 100) and (40, 200, 255). In order to properly set these values, firstly I selected some from one of the tools that can be found on Internet, then I have adapted the values using some test images.



Images after color filtering.

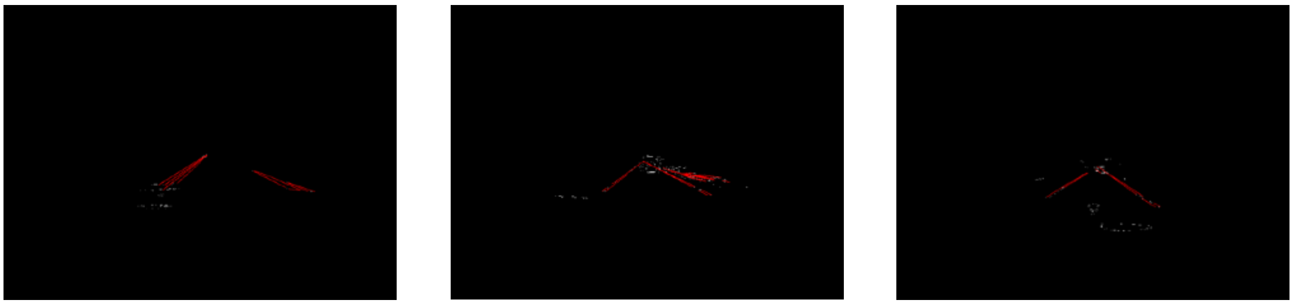
Then, starting from the filtered color image, I have written a method that defines the region of interest in which there are the lines. The region of interest is built as a triangle that covers the area in which there is the road. I have selected the interested area after some tests on the images and it is bigger than the area actually covered by the road in order to surely include it. The number of pixels included in the region of interest are 7'647'921 over 12'000'000; this, combined the color filtering, reduces a lot the possibility to find useless lines.

Images with the region of interest applied.

The image produced by the two methods above is converted in grey color space and blurred with a Gaussian filer with a mask 15x15 and passed to the Canny edge detector with min threshold 60 and max threshold 180. The image produced is passed to the Probabilistic Hough transform. Using the OpenCV documentation and some test images, I have set the method parameters as:

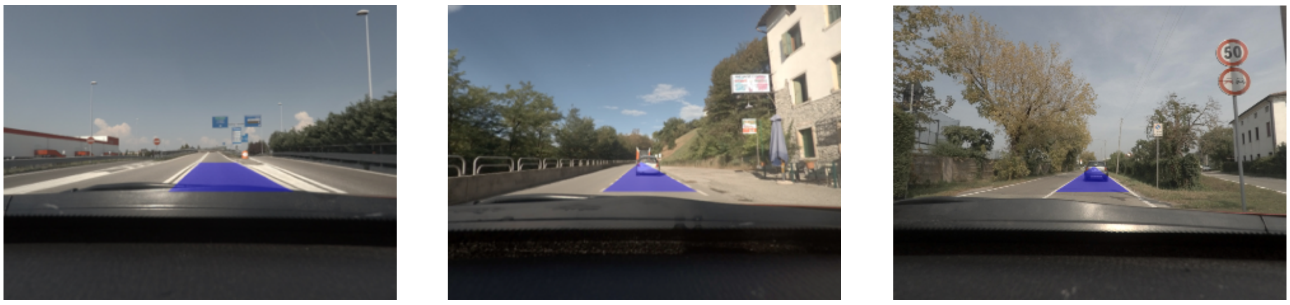
- Rho = 1;
- Theta = CV_PI/180;
- Accumulator threshold = 90;
- Minimum line length = 30;
- Max Line Gap = 50;

The most interesting parameters are the minimum line length and the max line gap. I have set them to the values that permits to find road non-continuos lines since they are short and spaced. Furthermore these values must be quite high in order to reduce the probability to find lines that describe smaller details such as the car in front, if there is.



Images with the lines found.

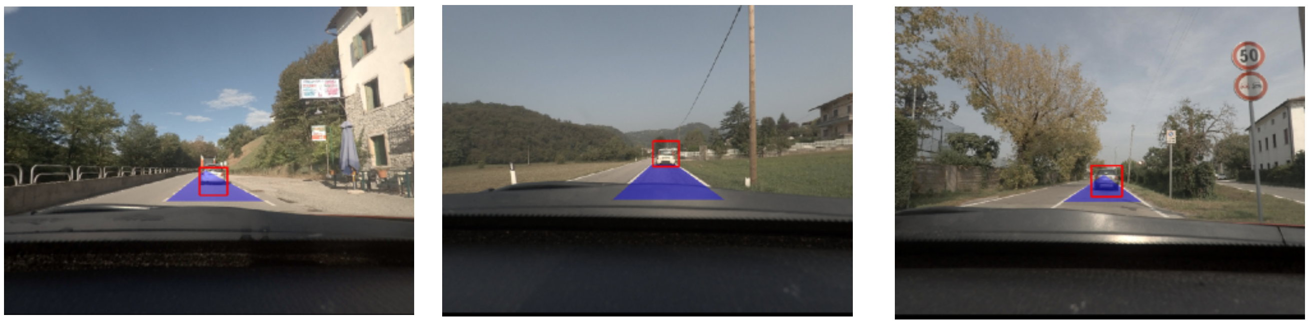
Not all the lines obtained in the previous step are useful to detect the road and they need to be filtered. First of all I have excluded all the vertical and horizontal lines. Afterwards I have analysed the slope coefficients of the remained lines. I have noticed that generally all the left lane lines have a low slope coefficient and the right lane lines have an high one. Furthermore this method permits to avoid to consider the lines that delimit the left side of the road since they have a slope coefficient less that zero but higher that the one that delimits left line of the lane interested. So I have selected a group of left lines that have a slope coefficient similar to the lower one and a group of right lines that have a slope coefficient similar to the higher one. For each group I have computed a mean in order to obtain two lines: the left one and the right one that define the lane limits.



Detected road.

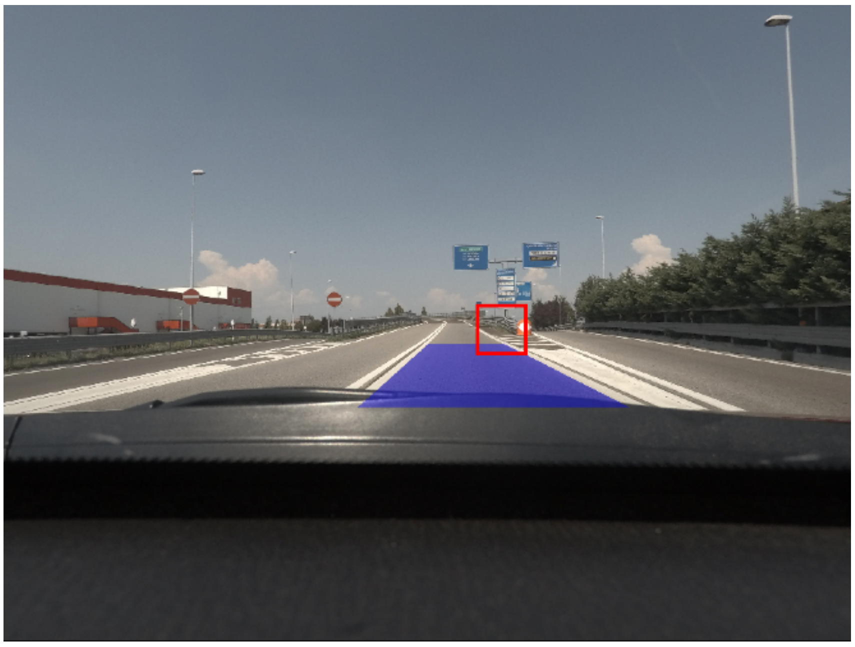
Car detection based on edge density

Once the lane is detected, I have developed a class that permits to detect the car in front, if there is. The main idea of the car detection algorithm I have developed is the edge density. I have noticed that, considering only the lane in which my car is, applying the Canny edge detector there is an high edge density where there is a car. So I passed the image in which there is only the lane interested to the Canny edge detector and then I have built a summed area table on it. Each value in position (x,y) of the summed area table contains the number of pixels that make edges in the rectangle between (0,0) and (x,y). Building a summed area table permitted me to perform faster the following methods since they need a sliding window that moves on the whole image a lot of times. In fact I have written a method that, based on a sliding window, searches the zone with the highest edge density. The edge of the sliding window becomes smaller of 10px at each iteration until it detects a zone with a edge density higher than 0.065 (calculated as $density = \frac{edgePixels}{windowSize}$) or until the edge of the window becomes of 200px an none high-density zone is detected. I have found 0.065 threshold after several tests on my images: this value is a tradeoff between a reliable car detection and reducing as much as possible the detection of some artefacts of the surrounding environment such as guardrails, trees etc.



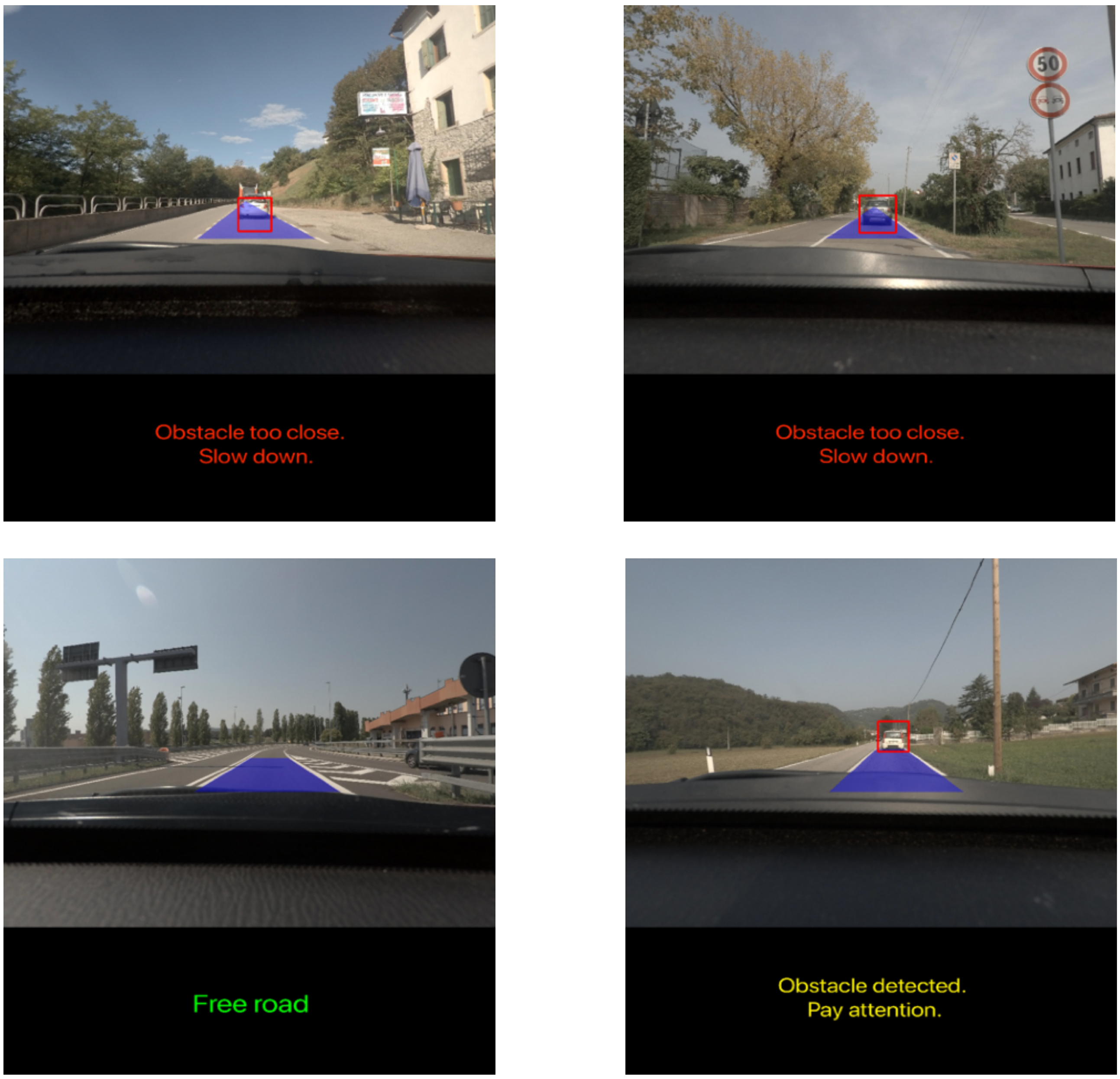
Car detection results.

The image below shows the presence of a false positive. This problem is due to the absence of a car and the presence of a high density region in correspondence of the guardrail near the lane (the window detected has a 230px edge and a 0.067 density). In fact, in the other two images of my dataset without a car and without details near the lane, the sliding window stops with a density of 0.056 and 0.052. This false positive could also be seen as the possibility to detect other obstacles different from cars that can be found on the way.



False positive.

In order to detect if a car is too close, I have applied a threshold to the bottom edge of the window. Considering the height of the lane detected, I fixed the threshold to the 40% of the height. If the edge considered is lower than the threshold, the car is too close and an alarm is triggered. The best the car is detected, the best this method works; if the car is not properly detected due to some artefacts of the surrounding environment, the window does not fit perfectly the shape of the car and the alarm could be triggered even though the car is not too close.



Final results.

Conclusions and evaluations

The application presented works well for lane detection; it is always properly detected except for image test #6 in which it does not correctly detect the depth of the road. The technique of car detection using edge density works fine: there is only one false-positive case (described above) and in some cases (especially for test images #2, #3, #5 and #12) the shape of the car is not properly detected because of the presence of strong edges around the car.

For each image both the lane and car detection are performed on average ~3500ms. Because of the multiples iterations of the sliding window, the car detection is the most expensive step: the more iterations are required by the sliding window, the more the algorithm is time expensive (when the sliding window reaches its minimum size are required more than 4000ms).

One possible future improvement is to implement the detection of other obstacles different from cars. In fact, as seen in the false-positive example, the algorithm detects every high edge density zone and that zone can be seen as an obstacle along the road.
In order to improve the reliability of the alarm, can be studied a relation between the sliding window coordinates, the disposition of edges among the window and the window size. In fact, as explained before, some windows could not express the proper car location due to some environment details. Finally, another improvement is to refine the lane detection by using other filters, by standing out better white pixels and by inspecting local brightness.