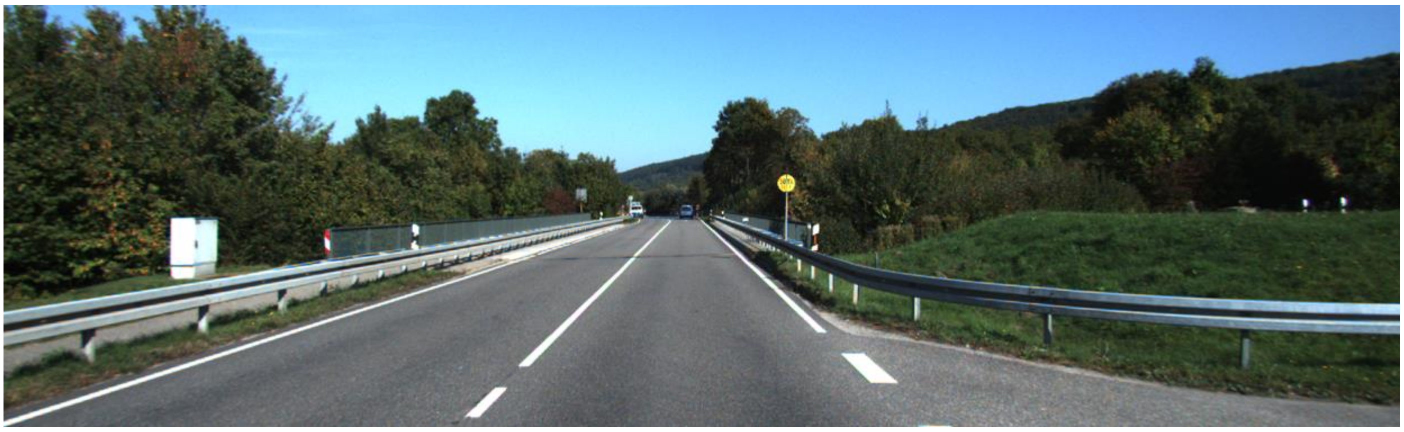
LAB 4

The goal of this lab is to detect the right-most line of the street and the round street signal from the following image.



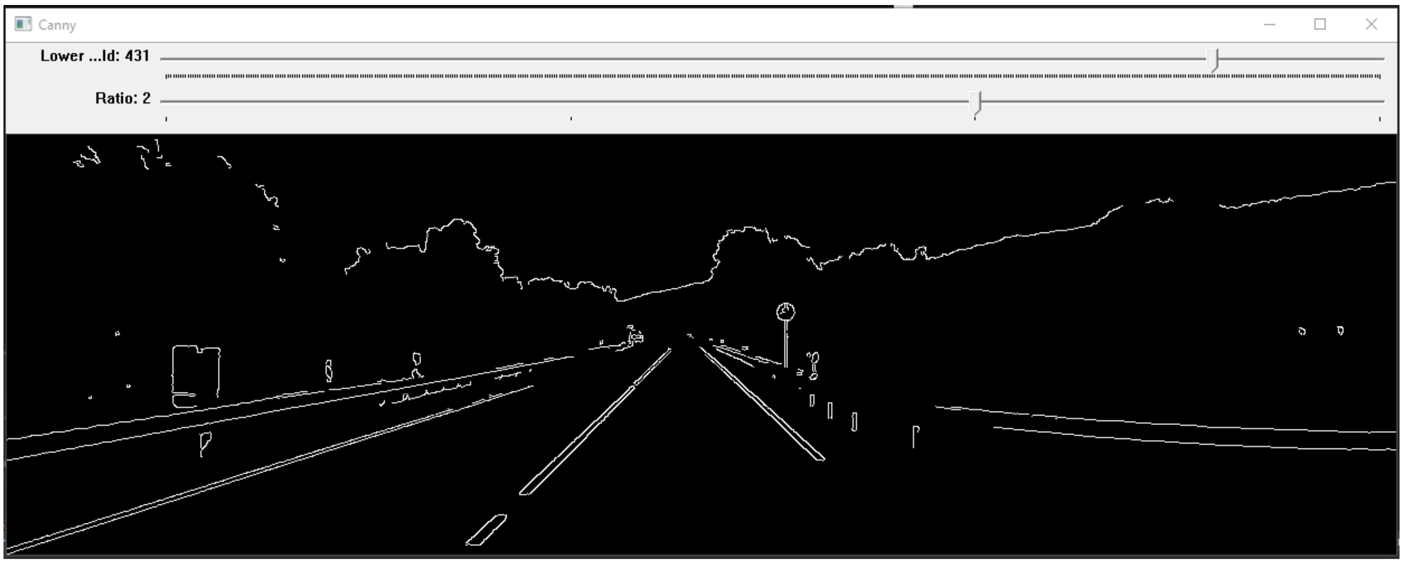
The process adopted is the one suggested in the exercise text pdf:

1. Line detection:
    a. Canny's edge detection algorithm
    b. Hough's Line Transform
2. Circle detection:
    a. Canny's edge detection algorithm
    b. Hough's Line Transform
3. Drawing the detected regions.

For Canny's and Hough's algorithm are used the function provided from the OpenCV library.

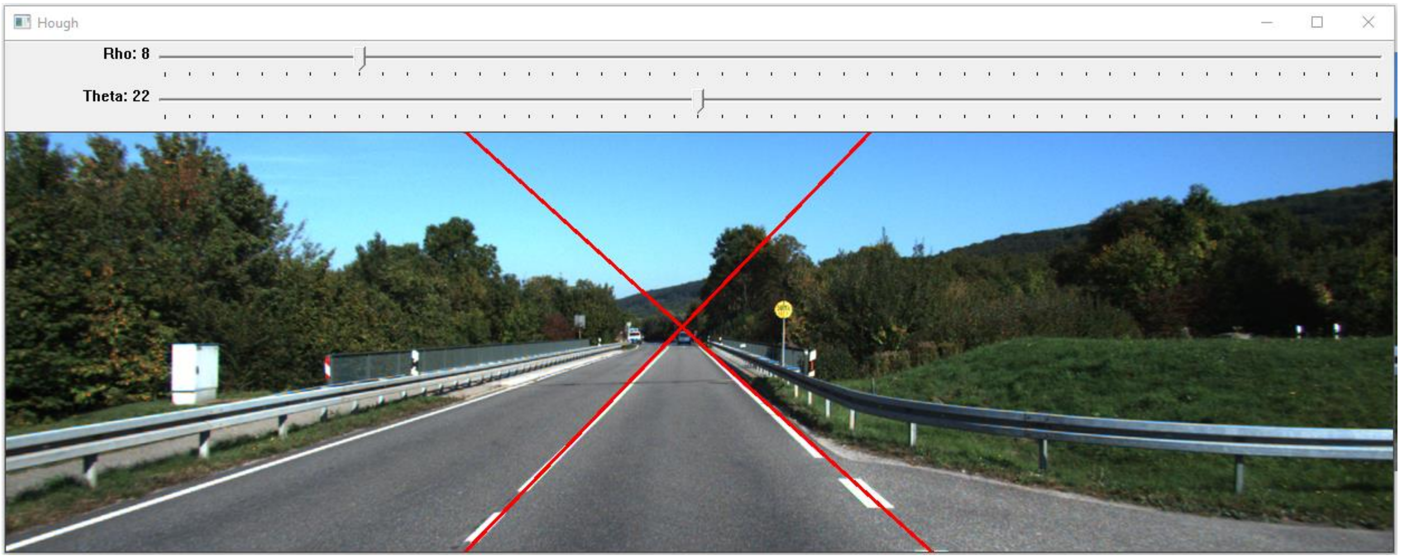
1. Line Detection

As Canny's recommendation, for the hysteresis threshold it can be used a ratio 3:1 or 2:1 with the *upper:lower* thresholds, then the code permits to select the values of the lower threshold and the preferred ratio through 2 trackbars (the value 0 and 1 in the trackbar are considered as a ratio 2:1). By changing values on the trackbars it's possible to visualize in a window the Canny's algorithm effects on the input image.
After some experiments it has been chosen as lower threshold the value 431 and a ratio 2:1.



Lower ...ld: 431
Ratio: 2

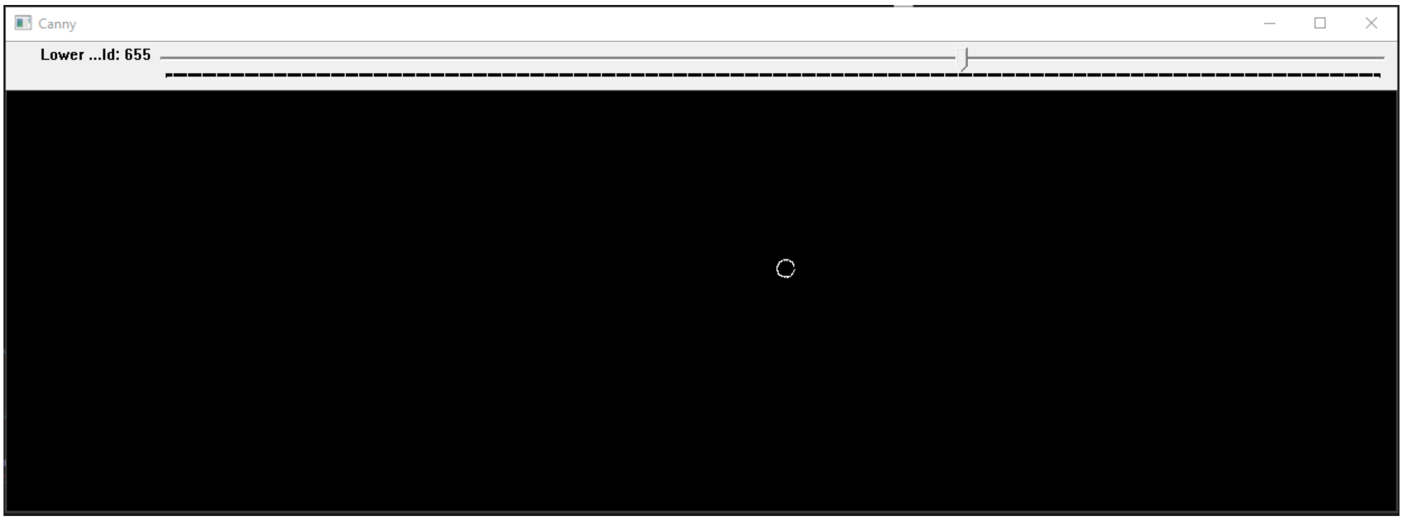
For the Hough transform instead we are considering the 2 strongest lines which are the first 2 lines returned from the OpenCV function. Because we are taking only the 2 strongest lines, the threshold value can be set to 0 and we allow the user to choose the distance (rho) and angular (theta) resolution as input parameters for the Hough transform algorithm (as before the trackbars permit to change the algorithm output and to show it on the window). Here we have found good lines with values: `rho = 8` and `theta = 22`.



Rho: 8
Theta: 22

2. Circle Detection

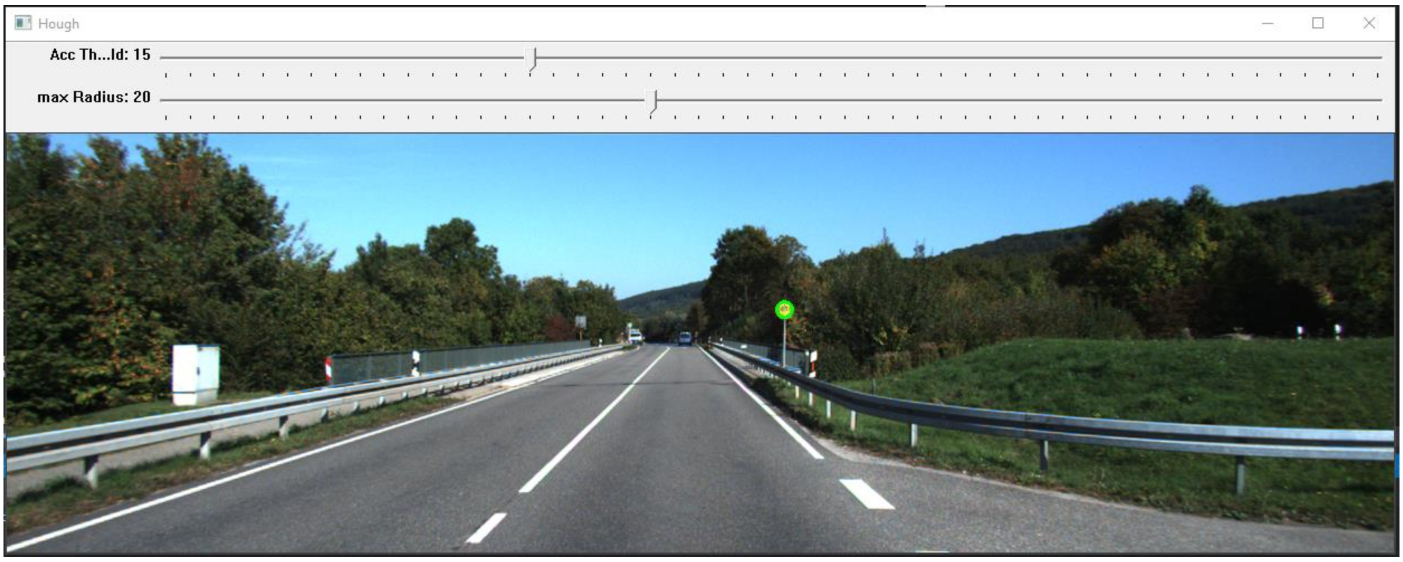
This time we keep fix the ratio to 2:1 and we allow only the selection of the lower threshold, this because the Hough Transform OpenCV function that detects lines already implements the Canny's edge detection method with a fixed ratio 2:1. The lower threshold found value is 655.



For the HoughCircles OpenCV function we use:

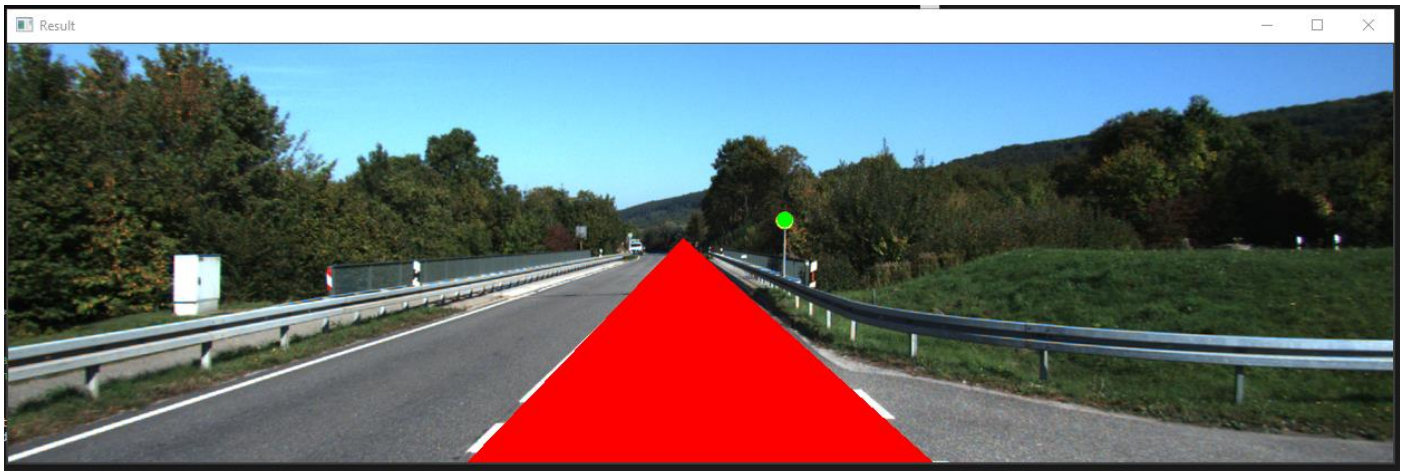
- as method the HOUGH_GRADIENT;
- as accumulator resolution the same resolution of the input image;
- as minimum distance between the centers of the detected circles we use $\frac{1}{16}$ of the image height (as used in the OpenCV tutorial)
- as canny threshold the one chosen on the previous step
- as minimum circle radius we use 0.

Therefore the user can change (always through trackbars) the accumulator threshold for the detected circle centers and the maximum circle radius allowed. The found values are: `acc_threshold = 15` and `max_radius = 20`.



3. Drawing detected regions

Through the OpenCV drawing functions the detected areas are drawn over the input image with the requested colors. Regarding the road lane: it's detected by two points in polar coordinates, which represent two lines. In fact, for each polar coordinate it's computed its line equation in the form $y = mx + c$, where $m = -\frac{\cos(\theta)}{\sin(\theta)}$ and $c = \frac{\rho}{\sin(\theta)}$.
Once the equation parameters are computed, we compute the intersection between the two detected lines and the intersection of each line with the bottom border of the input image; this to draw the triangle that covers the detected road lane.
This is the result:



4. Final note on running the program

NOTE THAT: the code is set with some default parameters that has been computed through some experiments and these values are the ones presented in this reports in the previous points. These values work only with the presented test image and the code can bring out unexpected results if the values are changed. In particular, this program works only with the 2 strongest detected lines and with one single detected circle.
Therefore, to correctly run the program is sufficient to press any key of the keyboard to go from step to step and to watch the results presented here.
If the input image and/or the trackbar parameters are changed, the program can lead to unexpected results.