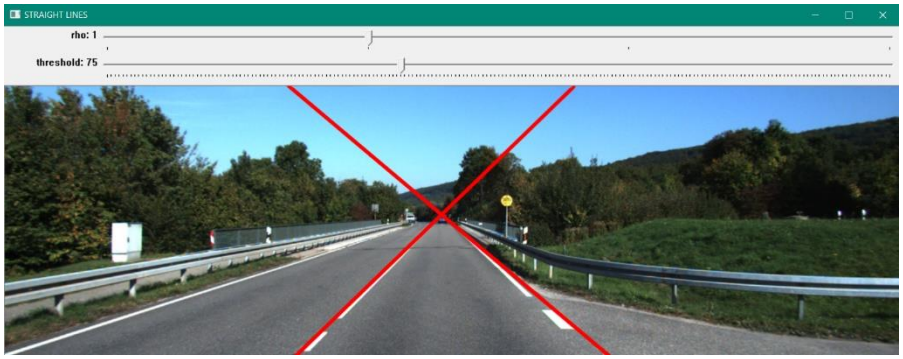
Lab 4 **Pietro Balzan 1241795**

In this lab, I learnt how to detect edges and shapes such as straight lines and circles in an image, using OpenCV with c++.

The file lab4.cpp contains the main function to be executed. Inside it, the program reads the input image from the command line (`argv[1]`), before showing it.

We need to extract the edges in the image, and to do this we can use a Canny edge detector by applying the `Canny(...)` OpenCV function on the grayscale version of the input image.

The parameters that can be tuned in this step are the kernel size of the `blur(...)` function that denoises the image (before canny calculation), and the lower threshold of the hysteresis procedure which is one of the canny function's arguments.



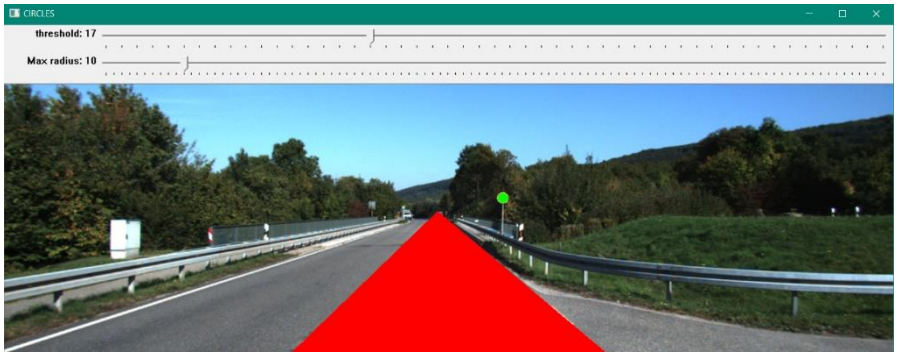
By setting the trackbars on the values shown in the image (73 for the threshold and 4 for the kernel size, these in my experience need to be very specific), we can specifically highlight the edges that we are interested in: that is, the boundaries of the road's right lane.

After this, the Hough Transform is applied to the edges matrix through the `HoughLines(...)` function to detect the straight lines. By setting rho to 1 and threshold up to 75 we obtain the two lines of interest.

In order to find the triangle to be filled, the intersection between those two lines needs to be found, and this is done by `linesIntersection(vector<Vec2f> lines, Point &r)`, which sets the point in `r` to the correct coordinates, computed from the `lines` vector. The two other points in the triangle (found directly

from the points that define the two lines' extremities) are saved in a global variable, an array of points, and, together with the intersection, they're used in the `fillPoly(...)` method.

A similar procedure is followed to calculate and find the circle corresponding to the roadsign: calculate another image of edges (this time a good setting for the parameters can be 2 for the blur kernel size and



100 for the lower threshold, but there is some room to change these and still find the circle), to which, instead of `HoughLines(...)`, the function `HoughCircles(...)` is applied with `threshold` equal to 17 and `max radius` set to 10 as shown in the image (with some room to change them).

The final result shows the right side of the road colored in red, and the yellow round road sign highlighted by a green circle.

I was surprised at how specific the parameters need to be for the detection to work properly in a particular image: it was especially difficult to find the right parameters for the canny function that would allow for a good detection of the road triangle (not as much difficult for the sign since it's the only perfectly round object in the image (lowering the blur from 4 to 2 in the edge detection was enough to find it consistently).

I tried with other images as well, but, again, the parameters need to be very specific for each image (couldn't find an image with the road sign):