Edge Detection and Hough Transform

University
of Padua

Computer Vision Report Lab 3
Alberto Chimenti

08/05/21

Abstract

This report discusses a naive implementation of a program to detect street lanes boundaries and circular road signs using Canny edge detector and Hough Transofrm algorithms.

1 Code Development

Parameter fine-tuning has to be considered as a necessary component for the program in question with a very wide range of possibilities, depending on the input that has to be processed. For this reason, the main goal of the whole implementation was to make it as flexible as possible. Therefore, the structure of it revolves around the use of OpenCV Trackbars.

1.1 trackbar.h

The `trackbar` module constructs custom classes to make use of the OpenCv Trackbar tool. To make the implementation cleaner, a parent class *Trackbar* has been created and it consists of a "wrapper class" for OpenCV Callback function. Such object stores three parameters, one *integer* and two *double*. Since the *cv::createTrackbar* method allows only the use of an integer parameter, for each of the *double* objects both an integer slider and an integer multiplier are allocated.

In the following is reported a snippet of the callback function defined for *double* parameters.

```
1 void Trackbar::on_trackbar(int pos, void* ptr) {
2         Trackbar* obj = reinterpret_cast<Trackbar*> (ptr);
3         obj->setParam(static_cast<double>(obj->slider)/static_cast<double>(obj->mult));
4         obj->doTask();
5 }
```

It is important to note that by construction, OpenCV callback functions have to be *static* which is the reason why pointers are used instead of the actual variables. Line *2* is used to reinterpret the passed class object as if it belonged to the parent class, therefore generalizing the usability to different subclasses. The other two functions called in line *3* and *4* are respectively used to set the correct value of the parameter corresponding to the selected slider variable, and perform the corresponding task.

The *Trackbar::doTask* function is defined as *virtual* in order to be able to override different versions of it depending on the subclass to which *obj* belogs.

Canny

The *cv::Canny* function from OpenCV has been wrapped in a subclass of the previously discussed *Trackbar* one. Two trackbars have been defined controlling the two *double* parameters from the parent class which vary the lower and higher threshold of the algorithm.

HoughCircles

Likewise the *cv::HoughCircles* function is wrapped in a specific subclass which uses three trackbars controlling: the internal Canny threshold (*double*), the accumulator threshold (*double*) and the maximum radius (*int*) of the circles to be detected.

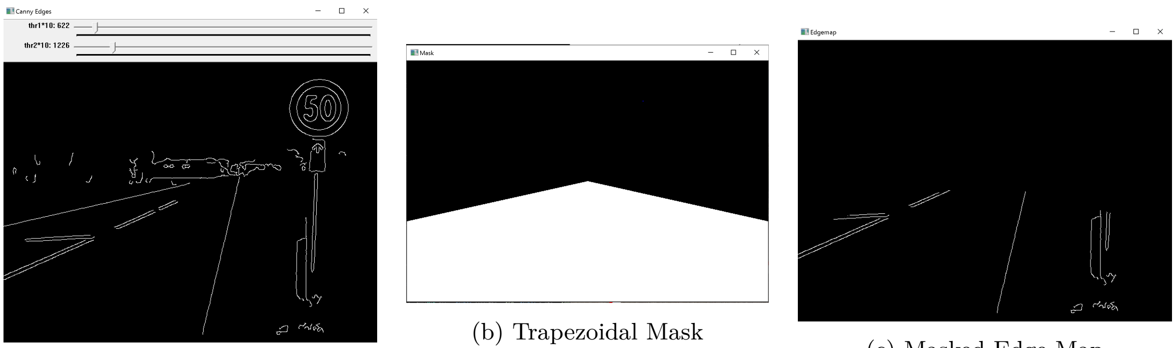
Also a specific *draw_circles* function is defined to show the results of the algorithm.

HoughLines

In this case the three trackbars control: the ρ and θ variables (*double*) of the polar coordinate representation of the line, and the minimum number of curves crossing a point in the line space for it to be detected.

The specific function *draw_lines* is responsible of overlapping the lines found to the original image. Since *cv::HoughLines* outputs the lines as vectors containing ρ and θ values, the actual line is drawn computing two very far away points which belong to it. To make the visualization prettier, the upper part of the resulting lines is cropped using a rectangular mask.

Additionaly, in order to detect only the correct lines, two operations are performed: filtering of the horizontal lines (*filter_lines*) and masking of the edge map. Indeed the masking is useful to avoid the detection of irrelevant components of the image. The class can load a custom mask with the *setMask* method but it also implements a specific "trapezoidal mask" covering only the lower part of the image.



thr1*10: 622

thr2*10: 1226

(a) Original Edge Map

(b) Trapezoidal Mask

(c) Masked Edge Map

Figure 1: Example of masking of the Canny Edge Map

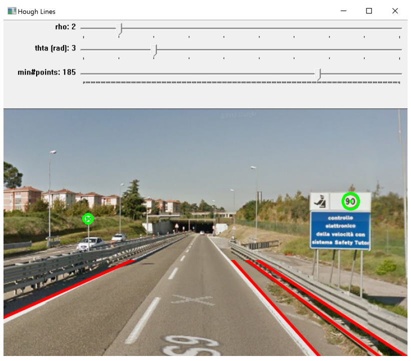
Lastly the program pipeline can be schematized as follows:

- Convert to Gray Scale and Resize
- Gaussian Blur: important to smooth the image and retain only the macroscopic features
- Hough Circles: which takes as input the grey scale original image
- Canny Edge Detection
- Hough Lines: which takes as input the edge map

For further information about other specific aspects, the reader is invited to take a look at the commented code.

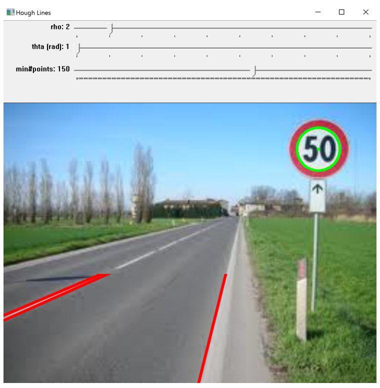
2 Example Results & Performance

One thing to note is that the white interrupted central street lines are more difficult to detect since they are not continuous. Finding them is still possible but it would need much more refinements since highering the sensitivity of the algorithm leads to many incorrect detections. Another possibility could be to fix a specific type of mask which has been implemented as an easy modification but not explored.



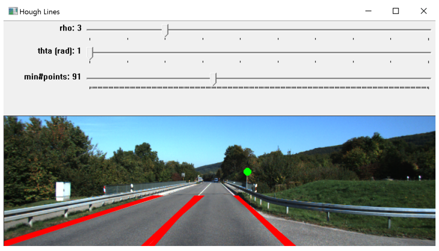
rho: 2
thta (rad): 3
min#points: 185

(a)



rho: 2
thta (rad): 1
min#points: 150

(b)

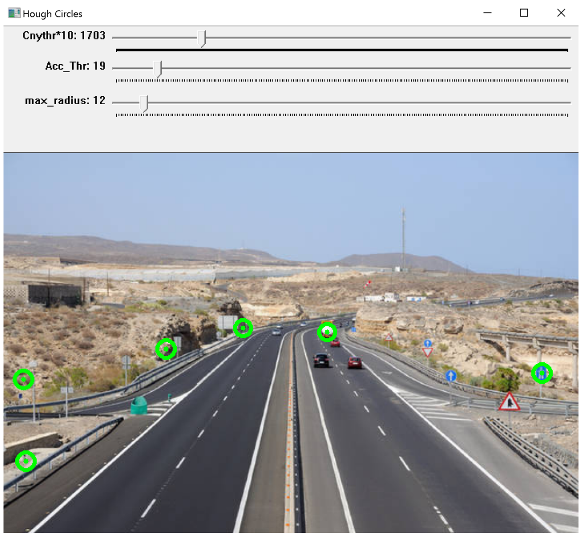


rho: 3
thta (rad): 1
min#points: 91

(c)

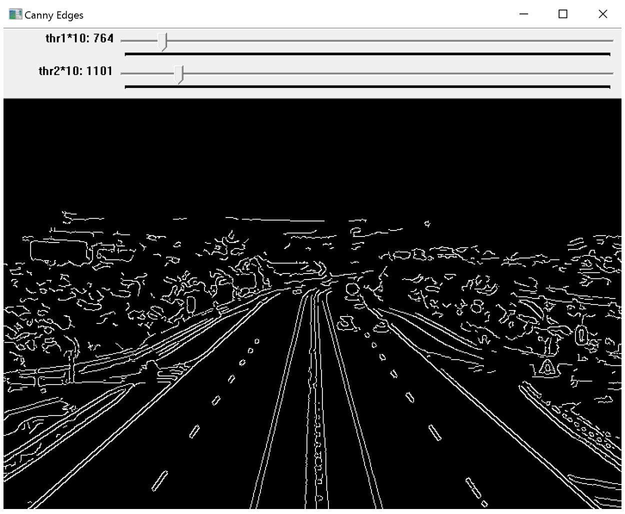
Figure 2: Examples of correct detection: see HoughLines chosen parameters

Finally one can clearly see that in case of *Figure(3)*, the detected circles are wrongly placed in the picture. By looking at (3*b*) that the problem here lies in the insufficient preprocessing of the image before the edge detection.



Cnythr*10: 1703
Acc_Thr: 19
max_radius: 12

(a) HoughCircles output



thr1*10: 764
thr2*10: 1101

(b) Edge Map

Figure 3: Example of incorrect circles detection