# Lab 3: Edges, lines and circles detection

Nicole Zattarin

**Abstract**

Before developing complex models to detect objects and people within an image, one should be able to identify the main characteristic of an image and its geometrical structure. In this report we explore simple, yet fundamental, algorithms that allow to detect edges, lines and circles in an image. In particular, we provide an interface which makes it simple to understand the meaning of the parameters of such algorithms and how these affect the detection of geometrical structures in the image.

## 1. Setup and parameters tuning

At this link we provide a possible implementation of a program that, given an image, performs edge, lines and circles detection. The executable is structured in order to receive from command line the following arguments:

- path to an image;
- path to a file of parameters;
- boolean variable to decide if tuning parameters or use those provided in the file;
- boolean variable that establish if tuned parameters must be saved in a `.txt` file.

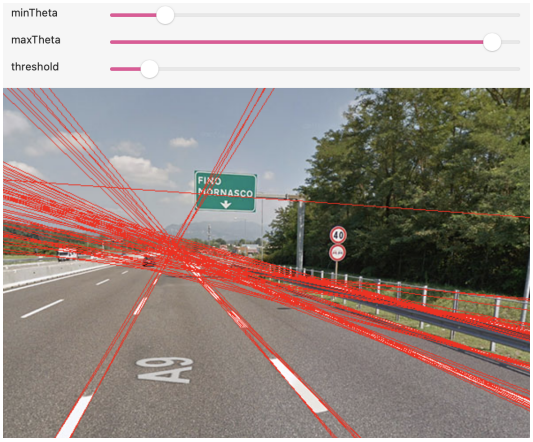
In summary, to lunch the executable one should first compile and link the code with make and then run the program providing the desired variables, as follows:

```
make
 ./main <image path> <params file path> <tune> <save>
```

When the tuning is performed the user can change in real time the values of the different parameters on the multiple windows that are displayed, in order to observe the impact of each parameter. Note that parameters change simultaneously on all the windows, so for instance if one changes the Canny threshold and then interacts with the trackbars of HoughLines, line detection is performed with the new Canny threshold.

The idea is that we can get a good result by tuning different detectors and observe in real time how changes affect the result. Anyway, the suggested path is to first we Canny, see an example of interface in Figure 1a and only after that moving to HoughLines and HoughCircles. Trackbars are useful to see in real time how the output of a detector is affected by parameters change, for instance in Figure 1b we can observe lines appearing and disappearing according to the angle range and to the threshold. At the end of this tuning process the chosen parameters are printed on the screen, and, if `save` is `true` these are saved in a `.txt` file. In this way we can run again the code to get only the final result with the parameters provided into the file anytime, note anyway that files may be overwritten. To summarize, the idea is to run:


**(a)** *Example of interface for Canny parameters tuning, i.e. lower and higher thresholds.*




**(b)** *Example of interface for HoughLines parameters tuning, i.e. threshold and angle range.*

**Figure 1:** *Example of interface for a run in which we ask to tune the Canny parameters (a) and the HoughLines (b). Note that parameters are set randomly in this example, just to show that the image changes according to values inserted through the trackbar.*

```
./main images/road2.png params/params_road2.txt 1 1 // manually tune params, save into file
./main images/road2.png params/params_road2.txt 0 0  // Get final result!
```

Trackbars callbacks are collected in `trackbars.h`, while `utils.h` contains all the functions that can be useful to manipulate images, lines, circles and draw on the image itself. Finally, note that each of the image requires a specific tuning and a specific processing in order to get the final result, this is performed with functions provided in `images_processing.h`. Therefore, running the code on a generic image won't give back the same result as on the sampled ones, unless one creates a specific function to deal with it.

## 2. Results

We test edges, lines and circles detection of three images provided in the folder `images`, in particular `road2.png`, `road3.png` and `road4.png`. The initial part of the processing is the same for all the images and consists of the following steps:

1. Load image and parameters, show the original image;
2. Apply Canny to the greyscale image, tune the parameters and save the edge map. This is quite a delicate point, since the output of Canny is fundamental to individuate lines in the next step. Indeed, we are interested in detecting edges which correspond to street lines, but we can also allow the detection of the edges of leaves and other minor objects, since these won't be identified as lines by HoughLines. The thresholds should then be fixed in order to make it possible for HoughLines to detect the desired lines, but we still have a tolerance due to the fact that many edges won't affect our final result.
3. Use the output of Canny, i.e. the edge map, to individuate lines within the image by means of HoughLines. We tune the parameters in order to individuate only the boundaries of the street or of the street lane, this can be done fixing a proper range for $\theta$ and the threshold: the lower the threshold, the more lines we detect. Moreover, once the parameters are fixed, we fill

| | | |
|---|---|---|
| Canny | apertureSize | 3 |
| | threshold1 | 350 |
| | threshold2 | 850 |

| | | |
|---|---|---|
| HoughLines | rho | 1 |
| | theta | 0.05 |
| | threshold | 130 |
| | min theta | 0 |
| | max theta | CV PI |

| | | |
|---|---|---|
| HoughCircles | dp | 1 |
| | minDist | 1 |
| | param1 | 100 |
| | param2 | 25 |
| | minRadius | 0 |
| | maxRadius | 10 |

**Table 1:** *Parameters set for road2.png image, for all the three algorithms.*

| | | |
|---|---|---|
| Canny | apertureSize | 3 |
| | threshold1 | 270 |
| | threshold2 | 400 |

| | | |
|---|---|---|
| HoughLines | rho | 1 |
| | theta | 0.05 |
| | threshold | 160 |
| | min theta | 0 |
| | max theta | 3 |

| | | |
|---|---|---|
| HoughCircles | dp | 1 |
| | minDist | 1 |
| | param1 | 353 |
| | param2 | 31 |
| | minRadius | 0 |
| | maxRadius | 32 |

**Table 2:** *Parameters set for road3.png image, for all the three algorithms.*

the polygon which is identified by the lower (i.e. points with larger $y$ in the openCV system) points of two intersecting lines and the intersection point,

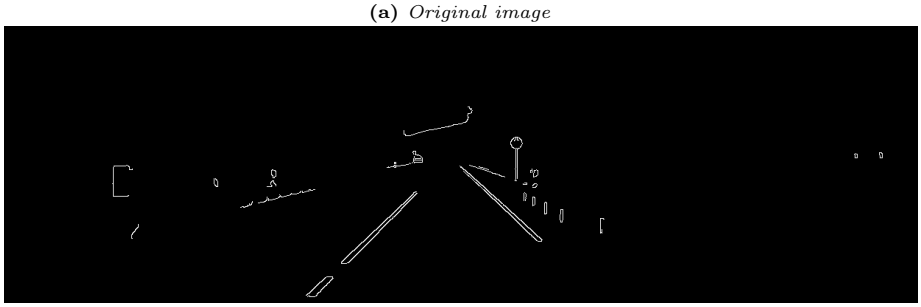
4. Apply HoughCircles to the blurred greyscale image in order to individuate circles, with particular attention to tuning in order to detect only round street signs. In this case the most important parameters to consider are the range of the radius, which allows to fix the size of the circles we want to detect, and the thresholds, which are used inside of HoughCircles to perform edge detection. Finally we fill circles with the same color.

In particular, parameters to process image `road2.png` are shown in Table 1, while in Figure 2 we provide three different steps of the image manipulation: original image in Figure 2a, edge map in Figure 2b and the final result in 2c. Note that in the final result we show only the relevant lines and we fill in between only up to the intersection point. This processing is designed specifically for this image and it is performed in `void road2Img_processing (Mat&, vector<Vec2f>)`, in which we also fix a condition of the coordinates of the intersection for plotting the lines in order to eliminate lines which have an intersection outside of the image.

For what concerns image `road3.png`, parameters are shown in Table 2, while images in Figure 3, in this image we detect the main lines that delimit the road and two circles. Note that the parameters for HoughLines are similar to the same fixed for the previous image, while Canny thresholds are pretty different. This is due to the fact that each image requires a specific tuning, both because of the intensity distribution of the image itself and on the objects inside the image. Also in this case we perform a specific processing in `void road3Img_processing (Mat&, vector<Vec2f>)`.

Results for `road4.png` are shown in Table 3 and Figure 4, in this case we identify both the limiting line of the road and two lines inside of the street itself. We then fill the two roadways with different colors, in order to highlight the line in the middle, moreover we detect circles corresponding to signals.

**(a)** *Original image*



**(b)** *Edge map after applying Canny to the greyscale image*

**(c)** *Final result: circle street signs are filled in green, while the area between the intersected lines is filled in magenta*

**Figure 2:** *Three different steps of the image manipulation: original image in Figure 2a, edge map in Figure 2b and the final result in 2c.*

| | | |
|---|---|---|
| Canny | apertureSize | 3 |
| | threshold1 | 450 |
| | threshold2 | 730 |

| | | |
|---|---|---|
| HoughLines | rho | 1 |
| | theta | 0.05 |
| | threshold | 62 |
| | min theta | 1 |
| | max theta | 3 |

| | | |
|---|---|---|
| HoughCircles | dp | 1 |
| | minDist | 1 |
| | param1 | 67 |
| | param2 | 31 |
| | minRadius | 0 |
| | maxRadius | 14 |

**Table 3:** *Parameters set for road4.png image, for all the three algorithms.*


**(a)** *Original image*

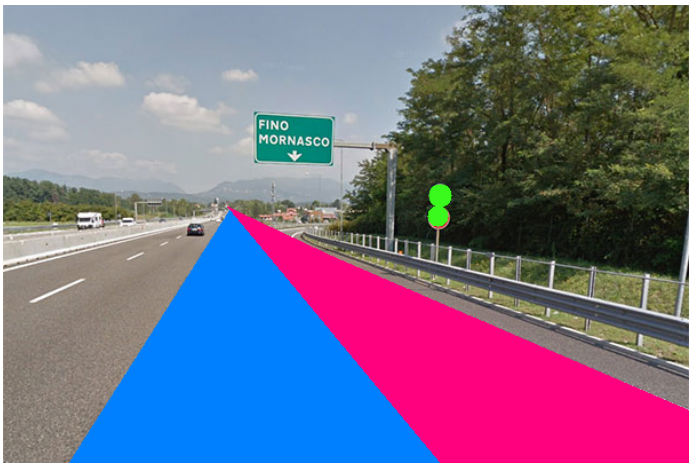
**(b)** *Edge map after applying Canny to the greyscale image*


**(c)** *Final result: circle street signs are filled in green, while the area between the intersected lines is filled in magenta*

**Figure 3:** *Three different steps of the image manipulation: original image in Figure 3a, edge map in Figure 3b and the final result in 3c.*


**(a)** *Original image*

**(b)** *Edge map after applying Canny to the greyscale image*




**(c)** *Final result: circle street signs are filled in green, while the area between the intersected lines is filled in magenta and blue. Note that in this case we fill differently the two different areas in order to highlight the line in the middle.*

**Figure 4:** *Three different steps of the image manipulation: original image in Figure 4a, edge map in Figure 4b and the final result in 4c.*