Computer vision: Final project

1 Task

Given a set of photos of cars, it is required to develop an application in c++ able to:

- detect the licence plate;
- read the various single characters in the licence plate.

Given the length of the task, two classes have been implemented:

- *ExamClass*: does the preprocessing of the input image, detects and extracts a plate;
- *ExamClassPlates*: detects and extracts the single characters from a plate.

The program has been tested only on the given dataset.

2 Detection:

Firstly, all the images have been loaded in a single vector; the main will run the same code for seven times, so for all the images.
The preprocessing has been decided purely on manually changing filters, their orders and parameters with the aim to highlight edges and, in particular, closed edges like a plate could be. So, a very simple and not too aggressive bilateral filter has been applied, followed by an equalization of spectrum in the Lab color space. The results, in Figure 1 shows how the details on the plane are ruined, but at this stage of the algorithm it's not important, the focus is on the contour.

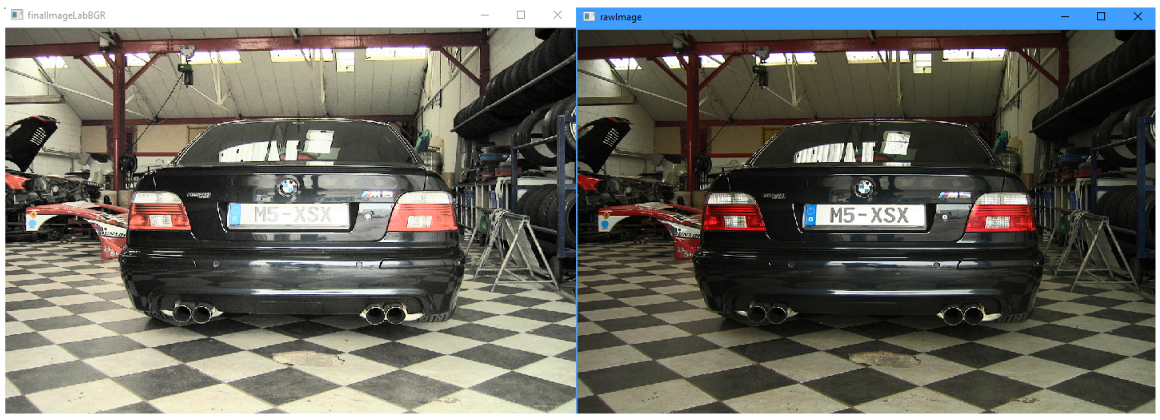


Figure 1: On the left, the preprocessed image; on the right the original image.

The image has been blurred and then the Canny algorithm has been applied to find edges. The threshold has been found experimentally, looking at the results and trying to remove as much noise as possible

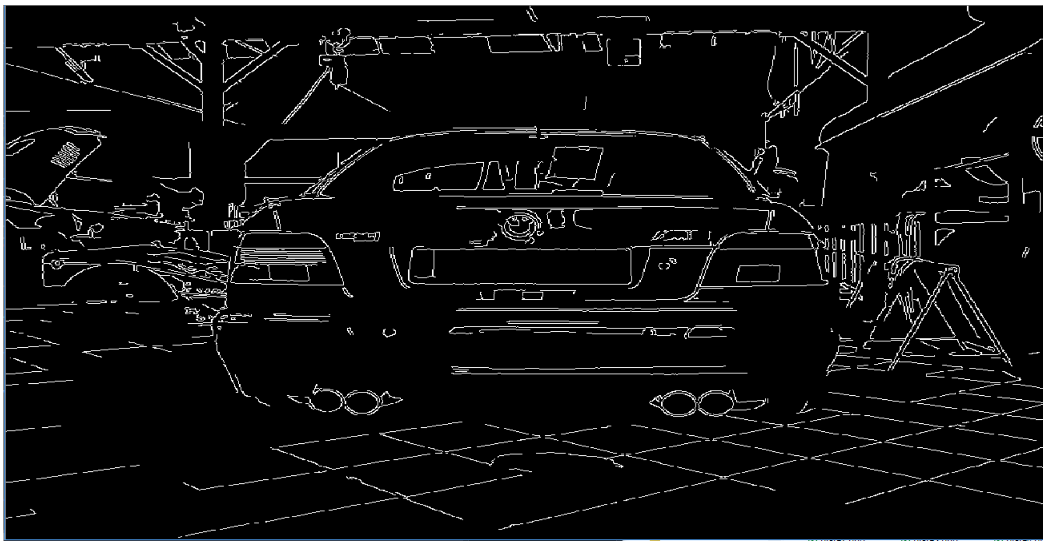


Figure 2: Example of Canny detect, the plate box is intact.

without ruining the box edges of the plates. In Figure 2 the plate box is safe, but there is still a lot of other edges.

To highlight the results, I used the *findContours* method to color the edges Canny found: we can see in Figure 3 all the plates except a) are inside a single border. At this point I realized I needed to produce multiple candidates for the plates and choose among them.

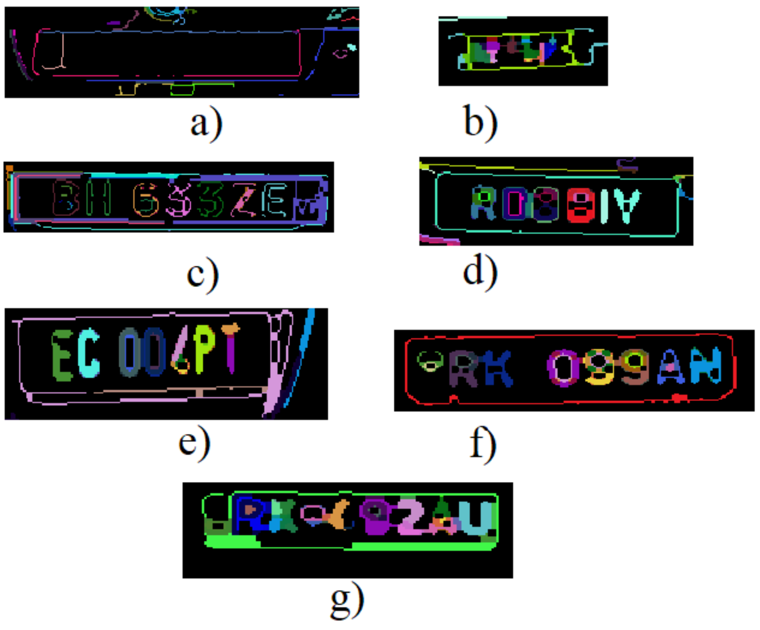


Figure 3: a), b), c), d), e), f), g) details of highlighted contours. No extraction yet.

Two methods has been implemented to extract the plates:

1. Starting from the contours of the previous described image:
    - find the biggest area contour: it's the candidate to be the plate;
    - simply extract the plate.
2. As already proved, some plates are not inside a single contour, so after many experiments, I've decided to just do a very light random selection of the borders to eliminate:

- starting again from the image with the coloured contours, convert the image to the HSV (Hue Saturation Brightness) color space, and just keep the lines inside a very large threshold, this eliminates very few borders;
- using *findContours* again on the new image, I do the same things as 1) and find the biggest area.

This procedure generates 2 plate candidates, and the final plate is just the biggest extracted image from 1) or 2). Also, if the plate is too small, number of columns under 110, then it gets resized with a *Lanczos* method, which shouldn't deform too much the original image and will help the char detector later.

3 Characters:

The extracted plates are sections of the original raw images, so now another preprocessing is needed before trying to extract the single characters. Again, the preprocessing can't be too heavy, so I've just:

- converted the image to gray scales;
- maximized the contrast. To do so, a very common procedure found on the OpenCV community forum was applied:
  - filter the image with a top hat and a black hat;
  - sum the original image with the top hat and remove the black hat version.
- used a *gaussianBlur* filter, with a big kernel size, 5x5;
- used a threshold to keep only the brightest colors;
- applied another threshold, *adaptiveThreshold*.

The parameters have been tuned manually based only on my personal opinion on the quality of the results. In the Figure 4 a), b), c), d), e), f) and g) are reported the extracted plates with the aforementioned processing.

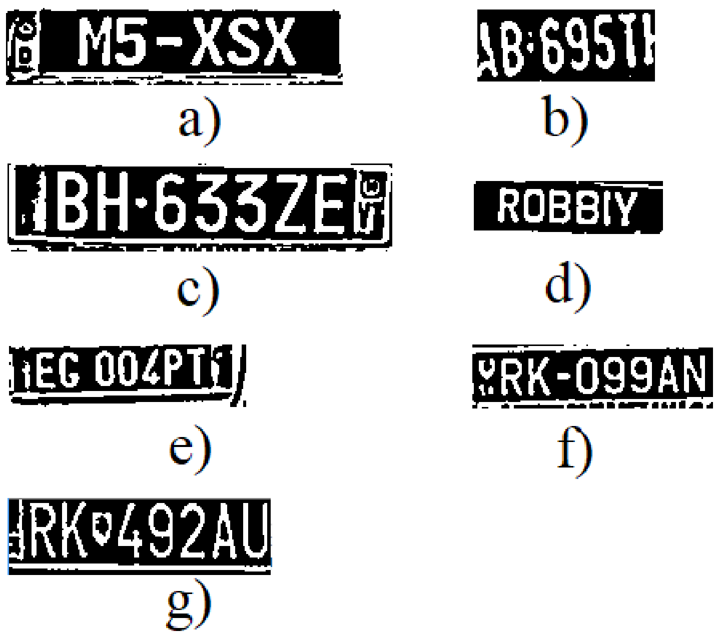


Figure 4: a), b), c), d), e), f), g) are the detected and extracted plates after preprocessing.

The next step is using Canny to find the edges and *findCountours* to find the contours. Then, for each contour I fill the box with a red color and extract the corresponding region in the preprocessed image: this is a potential letter.
To filter them, the rules deciding if a box contains a char are:

- the box must be a rectangle, with the number of rows bigger than the columns;
- the box should have a number of rows bigger than half the rows of the total plate;
- the box should have a number of columns smaller than the number of columns of the plate divided by 5;
- find the area of every box, compute the mean and keep only the boxes inside an interval of]mean area -220, mean area + 220[.

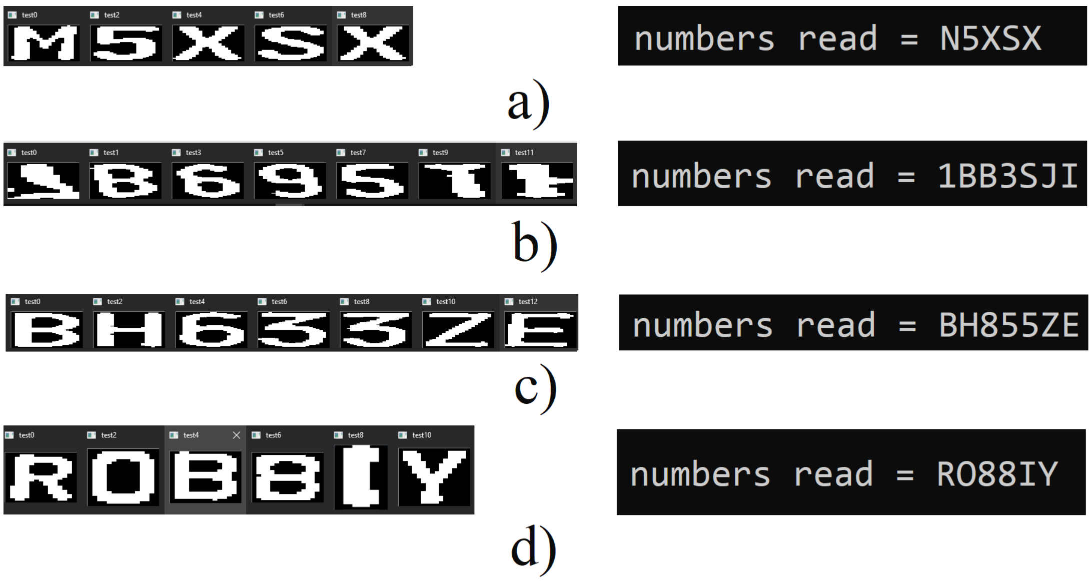
Unfortunately, the boxes are not found in the correct order and some are repeated, so I also had to:

- find the correct order of the boxes;
- eliminate the boxes too close with each other: the letter are well spaced one from another;
- eliminate the boxes in the same position;
- keep only the first 7 boxes, because the maximum number of characters is 7.

Now that a vector of images with only the characters is generated, the final step is to identify the correct char. To do so, I've used a *kNearest*'s OpenCV implementation with a dataset found on YouTube [1]. The method requires two files:

- *classifications.xml* represents the char position in the images file, coded in UTF8;
- *images.xml* contains flattened images used for the training, four for each of the 36 possible alphanumeric characters.

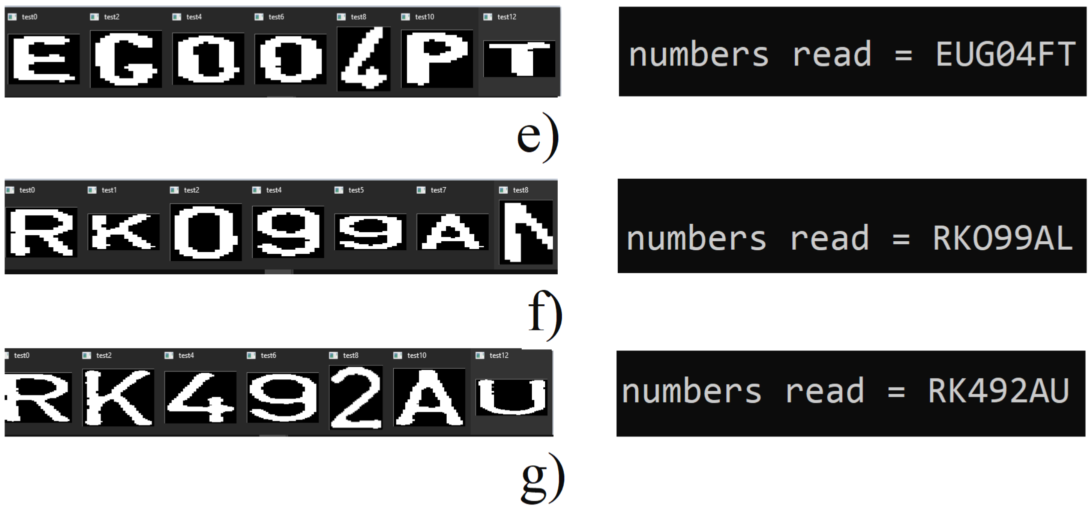
Before calling the classification algorithm, a simple reshape is performed; the results are showed in Figure 5 and 6.



```
numbers read = N5XSX
numbers read = 1BB3SJI
numbers read = BH855ZE
numbers read = RO88IY
```

Figure 5: a), b), c), d) final results.

[1]`https://www.youtube.com/watch?v=euG7-o9oPKg` and relative project on gitHub `https://github.com/MicrocontrollersAndMore/OpenCV_3_License_Plate_Recognition_Cpp`



```
numbers read = EUG04FT
```

```
numbers read = RKO99AL
```

```
numbers read = RK492AU
```

Figure 6: e), f), g) final results.

4 Conclusion:

The results can be commented in two section, one for the character extraction and one for the character identification:

- regarding the char extraction:
  - 5 b), has an issue with the initial A and final H of the plate. This is due to the preprocessing of the plate: because it has to work for every other plate too, it must has a general approach and in this case the car plate is the smallest of the dataset and it's not totally horizontal, so it's a peculiar case;
  - 6 f), the N is not well cut by the box even if the plate image is well extracted.
  - all of the other chars are well identified and cut.
- regarding the character identification, the *K-nearest neighbors algorithm* has been chosen simply because of it's easy to implement; the type of results, on a total of 46 chars, are:
  - correctly classified, 30 characters;
  - wrongly classified, 13 characters;
  - wrongly classified because of the character box, 3 boxes.

So the algorithm classifies the char with a success of 65%, which is definitely better than random, but not enough for a real application.