Image Processing
License Plate Recognition

[redacted]

1 Introduction

In this report, we describe our approach for an Automatic License/Number Plate Recognition (ANPR) system, which is still an ongoing research topic of high interest in the image processing field.

Given an input video of cars with license plates of various angles, sizes and shown in different scenes, the system will pick individual frames from the video and firstly try to localize the plate inside the frame and then isolate each character, which will be matched against our database of standard license plate characters. The best matches will be picked and put together character by character to form the final output string of the plate.

For better accuracy, we have applied some restrictions (see subsection 3.4.1) and a voting system which is further explained in Section 3.

2 Localization method

2.1 Data splitting

Out of the ≈ 40 plates in category I and II, we have used 30 plates as training data and the rest were used as test data. We have annotated 25 frames of the test scenes by manually writing down the results we were expecting to get. In this way, we have generated our own ground-truth data to check the algorithm's performance and thoroughly evaluate it.

2.2 How it works

The localization system receives as input a frame from the given video, which is converted into HSV and blurred with a Gaussian kernel (Mousa [2]) to get rid of the noise. We then look for the yellow color inside the frame (standard Dutch license plates have a yellow background and black letters, which makes it much easier to isolate the plate) by keeping all the pixels that are inside a given range [1] (we have predefined a range of values that covers different shades of yellow). After getting a rough mask of the yellow plate, we apply morphology techniques to remove more noise and get a better overall mask of the isolated plate.

[1] This and some other techniques used from OpenCV library (OpenCV [4])

Figure 1: Initial frame (left); Masked yellow (right)

Using the previously obtained mask, which is a white rectangle with as little noise around it as possible on a black background, we crop the white part out. This way, the size of the frame is considerably reduced and only the ROI (region of interest) is kept. Furthermore, we increase the algorithm's performance since we have a lot fewer pixels to loop through.

Next, we apply the Sobel edge detection technique (X et al. [8] and Dougherty [1]) to get just two parallel vertical white lines on a black background. We thought this is the easiest way to loop through each half and find the corners of the mask, which shall be used to straighten the plate.

We split the already cropped frame in half and loop through each of them, pixel by pixel until we find a white one. Then we loop in reverse, starting from the bottom to the top. In this way, we find the two corners in each half. This explains why we have previously applied edge detection to get just two vertical lines. If we were to keep the entire white rectangle, for the bottom part, the middle of the plate would have been found instead of the corners.

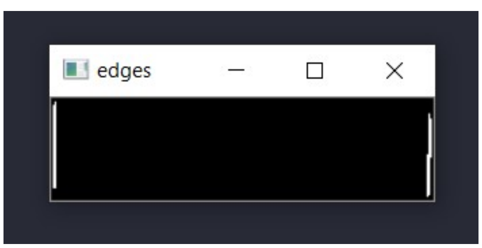

Figure 2: Edge detection on mask

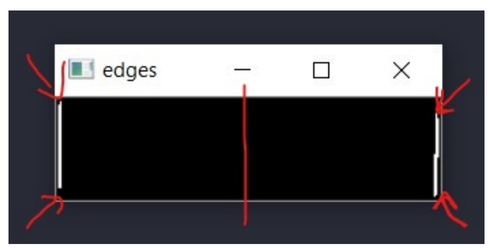

Figure 3: Split the plate in half to find corners

Now that we have found the corners, we draw lines between them to form a better-defined rectangle filled with white. Using the bottom corners, we calculate the in-plane rotation angle of the plate and straighten it out by using "ndimage.rotate" from the SciPy [6] library.

Figure 4: Rectangle from detected corners

Figure 5: Filled rectangle

Then we perform another check based on the ratio between what we had initially and the rectangle we have just defined. If that ratio is below a certain threshold, it means no plate could be found.

For multiple plates in a single frame, we perform the same algorithm with a slight variation. If we cannot correctly find a plate after running the entire algorithm, then it means there might be more than one plate in the frame, so we split the initial frame in halves and repeat the algorithm on each of them.

2.3 Localization evaluation

As previously stated, we have split the data and created our own ground-truths to check the results. We first ran our algorithm on the training data to see if there were any more improvements we could make and then we used the test data to evaluate its performance. For every test object we checked if the returned image contained well cropped licence plate. Besides that, we tested the intermediary step of the corner detection, where we checked if the corners found by our algorithm are within a radius of 10 pixels from what we had as our ground-truth.

2.4 Result analysis

After gathering the results, we concluded that our Localization algorithm was performing well.

Taking into account that we have also used frames where the plate was cut off or on the very edge of the frame, we have obtained a 93% localization rate. We had some issues with the noise and masking for some plates, which, unfortunately, could not be improved any further without affecting the intermediary steps.

The main problem was the extra noise generated by the initial yellow mask, which would sometimes detect the taillights, and if we extended the range, then other plates would perform worse. By trial and error, we have eventually found an optimal overall solution.

Figure 6: Localization final result

3 Character recognition

3.1 Data splitting

For splitting the data and annotating it, we followed the same process as we did for the localization part. We chose a training set and a test set of perfectly cropped plates by splitting the initial data and manually annotated them. We wrote down the final output strings that we expected to get as result in a ground-truth file. Using this file, we were able to check the character recognition algorithm's performance and apply a thorough evaluation.

3.2 How it works

Given a cropped and straightened plate, we start the recognition process by resizing it to a certain smaller size (setting height to 70 px) which allowed us to quickly[2] process the frame without much loss of pixel values. This size seemed like an optimal solution for a good recognition rate while maintaining good performance since the number of pixels is reduced considerably for most plates.

We convert the image to HSV and blur it using Gaussian kernel to mask out the yellow color better (the same algorithm used for the yellow mask in the localization process. See Figure 1). The initial mask is used to crop out the letters from the plate with a bitwise AND. An extra crop of 7% on the x-axis and 2.5% on the y-axis is applied to eliminate the potential noise and screws that keep the plate fixed on the car. The mask is turned into grayscale and we apply histogram equalization (Zhai et al. [9]) to increase the differences among the shades of gray to be able to put a threshold on what is the foreground (the letters) and the background (everything else). In this way we can better binarize

[2] By using NumPy (NumPy [3]) and Python speeding up techniques (Python [5])

the image to pure black and white by making all pixels above the picked threshold black and the rest white. See the mask improving process below.

Figure 7: Letters mask evolution

Now that we have perfected our mask's boundaries, we apply some morphology to eliminate the noise around the letters and further improve it. The mask is then used to find each individual letter and crop it out of the plate. We loop from left to right, column by column until we find one that has at least one white pixel and save the index of that column, then we keep looping until we find one with no white pixels at all and we save that index as well. If the difference between the start index and the end index of the previously found object is bigger than 20, then it means there was a big gap between the letters and it might indicate either noise or a dash. In this way we have saved the beginning and end of a letter and we just keep repeating the process until there are no more letters to find.

We store all those indices in an array and then use them to crop out the letters from the black and white mask of the plate (see Figure 7 bottom right). These are then matched against all the characters in our "database" (we have used our own letters, see subsection 3.4.1 where we describe the filtering rules that were applied to the final plates). Each char is resized to match the width and height of the characters we have created because sometimes the character crops are not of the exact same size. This is needed in order to perform a bitwise XOR between our characters and the cropped ones. Afterwards, we count the number of white pixels for each XOR and we pick the best match for each cropped character and append it to the final string.

We do this for all the frames in which the same plate is visible, which results in having multiple strings for the same plate. Sometimes, these results vary a bit and therefore we have counted which version was recognized most frequently and pick that one as the final result. Some extra filtering is applied before this, of course, because Dutch license plates have certain rules that help us filter out wrongly recognized plates. For more details see subsection 3.4.1.

3.3 Character recognition evaluation

In order to test and evaluate our algorithm, we have used the training and test data and our ground-truth file that were previously described in subsection 3.1.

We ran our algorithm against the training data first to make sure we have improved it as best as we could and then used the test data to output the performance of the recognition algorithm. We checked the strings that the algorithm generated against the results we were expecting and calculated the percentage of accurately recognized sequences of characters.

3.4 Result analysis

This time the results were worse compared to the localization part since some characters were easily mistaken for others, such as 8 and B, but the voting system that was described previously helped with filtering out little mismatches such as these. Unfortunately, this was not enough and hence we have filtered the results even more with some rules and restrictions that are described in the subsection below. Adding more constraints have considerably increased our overall score.

We have managed to achieve a 90% character recognition rate, given that the plate was perfectly cropped and the restrictions were applied. Because we wanted to evaluate only the Character recognition side, we have given it a perfect input. Unfortunately, this was not always the case and the overall score was a bit lower (see subsection 4.1).

3.4.1 Filtering rules

We have applied some rules and restrictions to filter out the wrongly recognized plates (Wikipedia [7]):

- there must be exactly 8 characters; 2 dashes and 6 other characters in every plate
- the dashes can never be the first or the last
- two dashes cannot be consecutive
- removed all banned characters. All vowels and others such as C, Q, etc.
- the 3 groups of characters formed by the 2 dashes are made out of only letters or only digits
- there must be at least one digit and one letter in the same plate

4 Analysis of the system

4.1 Analysis of failures and success rate

The main issue with our algorithm is the fact that we have based our solution on finding the yellow color in a frame since the Dutch plates have a yellow background, but this became an issue since it would sometimes detect tail lights, entire cars because they were yellow, or not detect the plate at all since the video had a green/purple overall tint. Moreover, we are not able to detect international plates since they do not have a yellow background.

Sometimes the plate localization is not perfect because of the out-of-plane rotation, which we were not able to fix, but we think we overcame this issue further down the pipeline when we filter the results and apply the voting system. Overall, we found that our algorithm has a 85% success rate of correctly identifying license plates from the given video.

The algorithm processed the trainingsvideo.avi in 2 minutes and 20 seconds on a MacBook Pro(Retina, early 2015), 2,7 GHz Dual-Core Intel Core i5, 8GB RAM 1867 MHz DDR3, Intel Iris Graphics 6100 1536MB.

4.2 Further improvements

The main improvements that could be made are the following:

- changing the localization method and try to find not only the yellow plates in a frame. Our algorithm cannot detect international licence plates which have a white background or any other color that contrast the letters.
- using better noise reduction (different kernels for morphology or blurring) as sometimes our algorithm removes the noise but also distorts the plate itself.
- taking better care of the out-of-plane rotation with something that warps the entire plate from a skewed plane to a straight, flattened one.
- a better voting system that is more precise. So instead of voting per final string, we could vote per character. In this way, we can get the most likely character at each position, instead of treating the entire string of 8 characters as a whole.
- the majority of the algorithm consists of ideas of how to go around a problem which could be easier solved by optimized library code. In many cases our algorithm performed worse when we had to exchange library methods with our own implementations. The algorithm could be improved a lot by adding some optimized library functionality and that would result in better general performance as now it relies on well characterized data sets.

References

[1] A. Dougherty. Magic of the sobel operator, 2020. URL https://towardsdatascience.com/magic-of-the-sobel-operator-bbbcb15af20d.

[2] A. Mousa. Canny edge-detection based vehicle plate recognition. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 5(3):1–3, 2012. doi: https://staff-old.najah.edu/sites/default/files/Canny_Edge-Detection_Based_Vehicle_Plate_Recognition.pdf.

[3] NumPy. Numpy documentation, 2020. URL https://numpy.org/doc/.

[4] OpenCV. Opencv documentation, 2020. URL https://docs.opencv.org/master/.

[5] Python. Python documentation, 2020. URL https://wiki.python.org/moin/PythonSpeed/PerformanceTips.

[6] SciPy. Multidimensional image processing (scipy.ndimage), 2020. URL https://docs.scipy.org/doc/scipy/reference/ndimage.html.

[7] Wikipedia. Vehicle registration plates of the netherlands, 2020. URL https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_Netherlands.

[8] A. D. X, O. V. J. Laura, S. P, S. Y, and S. R. P. License plate localization by sobel vertical edge detection method. *International Journal of Emerging Technologies in Engineering Research*, 4(6):51, 2016. doi: https://www.ijeter.everscience.org/Manuscripts/Volume-4/Issue-6/Vol-4-issue-6-M-13.pdf.

[9] X. Zhai, F. Benssali, and S. Ramalingam. License plate localisation based on morphological operations. *11th International Conference on Control, Automation, Robotics and Vision, ICARCV 2010, Singapore, 7-10 December 2010, Proceedings*, pages 1–5, 2010. doi: https://www.researchgate.net/publication/221144556_License_plate_localisation_based_on_morphological_operations.