Image Processing
<License Plate Recognition>

[redacted]

1 Introduction

Our main goal was to design a system that localizes yellow license plates and recognizes the characters found on them, such that one would be able to identify a license plate number.
The following section will briefly introduce an overview of the system, from the input image to the output characters.
As it can be observed in Figure 1, the pipeline of the system is straightforward: initially, the input video stream is pre-processed in order to be able to localize the actual license plate. After the plate is found, the next step is to standardize it, such that the input for the next pipeline stage is always as expected. Finally, the characters are as well standardized, then recognized and outputted.
More in-depth explanations about each of these stages will be presented in the upcoming sections of this report.

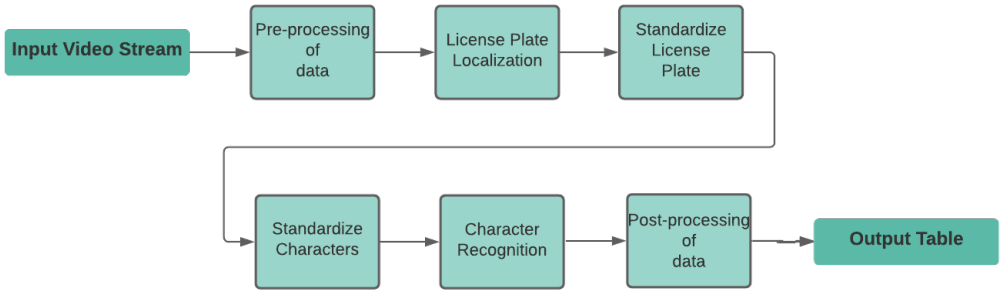

Figure 1: Schematic pipeline of the license plate recognition system.

2 License plate localization method

2.1 License plate localization data description

In order to design the model, the videos provided in the Training Set folder for category I, II, and III were used.

The splitting of the data was done by allocating 70% of the license plates present in the video for training. The rest of 30% were only used for testing purposes.

In order to annotate the data we designed a simple User Interface that allowed red rectangles to be drawn on top of the image and saved for each frame. This was used when checking how accurate our localization system was, by comparing the user annotated rectangles with the rectangles that resulted from the extrema points after applying fill on the masked images.

2.2 License plate localization system

Our license plate localization technique consists of several steps, which will be briefly explained below.

1. Firstly, we have considered a simple image segmentation approach, based on the HSV color space. We have tested a variety of ranges of yellow and we have arrived to the one that is currently in our source code. We have used this range of yellow to create a bit mask, that has white where we have found a color inside that range and black where the colors are outside of the range.

Figure 2: Mask of yellow.

2. Secondly, after identifying our yellow license plates, we have implemented a flood fill algorithm. (Please refer to Appendix A for a comprehensive explanation of how this algorithm works.)
3. To find the contours of our plate, we used a technique inspired by the Canny edge detection algorithm, specifically, we applied both vertical and horizontal sobel filters and created the magnitude matrix by combining their results, which can be seen in the following picture.

Figure 3: Contour identification using Sobel filter.

4. Afterwards, we extracted the positions of the pixels in the contour image and applied the minimum area rectangle function them to find the angle with which we need to rotate the plate.
5. Having the corresponding angle from the previous step, we have rotated the plate and reapplied the HSV mask once again to ensure that only the license plate is extracted from the image.

Figure 4: Rotated license plate.

6. Finally, we cropped the plate from the rotated image and normalized the height to a value of 100 pixels for all of the plates. This was necessary in order to make assumptions about the possible width and height of the letters in the recognition method.

Figure 5: Normalized license plate.

2.3 Evaluation metric for license plate localization

The evaluation metrics of our localization method were computed using the accuracy score. In order to determine it, we have used intersection-over-union of localized bounding boxes versus boxes annotated by us.
The way this is done is by calculating the area of both bounding boxes and outputting how much these two areas actually overlap.

(a) Our algorithm's detection.

(b) User annotated image.

Thus, the resulting output should be either 0 - or close to 0, if the plate is not found, or over 0.6 - 0.7, if the plate is correctly identified.
After computing the accuracy considering the formula: correctly identified plates/all testing plates, the resulting accuracy score was 96.5%.

2.4 Analysis of the license plate localization results

It should be noted from the beginning that our license plate localization method was designed specifically for the yellow category: thus, there is undoubtedly room for further improvement when considering those.
Moreover, other struggles we have encountered along the way while developing the functionality were regarding the presence of other yellow objects in the video frame. Thus, it was more difficult to correctly identify which of the objects was the actual plate.
Despite this, we have managed to overcome this issue by establishing a minimum width value: in this way, objects that do not have the width at least 80 pixels, are not taken into consideration. We chose this margin since we were told from the requirements that plates will be at least 100 pixels in width, and we are ignoring the blue part of the plate when calculating its width.

3 Character recognition method

3.1 Recognition data description

We used the cropped characters from the first 28 license plates for "training" our algorithm. The training / validation data was split similarly to the localization data: 70% used for training, 30% used for testing.

For annotations, we manually annotated each license plate and compared how many of the characters match in order to get a score for the plates. After implementing the part that keeps track of the frequency of the plates, we changed these scores from $\frac{correct\ characters}{total\ characters}$ to 1 or 0 depending on whether or not our algorithm detects all of the characters on the license plate.

3.2 Recognition system

3.2.1 Character localization system

For localizing the characters within a license plate, several steps have been taken, as follows:

1. The license plate was sharpened, then contrasted and thresholded to obtain a binary matrix.

Figure 7: Thresholded plate.

2. The fill algorithm was then applied (refer to Appendix A) in order to identify the regions which contain the characters. This is done by finding the extrema points: if the width and the height of these correspond to some pre-defined value, we can conclude that a character has been found.
   We chose these values to be in the range in which we expect our characters to be, so, for example max-height would be the height of the normalized plate, max-width is around $\frac{1}{8}$ of the width of the plate, since we have 6 characters and a bit of extra space. We have also used min-width and min-height to ignore the dashes or other imperfections.
3. The last step is to split the image into multiple images, each corresponding to a certain character. This is done by making use of the extrema points that resulted from our fill algorithm.

Figure 8: Plate split into multiple characters.

3.2.2 Character recognition system

In order to be able to recognize a character, we have normalized its size to match the one of a preset character. The reasoning behind this implementation choice is because afterwards, a pixel-by-pixel comparison between these two images is made.
Thus, to compute the distances between two same-sized images, the algorithm checks if they have different pixel values at the same corresponding position. If that is indeed the case, we count it as a difference. In the end, the result will be determined by the total number of differences over the total number of pixels.
In other words, if there are no differences between two images, then the resulting value would be 0, meaning those are identical.
After several trials, we have established a threshold value of 0.3: that counts for how much we allow images to differ, and still consider them identical. We also compare the image to all of the images from our preset characters data set and select the one that matches the most (has the lowest value for the difference between images) with the one from the plate.

Figure 9: Differences between pattern and our character.

3.3 Evaluation metric for recognition

A crucial step in the development of this method was computing the evaluation metrics at any step, in order to see if there was still room for improvement.
We have chosen to use the accuracy score for the numerical evaluation: thus, if all of the characters present on a license plate were correctly identified, the accuracy score would be a 1, if at least one single character has been misclassified, then the score was 0.
The overall accuracy on the testing data resulted in a score of 70% for categories 1 & 2.

3.4 Analysis of the recognition results

Although the recognition system generally produces very good outputs, there are some specific cases more prone to error. Some difficulties are encountered when the license plate contains one of the following characters:

- 3 / B / 8
- J / T / 1

Having such a similar shape, it makes it sometimes impossible for our system to differentiate between those. A further improvement that could possibly resolve this issue would be to make some changes in the training data set. We have already started doing this, by adding thicker and thinner letters, but the improvement wasn't major so we decided to focus on improving other parts of the system.

4 Analysis of system

4.1 Successes and failures

One strength of our project would be represented by the overall performance. One design choice that made this possible would be dilating the mask such that we are guaranteed that the entire plate gets recognized as a single plate. Thus, this ensures that the fill algorithm gets as input a plate that has all its parts connected. Additionally, before applying the fill algorithm, the plate was resized, which mostly contributed to a faster localization.
On the other hand, one weakness of the system can be seen when considering videos with multiple license plates in one frame: in this case, the resizing of a license plate becomes difficult. Having such a small size makes it even more blurry when we resize it, thus making it almost impossible for the system to proceed.

4.2 Future improvements

As for the future improvements, what we would consider accomplishing would be identifying the blue and white license plates as well.
One approach which would yield to good results in that sense could be implementing the Canny algorithm instead of relaying on the HSV color space. In this way, the focus would not only be on the yellow plates found, but on all of them, regardless of their colour.

5 References

Fill Algorithm:
https://en.wikipedia.org/wiki/Flood_fill.

Image rotation:
https://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/

Min Area Rectangle from OpenCV:
https://opencv-python-tutroals.readthedocs.io/en/latest/index.html

A Flood Fill algorithm

We use Fill for finding the license plates (1) and for separating the letters into multiple images (2). The basic idea behind Fill is that it finds all the white pixels that form an "island" (that are connected), we do this by pushing each white pixel in a stack and looking at all of its neighbours (which also get pushed): it is basically a depth-first search, but on a matrix. Since the images are quite big, we scale them down before applying fill.

1. For finding the license plates we scaled them down to 20% of the initial size of an image. We did this because for the localization step the position does not need to be very precise, we just need this to be able to separate the image into multiple yellow objects and to ignore the ones smaller than 80 pixels in width. Scaling it down also improves the speed of the algorithm.
2. For splitting the license plate into letters we scaled the thresholded plate down to 40% of the initial size (after some trials we concluded that this value worked the best). This required more precision than the localization of the license plate, but we still managed to improve the speed by scaling it down. After filling the different letters, we looked at the min-max coordinates and calculate the width and height of each letter. We assumed that an island is a letter when it fits within our pre-set letter dimensions.