Image Processing
License Plate Recognition System

[redacted]

1 Introduction *(0.5 pt)*

Our goal is to recognize license plates that appear in a video. To achieve this goal we designed a system that extracts frames at intervals and then passes them through a whole pipeline to verify if that frame contains a license plate and if we can recognize it correctly. This pipeline consists of first localizing any possible plates in the frame and extracting those parts of the images to pass them on to the recognition phase. In the Recognition phase a lot of operations are applied such that ideally only the characters of the plate remain in the foreground of a binary image. The next step is to segment this binary image into the different characters and then compare them to reference characters to identify which one it is. By merging the results together this gives us a full license plate. The only thing left is handling the results because the same plate can appear in multiple frames, and some results might be incorrect. By comparing scores and similarities we can make a smart decision and keep a final set of plates that will be the most accurate.

2 License plate localization method *(4.0 pt)*

2.1 License plate localization data description *(0.5 pt)*

For designing the model the *trainingsvideo.avi* video was used. A script (Annotator.py) was written to assist in creating data. The annotator picks frames from the video at a certain frequency and displays them on the screen of the user. The user than clicks on the corners of the license plate(s) to annotate the exact location(s). The annotator stores all frames as png files, metadata is stored as csv and an array containing all points is saved using *numpy.save*.

The data for training and validation is split in half by alternating frames (containing different license plates), this way both sets contain data for all categories.

2.2 License plate localization system *(1.5 pt)*

A small summary on how the localization system works: First, Canny edge detection. Secondly, finding and categorizing straight lines using the Hough space. Third, finding rectangles by intersecting lines, validating and categorizing the found intersections. Fourth, picking the right rectangles by voting and comparing size. And finally, performing a perspective transform to deliver a well formed image to the recognition system.

Yellow license plates

Because most categories contain yellow license plates, the localization system first detects yellow pixels and uses this binary picture as input for the localization algorithm. If no plates are found a gray scale image is used. The system works without this step but performs significantly better.

Edge detection

To localize the borders of a license plate we need to know where the edges are. For this a Canny edge tracking system is developed. The image is first reprocessed by gray scaling and blurring. Then a Sobel filter is applied in the X and Y direction. Next, trough non maximum suppression all edges shrink to 1 pixel width. Then a double threshold is applied for weak and strong edges. Lastly, weak edges are discarded or kept depending if they are connected to a strong line or not.

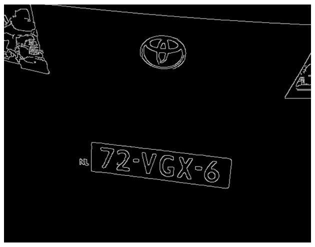

Figure 1: Canny edge detection

Finding straight edges

No matter the perspective, the borders of a license plate are always straight. This property can be used to identify which edges could be a border of the license plate. The edge frame is transformed into the Hough space. Local maxima indicate straight lines. To decrease the data size non maximum suppression is applied. Then all leftover lines are classified into a vertical set, horizontal set or are discarded based on their angle.

For each line the underlying pixel locations are calculated. Lonely pixels are discarded to remove noise introduced by perpendicular edges crossing the Hough line. Now the size of the line can be calculated by summing all pixels. The center of the line by averaging the pixels. The expected deviation can be predicted using the center point and the size of the line. The actual deviation can be calculated using the pixel locations and the center point.

All lines with a deviation exceeding a factor of the expected deviation are discarded. These lines are too noisy and the accuracy of the center point which we need for rectangle detection is too low.

Detecting rectangles

Now that the lines are classified and center points are known, its time to form rectangles. First all horizontal lines are intersected with all vertical lines. Next all intersections are categorized in corners (left up, right up, right down, left down). This is possible because the intersect locations are known and the center points of the intersecting lines are known. Also all intersections are validated by checking if the intersection lays within a factor of the deviations of the lines. This way only intersecting *edges* are kept.

Then, for each left up corner we walk along its horizontal line and look for all right up corners. This process is repeated for right down and finally to left down. The rectangle is complete when the vertical line of the left down corner is equal to the left up corner. Lastly all rectangles are validated on ratio.

Selecting rectangles

In the Hough space step, a very high angle count (around 1000 theta's) is used to perform the Hough transform. This means multiple Hough lines can fit the same line of a license plate. Due to the high angle count there are a lot of duplicate rectangles.

For all rectangles the center points are calculated and grouped. For all groups the size is counted and the corner points are averaged. Next a score is assigned to each unique rectangle. This is based on rectangle count in a group, *a vote*, and on the surface area of the rectangle. Bigger rectangles are favored because often characters in the license plates can have straight lines.

Now we evaluate the unique rectangles iteratively starting at the highest score. All rectangles overlapping with the highest score are discarded. Then we compare all plates to the second highest rectangle. This process is repeated until all rectangles are evaluated. The result is a few non overlapping rectangles with the highest probability of being a license plate.

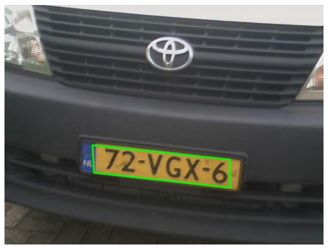

Figure 2: Canny edge detection

Extracting license plates

In the last part of the localization system the license plates is extracted and warped via a projective/perspective transform. A transformation matrix, obtained trough solving a linear system of equations, is used to map each pixel.

2.3 Evaluation metric for license plate localization *(1.5 pt)*

To numerically evaluate the localization system the validation data discussed in section 2.1 is used. The points found in the *selecting rectangles* stage are compared with the points in the validation data. A score is obtained trough a intersection-over-union operation. When no plate is recognized a score of 0 is assigned. All scores are averaged into one overall score for the current state of the system. This way each score has influence but frames without detected plates have the biggest impact.

The runtime is recorded as well.

2.4 Analysis of the license plate localization results *(0.5 pt)*

After evaluating the localization system a detection rate of 80.7 % was achieved. With an average intersection over union score of 0.69 where not recognized plates achieved a score of 0.

Challenges with the design and performance of the localization system lay in the trade off in runtime and memory usage. Due to the language requirement (python) the hough detection and other algorithms run slow. To overcome this problem numpy was used to vectorize nested loops. Though prearanging data in often large arrays increased the use of memory considerably. Often the best performing setups (high theta count, and large deviation errors) were the slowest and had often MemoryErrors due to arrays getting too large.

The localization system can be improved by choosing a faster language like c++. Or by using techniques like a CNN or other machine learning techniques.

3 Character recognition method *(4.0 pt)*

3.1 Recognition data description *(0.5 pt)*

For designing the model, we used the same data that was annotated during the localization process. There we had already set aside a list of corners in frames that corresponded to plates. Instead of using them to verify the accuracy of localization we now used them to pass perfectly extract plates from frames to input them in our recognition process. Then we extracted the truth values out of *groundTruth.csv* to be able to test if recognition worked correctly. The data for training and validation is split in half by alternating frames (containing different license plates), this way both sets contain data for all categories.

3.2 Recognition system *(2.0 pt)*

3.2.1 Character localization system *(1.0 pt)*

The goal was to first get to a binary image with only the characters in the foreground. To achieve this some pre-processing was applied that consisted of first scaling the plate to a fixed size to keep consistency, and then convert to gray scale as well as apply contrast so that the characters would distinguish themselves more. The image is now ready for adaptive thresholding to obtain our binary image.

Adaptive thresholding is similar to a simple thresholding but instead of applying a single threshold on the entire image, we determine on a small region of pixels what the best option for that region is. This way we get different thresholds for different regions of the same image.

Since adaptive thresholding is not perfect, a blur operation and a closing operation is performed on the result of it to get rid of any gaps that could have incorrectly occured. The image is now ready to begin localizing the characters. To achieve this, the system uses an algorithm called the Hoshen-Kopelman algorithm.

The Hoshen-Kopelman algorithm uses the Union-find algorithm to partition the pixels into different clusters. The algorithm scans over all pixels that have a value and labels them accordingly. This will separate the characters into different clusters and help us eliminate and noise in the image. This is also where we get rid of the hyphens. We have decided to add hyphens manually afterwards by keeping track of distances between characters and following rules instead of recognizing them since they look a lot like noise.

To finally extract the different characters in the image we create a bounding box for each cluster and then extract that part of the image which gives us the wanted character.

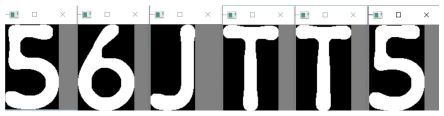

Figure 3: Example of character Localization

3.2.2 Character recognition system *(1.0 pt)*

The localized characters are first and foremost normalized. By this we mean converting them to a fixed format so that we can compare them to the references we have stored. We first try to scale them to the format that we want without introducing any distortion and then pad with black pixels so that it matches the size if needed. Then for every character in a plate we compare them to the reference characters we have stored. These are divided in three sets: reference characters that were provided to us initially, US characters and UK characters (both found online). The method used to compare is Intersection-Over-Union. What it does is that it computes the intersection of both characters and divides that by the union of both characters and this gives us a maximum of 1 if both characters perfectly match and a minimum of 0 if they have no points in common. Using this method, we select the character for which we had the highest result and use that in our final result.

3.3 Evaluation metric for recognition *(1.0 pt)*

Using the annotated data that was set aside as explained before, we can pass along the selected test plates to the Recognition process and compare its accuracy with the ground Truth. For the numeric evaluation we have two metrics: one is the percentage of license plates recognized completely correct. The other is the percentage of characters recognized correctly.

$$plateAccuracy = \frac{CorrectlyRecognizedPlates}{TotalNumberOfPlates} \tag{1}$$

$$characterAccuracy = \frac{CorrectlyRecognizedCharacters}{TotalNumberOfCharacters} \tag{2}$$

3.4 Analysis of the recognition results *(0.5 pt)*

Results:

- Overall: 41% plates correctly recognized and 59% characters correctly recognized
- Cat 1 and Cat 2 only: 61% plates correctly recognized and 79% characters correctly recognized

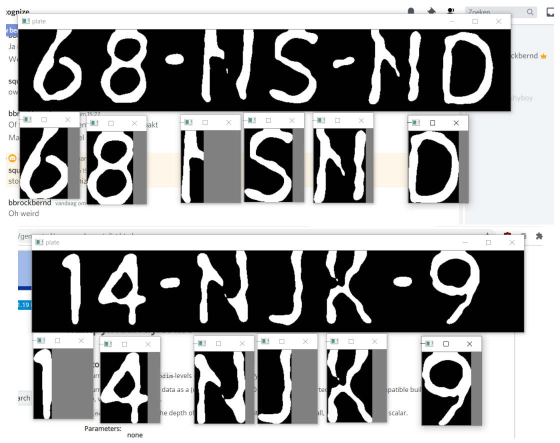

Figure 4: Fail vs Success during character recognition

The system often struggles with letters like X, N, and H because they tend to have gaps along the lines. Also B is very often mistaken by an 8 or 0 and vice versa. The actual recognition method we have now might not be good enough. This could be improved by replacing Intersection Over Union by something like SIFT for example.

The system also struggles a lot with international plates, we can clearly see this by looking at the results above. It happens probably because we don't have the right character references to correctly recognize them. Could be improved with a better dataset.

4 Analysis of system *(1.5 pt)*

4.1 Time analysis *(0.1 pt)*

Average Running Time = 8 min

Plate Localization is the part that takes up the most time. There are two ways by which our system might recognize a plate. The faster one is not an issue at all but it works only on Dutch license plates. The other method however is a lot slower and can sometimes become problematic.

4.2 Successes and failures *(1.0 pt)*

When we are faced with a Dutch License plate the system performs a lot faster and is more accurate too. This is due to the fact that we can use the color yellow to speed up the localization process and also that since we know it is a Dutch plate, we know the rules the plate has to follow. This is definitely where the system performs the best. It does worse when it comes to International license plates. This can be due to the way we recognize characters that has mostly been trained on Dutch plates. The issue is that there are so many formats and finding a method that fits all is very difficult. Also having a limited set of characters to work with is not helpful when you have to recognize foreign plates that use different sets of characters.

4.3 Future improvements *(0.4 pt)*

Our version of Adaptive threshold does not seem to work to convert the plate into a correct binary image when the characters of a plate are white. Instead it selects everything else but those characters. This can definitively be an improvement point that can better the accuracy of the system. Another way of comparing characters than Intersection Over Union is also what we would have changed. We saw that it had a lot of trouble with similar characters and this was one of the major reasons why our accuracy was not the best. A different technique could have been using a SIFT descriptor to compare characters.