License Plate Recognition: Final Poster

Introduction

In the following poster, we describe the pipeline for the automatic recognition of Dutch license plates. The problem we're trying to solve is that given an input video of parked cars we want to recognize all the license plates and the first appearance of the plate in seconds and frame number. The problem is split into 4 categories of increasing difficulty. In Category I the camera and car are stationary and only a single license plate is always in view. Category II has a person walking from car to car making the camera move. In Category III we have a stationary camera but two license plates in view further away. Category IV includes international plates.

The poster's sections follow our development process which was split into three phases. Firstly we describe the localization of the plate, secondly the character recognition and lastly aggregating the predictions across multiple frames using majority voting.

License plate localization

The following section describes the implementation and performance of the first part of the license plate detection project pipeline – localising and cutting out a license plate from a video frame. The pipeline has the following steps:

1. Preprocessing For localising the plate it is not important to maintain details so we blur the image to remove noise with a 5 by 5 Gaussian filter and then with a median filter. To localise the license place we start by converting the image into the HSV spectrum.
2. Segmentating licence plates All orange license plates should have a similar hue yet different saturation and intensity e.g. due to different lighting conditions. We therefore create a binary mask where each pixel is true if its hue is between 15 and 30. We then apply the morphological operations of erosion to eliminate any isolated orange pixels and dilation to fill in holes in the mask.
3. Contour finding To find clusters of pixels we used the SUZUK [2] algorithm implemented in OpenCV's findContours function. It runs a breadth-first search on the image mask, finding connected components/clusters of pixels. Each cluster then represents a contour.
4. Contour selection We sort the contours by their area and remove contours that have an area smaller than 1000. For categories I and II we take the largest cluster and fit a bounding box around it. If we find a second contour of similar shape and size as the first one we know we're looking at a frame from Category III. If the aspect ratio of the box is close to a certain ratio, we likely found a license plate and we crop it out of the original unblurred image.
5. Removing plate rotation For many of the frames the camera is slightly tilted so to improve the performance of the recognition pipeline we needed to get rid of it. We apply Hough Transform to get horizontal lines in the image. We start with a larger threshold of 175 and if we find only a few lines we decrease the threshold dynamically until we find a larger sample. We average the lines find their angle and rotate the plate using `ndimage.rotate`.

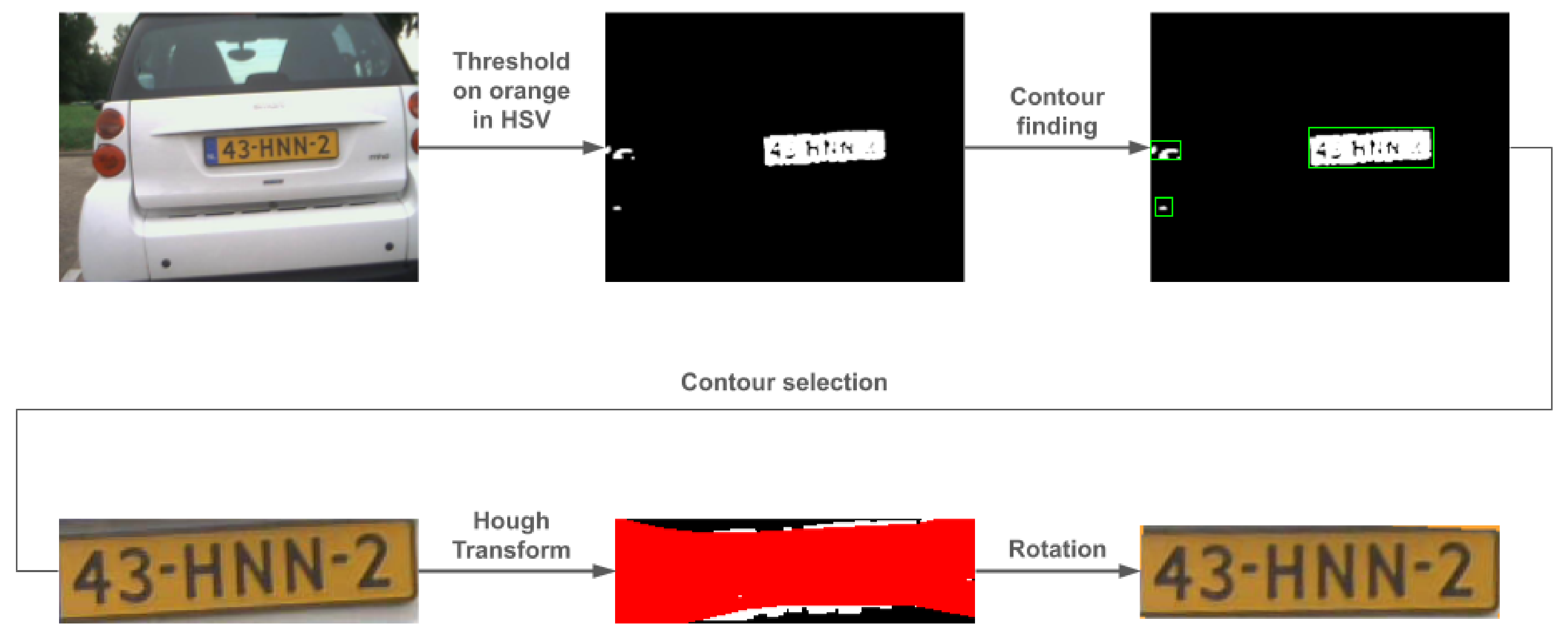


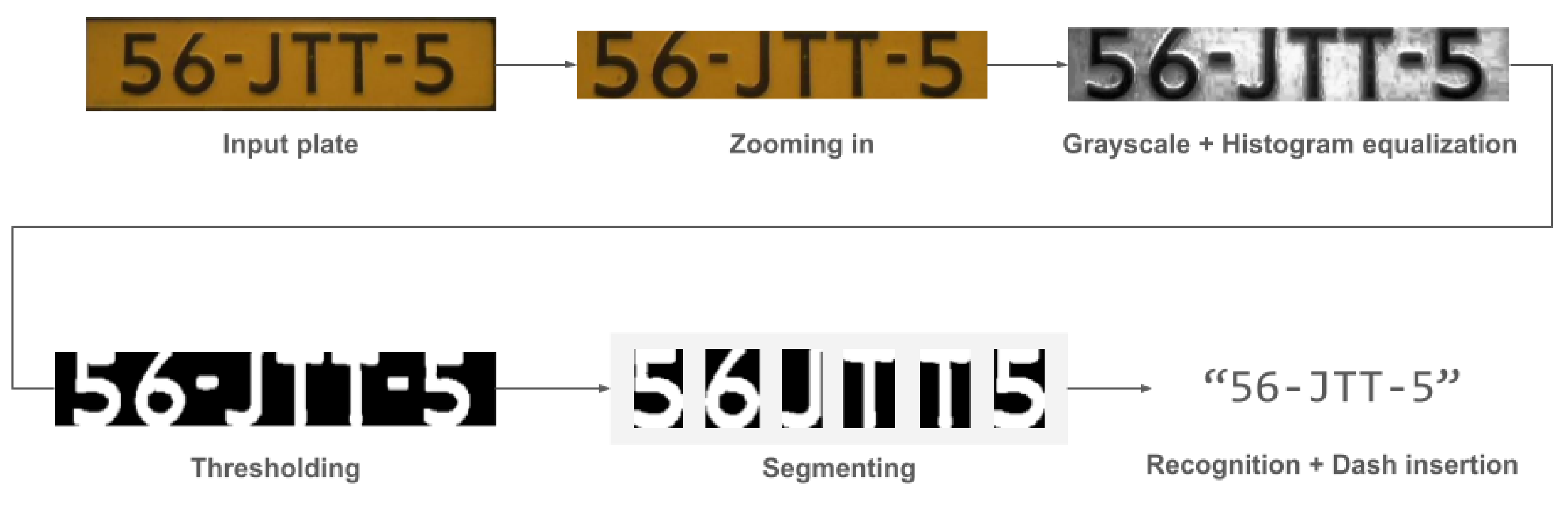
Figure 1. Localization pipeline.

To evaluate our localization approach we took a single frame for each license plate in categories I and II, giving us 60 plates in total. We randomly put 50% into our validation set and the rest into a training set. We obtained ground truth labels by manually marking bounding boxes for all 60 frames with the Label Studio[a] tool. We fine-tuned the hyperparameters such as ideal aspect ratio or minimum contour area by trial and error on the training set.
As a metric, we implemented intersection over union[1]. The localization is successful if the IoU of the ground truth and predicted bounding box is larger than 0.6. If no plate is found IoU is 0. We obtained a validation accuracy of 93.3% – 2 plates out of 30 were misclassified.

Character segmentation and recognition

This section the process of segmenting the license plate into characters and recognizing them. At the input, we receive a cropped and horizontally aligned image of a plate.

1. Preprocessing, We apply 5 by 5 Gaussian blur to remove noise. Then we scale the license plate to have a height of 70 pixels. This standardisation makes later recognition easier. We convert the image to grayscale and equalize its histogram to increase the contrast between the text and the characters and license place background. To binarize the image we use an inverse threshold for the black characters to become foreground. Morphology is then applied to get rid of the remaining noise.
2. Character segmentation. We loop over all columns of the binarised plate image. If the sum of the column is greater than 4 we know it's a start of a character. If we have a start of a character and we encounter a column with fewer than 4 white pixels we label the previous column as the end of a character. We use these indices to crop out each character. For each, we then also perform a horizontal cropping by looping over rows and with the same method identifying the start and end of a character. We therefore get a tightly cropped character.
3. Character recognition. We perform a binary XOR between the cropped character and each template character. The template character with the greatest overlap is appended to the string.
4. Dash insertion. Upon receiving a 6-character long prediction we try to fit one of the possible formats of Dutch license plates. If the sequence of digits and letters match with one of the formats we fill in the dashes accordingly.



56-JTT-5
Input plate
Zooming in
Grayscale + Histogram equalization
Thresholding
Segmenting
"56-JTT-5"
Recognition + Dash insertion

Figure 2. Segmenation and recognition pipeline.

We created a dataset with cropped license plates by reusing correct predictions from the localisation step to simulate the conditions of the complete pipeline. If the license plate was rotated or wrongly predicted in the localization pipeline we fill it in manually. We then label all the plates ourselves. This gives us 20 plates in total. We randomly put 3 into our training set which is used for experimentation and hyperparameter tuning. The rest goes into the validation set.

[a]labelstudio.io

As an evaluation metric, we use accuracy. We obtained a validation accuracy of 56.2%. We also gained a deeper insight into what kind of mistakes the algorithm is making using a confusion matrix. We found characters were correctly detected, but problems usually arise when dealing with characters that have similar shapes. In particular, 1 and J, D and O, K and X were problematic. The accuracy when looking at characters (correct characters predicted/total characters) is 89.5%.

Majority voting algorithm

At the end of the recognition pipeline, we have several predictions for each license plate. To find out which frame predictions belong to a single car we find a plate that has 6 characters and is in the correct format, we add it to a dictionary where we count the number of appearances of each plate. We also keep a dictionary where for each plate, we record the earliest frame where we saw that plate. If we find a plate that we have already seen and that appearance was more than 5 seconds ago, the plate likely appears in 2 different videos. So we do majority voting on the plates already in the dictionary. This is done by going through all combinations of plates, and if 2 of them are similar (1 or 2 characters are different) we eliminate the one with fewer appearances. Majority voting can help deal with random edge cases because the correct character is predicted more often than not.

Results analysis

Accuracy

The accuracy on the training video ended up being 90% for Category I, 100% for Category II and 73% for Category III. We think the accuracy for Category II is higher than I's since if there's one bad angle we get multiple others which helps the majority voting. Since we increased the performance of our pipeline by taking advantage of features unique to the Dutch plates such as finding orange pixels or using Dutch plate syntax we got 0 percent accuracy for Category IV.

Time

We found that a higher sampling rate increases the accuracy. We hypothesise that it gives us more samples to do majority voting on which reduces the chance of the erroneous prediction being in the majority. We settled for a sampling rate of 12. When we ran the pipeline on the 173 seconds long training video it took our pipeline 70 seconds to finish on Intel i5-11400H and 16GB RAM.

Possible further improvements

- International plates We could make our pipeline work for international plates if we localized the plate not based on orange pixels but e.g. by running edge detection and finding clusters of edges that have dimensions like a license plate.
- Improving character recognition We found XOR method is not robust towards input characters being rotated, incomplete, or similar to other characters. Using skewed letters worsened the performance of our pipeline but perhaps we could use multiple examples for each character or use a different recognition method altogether like SVM.

References

[1] Adrian Rosebrock.
Intersection over union (iou) for object detection.
`https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`.

[2] S. Suzuki and K. Abe.
Topological structural analysis of digitized binary images by border following.
CVGIP, 30:32–46, 1985.

Vaclav Kubon, Matei Hristodorescu, Group 42