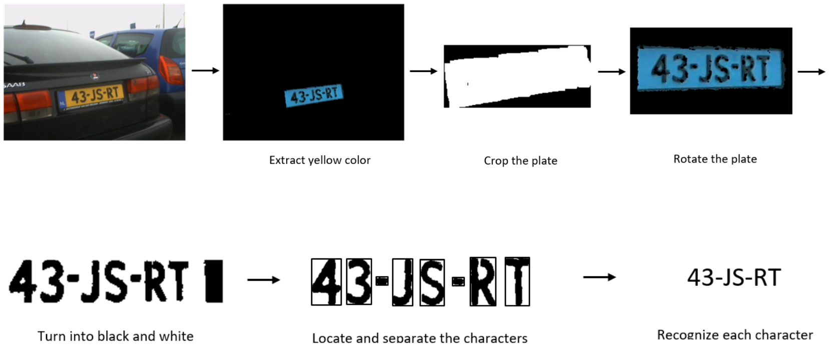
Image Processing
License plate recognition

A. Shcheglova
4917456

1 Introduction *(0.5 pt)*

In our plate recognition project, we implemented a sequence of simple steps. We start off by locating the plate in a frame using color extraction, rotate the plate if needed to make it straight. Then we convert the picture to black and white to make it easier to distinguish and extract the characters. We put the extracted characters into a list. After that we get a character from the list and compare to all the characters from the dataset, by using "bitwise and" operation. The closest match gives us the character.

Below you can see a simplified pipline of the process.



2 License plate localization method *(4.0 pt)*

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

The data was provided to us and consisted of 38 plates (for category 1 and 2). Plates of categories 1 and 2 have certain specifications, like size, rotation and color. We implemented our plate recognition method, based on this information.

We used 15 plates for training and 23 for testing. We took different plates into training set. So in our training set we had plates rotated in different directions, located on top or bottom of the picture and being closer or further from the camera etc. We had to make sure we develop a method which will work for as many cases as possible.

2.2 License plate localization system *(1.5 pt)*

The localization method is based mainly on color extraction. In order to extract the yellow color of the plate from the image, we first convert the image to hsv and then set the mask with the following values: hue [17,30], saturation [100,255], value [50,255]. This mask leaves only yellow color in the image, and turns everything else to black. Since the license plate is not the only thing that may have yellow color (for example, car lights, license plates of the cars in the distance), we proceed with erosion and dilation. Erosion removes the small patches of yellow which don't belong to the car plate, dilation will help to restore the license plate of the car. Dilation leaves almost perfectly rectangular shape of the plate. After that we crop the plate. To do so, we convert the image into grayscale image and then turn it into black and white picture (we set all pixels which are smaller than the mean value of the image to 0, all the rest to 1).

Mask applied

After erosion and dilation

After grayscale is turned into black and white

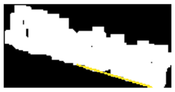
At this point we have white rectangular shape and the black background. However, the plate sometimes is not the only thing that is left in the picture, some yellow spots may still be present. We assume the plate is the biggest object we have at this point in the picture. To determine the plate, we look for the row with the maximum sum (we go row by row and sum all columns).

Then we go up and down from that maximum row, until we come across the completely black row, which indicates the end of the plate. This operation crops the plate from top and bottom. We do the same but for the columns next. As a result, we get a cropped plate (Because of previously applied dilation, we don't only get the plate but also some parts of the car around it).

Now we need to rotate the plate. First, we determine the angle at which the plate is rotated. We take the black and white image of the plate from the previous step. We go from bottom line up, column by column, until we encounter the first white pixel - it indicates the beginning of the plate. Then we look at the middle. We go from bottom to the top until the white pixel is found, which indicates the middle of the plate.

Now having this two points (beginning and middle), we draw a line between them and compute the angle using math.atan2(). Knowing the angle, we know how much we need to rotate the plate to make it straight.

Cropped



Find the line to determine the angle

After rotation

Now we need to crop the image again as it still contains redundant information. To do that we take the already rotated image, turn it into hsv (to leave only the plate) and then to grayscale. We determine the mean value of the picture. Then we go from top row by row, until we encounter the row, which mean value is bigger than the mean value of the picture (that's the beginning of the plate). Afterwards we do the same from the bottom. This crops the plate from top and bottom.

Now we need to turn our grayscale image into black and white image (to make extraction of the characters easier). So first, we find the mean value of the picture again, and then turn all pixels that are less than the mean value to black and all the rest to white. This ideally gives us the white plate and black characters.

However, quite often we still have black around the white plate (which we don't need). At the training stage we found out that we can easily remove these black parts, by cropping out 1/6 from top and bottom of the plate. In case if the plate doesn't have any black, it will just remove white parts of the plate.

To make the further localization and recognition of characters easier, we want to make sure that the leftmost column of the picture is the beginning of the first character, so we crop out the left part of the

HSV to grayscale

Crop based on the mean value

Black and white

image until the start of the first character. We do not crop out the black on the right side of the plate as it won't be reached while performing character localization, and hence doesn't influence the result.

Cropped from top and bottom

Cropped out the left part

In case the height of the picture is less than 32, we enlarge the picture. It is needed to make the further character localization possible.

As a result we get a picture containing a white plate with black characters.

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

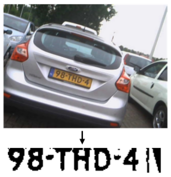
We consider the localization successful if the resulting image is a white plate with black characters. To evaluate the method, we manually calculated the amount of successfully located plates. As a result of our localization method we got 21 out of 23 (testing) plates.

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

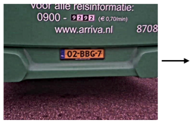
All in all, our localization method works good for the dutch yellow license plates. However, it is not perfect. This method will encounter problems if the image contains a big yellow element (like a car itself), then it won't be able to locate the plate. That is due to the assumption we make in the beginning, where we decide that the plate contains the largest yellow row, which won't be the case if the car is itself yellow. Another challenge is if a plate is more orange than yellow, in this case, the method is not able to locate the plate, because of the mask we use.

The method can be improved by using findCountours method form cv2. If we apply the findContours method and color extraction we will be able to locate the plate more reliably.

Success case



Failure case (plate is orange)



Fails when yellow mask is applied

3 Character recognition method *(4.0 pt)*

3.1 Recognition data description *(0.5 pt)*

We applied the data that we got from the plates that had the localization method used on them. These again consisted of 38 plates (from category 1 and 2). These all had different sizes still, even after applying the localization method. Which means that this algorithm was tested for bigger and smaller characters and the ability to recognize small and big characters. We trained with the same 15 plates we trained the localization method and tested with the remaining 23.

We had plates that were slanted more inwards and outwards which warped some characters and therefore had diversity in their characters because of this and other factors like light, distance and etc. We again strived to develop an algorithm that would recognize as many characters as possible in as many as possible situations that these plates were in.

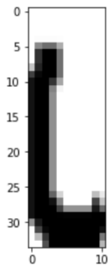
3.2 Recognition system *(2.0 pt)*

3.2.1 Character localization system *(1.0 pt)*

We take the localized plated that should now only show the plate itself with all the characters. This localized plate has an white background and black characters (this is necessary for the character localization). We first make 8 boundary boxes, because every Dutch license plate has 6 characters and 2 hyphens on the license plate. So we know there can't be more than 8 characters on the plates.

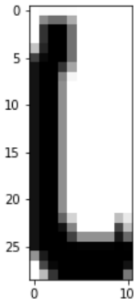
We calculate an threshold and use this one for every plate and every character on this plate. The threshold here represents the amount of rows the image has minus 5, this number is chosen after experimenting with the training plates. It first makes the bounding boxes by looking for the left and right boundaries of the character bounding box. It does this by starting at the start of the plate and checking if the amount of white becomes less than the threshold. If this is true, that means there is enough black to start the left side of the bounding box and a new search will be made for the right side of the bounding box, which is found when the amount of white is more than the threshold plus 2 (it should be more than the value that we used before, because otherwise it would sometimes stop again and have a close to zero length bounding box). If this is the case, that means there isn't enough black to contribute to an actual character, so we can safely cut it off.

Full localized plate



Bounding box with tight bounds on the left and right side of the character

We then save these characters in an separate array. And loop through all of the characters we saved again to cut them from top to bottom, to get the most tight bounding box we can manage. We do this by looking for the row with the most black pixels. We start by making an zero array that gets filled with the sum of the rows of the characters we saved (the larger the number, the more white there is again). The row with the smallest sum has the most black. We then construct the begin row by looking from the row with the most black and keep going up until we find a row that has as much white as the amount of rows in the character. This means we have found the top part of the bounding box. We also look for the bottom part of the bounding box by doing the same but going in the opposite direction and going down until we find a row which sum of white is just as much as the amount of rows in that character.



Bounding box with tight bounds on all four sides

If a character is a hyphen, then their top and bottom difference isn't that much, we calculate that and if we see it is smaller than 8 rows, then we can assume that it is hyphen and update the character bounding box to be 8 rows and 8 columns long. This makes identifying an hyphen easier.

We then save every character we have bounded this way in a separate folder.

3.2.2 Character recognition system *(1.0 pt)*

After we have made our folder with every character separated and bounded strictly. We now try to match every character and see which one matches which character the most from our data set of characters.

We begin by looking if the character is a hyphen or not. We do this by checking for the two smallest images in the folder and declaring those to be the hyphens and remembering their indexes, this works because in the character localization, we made it certain that the hyphens are the smallest characters, therefore these characters are sure to be hyphens.

We loop through all the characters in the folder we need to identify (that were bounded in the character localization step) and check which character aligns the most with the characters we have in our data set. We do this by first by looping through all the data set characters we have. Every loop pick an character from the data set. We then resize our to-be-identified character to our data set character so they are the same size. We then perform an "bitwise and" operation and compute the mean of that result afterwards. We then finally take the mean value we calculated and subtract that from the mean value of the resized character that we have yet to identify. This number gets saved in a separate array, and we do this until we have looped through all characters in the data set. This gets done for every character that needs to be identified.

After we have done that, we then check which to-be-identified character fits the most with which character. We first start by checking if the character isn't a hyphen. If it is the first character we check, then we can just pick the minimum value from the array and say that the index for the to-be-identified character is that index, which means that the character with the smallest difference is the closest to that character from the data set. Therefore it implies it is most probably that character. The to-be-identified characters afterwards get identified by looking if the previous character is a hyphen or not. We do this, because of the strict rules dutch license plates have and therefore can more easily identify the characters this way.

After all characters are identified, we then have an extra check to see if we don't have any weirdly recognized or invalid plates, like two hyphens in a row for example. If this is the case, we just return an empty string. If not, then we return a string that resembles the plate with recognized characters and hyphens.

3.3 Evaluation metric for recognition *(1.0 pt)*

We used a mean absolute error for our characters. So the average of the difference of the data set characters and to-be-identified characters. This was done for every character.

The formulae we used to compare how close the to-be-identified character was to the data set character was:

x = mean(bitwise_and(to-be-identified character, data set) character)

y = mean(to-be-identified character)

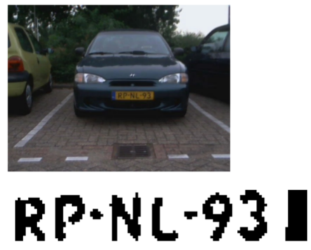
|x - y| = mean absolute error.

The closer to the zero, the more similar the characters are and therefore can be linked to the data set character they most resemble.

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

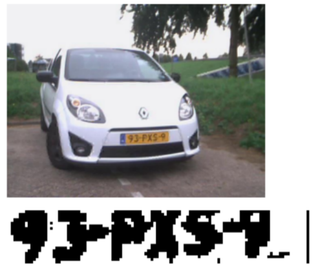
Analyze the results: show examples of failure cases and success cases.

It mostly struggles for characters that look similar in a way. Like 3 and B, both have 2 rounding parts around the same place or P and B, where both have similar forms but the only difference is the second rounding part not being there for P, but in certain angels it not being very distinguishable for the algorithm and X and K, where they in certain angles and number plates have sort of the same form (both have two "sticks" that make a character, in some angels or distances, it is harder to distinguish) and more of these similar characters. Because of these similarities, the mean value difference of these characters could be very close to zero, even though they aren't the character that the to-be-identified character actually is.



```
RP-NL-93
```

Succesful case



```
977--000
```

Unsuccessful case

It could be improved if we were able to implement an certain machine learning algorithm that would be better equipped to recognize the characters and more accurately distinguish which character it is supposed to be than use our method of the absolute mean error. Being able to use more library methods from for example cv2 would have also made the recognition quite a bit easier to implement.

4 Analysis of system *(1.5 pt)*

4.1 Successes and failures *(1.5 pt)*

It is quite an fast algorithm, running in less than 30 seconds often trying to identify the plates and generating the csv file. Problem is that it is not quite accurate depending on quite some factors like lightning, distance, the amount of yellow in the picture that is not the plate itself and how much the plate is slanted.

Also the quality of the frame that it was looking at was important. If it was a low quality, it would just discard it often, because it was difficult to read what is actually on the plate, even if the plate was correctly cropped. The characters were just not distinguishable because of it being too smudged or from information loss because of not all noise being removed after noise reduction methods.

Bounding the characters was then going to be a problem, and even if that went well, then some characters could still be very similar because of some slanting, thinner letters (from the distance that the frame of the video was) or not being properly cropped because of some noise not being able to be removed.

The system does quite well for most plates in category 2, we assume because they were at a reasonable distance away and all had around the same lightning and noise, which our algorithm could handle. Category 1 it had more troubles with, especially with if the plate wasn't exactly yellow.

The accuracy being a bit lower might be, because of our method of trying to detect yellow and it having trouble if the yellow is not exactly what we want or too abundant in the picture. And it sometimes seeing noise as valid and counting that in with the localization or recognition. Also in the recognition, using absolute mean error can cause troubles if characters are quite similar and because of some factors might lean more to similar but inaccurate character instead of the correct one.

4.2 Future improvements *(0.5 pt)*

Using an machine learning algorithm to detect the plate, bounding boxes and for the recognition of the characters would have maybe benefited us more. This could be more accurate, but taking longer than our current algorithm. Maybe an even more accurate data set, that resembled the characters on plates in general more. Even though we did this manually and think we came quite close, it can probably always be improved upon. Which would in turn have made our recognition more accurate and maybe covered our weakness of identifying characters that are more similar.