**ECE418 Final Project Report**

**License Plate Recognizer**

Kexin Hui (khui3)

Yang Liu (yliu160)

May 10, 2017

**Abstract**

License plate recognizer can be used in many aspects, such as assisting police to check whether a vehicle is registered and identify the vehicle in case that its diver fails to pay when passing the electronic toll collection on the pay-per-use road.

Our application is made up of three different components: image deblurring, license localization as well as text recognition. To start with, image deblurring technique helps to remove blurring defects that might be caused by the motion blur or defocus photography. Restored image is then passed to the next step-license localization. In this step, vertical edge detection is used to identify the location of the license plate, considering the special feature and structure of the license plate. After successfully cropping and keeping only the desired license region, we apply the character segmentation and text recognition to the image. To be more specific, we utilized MATLAB default function-*regionprops* to extract features and produce each character or digit individually. After segmenting the extracted texts, we “recognized” each single text by searching for the closest text template of the license with the aim of different strategies - Mean Squared Error Estimation (MSE), Structural Similarity Index (SSIM), and Complex Wavelet (CW-SSIM). We also try MATLAB’s default function-*ocr* to recognize the data and make a comparison with the methods mentioned above.

**Introduction**

This problem, license plate recognition, could be solved by using convolutional neural network. That is, given a set of training data of images containing license plate, we can implement a neural network to apply on the test images. The network should output the bounding box region of the license plate and give the character predicting. Yet due to our limited time and knowledge, we are not able to implement the neural nets ourselves. So we try some mathematical algorithms learnt in the class to solve the problem.

The problem can be scoped down into four subproblems: recover the image from blurring, localize the license plate region, image segmentation and character recognition.

Since the image from the road-side camera or surveillance video cameras cannot be perfectly captured, there are two main problems we need to consider. First, the image may be blurry due to the fast motion of the vehicle relative to the still camera. It would be problematic if we do the recognition on the blurry image. It is crucial to have deblur algorithm to recover a clean image. Second, the image captured cannot only contain the license plate to the full view, we are not able to rely on the entire image to recognize the license plate number. Accordingly, it is essential to localize the license plate region for better digit recognition. Yet it is quite challenging to find and isolate the license plate from the entire image due to huge variations in size, shape, color, texture, and spatial orientations of license plate regions in such region. [1] It also depends on the background, for example, outdoor lighting conditions of the scene and the viewing angle.

This localization problem can seemingly be solved by finding a rectangle region in the image as the license plate is always of a rectangular shape. However, we are not able to have a perfect image segmentation algorithm to partition the image to multiple sets of pixels which share similar characteristics or belong to the same object. In addition, many other objects in the image, for example, the front/rear window of the vehicle, may be recognized as in the shape of rectangle, which will cause a lot of problems for us to decide the true positive one. On the other hand, we notice that the characters written horizontally in the license plate region always have very strong vertical edges. Therefore, we only need to look at small regions with continuous strong vertical edges.

All the related work is done in Matlab.

**Dataset**

Our testing dataset is downloaded from the Medialab LPR database which provides a common testing set to researchers working on License Plate Recognition (LPR) problem. The data we are using in this project is a small sample set of color vehicle images that contains 35 image samples.

**Method**

1. **Blur & deblur**

In this section, we set up a model for solving the possible blurry image of the license plate by exploiting the point-spread function (PSF) and blind deconvolution.

Firstly, we simulate a blur by convolving the image with a Gaussian filter using the default MATLAB function *imfilter*. The Gaussian filter represents PSF. However, in practice, we do not have the pre-knowledge about the real size of the PSF. Therefore, in the next step, we may need to uniformly explored the possible sizes of the PSF within a certain range, and search for the optimal size by comparing their corresponding error estimation such as the mean squared error (MSE) estimation. After finding the true size of the PSF and pass this parameter to MALTAB’s default function-*deconvblind*, we would still see that the ringing effect is very obvious in the image. In the following step, we suppress the ringing effect by specifying a weighting function. To begin with, we utilize the function *edge* to find pixels with high contrast. After that, we use *imdilate* function passing in a structuring element, and then try the deconvblind function again while calling with the constructed weight array. After 40 iterations, the ringing is greatly reduced.

1. **Localization**

After obtaining a clean image from the first step, we need to localize the license plate from the input image. At first, we need to do some preprocessing work in order to normalize the image for later processing.

1. *Gray Scale Conversion*

As we know, luminance is far more important than chrominance in distinguishing visual features. We need to convert the image to grayscale version using the default Matlab function *rgb2gray*.

1. *Median Filter*

Median filter is a non-linear filter, which can effectively remove salt-and-pepper noise without destroying the edge information. Median filter will run through the image pixel by pixel and replace each pixel with the median of its neighboring pixels. The Matlab function we are using is *medfilt2*.

1. *Image Binarization*

Binary image is required since we want to have the edge information to better detect the digit. In addition, we also want to filter out some unrelated information in the image. We are using simple thresholding for binarization, and the threshold value is set to 120.

1. *Vertical Edge Detection*

Conceivably, the characters written horizontally in a license plate always have very strong vertical edges. Sobel filter is used for vertical edge detection. Sobel edge detection algorithm works by amplifying changes in its neighboring pixel values. In terms of strong vertical edge, the pixels that make up the edge will differ greatly from the ones left and right to them. It is worth mentioning that the output, the vertical edge map, is a logical array where 1 indicates discontinuities in image brightness and 0 indicates smooth transition in image brightness.

After getting the vertical edge map, we continue to find and isolate potential license plate regions. Again, these potential regions will have continuous strong vertical edges. We are interested in those 1 locations which imply strong vertical edges. There are in total four stages to determine the license plate region from coarse to fine based on paper [1].

1. *First Stage: Select Rows*

For each row, we count the number of 1 in vertical edge map. If the count is greater than certain threshold value, it means this row contains a lot of strong vertical edges. In other words, this row is very likely to be a license plate region. Next, we calculate the mean and variance for each potential row with their count greater than threshold from the vertical edge map. If the variance is continuously larger than a threshold value, a band region is selected with *top* as its starting row and *bottom* as its ending row. It makes senses since the license plate region will appear somewhere in the row, not to the full extent of the row. Thus, the variance will be somewhat larger with 1 centered at a certain region, and 0 elsewhere. In the end, we will have several potential license plate band regions. Both the count threshold and variance threshold value are obtained through the observance of our own dataset.

1. *Second Stage: Select Columns*

For each potential band with selected top and bottom row, we calculate the mean and variance for each row. Note that it is not the same mean and variance we get in the first stage. Instead, we want the mean and variance of the locations of 1 for each row in one band. Then, we will have a potential left and right bound for each band, given by

and

where denotes the mean, and denotes the variance.

The initial license plate region is found, which indicates the maximum region in which the license plate can appear.

1. *Third Stage: Refine the Regions*

For each initial license plate region, from left to right, we find the first significant vertical edge as left; from right to left, we find the first significant vertical edge as right. That is, if the number of continuous vertical edges from top to bottom in one band is greater than certain threshold value, it is said to be a prominent vertical edge and we will move the left/right boundary to this. The threshold value for this significant number of edges is decided based on our own dataset.

1. *Fourth Stage: Finalize the Regions*

According to our preliminary knowledge of a license plate dimension, we can rule out some false positives. First, the height and width of this region should fall in some reasonable range. Second, the aspect ratio denoted by width divided by height should also fall in some reasonable range.

1. **Image segmentation**

After plate location, we got cropped images most of which got the license plates of interest. In this part, we need to segment each individual character or digit. The main steps are shown below.

1. *Image rescaling*

The image we got after cropping is rather small, which makes it hard to recognize the desired object. Therefore, we rescale the image samples to 200x600, which keeps the height to width ratio of the license plate.

1. *Blur Image*

Since the image noise cannot be fully removed, the noise on the license template may be considered as part of the text while we try to convert the image from RGB to gray scale in the following step, which would easily lead to misclassification in the text recognition section. Thus, a Gaussian filter is needed to filter the image and blur the noise.

1. *Binary Conversion*

Resulting from the fact that luminance is more important dealing with distinguishing visual features in image processing, as well as we need to *bwlabel* in later procedure, we convert the image to binary version here.

1. *Removing Small Objects*

After transforming image version, we notice that noise becomes more obvious under gray scale. For the sake of avoiding such interference, we remove single object with fewer than 50 pixels. This would not discard our character or digit by mistake since the area of interest is much larger than 50 pixels.

1. *Label Connected Components*

The MATLAB default function-*bwlabel* that can only be used under gray-scale condition helps us to separate the unconnected components, and gives back the positions of the connected components in the image.

1. *Bounding Boxes*

In this step, we use label the connected objects in the image with a bounding box of which vertices are given from the above procedure.

1. *Re-decide Bounding Boxes*

Under normal condition, the region of the desired character or digit is of a standard size. By utilizing this property, we can get rid of the impossible candidates provided by the bounding boxes. Here, we take advantage of the median value. That is to say, if both the width and height of the boxing boxes is within pre-defined deviation from the median width and height among all possible boxes, then we keep these candidates. We also check the width to height ratio that should fall into a reasonable range.

1. **Text recognition**

We have tried different methods to detect the character and digit in the license plates, and compare them by checking the similarity with regards to the template we create earlier.

1. *Text template*

We make 26 English character templates and 10 digit templates, which are specifically used in our testing license plates.

1. *Testing with OCR*

OCR stands for optical character recognition. It could covert images of typed text into machine-encoded text. Meanwhile, it is a MATLAB tool that we could directly use without further processing. We simply implement it by calling the default function-ocr with extracted text region as its only input parameter.

1. *Testing with Different Similarity Methods*

Such method is different from testing with OCR, since it requires to set up the template, while on some degree, the accuracy is highly dependent on the templates we choose to use. In this method, we compare the text we find within the bounding box with each template by calculating their corresponding MSE or SSIM index or CW-SSIM index.

1. *MSE*

The MSE has been deemed as a prevailing quantitative performance metric in the signal processing area. Suppose that Y=[, ,...,] is a vector of ideal output signal values, =[, ,...,] is a vector of modified signal values with the same number of signal samples as Y, and and are the ith sample value in Y and respectively. Then, MSE criterion is calculated as below

1. *SSIM*

A very common measure used in image processing applications is the Structural similarity index, which highly considers the structures in an image instead of only calculating the error according to pixel- by-pixel differences. This measure is calculated as below, using a combination of three different parameters: luminance, contrast, and correlation.

: Mean-Luminance shift

: Standard deviation-Contrast

: Mean-Luminance shift

SSIM considers two patches of the original and distorted images, considers aforementioned three parameters for these patches, and gives the SSIM index as stated above. The result is always a number between 0 and 1 ( 0 if they are completely different, and 1 if they are exactly the same).

1. *CW-SSIM*

Though the basic SSIM mentioned above is proved to be a better option compared to the MSE in the context of image quality measurement, the drawback of being sensitive to translation, rotation and scaling should not be ignored. Luckily, the Complex Wavelet-SSIM could help us deal with this problem, since it is based on local phase measurements in complex wavelet transform domain. The local phase used in this measure contains more structural information than the local magnitude, and non-structural image distortions such as translation will cause a shift in the local shift, which could also be captured by the CW-SSIM. Hence, it is a robust measurement considering image translation, rotation and scaling, and it gives out better performance than the SSIM in this aspect.

Furthermore, for MSE, we say that the minimum value gives the best performance. Therefore, the text is detected as the specific template that has the minimum MSE, while the text should be recognized as the one with highest values in case of SSIM or CW-SSIM indexes.

**Results**

1. **Blur & deblur**

Since in real life, we do not know the true size of PSF. Therefore, we set up a model searching the proper size for PSF. After importing the guess size number into the default MATLAB function-*deconvblind*, in return it gives us the MSE between the restored and original image. By comparison, we need to find the optimal size of PSF, which can give the minimum MSE.

The size values we have tried are half the real size, the real size and twice the real size. In Table 1, it is shown that the real size can give us the minimum MSE.

Table 1 MSE Result

|  |  |  |  |
| --- | --- | --- | --- |
|  | Undersized PSF | True PSF | Oversized PSF |
| MSE | 28.417 | 21.511 | 52.329 |

As we observe the images in Figure 1, the real size indeed gives us a rather clear image, while that of half the real size is too blurry and that of twice the real size contains too many ringing effects.

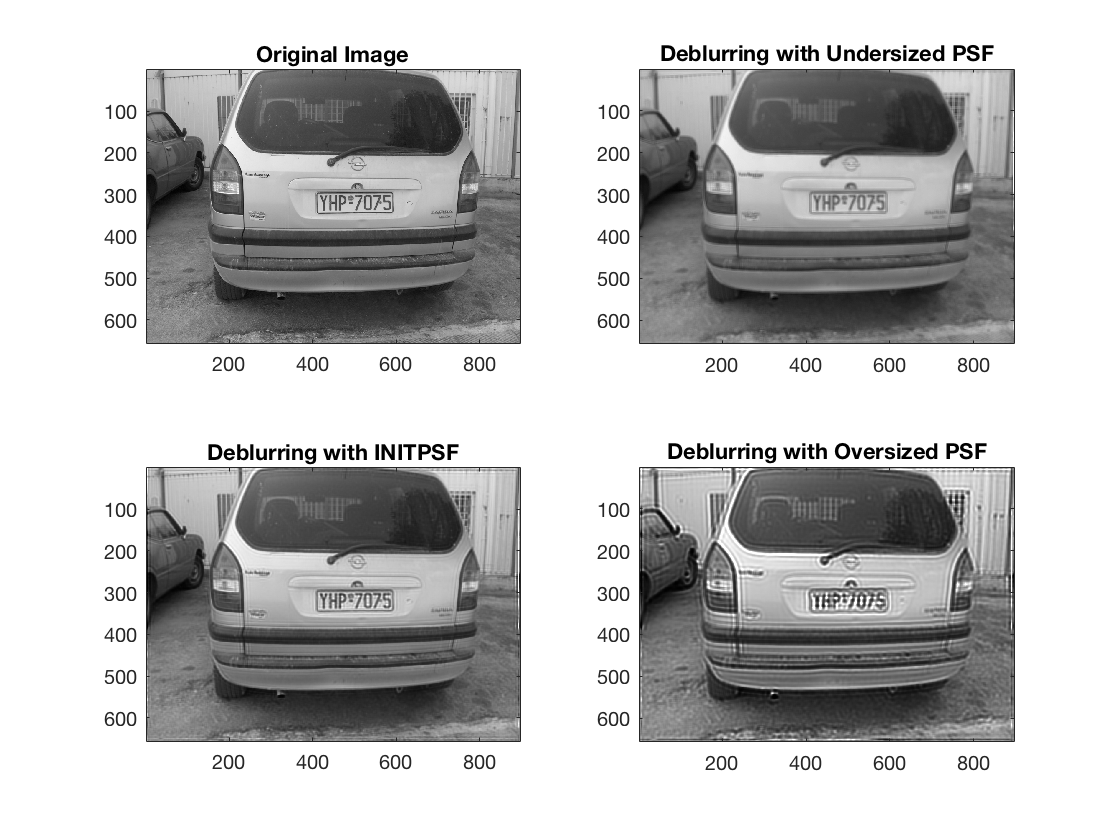
****

Figure 1 Deblurring Results

The previous result can be further improved by changing the iteration number for the *deconvblind* function. Results are shown in Table 2, it is obvious that as the iteration number goes from 20 to 40, the MSE decreases; as the iteration number keeps increasing, the MSE also increases, where we pick the rather optimal iteration number 40 for the following analysis.

Table 2 MSE Result

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Iter=20 | Iter=25 | Iter=30 | Iter=35 | Iter=40 | Iter=45 | Iter=50 |
| MSE | 15.273 | 14.222 | 13.637 | 13.281 | 13.125 | 13.1433 | 13.313 |

Original, blurred, deblurring with initial PSF size and improved deblurred images are shown in fig[]. As we can see, the improved deblurred image gives a better performance from human eyes, and its corresponding MSE equals to only 13.125, compared to the initial PSF size deblurring image with 21.511 MSE. Thus, the improved deblurred technique outperforms, and could be used in the real life application. However, in our project, in order to better testing the plate localization, we assume we could completely restore the original image.

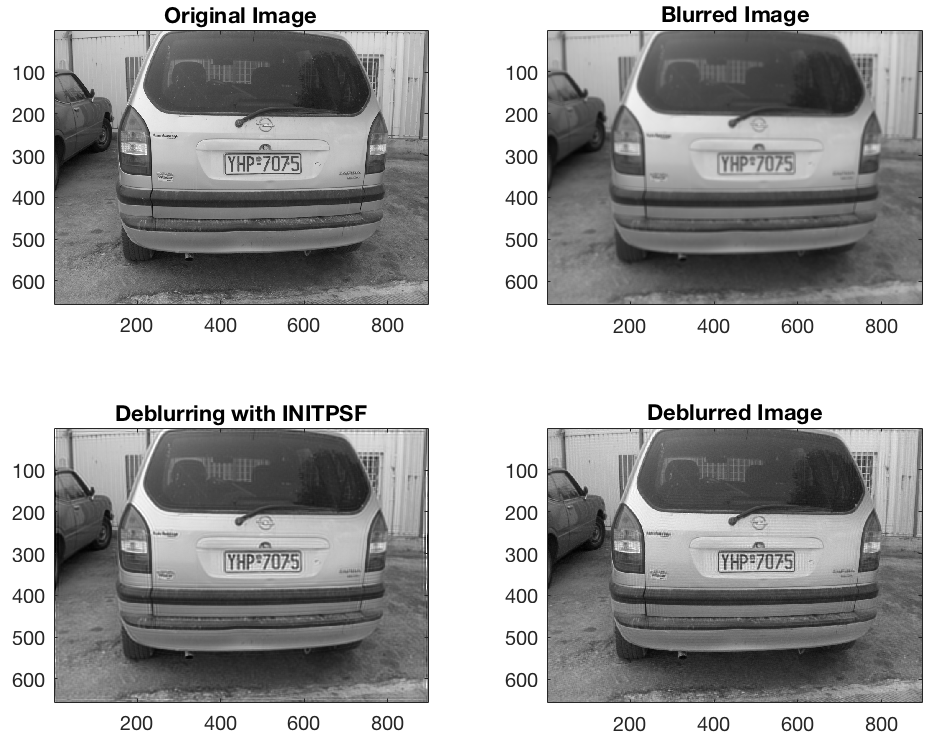


Figure 2 Deblurring Results

1. **Localization**

The method for localizing the license plate region works quite well. In general, the accuracy is 18/35 = 51.43% for our dataset. The accuracy could be improved to 27/35 = 77.14% if we do minor changes on the threshold values and constraints for specific images. More details will be discussed as follows.

We resize all the images to 600x800 for consistent parameter choosing. In terms of pre-processing, we could get a relatively clean binary image, and clearly see the license plate number in the middle image in Figure 3. In order to get a sharply contrasted image with much more obvious license plate number, we also tried histogram equalization to enhance the image after smoothing. However, this is problematic since the license plate in the foreground is usually darker in our dataset. Enhanced result would be even worse with entirely black. Vertical edge map clearly shows the locations where strong vertical edges lie. However, there are some annoying noises in the background, which negatively affect our detection result. This could be solved if we try expectation maximization algorithm or other image segmentation algorithm to differentiate the foreground, namely the car, from the background. Then we could only implement the localization algorithm on the car body.



Figure 3 Preprocessed Results

Notice that some images have much darker foreground than its background due to the lighting condition. The conventional simple threshold doesn’t work since it only sets a global threshold value to binarize the image. Therefore, we need to use adaptive thresholding to accommodate the changing lighting conditions. Based on Otsu’s method [2], we separate the foreground pixel class and background pixel class by setting the sensitivity in MATLAB to calculate the optimum threshold value. This local thresholding effectively works as the detection result in Figure 4.



Figure 4 Different Binarization Results

After pre-processing, we mainly apply mathematical model on the vertical edge map to decide the region of license plate.

In the first step, we generate two plots for edge pixels per row and variance per row for the sample image. The larger number the edge pixels per row have, the higher variance per row is, the higher probability that row is part of the license plate region. Since we resize all the input images to the same size, the parameters are quite consistent among all image data. As we can observe in Figure 5, we choose the threshold for edge pixels per row as 20, and threshold for variance per row as 0.15. These two parameters work perfectly for most of the test images. We find several continuous bands with selected top and bottom row after these two thresholding and remove some impractical results due to the dimensionality constraints. For example, band with height = 1 is ruled out.

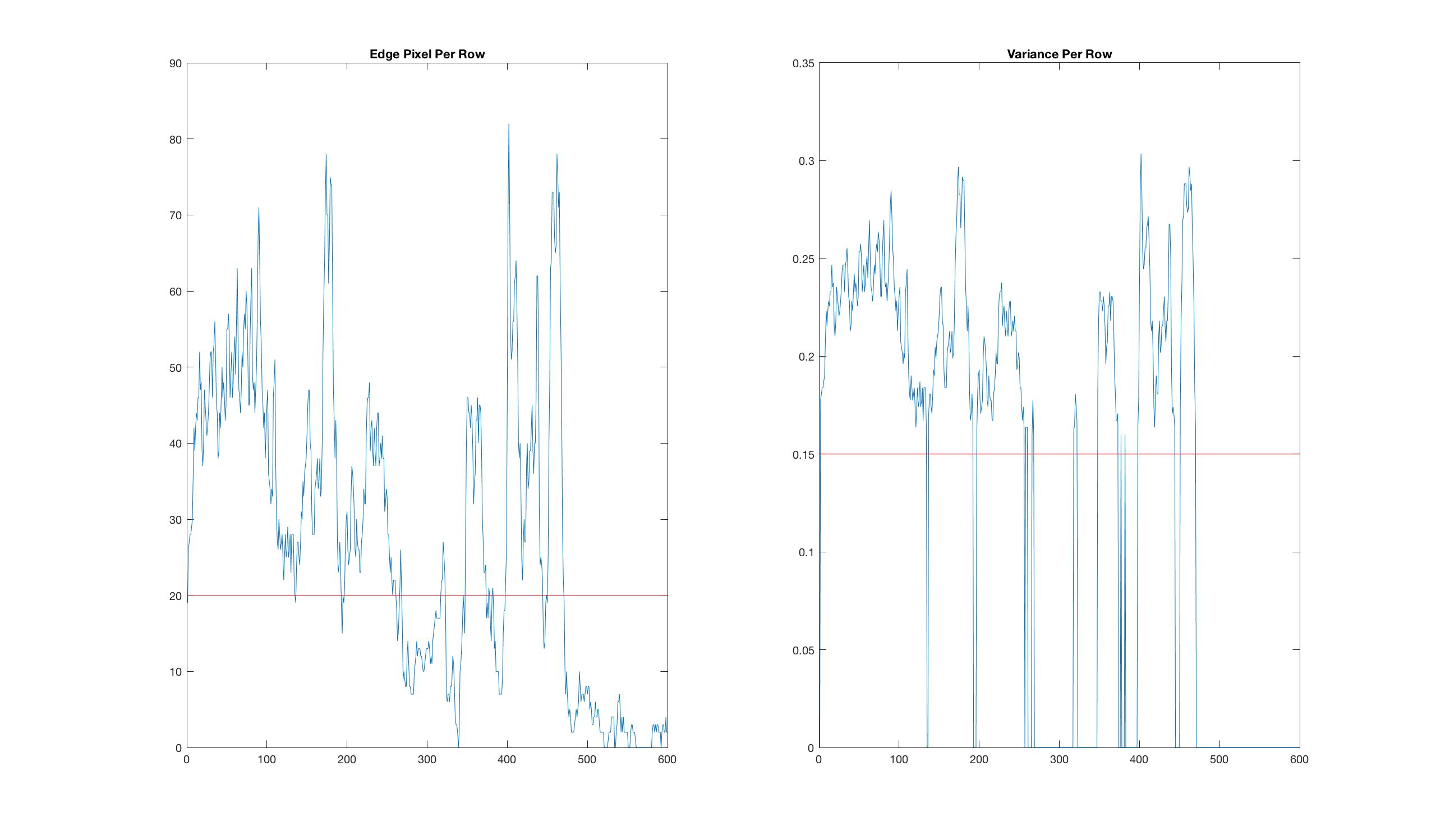


Figure 5 Mathematical Statistics per Row

For the second step, we follow the method and get potential license plate regions as expected, which is shown in the middle in Figure 7.

In the third step, we need to refine left and right boundary of each band based on our method. As proposed before, we define a prominent vertical edge as the column with number of edge pixels greater than half of its height. This works for most of the images. However, if the image is oblique with tilted license plate, we are not able to detect that prominent edge. For example, in Figure 6, we find the potential region after the first two steps. Yet the boundary is not a vertical line in this case. Thus, we would get nothing after the third step. In this case, we could possibly solve this by applying image rotation and translation to normalize the orientation on this detected region to further detect the license plate.

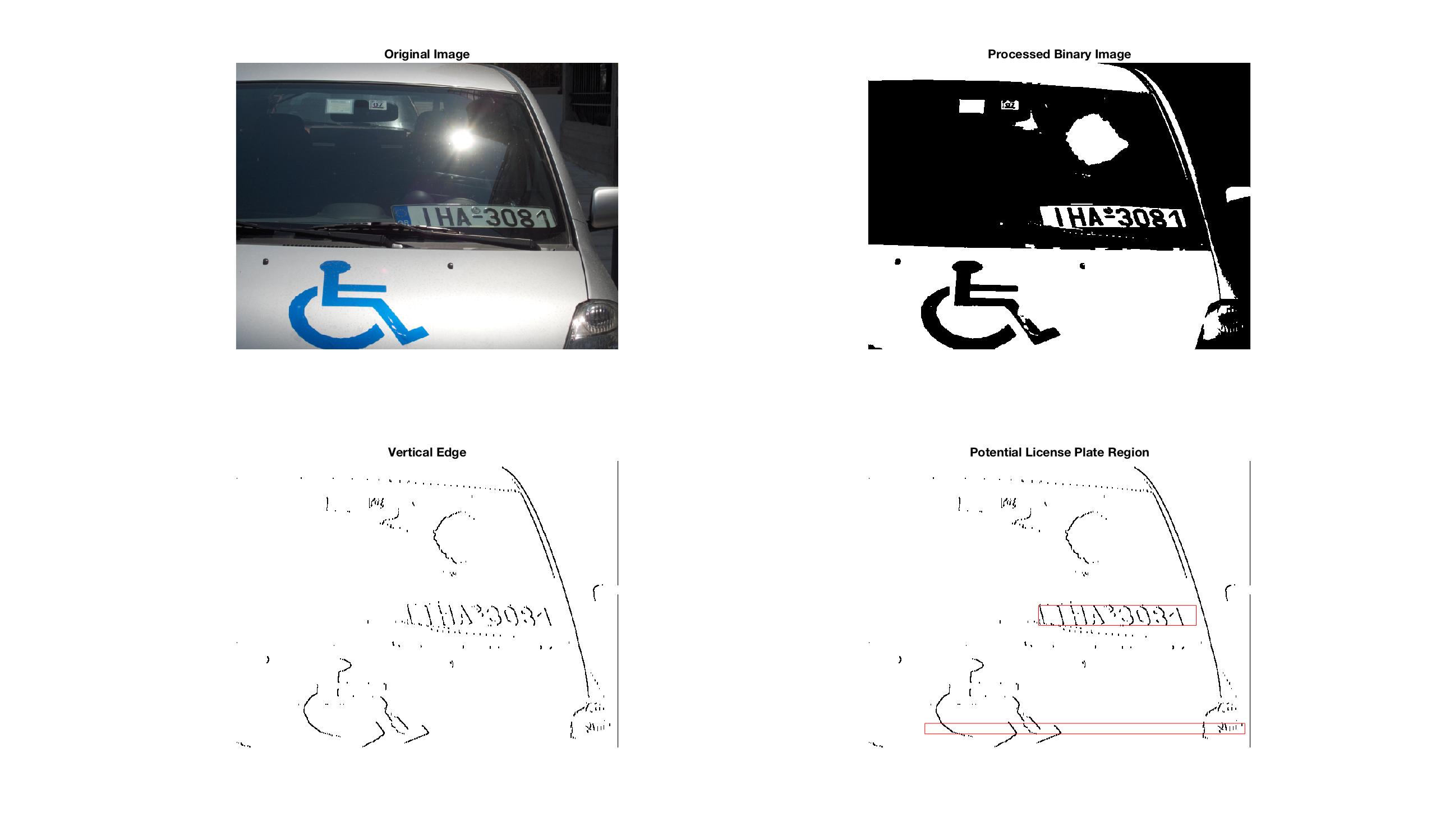


Figure 6 Tilted Image Case

For the final step, we set several dimensionality constraints to remove some false positive detection results. In details, the height of the detected region should be greater than 10 and less than one third of the image height, and the width of the detected region should be greater than 20 and less than two thirds of the image width. The aspect ratio denoted by width divided by height should be greater than 2. The final detection results are shown in Figure 7.



Figure 7 Final Recognition Result

These constraints work okay generally. Some outputs may not have perfect edges as we want. We are not able to set more restricted constraints since we could always detect some unexpected edges, for example, the boundary of the car. Even for a small number of images, we might need to relax the constraint, especially the upper bound of the region size, so that we could accurately detect the region with license plate. Figure 8 shows several imperfect detected regions.









Figure 8 Imperfect Detection

Besides that, we also encounter the problem with extra false detection, which decreases the precision of detection. However, this problem could be easily solved. It is conceivable that other detected regions are regions without license plate and they don’t contain license plate numbers. We could easily perform an OCR algorithm to detect the characters as described in digit recognition. Most often, OCR will detect nothing. Even if it detects something, the OCR results will tell us the confidence of predicting the character, and the confidence for those without characters will be considerably low around 0.1 and 0.2. Then we can simply remove the false detection by using the true positives.

1. **Image segmentation**

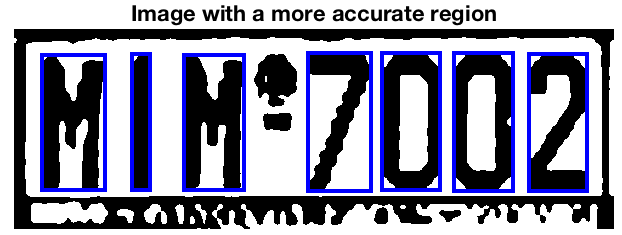
Image segmentation gives a rather good performance, and the outcome for each major step is shown below.



↓



↓



↓

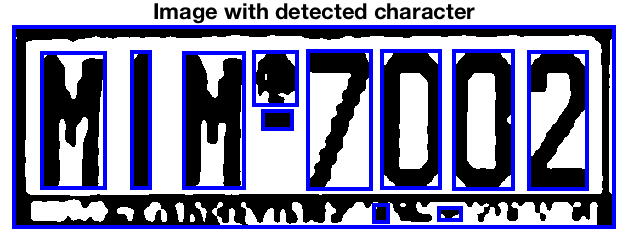


Figure 9 Image segmentation

The first important procedure is image blurring. It is used to smoothen the image, especially the edge of the text. This is because too clear image will possibly include the noise as part of the character or digit after doing the binary version conversion, which will degrades the accuracy of test recognition in our experiment.

The following major step after is deciding the threshold for converting the binary image. If the pixel value in the image is larger than the threshold, 1 (white color) will be assigned; if the pixel value is smaller or equal to the threshold, 0 (black color) will be assigned. By adjusting the threshold value, we notice that this will cause a great impact on whether we could correctly label the connected components in the image or not. Sometimes, there is shadow on the top of the plate region. In such case, the extraction will work better if we lower the threshold value; otherwise, the upper half of the plate may be totally blackened after it is converted to a binary image, in which case we can hardly identify the right symbols. An example is shown below.

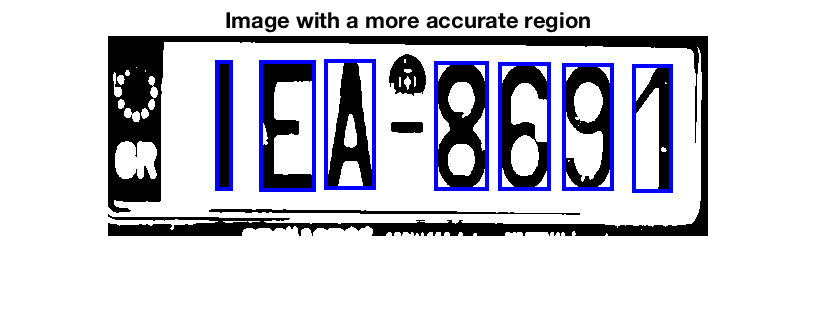
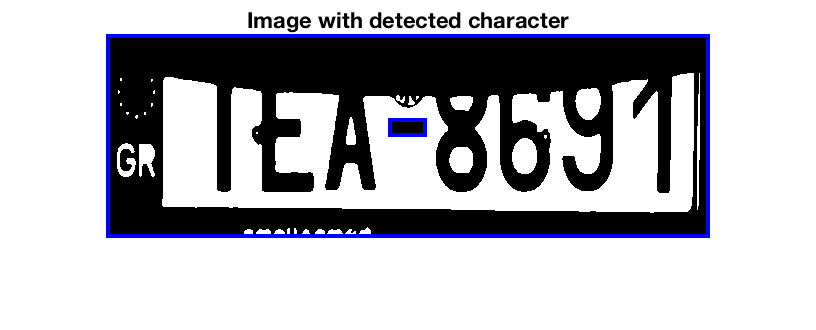


Figure 10 Problematic Segmentation Figure 11 Improved Segmentation

The last significant step is the possible candidates selection for image segmentation. After labeling the connected components, we got several potential regions that may possibly be our desired digit or character region. However, in most of the situations, the number of detected areas is more than the real number of regions. Hence, we think about using the feature of the text itself. To be more specific, after labeling, we observed that the size of the text is quite standard. Comparatively, other selected regions such as the area caused by shallow are very likely in a random shape. Thus, we decide to utilize the median value. If both the width and height of the labeled region are within pre-defined deviation from the median width and height among all possible regions, we store their position information. We also check the width to height ratio and make sure that they fall into a reasonable range. In total, this method works very well under the condition that the image is not tilted.

1. **Text recognition**

Table Similarity Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | OCR | MSE | SSIM | CW-SSIM |
| Accuracy | 43% | 30% | 50% | 65% |
| Execution time per character or digit | 0.169873529 | 0.229079412 | 0.210291176 | 1.386238235 |

In the Table 3, it can be easily seen that when only considering the accuracy of finding the correct text, the MSE gives the worst performance, while the CW-SSIM can provide highest accuracy among the all. When simply taking into account the execution time per character or digit, we notice that CW-SSIM, on the contrary, gives the slowest running time. However, OCR performs the best. The result data we got is reasonable for the following reasons:

1. *Accuracy Aspect*

MSE is the worst in accuracy because MSE only considers the square of difference between two pixels of each from two images. It is independent of any temporal/spatial relationship between original signal samples. Thus, although MSE is considered as the standard criterion for signal quality assessment and the preferred choice of engineers trying to optimize signal processing algorithms, the MSE is not always the perfect evaluation criterion, especially in image processing applications.

However, CW-SSIM is robust to image rotation, translation, and scaling, which not only outperforms the MSE, but also gives better result compared to SSIM. During the recognition, the character or digit in the plate may slightly tilted or even distorted due to pre-preprocessing steps. Therefore, it will be a good idea to try CW-SSIM at first.

1. *Running Time Aspect*

The reason that ORC has the shortest execution time is that it is a mature software tool, however, its accuracy is highly based on the structure of the testing image. CW-SSIM is superior in accuracy, however, taking into account the running time per character or digit, CW-SSIM is the worst. This is mainly because it not only needs to calculate the mean-luminance shift, standard deviation-Contrast, and Mean-Luminance shift for each pixel, it also requires to take into account of consistency of phase changes. The computation is comparatively complicated, so it takes the longest running time.

What is more, the accuracy of the text recognition is largely depends on the templates we find. If the font of the template and testing sample is different, we will find it hard to match the testing samples with the correct template. However, OCR does not have such problem. But the problem for it, as we can see from the table, is that its accuracy is much worse than that of CW-SSIM. For the license plate recognizer, we care more about the accuracy instead of the running time. Also, the running for CW-SSIM to test each character or digit is within 1.5 seconds, which is still acceptable in real life. Therefore, after the experiment, we think CW-SSIM is a more appropriate choice that should be used in our application.

**Conclusion**

To conclude, our recognition works great as expected.

The license plate localization algorithm is quite robust. We could accurately find and isolate the license plate region, ready for digit recognition. Yet this method is greatly sensitive to the conditions of the dataset, especially the parameter selection. For example, we would expect the image to be not oblique with the license plate in the right orientation. In addition, we would expect the image under the normal lighting setting with foreground brighter than background. The detection accuracy could be further improved if a much larger dataset is provided so that we can better tune our parameters. In future, we want to fix oblique cases by implementing the image translation model on oblique license plate. We would also research on image segmentation algorithm to extract car body from the image for better result.

**Bibliograpy**

|  |  |
| --- | --- |
| [1] | S. Saha, S. Basu, M. Nasipuri and D. K. Basu, "License Plate Localization from Viechle Images: An Edge Based Multi-stage Approach," *International Journal of Recent Trends in Engineering,* vol. 1, no. 1, pp. 284-288, 2009. |
| [2] | O. Nobuyuki, "A Threshold Selection Method from Gray-level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 9, no. 1, pp. 62-66, 1979. |

[3] Z. Wang and A. C. Bovik, *Mean Squared Error: Love It or Leave It? A New Look at Signal Fidelity Measures*, IEEE Signal Proc. Magazine, vol. 26, no. 9, pp. 98-117, Jan. 2009.

[4] Y. Gao, A. Rehman, Z. Wang, *CW-SSIM based image classification*, Proc. 18th IEEE Int. Conf. Image Process., pp. 1249-1252, 2011-Sep.