LICENSE PLATE LOCALIZATION USING CONCURRENT COLUMNS

Nathan Woods

Carroll College Helena, Montana
REU 2010 at Utah State University
Email: Nathan.J.K.Woods@gmail.com

1. ABSTRACT

In this paper, we propose a novel approach to localizing the license plate of an image. First, we apply the pre-processing to the initial image. Second, we apply the vertical gradient finding to the pre-processed image. Third, we localize concurrent columns to generate possible plate regions. Next, we remove geometrically impossible plates. Using Haar-like features, an AdaBoost learner classifies plate regions as a plate or not. The remaining plate regions are checked for overlapping and merged if so.

Our extensive experimental results show that even with an algorithm that produces the proper license plate 50% of the time our accuracy is 99.6% with relatively quick runtimes. With a few adages to what is already created, it is possible for this algorithm to be very effective and efficient.

2. INTRODUCTION

In modern day society, License Plate Localization and Recognition is becoming a larger part of many transportation systems. This technique is used in several systems to do anything from computing throughput of traffic structures, billing patrons for the use of certain traffic structures, or for overall traffic management. Improving this technology would allow these systems to run on higher speeds of traffic and with less cost.

3. PAST WORK

Many methods for license plate localization have been proposed. Shen-Zheng Wang and Hsi-Jian Lee [1] suggest the use of cascading rejecters, including an AdaBoost Learner trained with Haar-like features. While this method is fast and efficient, having AdaBoost classify each possible plate region takes time. While this time could be cut in more than half by simply excluding regions that do not satisfy the geometric conditions of a license plate.

Our goal is to investigate the effectiveness of our approach as compared to past work done in license plate localization.

4. PROPOSED WORK

The proposed system consists of six components.

1. Pre-Processing
2. Vertical Gradients Finding
3. Concurrent Columns Localization
4. Geometrically impossible plates Removal
5. Adaboost Learning
6. Overlapping plate regions merging

In the following sub-sections, we will explain each component in detail.

4.1. Pre-Processing

In the world of image processing, the biggest necessary evil is pre-processing. Pre-processing can reduce the amount of time it takes for an application to process an image, immensely. If a well constructed pre-processor heads a program, is should ensure high accuracy and fast program execution.

First, we convert a color image to grayscale to reduce the size of the input image array by 2/3 and combine the color layers.

Second, we shrink the image to 600X800px allowing for minimal data accuracy loss and reduce processing time of the entire algorithm.

Figure 1: Example bumper to evaluate multi-directional gradients

Figure 2: Acc clarification

Original			
0	1	1	1
1	1	1	0
1	1	0	1
1	1	1	1

(a)

Acc			
0	1	1	1
1	2	2	0
2	3	0	1
3	4	1	2

(b)

Acc-Reverse			
0	4	2	1
3	3	1	0
2	2	0	2
1	1	1	1

(c)

Finally, we apply histogram normalization to increase the contrast between the darks and lights of all elements. This gives the possibility that the difference between the highs and the lows of the license plate could be accented ever so slightly by this scaling process.

4.2. Vertical Gradients

As mentioned in (Shen-Zheng Wang and Hsi-Jian Lee's) paper on page 5 section A, Vertical Gradients are the best choice when it comes to distinguishing license plate from other objects in the image.

A license plate is traditionally located in the center of the bumper. Looking at a bumper [Figure 1], it is easy to see that scanning from the top-down (horizontal gradients) there is quite a bit of change happening all along the entire scan line. But scanning from left to right (vertical gradients) there is very little change in the bumper until the license plate is reached and then it goes back to fairly static. Since horizontal gradients are not acceptable for this application, neither is the sobel filter, because it is a combination of both horizontal and vertical gradients.

Vertical gradients can be calculated quite easily by using a filter. You can summarize this process using several steps:

1. Apply the filter [1,0,-1] to find vertical gradients
2. Compute the absolute values to keep gradient calculations positive
3. Apply Ousts method to find appropriate threshold to convert the gradients to a binary image
4. Binarize the image to contain vertical gradients

4.3. Concurrent Columns

Finding concurrent columns is just a short way of saying finding vertical columns that are of similar height and similar horizontal location in an image. In order for this method to work, some maps are generated using the output from the vertical gradients function.

An acc map is similar to a binary image [Figure 2.a], but not quite the same. An acc map is a form of vertical cumulative summation, but with the adage that every time the binary image switches back to false, the counting resets at zero [Figure 2.b]. During the coding process, I found that finding concurrent columns would also require the use of a Reversed Acc map. Which, is not the image flipped upside down, but the counting for each individual counting column reversed [Figure 2.c]. The Acc regular map allows us to find the top of a column that was crossed where the Acc Reverse map allows us to find the bottom of any column.

After generating both the Acc and Reverse Acc maps it is time to pull possible plate regions from the image. This is where the concurrent column idea comes into play. Looking at the output of the vertical gradient function, two vertical lines appear on the edges of the license plate. Another property that these particular lines have is that they are of similar height and horizontal position. Group these two lines into a possible plate run and we have a license plate. Using these two facts along with the traditional proportion of a license plate, the following algorithm was generated.

Input: Acc and Rev-Acc, where Acc is a matrix the same size as the input image and contains values of counted columns from the top down, and Rev-Acc is a matrix the same size as Acc and contains values of counted columns from bottom up.
Output: Coordinates of possible plate regions
(x, y) = search Acc for 1's
For I = 1: length(x),
  curcol = (x(I), y(I)) *(current column)*
  Grab box to right of curcol (Rev and Acc)
    *(4.5*height of curcol is the width for this box where ±5px of current horizontal position is height)*
  If (values within tolerance of value at current location)
    (looking for values within 10% of curcols height)
    Get location of values
    Use location to store top left and bottom right corners of possible plate run
  end If
Next

Once this function finishes executing, the result is an array of elements which each contain coordinates for the upper left and bottom right corners of each possible plate. 4.5 times the height was chosen because it is larger than the maximum license plate ratio of 1:4. The output of this method can be seen in Figure 3.

4.4. Remove Impossible Possibilities

We observe that all license plate's posses the following properties:

1. A plate is at least two times wide as it is tall
2. A plate cannot be more than half of the image
3. A plate cannot be smaller than 15X15px

Based on these properties, we use the following conditions to remove all the candidates which is impossible to contain a plate.

1. 2*width of region < height of region
2. Height of region > ¼ * height of input image
3. Width of region > ½ * width of input image
4. Width or height <= 15px
5. Height*Width <= 40px

The output of this method can be seen in Figure 4

4.5. AdaBoost classification of plates

AdaBoost is a machine learning algorithm [2]. Meaning that AdaBoost is a package designed to allow computers to learn from the features of data input. In this project, AdaBoost was used to take possible plate regions and classify them as a Plate or not.

In order to come up with data to train the AdaBoost package with, the program runs through all the steps covered thus far and passes the output into the following training algorithm.

At this point there are many sub-sections of the image, some of them plates and some not. A human is used here to classify them as a plate or not. This data is stored in a separate array called labels and is used for the training of the AdaBoost learners. Using Haar-like features, we can see how an image changes.

Figure 3: Concurrent column output

Figure 4: Impossible possibilities output

Figure 5: AdaBoost Learner output

Figure 6: Final Program Output

To extract the features, 10 different filters are used to obtain sums from the possible plate region [Figure 7]. Since the image is size-normalized, as long as the same filters are used for finding the differences, each filter should have the same size of output data. This allows all the data to be concatenated together in one long array. Each of the 10 filters gives a specific difference that aids in the classifying process. Type 1 filters assist with finding edges. Type 2 filters assist with finding lines. Type 3 filters help find specific identifying marks, such as bolts or tags. Type 4 filters assist to find text on the plate.

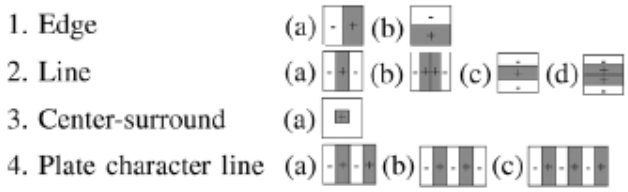

Figure 7: Explanation of Haar-like filters [1]

Now, there is a large array of data generated in the paragraph above, that holds all the differences for each plate and an array for true or false values stating if the differences correspond to a plate image or not. This data is used to train the learners. The output for the trained learners is an array of learners and an array of weights.

The process for returning true or false images during the runtime of the program is fairly similar to the

training process. Everything is the same except for the human interaction, and the training. The program takes the saved weights and learners from the learning process and the differences that were re-calculate for each possible plate in each run of the program, and passes them into a classifying function. This classifying function tells the program whether or not the region in question is a plate or not. Removing the non-plates from the plate holder array the program moves forth. The output of this function can be seen in Figure 5.

plate over all	non-plate over all	true positive	false positive
332	145440	150	294
		false negative	true negative
		182	145146
TPR (%)	FPR (%)	Accuracy (%)	Precision (%)
0.451807229	0.002021452	0.996734627	0.337837838

Figure 8: Statistics

4.6. Merge Overlapping Plate Regions

Since the plate candidates may overlap to each other, we apply a plate merging function to merge plate regions that were overlapping considerably and return one highly possible plate.

This algorithm starts working by creating a blank mat, the same size as the input image. Each plate run is handled in order. First we look on the mat where the plate run is from. If there are any numbers there, we find which ones edges are closer to the image border (bigger region) and re-adjusts the plate to accommodate for both (or more) of the intersecting plates. Finally we place the index of the plate currently being tested, onto the mat. This process is repeated until all plate regions are dealt with. The final output of the algorithm can be seen in Figure 6.

5. EXPERIMENTAL RESULTS

Upon initial statistics [Figure 8], the results do not look anywhere near what the paper had. But upon further statistical analysis, a different outcome can be gathered. Two different sets of images were used for the testing, images used for training the learners and images that were not used for training the learners.

The training data used for the AdaBoost learners were 48 images of vehicles with license plates in them pulled from the internet that were of assorted sizes and cropped license plate regions of size 431X214px [Figure 9].

Figure 9: Training data 1, Training data 2, Testing data

Terminology from table:

- Plate over all: # of license plates
- Non-plate over all: # of non plate regions
- True positive: # of true plates program returned
- False positive: # of false plates program returned
- False negative: # of true plates program rejected
- True negative: # of false plate that program rejected
- TPR: True plate ratio
- FPR: False plate ratio
- Accuracy is the degree of veracity while precision is the degree of reproducibility.

The accuracy is calculated with this formula [2]:

$$\frac{\text{True}_{(+)} + \text{True}_{(-)}}{\text{True}_{(+)} + \text{False}_{(+)} + \text{False}_{(-)} + \text{True}_{(-)}}$$

The precision is calculated with this formula [2]:

$$\frac{\text{True}_{(+)}}{\text{True}_{(+)} + \text{False}_{(+)}}$$

Having a high accuracy is very good, but with low Precision and TPR, this algorithm needs a little bit of work. Some things that can increase the TPR and Precision of this algorithm are as follows.

6. FUTURE WORK

In the future, we will improve the system from the following perspectives:

1. Modify the Acc counting in order to accommodate for gaps in the data and neighboring columns

2. Have all variables in the concurrent columns localization be percentages of current column
3. Adding more Haar-like features to the AdaBoost training process
4. Suppress non-plate regions and re-run the program to remove problems with Ousts method

7. CONCLUSIONS

Besides creating an entirely new way to grab possible plates from an image, this algorithm adds a rejecter based on geometric qualities of a license plate and a merging method that ensures a single plate region is returned. These additions to the traditional algorithm of license plate localization have the possibility to decrease runtimes, while increasing accuracy if implemented in an existing system or by pushing forward on the items mentioned in future work.

8. REFERENCES

[1] Wang. Shen-Zheng, "A Cascade Framework for a Real-Time Statistical Plate Recognition System," IEEE Trans. IFS, vol. IFS-2 no. 2, pp. 267-282, 2007.

[2] - wikipedia.org