License Plate Number Detection Algorithm for Qatari License Plates

Hermione Granger
Hogwarts School of W and W
Scotland, UK

Rancho Chanchad
Imperial College of Engineering
Pune, India

Saquib Razak
Carnegie Mellon University
Doha, Qatar

ABSTRACT
As traffic accidents increase due to over speeding and other traffic violations, it becomes important for law enforcement agencies to find technological solutions to detect these violations. In Qatar, the government has placed high resolution cameras on various roads and highways that detect speeding motorists and take a picture of their vehicles. These images are then processed to detect the license plate numbers and issue tickets accordingly. This paper describes an algorithm to extract and determine digits from a Qatari license plate.

Categories and Subject Descriptors
I.4.6 [**IMAGE PROCESSING AND COMPUTER VISION**]: Edge and feature detection.

General Terms
Image Processing, Optical Character Recognition,

Keywords
License Plate, Qatar, Arabic Numerals.

1. INTRODUCTION
The issue of traffic accidents is becoming an ever increasing phenomenon especially in rapidly developing countries. Several different approaches have been adapted by governments of these nations to bring traffic violations under control. These approaches include better awareness among drivers, better driving education, and strict penalties for traffic violations. In Qatar, the local administrators have installed traffic cameras and sensors around major roadways and highways. These sensors are triggered by cars that are going over the speed limit or breaking through a red signal. The cameras then take a picture of the license plate of the offending car. The image of the license plate is then processed through a software that determines the number on the license plate. This number is then searched through the database of all registered vehicles and appropriate people are levied with fines. One of the most important parts of this system is the software that detects license plates and determines the plate number. In this paper we will present one such algorithm that takes as input the image of a license plate and outputs the license plate number.

2. Qatari License Plates
There are two main kinds of license plate designs that are used on private vehicles in Qatar. Figure 1 and 2 shows these two designs.

Figure 1: License Plate Type 1

Figure 2: License Plate Type 2

In this paper we will concentrate on Type 1 license plates only.

3. Image Processing
There are five primary steps that are required for extracting characters from a license plate, the following sections describe each phase.

3.1 Plate localization
This step involves finding and isolating the plate on the picture. When the traffic cameras take picture of a moving vehicle, often the license plate is embedded as part of the image. The part of image that contains the license plate needs to be extracted and isolated in order to avoid unnecessary noise in the image being processed. For the purpose of this paper, we will assume that all the images being processed have already gone through this phase and we have access to extracted license plate images.

3.2 Plate orientation and sizing
This phase compensates for the skew of the plate and adjusts the dimensions to the required size. Again we assume that the images we use have gone through this phase.

3.3 Normalization
This step involves adjusting the brightness and contrast of the image. We will discuss algorithm to perform this step in detail in section 4.1

3.4 Character segmentation
This phase finds the individual characters on the plates. Once the characters have been segmented, we can go through individual characters to determine their value.

3.5 Optical character recognition

This phase takes a single character and determines its value. We use feature extraction and then statistical techniques for this step as mentioned in section 4.3. Figure 3 shows a pictorial view of the first 4 steps.

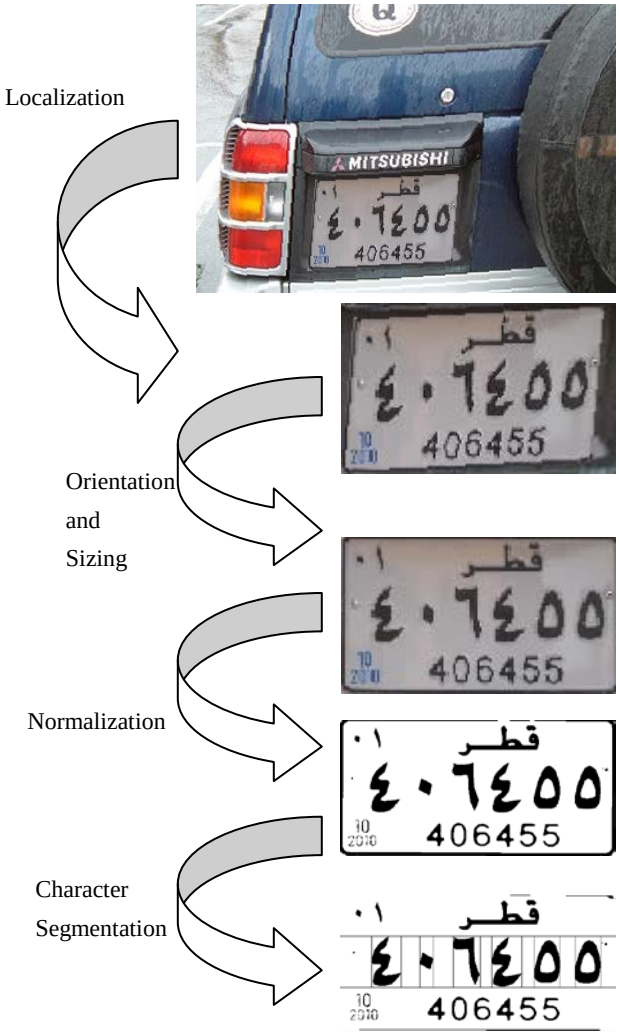

Figure 3: First four steps of License plate character recognition

4. Algorithm

In this section, we will present the algorithm that we developed to detect numbers on a Qatari license plate. As mentioned earlier, we assume that license plate images have been extracted from the images of the vehicles and have the correct orientation and some level of consistency in size. We also assume that the image has minimum amount of noise. Removing noise from images is a well studied topic and is out of the scope of this paper.

4.1 Normalization Algorithm

In this section, we will present our technique for adjusting the brightness and contrast of a given image. We assume that the image is loaded as an uncompressed bitmap file, where we have access to each pixel of the picture as an RGB coded color value. The **RGB color model** is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue[2]. The rows of pixels represent the height of the image and columns represent the width. Figure 4 shows an image that would be a good candidate for input to our algorithm.

Figure 4: A sample input image for our algorithm

We start normalizing by converting each pixel to a gray scale color. A **grayscale** digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest[1]. This is done by going through each pixel of the image and then determining the average brightness of the original color of the pixel by averaging its red, green, and blue parts. This will tell us what shade of gray to use. For example, if the original color is (20, 0, 100), then its average brightness is 40 ((20 + 0 + 100) /3), so we'll replace that color with (40, 40, 40), which will appear as a very dark gray. Applying this technique to Figure 4 will give us the following image. As you can see, all colors have been converted to various shades of gray.

Figure 5: Image after converting to grayscale

Conversion to grayscale is an intermediate step as we eventually want to convert the image into a black and white image. This helps us reduce the amount of information that we have to process. So the next step is to convert this gray scale image to black and white image. We take each pixel of the image and if the proportion of all three values: R, G and B is less than 100, we set the colors to 0 otherwise we set each component to 255. Notice that if R, G and B values are 0, we get the color black, and if all values are 255, we make the white color. So in essence, all values that are below 100 are converted to black color and all other values are changed to white. Applying this on our example image will produce the image as shown in Figure 6.

Figure 5: Image after converting to black and white

4.2 Character Segmentation

This step involves separating out the characters, so that each character can then be processed for identification. Assuming, that our input for this step is a black and white image, we use the following steps for character segmentation.

4.2.1 Removing Border

The first step is to remove the border from the image. This allows us to reduce the amount of information that we will have to process. To remove the border, we start from the left side of the image. We observe that the left border is made up of contiguous black pixels and the first white pixel indicates the end of the border. This is a trivial observation but nonetheless important for our purposes. We take the image and start at the left most part of each row. Start by looking for the first black pixel. The top left part of Figure 5 is zoomed in and shown in Figure 6.

Figure 6. Top Left corner of Figure 5.

We iterate horizontally through each column of this row and wait for the first black pixel. When we see the first black pixel, we set its value to white and after that every black pixel is changed to white until a white pixel is encountered. This white pixel would indicate the end of border. This process will produce the image as shown in Figure 7.

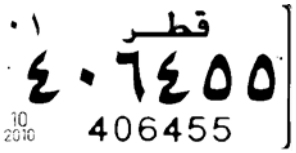

Figure 7. Image after removing partial boundary.

Notice that we still have the border on the right side leftover. This is because in the previous step, we started looking at the black pixel from left corner which ignored those black pixels that are on the right side. To remove this bit of pattern, we use a similar technique but start from the right side. So start looking at the pixels from the right most edge of the image and iterated backwards. The first contiguous set of black pixels should be converted to white resulting in the image shown in Figure 8.

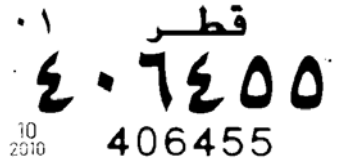

Figure 7: Image of the license plate with borders removed

4.2.2 Horizontal Segmentation

Horizontal segmentation is the first main phase of locating the numerical characters (the Arabic characters) on the license plate after all the surrounding noise has been reduced. First we will try to detect the starting row and ending row of the digits and hence the name horizontal segmentation. This involves writing an algorithm that will determine the location of imaginary lines "start" and "end" as shown in red in Figure 8.

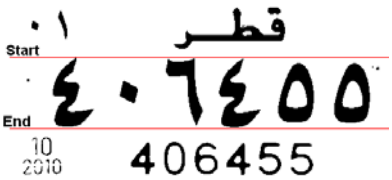

Figure 8: Locating the start and end rows of the digits

This task is accomplished by treating any series of horizontal lines that have any black pixel as one "blob". So looking at Figure 7, if we start at the top of the image, there are a few lines that are all white. Then there is some text which is made up of a group of lines where each line has some black in it. This is followed by another small group of white lines and then some text and then a few empty rows and then some text. So in all we have three blobs on the image. If there is some noise in the image, then we might get more than three blobs as shown in Figure 9. The image in this figure will result in 5 blobs. One extra blob because of thin line at the top and one because of the line at the bottom.

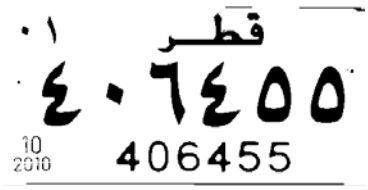

Figure 9: License plate with some noisy data

Our goal is to find the biggest blob. The following algorithm outlines our methodology for finding the biggest horizontal blob.

```
#initialize variables
inBlob ← False
startOfBlob ← 0
maxBlob ← 0
result ← Empty List
blobstart ← 0
Loop through all rows of the image
        CurrentColor ← WHITE
        For this row, go through each column and check if there are any black pixels in this row
        If black pixels found and inBlob is False
                # This is start of a blob
                Set isBlob to True
                # save the start of blob
                Set startOfBlob to current row
        If no black pixels and inBlob is True
                Set inBlob to False
                Find size of the blob we just finished
                If size is bigger than biggest blob we have seen
                        result = [startOfBlob,current row]
                        maxBlob  = size of this blob
End Loop
```

Figure 10: Algorithm for horizontal segmentation

4.2.3 Vertical segmentation

Once we have the numerical segment located, we will do the rest of the work with only this segment. Our task now is to figure out the location (start and end) of each of the digits in this vertical blob.

For vertical segmentation, we start from the left most column of the image and step through each column. For each column we analyze each pixel in this column. In the beginning we will find that all pixels are white. As soon as we hit the first digit, the color will change to black. The first time we collect a black pixel we mark it the start of the digit and then keep on searching until we see a column that is all white. This would mark the end of this digit. We repeat this step six times, each time starting from the end of the last digit in order to search for the next digit. In order to avoid detecting small specs as digits, we make sure that our digits have a minimum width. For our cases a minimum width of 5 works best. This helps us avoid detecting the small circles towards the left and right of digits as shown in Figure 11.

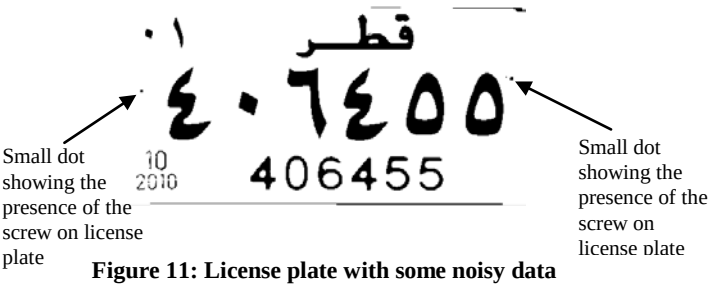

Figure 11: License plate with some noisy data

4.2.4 Character Recognition

Now that we have isolated each digit, recognizing each digit is a simple feature extraction and statistical analysis. Feature extraction is just a fancy name for a simple process described below. Lets take one digit as an example as shown in Figure 13a which represents the number "3". We take each digit and divide it in four quadrants as shown in Figure 13b.

Figure 13a: Shows the digit 3

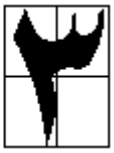

Figure 13b: Four quadrants

Now we calculate the percentage of black pixels in each quadrant of the digit. Percentage of black is taken by counting all black pixels in a quadrant and dividing it by the total number of pixels in the quadrant. We do this step for each of the four quadrants. Once we have these values for each quadrant, we compare with the measured value for each digit as shown in table 1. The values that match most closely is our prediction for this digit.

We use statistical analysis for pixel matching. This is again a fancy word for the following algorithm. Table 1 shows measured values for percentage of black pixels in each quadrant of each digit. For example, the fourth row in the table shows the percentages for digit 3. As can be seen from Figure 13b and Table 1, there is a high number of black pixels in Quadrant 1, and the corresponding percentage is 47% in the table. Quadrant 4, has no black pixels and the corresponding percentage in Table 1 is also 0%. Also notice that several quadrants have very similar percentages for different digits.

For example, in Quadrant 1, the percentage of digit 0 and 8 are very similar. Hence we have to use the knowledge of all four quadrants to determine the digits. We use the following statistical method: Let's say we have a digit that we are trying to decode. We take this digit's Q1 value, and subtract it from Q1 value for each digit. This shows us how close the two values are. If the difference is small, then the value are similar otherwise they are different. We divide this difference by 4 since we have four quadrants. We repeat this step for all four quadrants and add the results for each digit. The digit with the minimum overall difference most closely matches the sample digits and is our estimation for this digit. Lets say that for a given digit, our values for Q1, Q2, Q3, and Q4 are 0.12, 0.58, 0.72, 0.34.

```
// initialize variables
// two variables digitstartn and digitendn for each digit
digitstart ← 0
digitend ← 0
start ← input parameter
blobstart ← 0
CurrentColor ← WHITE
PrevColor ← WHITE
// go through the current column,
Loop j from start to TotalColumns
    // we assume that there are no black pixels in this
    // column.
    CurrentColor ← WHITE
            // step through each row of this column
            // check if there are any black pixels
            // notice that we should only check
            // between the start and end locations of
            // the biggest blob we found in previous
            // step
            Loop r from maxblobstart to maxblobend
                    if Color in row r, column j is BLACK
                                CurrentColor ← BLACK
            end Loop
            // now if the current color is white then in
            // this column there were no BLACK pixels
            if CurrentColor is not Equal to PrevColor
                    if CurrentColor is BLACK
                                blobstart ← j
                    else
                            // if color is white then
                            // prev color is black
                            // so we just finished
                           // a digit. Check if the size
                           // of the digit is big enuf
                          if j – blobStart > 5
                                digitstart(i) = blobstart
                                digitstop = j-1
                                start = j + 1
                                // break from loop
                                j ← TotalColumns


            end Loop j
End Loop i
```

Figure 12: Pseudocode for Vertical Segmentation

Then for Digit 0, the absolute difference between corresponding quadrants is 0.09, 0.27, 0.45, 0.08, with a total difference equal to 0.09/4 + 0.27/4 + 0.45/4 + 0.08/4 = 0.21. Similarly, for Digit 4 we get values 0.02, 0.00, 0.00, 0.02 with an overall difference of 0.01. When we get these overall difference values for all 10 digits, we can search for the smallest different and that is our estimate for this particular digit.

Digit	Q1	Q2	Q3	Q4
0	0.21	0.31	0.27	0.26
1	0.16	0.58	0.12	0.54
2	0.38	0.80	0.33	0.23
3	0.47	0.58	0.34	0.00
4	0.10	0.58	0.72	0.32
5	0.52	0.37	0.59	0.51
6	0.45	0.44	0.04	0.43
7	0.33	0.37	0.25	0.25
8	0.22	0.26	0.40	0.36
9	0.52	0.80	0.22	0.55

Table1: A measure percentage of black pixel percentage for each quadrant of each digit

5. Future Work

In this paper, we have presented a technique for determining digits from a Qatari License plate that is rectangular. Our next goal is to develop an algorithm that is capable of handling both types of license plates that are used in Qatar. We also want to study the effect of having noisy data where the images of license plates are blurry and measure the effectiveness of our algorithm.

6. REFERENCES

[1] Stephen Johnson (2006). *Stephen Johnson on Digital Photography*. O'Reilly. ISBN 059652370X.

[2] Wikipedia – RGB Color Model. http://en.wikipedia.org/wiki/RGB_color_model

[3] Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.

[4] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= http://doi.acm.org/10.1145/332040.332491.

[5] Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[6] Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[7] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[8] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In

Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697.

[9] Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= http://dx.doi.org/10.1016/j.jss.2005.05.030.

[10] Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= http://doi.acm.org/10.1145/90417.90738.