Automatic License Plate Recognition

Sarah Sterchele

CMPT 414

April 7, 2017

Introduction

License plate recognition is a commonplace use of computer vision in everyday life that often people take for granted. Through use of Automatic License Plate Recognition (ALPR), bottlenecks on toll bridges and roads are avoided, and tickets issued for traffic violations have a faster turnaround time for example. ALPR is a tricky business, however. Cars are moving, plates come in different fonts and colors, the time of day and weather affects lighting conditions, dirt, rain or snow can obscure the plate; an implementation of ALPR has to overcome many obstacles to be an efficient way to gather plate information. Because of these obstacles, no form of ALPR is 100% robust. Fortunately, the subject has been deeply researched, and in combination with machine learning and high speed GPU's, the accuracy of a positive detection increases as more data is accumulated and image-processing power grows.

Method

ALPR has seven primary steps: (Wikipedia, n.d.):

- **"Plate localization – responsible for finding and isolating the plate on the picture."**
- **"Plate orientation and sizing – compensates for the skew of the plate and adjusts the dimensions to the required size."**
- **"Normalization – adjusts the brightness and contrast of the image."**
- **"Character segmentation – finds the individual characters on the plates."**
- **"Optical character recognition."**
- "Syntactical/Geometrical analysis – check characters and positions against country-specific rules."
- "The averaging of the recognized value over multiple fields/images to produce a more reliable or confident result. Especially since any single image may contain a reflected light flare, be partially obscured or other temporary effect. "

The steps in bold are the algorithms I tackle to some degree in this final project. I will limit my research to simple British Columbia license plates.

Step 1: Plate localization

There are many places to begin in ALPR, depending on the use case. The camera may be in a fixed position, taken from a mobile device, or from a moving vehicle such as a police car. Each of these situations warrant a different approach. I begin at a relatively safe starting point: I use a fixed position close to the rear of the vehicle. There is no angle rotation.

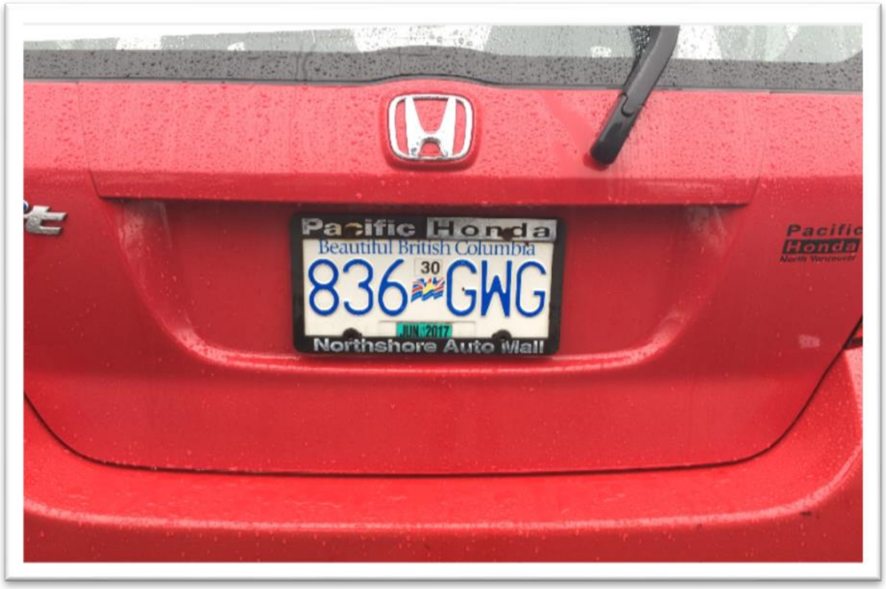


Figure 1- The starting point.

The first thing to do is image processing. This entails,

- Resize the height of the image, while maintaining aspect ratio
- Converting the image to Grayscale
- (optional) Histogram Equalization
- Thresholding
- Edge Detection
- Eroding/Dilation

First, resize the image so that we have consistency for what size of plate we are looking for. After resizing, convert the image to grayscale. The next step is histogram equalization, which has the possibility to work to one’s detriment. Histogram equalization spreads out the range of grayscale intensities, which can make thresholding difficult. The purpose of thresholding is to pull out the white plate, (usually whiter than anything else in the image), and make everything else receded. Equalization can make th e plate extraction difficult, unless in the case of a white plate on a white car, which I will discuss later.

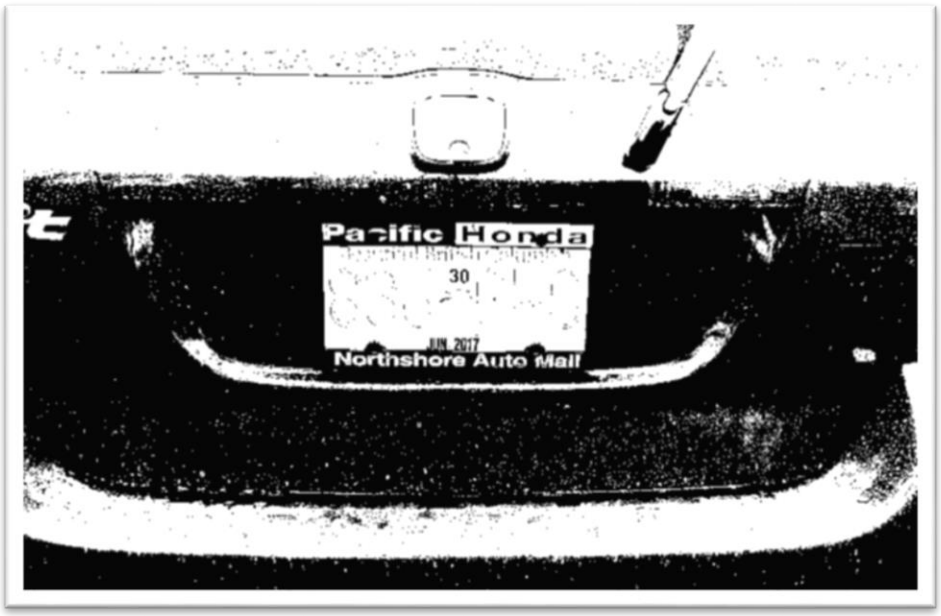


Figure 2- Using thresholding to extract the plate

After thresholding, a heavy blur filter is applied. We want to remove as many fine details in the image as possible. Afterwards, Canny edge detection reduces the image to a set of lines. We want bold, thick line, so we erode the image, making the white lines thicker.

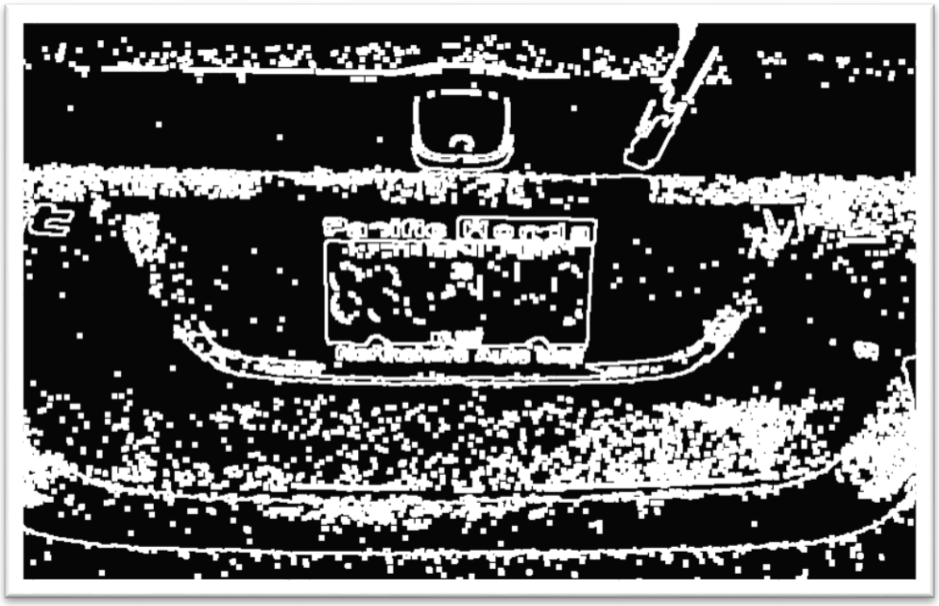


Figure 3- Canny Edge detection for plate localization

OpenCV has a function that detects closed contours. We apply this function on the edge map, and surround each contour with a bounding box. For the contour to qualify as a license plate, it must meet certain requirements: it must have a ratio of weight to height of between 2 to 4; it must meet a certain width and height criteria as well in respect to the image size. Afterwards, what is left is the bounding box for the license plate.

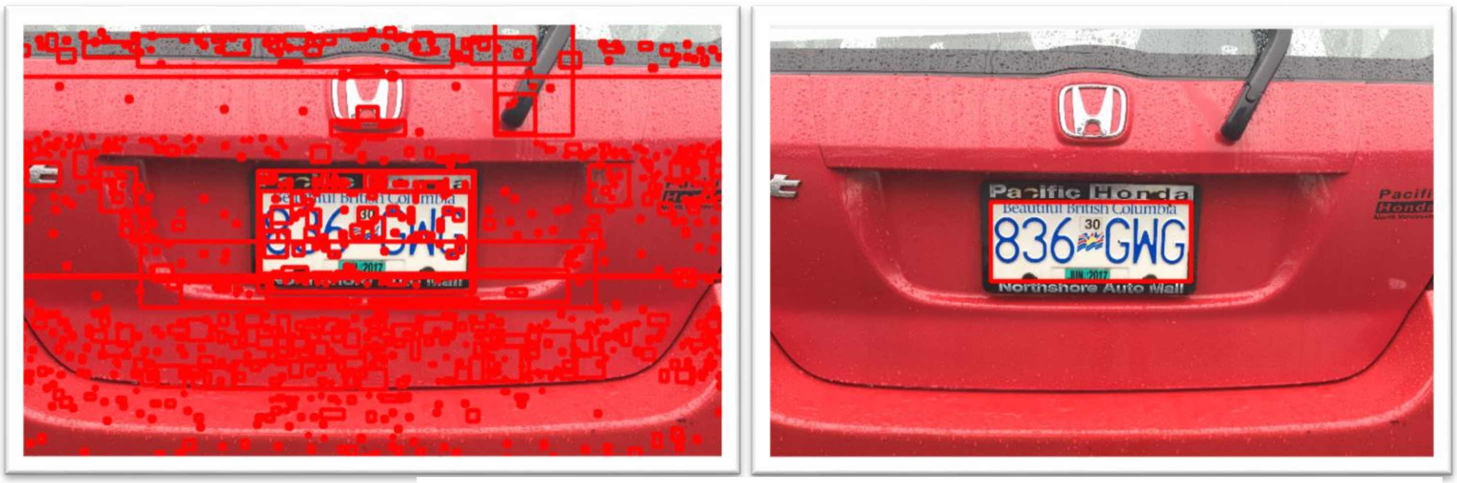


Figure 4- Using contours to find plate

Step 2: Horizontal Projection

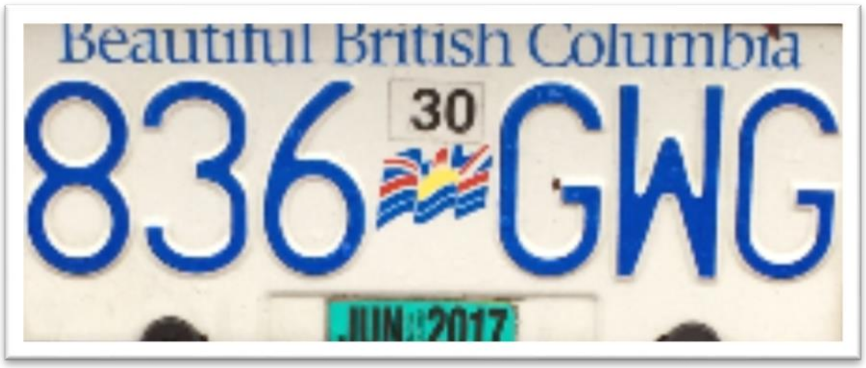


Figure 5- The plate has been localized

We have reduced the image down to the plate. The next step is to find the glyphs (characters). After again resizing the image, we can further reduce the area by detecting the horizontal lines on the plate. The letters are bounded above by “Beautiful British Columbia” and below by the registration sticker. Applying a dy Sobel edge detection, we then calculate the horizontal projection of the image. (A. Badr, 2011)

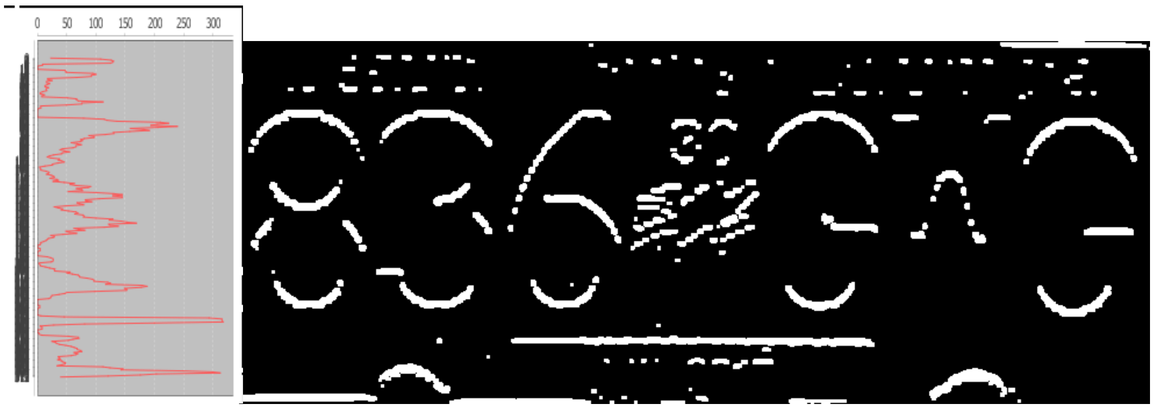


Figure 6- Horizontal projection of plate

From the horizontal projection, we can see peaks in the first half and second half of the image. Those are the places to crop the plate. Afterwards, all we have left are the glyphs that need segmented.

Step 3 - Character Segmentation

This step is simple since we have reduce the image to just the characters, which contrast well against the white plate. We simply apply Canny edge detection, and use OpenCV to find the contours. The characters will have a certain ratio of width to height (all the glyphs are the same height but differing widths; a ‘W’ is wider than a ‘1’) and a minimum height to satisfy. Applying a bounding box to the contours, we are able to extract the letters from the plate via further cropping.

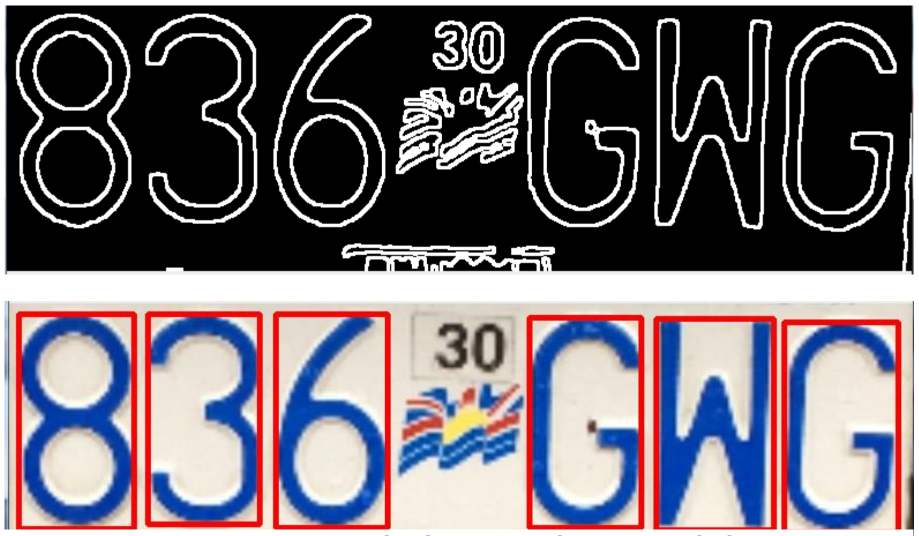


Figure 7- Canny edge detection used to segment glyphs

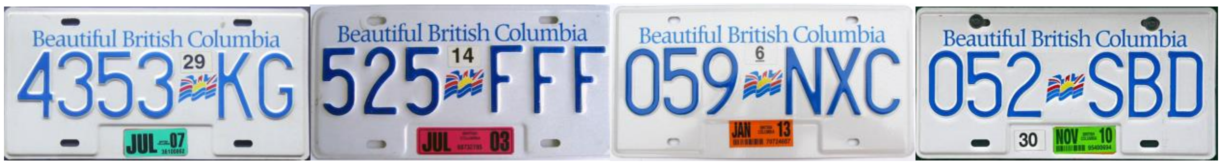
Step 4 – Optical Character Recognition

Lots of clever tricks and image processing has gotten us down to the meat of ALPR – optical character recognition. It does not matter how well you are able to segment your characters if it results in an incorrect reading.

I compare two methods of OCR – one I implemented myself, and one from a well-known OCR engine, Tesseract.

Tesseract is currently sponsored by Google and comes with an OCR API and trained data for the English alphabet. The results are, more often than not, robust, although it does make mistakes sometimes. In this project, each character segment is fed into the Tesseract OCR engine. The whitelist is set to be [A-Z,0-9] excluding ‘O’, ‘U’, and ‘I’ which are omitted from BC license plates.

I also implemented an unweighted K-Nearest Neighbors algorithm. I took a set of license plate images from the internet and, for each character, calculated a feature vector. The feature vector contained the following information: The area, centroid, the number of horizontal and vertical strokes the character had, and the perimeter. There are many other features that could have been extracted to improve accuracy, which I later realized.


Beautiful British Columbia
4353 KG
Beautiful British Columbia
525 FFF
Beautiful British Columbia
059 NXC
Beautiful British Columbia
052 SBD
Figure 8- Examples of training plates

A feature vector is extracted from each character segment from the test image. The Euclidian distance between the test segment and each training vector is calculated, where a higher weight is placed on the number of horizontal/vertical strokes. The strokes are size/scale/transformation/etc. invariant, while the others are not, although the glyph size should always be the same (if the segmentation process was performed correctly) because a resize step is done for each glyph segment. The top k smallest distances are tabulated, and thus resulting in a simple K-N-N algorithm that determines a glyph.

Geometric Features of a Glyph

A paper I researched suggested that English character recognition could be done solely by the geometrical characteristics of an image, such as curves, and the number of strokes in the horizontal, vertical, and diagonal direction (S. Bhooshan, 2009). The diagonal direction algorithm was very complicated, so I implemented the paper's algorithm of determining horizontal and vertical strokes:

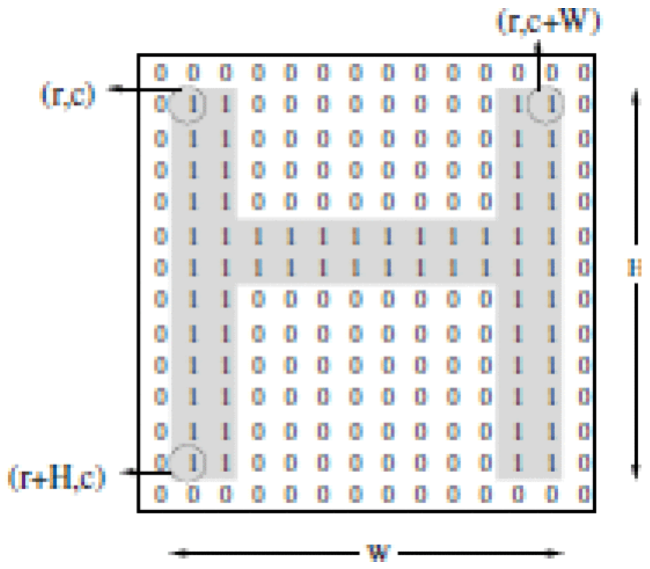


Figure 9- Visualization of line stroke algorithm

For vertical strokes, I traverse the image from left to right, top to bottom until a "true element" is reached. Then, from the true element's position, traverse downward until a non-element is reached. Count the number true elements encountered; if it is as many as the height of the character, then it is a stroke. Recognizing a horizontal stroke is similar in process. (S. Bhooshan, 2009)

Results

The results of my OCR implementation were, as expected, not robust. My training data set was far too small, and more feature points were needed in the feature vector, which I realized after the initial results. What was surprising that the detection of numbers was around 70% correct. My training set had more numbers than A-Z letters, and proved to detect them better than letters, although there was confusion in some numbers in the OCR step for '8' and 'S', '9' and '0', '9' and '5', '8' and 'S', '5' and 'S', and 'W' and 'M' (both have 2 vertical lines) among many other mistakes.

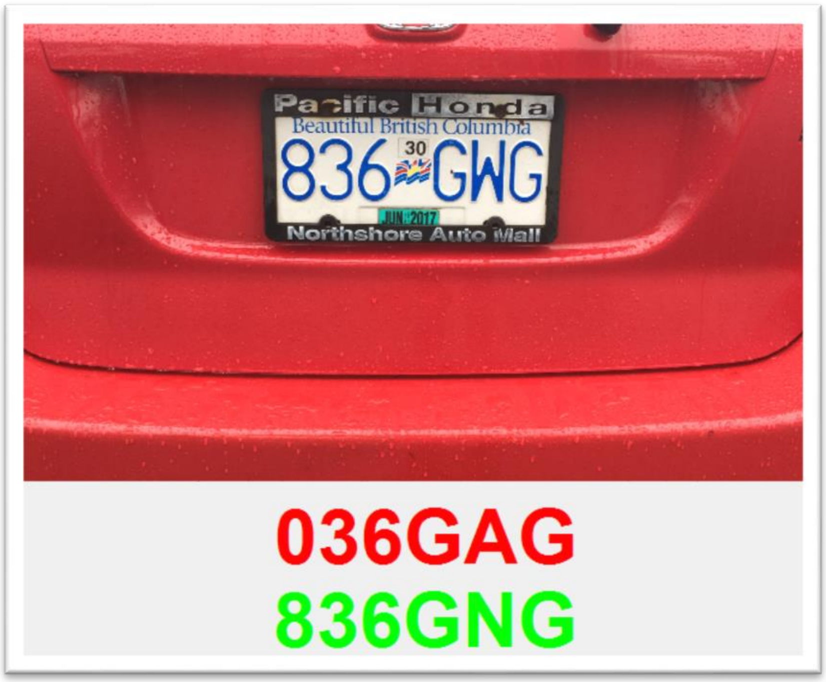


Figure 10-KNN in red, Tesseract in Green. Tesseract makes mistakes as well.

Tesseract OCR made some mistakes as well, although it was far more robust.

Discussion

What was very clear from the first initial results was that there is not a “one-size fits all” solution to ALPR. What works for one color of vehicle in a specific lighting condition will not work for a differently colored vehicle under the exact same lighting conditions. A full range of thresholds, kernel sizes, and image processing strategies needs to be tested until a “best value” is found. In my case, all useful thresholds and kernel sizes are tested, and histogram equalization either done or not done, until 6 objects that are thought as alphanumeric glyphs are found. (There is no guarantee they are glyphs, but usually they are).

After realizing that all combinations needed to be tested to detect more plates under different conditions, I wanted to harness NVIDIA’s CUDA parallel processing as it was designed for cases such as this where nested ‘for’ loops are used to process images. This realization came too late for implementation in this assignment, although given more time, this could be done easily.

There was one condition that persisted to fail. Consider the image below.

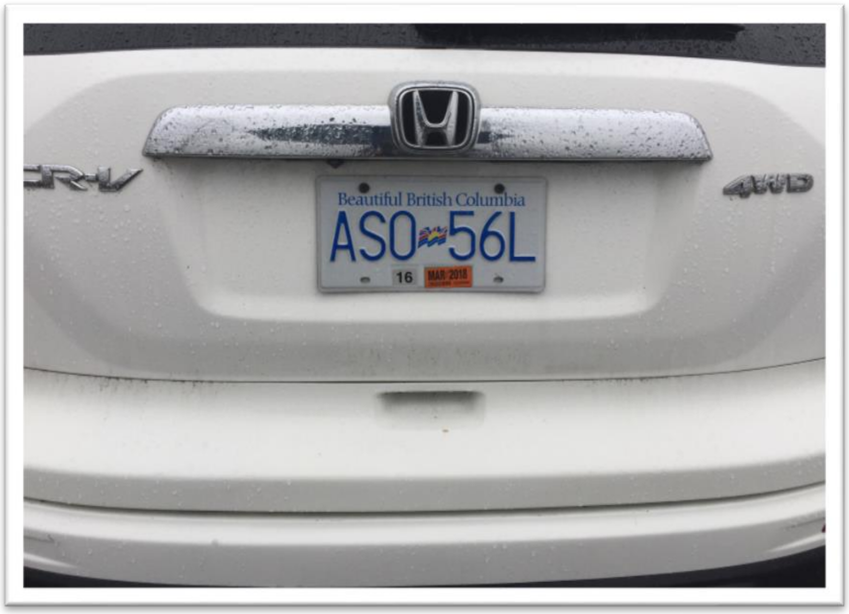


Figure 11- Difficult

There is no defining frame around the plate, and the vehicle is whiter than the plate. Even applying several known image-processing techniques, the plate always blended into the car because the Gaussian image blurring eliminated the fine distinction between vehicle and plate.

An idea I had considered was implementing Hough Transform to detect rectangles. OpenCV's contour detection and bounding box function eliminated that step and provided results that are more accurate.

ALPR is a testament to image processing as the foundation for computer vision. Every stage of ALPR required moderate to extensive image processing. While image processing is not an exciting area of computer vision study like neural networks, it is extremely important in order to have successful results.

If I had more time I would like to get a larger training data set to see better results in my KNN OCR implementation.

References (JNA Wrapper for Tesseract, n.d.)

A. Badr, M. A. (2011). Auomatic Number Plate Recognition System. *Annals of the University of Craiova, Mathematics and Computer Science Series*, *38*(1), 62-71.

JNA Wrapper for Tesseract. (n.d.). Retrieved from http://tess4j.sourceforge.net/codesample.html

P. Jain, N. C. (2014). Automatic License Plate Recognition using OpenCV. *International Journal of Computer Applications Technology and Research, 3*(12), 756 – 761.

S. Bhooshan, V. K. (2009). Character Recognition Using Geometrical Features of Alphabet: A Novel Technique. *International Conference on Communication Software and Networks*, 119-125.

Tesseract OCR Engine. (n.d.). Retrieved from https://github.com/tesseract-ocr