MINHLP: Module to Identify New Hampshire License Plates

Sunitha Raghurajan

University of New Hampshire, Durham

Follow this and additional works at: https://scholars.unh.edu/thesis

Recommended Citation

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact nicole.hentz@unh.edu.
MINHLP: Module to Identify New Hampshire License Plates

Abstract
A license plate, referred to simply as a plate or vehicle registration plate, is a small plastic or metal plate attached to a motor vehicle for official identification purposes. Most governments require a registration plate to be attached to both the front and rear of a vehicle, although certain jurisdictions or vehicle types, such as motorcycles, require only one plate, which is usually attached to the rear of the vehicle.

We present analysis of Automatic License Plate Recognition (ALPR) of New Hampshire (NH) plates using open source products. This thesis contains an implementation of a demonstrated model and analysis of the results. In this paper, OpenCV (computer vision library) and Tesseract (open source optical character reader) is presented as a core intelligent infrastructure. The thesis explains the mathematical principles and algorithms used for number plate detection, processes of proper characters segmentation, normalization and recognition. A description of the challenges involved in detecting and reading license plate in NH, previous studies done by others and the strategies adopted to solve them is also given.

Keywords
Information Technology, Graduate School

This thesis is available at University of New Hampshire Scholars' Repository: https://scholars.unh.edu/thesis/774
MINHLP: MODULE TO IDENTIFY NEW HAMPSHIRE LICENSE PLATES

BY

SUNITHA RAGHURAJAN

B.Tech., Regional Engineering College, 2001

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

in

Information Technology

May, 2013
This thesis has been examined and approved.

Thesis Director, Michael Jonas, Assistant Professor in Computing Technology

Mihaela Sabin, Associate Professor in Computing Technology

David Forest, Associate Professor in Engineering Technology

May 14, 2013
Date
# TABLE OF CONTENTS

**ACKNOWLEDGEMENTS** ........................................................................................................... iii  
**ABSTRACT** ........................................................................................................................ iv  
**CHAPTER 1** ......................................................................................................................... 1  
  1.1 Introduction .................................................................................................................... 1  
  1.2 APLR System Overview .......................................................................................... 2  
    1.2.1 Why We Need Automatic License Plate Detection ........................................... 4  
    1.2.2 Common Uses ..................................................................................................... 5  
    1.2.3 Parking Lot Management .................................................................................. 6  
    1.2.4 Traffic flow and HOV .................................................................................... 6  
    1.2.5 Weigh in Motion Systems ............................................................................... 6  
  1.3 An Open Source Solution ............................................................................................ 7  
**CHAPTER 2** .......................................................................................................................... 8  
  2.1 Commercial-Off-The-Shelf Solutions ...................................................................... 8  
  2.2 Previous Research ...................................................................................................... 9  
  2.3 Performance Issues .................................................................................................. 13  
**CHAPTER 3** ........................................................................................................................ 15  
  3.1 Tesseract Optical Character Recognition System ................................................. 15  
    3.1.1 History of Tesseract ......................................................................................... 16  
    3.1.2 Tesseract Architecture .................................................................................... 17  
  3.2 Image Processing ....................................................................................................... 18  
    3.2.1 OpenCV ........................................................................................................... 19  
    3.2.2 OpenCV History ............................................................................................. 20  
**CHAPTER 4** ........................................................................................................................ 22  
  4.1 MINHLP Explained ..................................................................................................... 22  
  4.2 Noise Filtering ........................................................................................................... 23  
  4.3 Problems with Feature Variation ............................................................................ 24  
    4.3.1 Results of Images from Various Steps ........................................................... 25  
    4.3.2 Image Smoothing ............................................................................................ 26  
    4.3.3 Image Cropping ............................................................................................... 28  
**CHAPTER 5** .......................................................................................................................... 30  
  5.1 Performance ................................................................................................................ 30
ACKNOWLEDGEMENTS

First of all, I am incredibly thankful to my advisor, Dr. Michael Jonas, who gave encouragement, supervision and support, enabling me to learn more about image analysis processing and how to apply it to this project. Dr. Jonas provided me with a great quantity of advice and inspiration on my research and always encouraged me to become an independent researcher. Not only did I learn a lot from him in how to become a successful researcher, but also how to become an IT professional. Aside from my advisor, I take this opportunity to convey my heartfelt gratitude to Dr. Mihaela Sabin. Both helped me tremendously with my master thesis.
ABSTRACT

MINHLP: MODULE TO IDENTIFY NEW HAMPSHIRE LICENSE PLATES

by

Sunitha Raghurajan

University of New Hampshire, May, 2013

A license plate, referred to simply as a plate or vehicle registration plate, is a small plastic or metal plate attached to a motor vehicle for official identification purposes. Most governments require a registration plate to be attached to both the front and rear of a vehicle, although certain jurisdictions or vehicle types, such as motorcycles, require only one plate, which is usually attached to the rear of the vehicle.

We present analysis of Automatic License Plate Recognition (ALPR) of New Hampshire (NH) plates using open source products. This thesis contains an implementation of a demonstrated model and analysis of the results. In this paper, OpenCV (computer vision library) and Tesseract (open source optical character reader) is presented as a core intelligent infrastructure. The thesis explains the mathematical principles and algorithms used for number plate detection, processes of proper characters segmentation, normalization and recognition. A description of the challenges involved in detecting and reading license plate in NH, previous studies done by others and the strategies adopted to solve them is also given.
CHAPTER 1

MINHLP: MODULE TO IDENTIFY NEW HAMPSHIRE LICENSE PLATES

1.1 Introduction

There are two types of technologies commonly being used for the recognition of license plates. Automated License Plate Recognition (ALPR) systems use the approach of passively reading license plate characters and Electronic License Plate Recognition (ELPR) systems that do so actively. ALPR technology utilizes cameras and alphanumeric recognition software to read license plates as they pass whereas ELPR technology uses Radio Frequency identification (RFID).

An ALPR system is a combination of image processing, character segmentation and recognition technologies used to identify vehicles by their license plates. Since only the license plate information is used for identification, this technology requires no additional hardware to be installed on vehicles. RFID based ELPR systems consist of three main components: readers, antennas, and tags. The antenna emits radio signals, the tags respond to their own unique code, and the reader receives the signal from the tag, decodes the tag information, and sends it to a processor through standard digital interfaces. RFID tags emit a radio frequency that can be read by an RFID reader.
RFID technology has become very prevalent as a means of payment for tolls in the United States, giving easy access to HOT (High Occupancy Toll) lanes. Perhaps E-ZPass is the most well known RFID application in the United States. E-ZPass is a voluntary program that allows toll users to set up a pre-paid account to pay tolls. When using a toll that accepts E-ZPass drivers enter a special lane. After entering the lane, the user pulls up to an E-ZPass reader, the reader identifies the vehicle and corresponding E-ZPass account, and the toll user is electronically charged and subsequently allowed to pass. All of this is done without interacting with a human having to physically exchange money.

The hardware for ELPR consists for RFID tags and a reader. Each license plate requires an RFID tag. The tags and readers are expensive, so the initial cost will be high when compared to ALPR. The law does not permit some states to install RFID tags in license plates. There are other disadvantages of RFID tags, including:

- Susceptibility to damage by accidents, wear and tear or damage caused by misuse
- Equipment / component theft
- Limited power sources for RFID equipment-Internal battery

1.2 APLR System Overview

ALPR systems utilize computer vision and pattern recognition technologies. The system consists of two components: the hardware component and the software component. The robust performance of both hardware and
software components results in a successful ALPR system. The functionalities of
the hardware components are:

1. Vehicle Detection
2. Vehicle Image Acquisition

The software component is a part of more general research called Text
Information Extraction (TIE). The Software component involves localization,
extraction, enhancement and recognition of characters in a given image. In a
regular TIE process, document recognition is different from TIE of the license
plate recognition. The APLR systems generally operate on noisy and low quality
images, in which illumination conditions may frequently cause difficulties.

Character recognition is one of the most critical issues associated with
ALPR systems. Typically, consumers are free to choose from a number of
commercial video imaging subsystems. The pattern recognition algorithm used in
the imaging systems is the most important component in the ALPR system. The
correlation matching approach takes each character, and attempts to match it to
a set of predefined standards.

If we summarize the hardware and software packages, the system
consists of:

- An illumination source
- A camera
- A vehicle sensing device
1.2.1 Why We Need Automatic License Plate Detection

The application of automatic license plate detection is numerous. Typically it is used for enforcement and data collection. ALPR and ELPR have increasingly been used by the Departments of Transportation (DOT’s), tolling authorities, and law enforcement agencies to find innovative ways to achieve their unique objectives. Countries like Singapore, Japan, Canada, Germany, Italy U.K, and France have developed license plate recognition systems and successfully applied them to their traffic management. ALPR technology has been used by Transport for London in implementing the congestion charge. In London, there is a network of cameras that surround what is the most congested part of London, called the charging zone. As vehicles enter the charging zone they pass by ALPR cameras that read the license plates. The London congestion charge is a flat fee of £10.00 (or approximately $20) that road users entering the charging zone must pay daily. No matter how many times the camera systems recognize a particular vehicle each day; each vehicle is only charged once per day [1].

In 2007, New Hampshire passed SB41, an act relative to the authority of law enforcement officers to obtain registration checks on motor vehicles for official purposes and prohibiting the use of automated number plate scanning.
devices. Current law does not permit the law enforcement officers not to use ALPR.

1.2.2 Common Uses

Three of the most common uses of license plate imaging system are origin-destination studies/trip surveys, cordon studies, and travel time studies. The purpose of origin-destination studies is to determine the travel patterns along a given transportation network. Often, trip surveys are used to implement these studies. Trip surveys require three main parts. First, all license plates selected should be done randomly. Second, there can be no incorrect interpretations of the license plates by ALPR systems. This type does not require a high level accuracy. As long as the system is consistent in its plate identification, it will be successful.

In a cordon study, the traffic patterns into and out of a given area are analyzed. The key to these studies is correct placement of the imaging equipment, located at pertinent locations on the boundaries of the area being studied.

Another common application is travel time studies. ALPR systems are used to record the location of a vehicle at two different points in time and from this data an average speed can be acquired [2]. It has been concluded that this technology can "reduce the cost and greatly facilitate the conduct of travel time and small area origin-destination studies."
1.2.3 Parking Lot Management

ALPR systems have been used in parking lot systems in order to overcome most of the shortcomings of normal parking systems. When a car enters a lot, the system captures the license plate number and logs the time and date. When the car later exists, the system again captures the license plate number and computes the fee. This system eliminates the problem with lost or swapped tickets, cashier fraud and stolen cars. This system requires a high accuracy rate. Whenever enforcement issues are involved, the accuracy of the system is very important.

1.2.4 Traffic flow and HOV

Traffic flow studies are another application of ALPR's. A license plate reader can eliminate some of the problems faced when attempting to conduct a travel flow study. ALPR systems can also help determine relevant information for HOV lanes. They can determine the potential demand for such a lane through matching license plates at enter and exit points of such facility, and subsequently determining the traffic volume.

1.2.5 Weigh in Motion Systems

A weigh station is a checkpoint along a highway to inspect vehicular weights. Usually, trucks and commercial vehicles are subject to the inspection. They provide an opportunity for the enforcement personnel to check a vehicle's weight, dimensions and credentials and to ensure that the vehicle is safe to
operate. The inspection process can be time consuming. The use of transponders can be uneconomical. An ALPR is a logical choice to eliminate the inadequacies of transponder based systems. The trucks require no extra hardware, therefore there is less cost and all trucks can be monitored, as opposed to only those with transponders installed.

1.3 An Open Source Solution

A common misconception is that ALPR systems are only used by law enforcement. Because of all of the above mentioned uses, such systems would be an incredibly useful device beyond that of law enforcement. We developed an open source solution to ALPR using Java and by combining open source packages such as OpenCV, JavaCV and Tesseract. We are able to successfully adapt these systems to capture license plates with a high accuracy.

In chapter 2 we discuss market solutions and researches done on this subject. In chapter 3 we describe OpenCV and Tesseract. We then demonstrate our system in chapter 4 and give results and concluding remarks in chapter 5. Also see the appendix for related resources.
CHAPTER 2

HISTORY OF ALPR SYSTEMS

2.1 Commercial-Off-The-Shelf Solutions

There are several COTS solutions available for license plate reading. "COTS" stands for Commercial-Off-The-Shelf (COTS) implementation of commercially available technologies for traditionally customized applications. The companies mentioned below have quoted on their websites that they have COTS solutions available for ALPR systems. There are no trial versions available and all require heavy customization.

1. http://www.ndi-rs.com

Due to the complexity of license plate formats, syntax rules, font types and sizes, and special characters, heavy customization is required for ALPR to work. In the ALPR business, it is not a "one size fits all" approach regarding OCR. The North American market is particularly complex as it is not uncommon for an
individual state to have 150-200 individual plate types. Dedicated OCR resources and expertise are needed to keep the technology at its optimum performance.

2.2 Previous Research

Some work on the scientific side, from the research papers quoted below, have good accuracy with more complexity than others. Some of those algorithms are computationally intensive. However, selecting one of them based on criteria such as execution time, memory usage, complexity of the algorithm, and its accuracy in different situations is a challenging problem.

In the study described in [3] by Clemens Arth, Florian Limberger, Horst Bischof, the system was embedded on a DSP platform and the system processed the video stream in real time. The system consists of detection and character recognition modules. The algorithm used for detecting license plates was AdaBoost by Viola and Jones [4]. Detected license plates were segmented into individual characters by using a region-based approach. In order to improve the embedded platform processing speed, a Kalman tracker (based on Kalman filter Algorithm) was inserted into the system and was used to forecast the position of the license plate in the next frame image. The real-time processing was the biggest advantage of this system. Additionally, it didn't require any additional sensor inputs (e.g., infrared sensors), in addition to a video stream. However, the systems were only capable of processing a single line of large characters with no background image, even if there were two lines of characters.
on the license plate. Since the NH license plate has a background image (Man embossed in the mountain), we could not adopt this algorithm.

From the study [4], by Anagnostopoulos, C. -N. E.; Anagnostopoulos, I. E.; Psoroulas, I. D.; Loumos, V.; Kayafas, E, the license plate recognition system was advanced by adopting a cascade framework based on a double-layer hidden Markov model. A method of fast identification algorithms was developed by using characteristics of license plate characters. The system, which was composed of three cascading modules for plate detection, character segmentation and post processing, could recognize the license plate at over 38 frames per second. This algorithm is better suited to detect license plate from a wide range of images. But can not be used to read the license plate characters.

Hongliang, B.; Changping [5] paper discusses an algorithm that can detect the license plate using edge statistics and morphological operations. However this algorithm is suited for the object detection part but not the reading the license plate. A disadvantage is that edge-based methods alone can hardly be applied to complex images, since they are too sensitive to unwanted edges; this may also show high edge magnitude or variance (e.g., the radiator region in the front view of the vehicle).

In the study described in [6] by Ying Wen, Yue Lu, Jingqi Yan, Zhenyu Zhou, Karen M. von Deneen, Pengfei, a new algorithm for vehicle license plate identification is proposed, on the basis of a novel adaptive image segmentation technique (sliding concentric windows) and connected component analysis in conjunction with a character recognition neural network. The optical character
recognition system is a two-layer probabilistic neural network (PNN) with topology 108-180-36, whose performance for entire plate recognition reached 89.1%. The PNN is trained to identify alphanumeric characters from car license plates based on data obtained from algorithmic image processing. Combining the above two rates, the overall rate of success for the license plate-recognition algorithm is 86.0%. This algorithm is not suited to use with NH license plates due to complex backgrounds and additional objects such as decal stickers in the NH license plates.

Lei He, Chang-fu Zong, Chang Wang [7] applied a fuzzy logic to the problem of locating license plates. The authors made some intuitive rules to describe the license plate and gave a few membership functions for the fuzzy sets “bright,” “dark,” “bright and dark sequence,” “texture,” and “yellowness” to get the horizontal and vertical plate positions, but these methods are sensitive to the license plate color and brightness and need longer processing times from the conventional color-based methods. Despite achieving better results, they still carry the disadvantages of the color-based schemes.

Chaddha, Sharma, Agrawal, and Gupta [8] propose a method to detect text from JPEG compressed images. The method extracts the DCT coefficients for each block of a macroblock in the JPEG image. The absolute values of a set of DCT coefficients are combined. This measure reflects the high spatial frequency content of blocks containing text. This particular set of coefficients was empirically determined as the optimum to discriminate text and non-text blocks. Next this sum is designated as a threshold using an empirically determined
value. All blocks having a measure greater than this value are marked as text blocks. Because the JPEG compressed information is not decompressed, the method is quick and capable of running in real-time. The algorithm finds artificial and scene text within a range of scales. However, it fails with text that has low or varying contrast within the string or on text that is much larger than the block size used (8x8).

Zhong, Karu and Jain in their paper [9] described a method to locate text in complex color images. They propose a method based on quantizing the color space based on peaks in the histogram. Adjacent colors are merged into these peaks. This assumes that the text occupies a narrow region of color space and a significant portion of the image. Color components are then labeled as text based on their geometrical properties and if there are at least 3 characters aligned horizontally. If the color quantization step fails because the text has low contrast or occupies a small portion of the frame, then a character may be broken into multiple segments.

Jun-Wei Hsieh, Yuan Ze, Shih-Hao [10] proposed a license plate detection using a morphology based solution. Morphological operators often take a binary image and a structuring element as an input and combine them using a set operator (intersection, union, inclusion, complement). They process objects in the input image based on characteristics of its shape. Those characteristics are then encoded in the structuring element. The mathematical details are explained in Mathematical Morphology. This algorithm expects that the car plates have clear backgrounds and dark foregrounds.
2.3 Performance Issues

Most of the research papers for license plate detection are based on neural networks, Gabor transform, Hough transform, and Ada-Boost models. But there are some common features affecting the performance of these algorithms including [3, 10] Lighting conditions such as cloudy weather, nighttime, and reflection of sunlight off of the car's taillight or front light. Complex backgrounds affect the speed of detecting the real region of plate. A damaged or dirty license plate fails in identification numbers. Varying view angle and efficient distance between camera and moving car is needed in successfully determining and reading the license plate characters. Therefore, in describing the algorithms we point to practical criteria in addition to technical measures such as run time and performance.

Even though the above algorithms are efficient in locating and reading the license plate characters, there are some constraints in using those to read license plate characters of New Hampshire. The aforementioned algorithms require high quality images. It is also important to mention that the above algorithms require that the noise of the images should be at a minimum. In order for the noise to be at a minimum, appropriate lighting conditions are required. Poor lighting conditions can result in inadequate exposure, distorting the colors and creating shades that obfuscate the real object. There is no universal solution available for the Automatic License Plate detection. The decal sticker on the NH license plate makes it difficult to read the characters. Some of the plates have
five or more stickers attached to the plate. The License plate readers read these characters too. New Hampshire has different license plates so it is difficult to come with a common computational algorithm. The simple binarization and algorithms of the text documenting will not be insufficient. License plates are produced by the state prisons. In the United States, there is no single official license plate font. So again, it is difficult to train the Optical Character Reader with a particular font. Most license plate fonts are created first as AutoCAD files and then directly incorporated into the embossing dies, or other manufacturing equipment used to produce plates. Because of this, the actual original fonts are not generally available to the public.
3.1 Tesseract Optical Character Recognition System

One of the two systems used in license plate recognition is Tesseract, an open source optical character recognition (OCR) system. OCR is the mechanical or electronic conversion of scanned images of handwritten, typewritten or printed text into machine-encoded text. It is widely used as a form of data entry from some sort of original paper data source, whether documents, sales receipts, mail, or any number of printed records. It is a common method of digitizing printed texts so that they can be electronically searched, stored more compactly, displayed on-line, and used in machine processes such as machine translation, text-to-speech and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Analyzing the printed text is a tricky problem since documents are printed using different fonts. Analyzing the handwritten document is trickier because each person has a unique style of writing alphabets. The process follows a pipeline architecture decomposing the process into multistage process.

1. Format analysis: used to validate the pattern of characters
2. **Character segmentation:** artificial neural networks are commonly used to perform character recognition due to their high noise tolerance

3. **Feature Extraction:** Detecting the individual lines and strokes characters are made from (feature detection) and identifying them that way

4. **Classification:** Character Properties (Character ID)

Today many types of OCR software available in the markets like: Desktop OCR, Server OCR, Web OCR etc. Accuracy rate of any OCR tool varies from 71% to 98% [11]. In this project we used Tesseract, an open source OCR Engine written in C++. The application for reading NH license plates is written using a Java wrapper for Tesseract, Tess4J.

3.1.1 **History of Tesseract**

Tesseract is an open-source OCR engine that was developed at HP between 1984 and 1994. Tesseract began as a PhD research project [12] in HP Labs, Bristol, and gained momentum as a possible software and/or hardware add-on for HP's line of flatbed scanners. Motivation was provided by the fact that the commercial OCR engines of the day were in their infancy, and failed miserably on anything but the best quality print. At the end of this project, at the end of 1994, development ceased entirely. The engine was sent to UNLV for the 1995 Annual Test of OCR Accuracy, where it proved its worth against the commercial engines of the time.
In late 2005, HP released Tesseract for open source. As of now Tesseract version 3.01 is released and available for use. Now it is developed and maintained by Google. It provides support for various languages and is available at:

http://code.google.com/p/tesseract-ocr

3.1.2 Tesseract Architecture

The working process of Tesseract is a traditional step-by-step process adhering to pipeline architecture. Below are the steps:

1. Adaptive Thresholding, which converts the image into binary images.
2. Connected component analysis, which is used to extract character outlines. Then after, the outlines are converted into Blobs. Blobs are organized into text lines, and the lines and regions are analyzed for some fixed area or equivalent text size. Text is divided into words using definite spaces and fuzzy spaces.
3. Recognition of text is then started as two-pass process. In the first pass, an attempt is made to recognize each word from the text. Each word passed satisfactorily is passed to an adaptive classifier as training data.
4. As adaptive classifier has received some training data it has learn something new so final phase is used to resolve various issues and to extract text from images.
3.2 Image Processing

Another important element in license plate recognition is the preprocessing of the image. The license plate image acquired through camera is a 2D view of a 3D world. Images acquired through a digital camera may be contaminated by a variety of noise sources. When a computer receives the image it is a grid of number. This data is corrupted by noise and distortions. Such corruption stems from variations in the world (weather, lighting, reflections, movements), imperfections in the lens and mechanical setup, finite integration time on the sensor (motion blur), electrical noise in the sensor or other electronics, and compression artifacts after image capture.
While most current cameras will reduce the noise levels to a very low level, noise can never be eliminated completely. Image resolution is another factor to consider, lower resolution images can be difficult to recognize, and even though higher resolution images will provide much more information helping recognition rate, the resolution increase will translate directly into longer processing times. We may want to remove noise or damage from an image so that we can read the license plate number.

Computer Vision is the transformation of data from a still or video image into either a decision or a new representation [13]. The decision might be there are ten characters in the license plate image. A new representation might mean turning a color image into grayscale image. Computer Vision is a science and technology that is used to process images to obtain relevant information.

3.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source C/C++ library for image processing and computer vision developed by Intel. It is a library of programming functions mainly aimed at real time image processing. It is free for both commercial and non-commercial use under the open source BSD license [14]. OpenCV was originally written in C but now has a full C++ interface and all new development is in C++.

Other libraries available are Matlab and AForge. OpenCV is fast and efficient when compared to MatLabd and AForge. The code is highly optimized for image processing. JavaCV is a wrapper that allows accessing the OpenCV
library directly from within Java Virtual Machine (JVM) and Android platform. JavaCV wraps C API wherever possible, and C++ API when necessary.

3.2.2 OpenCV History

Toward this end, Intel launched many projects including real-time ray tracing and 3D display walls. One of the authors working for Intel at that time was visiting universities and noticed that some top university groups, such as the MIT Media Lab, had well developed and internally open computer vision infrastructures—code that was passed from student to student and that gave each new student a valuable head start in developing his or her own vision application. Instead of reinventing the basic functions from scratch, a new student could begin by building on top of what came before.

Thus, OpenCV was conceived as a way to make computer vision infrastructure universally available. With the aid of Intel’s Performance Library Team,* OpenCV started with a core of implemented code and algorithmic specifications being sent to members of Intel’s Russian library team. This is the “where” of OpenCV: it started in Intel’s research lab with collaboration from the Software Performance Libraries group together with implementation and optimization expertise in Russia.

Although Intel started OpenCV, the library is and always was intended to promote commercial and research use. It is therefore open and free, and the code itself may be used or embedded (in whole or in part) in other applications, whether commercial or research. Intel’s OpenCV library includes most of today’s
computer vision algorithm. It has windows and Linux versions, which accounts for portability across platforms.
CHAPTER 4

MINHLP SYSTEM DESCRIPTION

4.1 MINHLP Explained

Each ALPR system consists of four basic sections: Image Acquisition, License Plate Detection (LPD), Image Processing Module (IPM) and Optical Character Recognition (OCR) which can be further subdivided into eleven distinct parts. Our system, Module to Identify New Hampshire License Plates (MINHLP), focuses primarily on IPM and the processing steps that have to be done using open source products after detecting and acquiring the license plate, so that OCR module will read the characters correctly.

The first step is to load the image. The image must then be resized to a constant size. Then next step is to remove the noise in the image. It is important to understand that we are capturing the image from a moving vehicle, and neither a human eye nor a digital camera can perfectly capture all the information clearly. Our challenge to read the license plate is to reduce the noise in the image. Noise occurs when individual pixels in the image appear brighter or darker than they should be, due to the interference in the electronic circuits inside the camera. Noise mainly appears as random changes to pixels. Having noise in the image can make it harder to recognize the license plate number in the image. Noise is anything in the image that we are not interested in the image. It is an
important step in the algorithm to reduce the noise in the image to read the license plate.

4.2 Noise Filtering

The OpenCV system comes with several functions to reduce noise. Smoothing is one such operation. The premise of data smoothing is that one is measuring a variable that is both slowly varying and also corrupted by random noise. Then it can sometimes be useful to replace each data point by some kind of local average of surrounding data points. Since nearby points measure very nearly the same underlying value, averaging can reduce the level of noise without (much) biasing the value obtained. The smoothing operation has the effect of removing small-scale bright and dark structure from an image. The images are not smooth because the adjacent pixels are not smooth. The smoothing making adjacent pixels look more similar. Two common smoothing filters are Bbox filter (simple averaging) and Gaussian filter (center pixels weighted more).

In order to reduce the noise in the image the first step is to convert the image to gray scale using the \texttt{cvCvtColor} system call. After converting to gray image \texttt{cvSmooth} [15] is used to smooth the image. After smoothing we next segment the image. This is a process where meaningful features are extracted from the image. To get the meaningful features that is license plate characters we need to identify the pixels that correspond to particular features of interest.
The techniques can be contextual or non-contextual. In non-contextual mode, pixels are grouped together based on some global attribute such as grey level. In contextual mode, the relationships between images features are exploited [16]. Image thresholding is a segmentation technique that classifies pixels into two categories: those at which some property measured from the image falls below a threshold, and those at which that property equals or exceeds the threshold. Because there are two possible output values, thresholding creates a binary image. This feature was used to improve results in MINHLP.

4.3 Problems with Feature Variation

The problem with NH license plates and their identifying tag characters is that there are other characters also embedded in the license plate. The decal stickers and the NH logo (Live Free or Die) are also embedded along with license plate characters. In order to find the license plate characters, contour mapping is used. Each contour is mapped, tested and approximated with accuracy proportional to the contour perimeter. Square contours should have four vertices after approximation to a relatively larger area (to filter out noisy contours) and to be convex.

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. Assuming the image is a binary image, we can say, its contour is the curve joining the all the boundary white points. So if we find a contour in a binary image, we are finding the boundaries of objects in an image. As stated in the OpenCV documentation, "The
contours are a useful tool for shape analysis and object detection and recognition." The OpenCV system call cvFindContours() does the job for us and returns the number of contours detected in the image. If we iterate through a typical NH License Plate Image, our program detects 118 rectangle contours. We iterate through the data structure and filter out objects as necessary. As only rectangle objects were needed, we used cvRectangle function to get them all.

Based on the rectangle contours we draw, we needed to get the license plate characters. This step required experimentation to determine what features to look for. Going through all rectangle contours, a formula was developed to determine if the width and height fell in the proper category demonstrating this to be a license plate character. After getting all license plate characters, copy the characters in a blank image. This will give a clean image for Tesseract to read.

4.3.1 Results of Images from Various Steps

We developed a Java AWT application, MINHLP, to load the image and to pre process the images. After that the program will call java wrapper for Tesseract to read the image and output the results in a text file. Figure 1 below is a screen shot of MINHLP:
4.3.2 Image Smoothing

The 6 steps used by MINHLP is as follows:

1. First we load the image to the program where OpenCV stores it as a C structure, IplImage.
2. Next we check the image size and if it is not the size we defined we resize it. We used the OpenCV system call cvResize.
3. Tesseract requires a binary image with no noise so we converted the image to a gray scale using OpenCV’s cvCvtColor which takes three parameters: source image, destination image and color conversion code. Figure 2 shows the original license plate image and the gray image.
4. This is the most important process where we take all the noise from the image to get a clean image to detect the license plate characters. The first part of this process is to smooth image and after that apply adaptive threshold (see Figure 3.)

5. After getting the threshold image, we detected all contours. This is the most complicated process. We need to get all contours using OpenCV's cvFindContours. Then get all bounding rectangles from the contours detected using cvBoundRect. This will give all bounding rectangles in the image. We need only the bounding rectangles surrounding the license plate characters. In order to find that we applied the logic if the area is
between a certain range and width and height is between certain ranges then this will be a license plate character. Refer Appendix A for the code.

6. In this process we call Tesseract to read the image and get the results which it was able to recognize accurately showing result “UBANTU” (see Figure 4). This is done using Tess4j, an open source Java JNA wrapper for Tesseract OCR API. Tess4j is released and distributed under Apache License v2.0. The required libraries can be downloaded from the internet.

![Figure 4: Tesseract recognition result](image)

4.3.3 Image Cropping

In another sample, where we got the image slanted, we took a different approach after getting our image. The quality of the image was also not good. Once we got the location of the character, we were able to come with a logic to set the area needed. We got the location of the character by applying contour detection logic to find all bounding rectangles. We were not able to copy all contours since the contour method detected several other bounding rectangles, since the quality of the image was not good. We applied OpenCV's
cvSetImageROI which cuts the image to the desired portion (i.e. the region of interest). We know the license plate character will be in center portion of the image and we can cut the other parts to get a clean image. After that we applied the same logic for the first image. The Tesseract OCR was not able to identify the character 6 and 0. Out of seven characters we got only four characters correct. Figure 5 shows the various transformations of the image.

![Figure 5: Cropping image first](image.png)
CHAPTER 5

RESULTS AND SUMMARY

5.1 Performance

The performances of MINHLP can only be understood by a set of experiments. The analysis was done using fifty NH license plate images. The images were processed using the module developed using JavaCV/OpenCV code and then the images ran the outputs through Tesseract. The first set of analysis was done using OpenCV system calls cvSmooth, cvDilate and cvThreshold and then passed to Tesseract. The detection failed with 50% of images. The second set of experiments was done with a contour detection process where after doing pre-processing of the image, all bounding rectangles were detected through OpenCV contour detection method. After finding rectangle areas that match, the number plate characters were detected and then copied the detected license plate characters to a blank image. The detection failed for six license plates and the percentage of correctly detected plate numbers is 88.8%.

The precision level increased from 56% to 89%. The precision level is the measure expressed as a percentage based on the total number of characters of each license plate, aiming at a character identification precision. This measure, (hereby referenced as character precision) can be expressed as:
5.1.1 Font Problems

The most common recognition problems were distinguishing between:

- '5' and '6'
- '8' and '9'
- 'O' and 'U'
- '6' and 'B'
- 'W' and 'H'
- '2' and 'Z'
- distinguishing letter 'A'

NH license plate characters are curved characters that have straight left and right sides joined by semicircles (often perfect semicircles) on the top and bottom. The counters are box-shaped with rounded corners. The license plate fonts are different from the fonts we see in our everyday items—such as books, magazine, newspapers, brochures, direct mail pieces or in internet communications such as email, websites and PDF documents. Fonts are created by people who have experience in graphic design, commercial illustration. License plate fonts however are created by those trained as draftsmen, mechanical engineers, or product engineers working with the design
of the industrial equipment and processes. The approach is totally different for each.

5.1.2 Training to Detect Fonts

The Tesseract engine is trained to read the regular fonts we find in our everyday items. The analysis results indicate that Tesseract is having difficulty in distinguishing between characters that look similar. It is because of the License Plate font. We can improve the accuracy by training the Tesseract. The word ‘training’ means providing Tesseract with the font of the license plate characters so that when it sees the digit six of the license plate character, it will give the result as ‘6’. In order to train Tesseract you need two things: an image of the character and the name of the character. This will provided to Tesseract with the help of “box files.” The box files contain the coordinates of the bounding boxes around characters with the labels as what those characters are. The documentation to train Tesseract is provided on its website [17].

The process has two major steps. First, create the training documents and then teach the Tesseract about the documents. The first step is to determine the full character set to be used, and prepare a text or word processor file containing a set of examples. Each character requires a minimum of five to ten sample files. We used GIMP [18] to extract each character from the license plate Image. All samples of a single license plate character should go in a single tiff file. It is important to save your training text as a UTF-8 text file for use in the next step, in which you must insert the codes into another file.
For the next step, Tesseract needs a 'box' file to go with each training image. The box file is a text file that lists the characters in the training image, in order, one per line, with the coordinates of the bounding box around the image. Tesseract 3.0 has a mode in which it will output a text file of the required format, but if the character set is different to its current training, it will naturally have the text incorrect. So the key process here is to manually edit the file to put the correct characters in it [19]. We used an open source product called jTessBoxEditor to edit the box files [20].

5.2 Results

After training Tesseract for NH license plate characters we found the results improved from 86% to 96%. Table 1 below shows test results from training Tesseract with 30 images.

<table>
<thead>
<tr>
<th>Image #</th>
<th>Image</th>
<th>Result</th>
<th>Characters</th>
<th>Level</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1.jpg" alt="Image 1" /></td>
<td>71:09?</td>
<td>4</td>
<td>1</td>
<td>75%</td>
</tr>
<tr>
<td>2</td>
<td><img src="image2.jpg" alt="Image 2" /></td>
<td>XPLR</td>
<td>4</td>
<td>0</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 1: Training results for Tesseract
5.3 Conclusion

We have successfully demonstrated that given reasonable lighting conditions, reasonable image size, and perspective, it is possible to create an application using Open source products like OpenCV and Tesseract to read New Hampshire license plates. With some improvements in robustness and performance, this application should be more convenient and user friendly than the existing similar apps which require heavy customization.

The current license plate image processing algorithm has two major weaknesses that we would like to improve. In situations where the image captured is not clear and if there is too much noise in the image due to uneven lightning conditions, the results will fail in 50%. In order to improve this condition, the image pre processing techniques should be enhanced.

The second major weakness is that Tesseract does not provide reliable results unless the image given to it is very clean. The images we are feeding to Tesseract often have missing pixels, which result in incorrect outputs. Since the font type and font size are fixed we plan to try using template matching as a replacement to Tesseract. In addition to accuracy, the user has to click another button to call Tesseract to get the final results. There should be only one button and when the user clicks it, the result should be given. We had issues with the java wrapper for the Tesseract library. The future version should be Java JNI interface which should call Tesseract behind the scenes.


19. JTessBoxEditor: Tesseract Box Editor Available from: http://vietocr.sourceforge.net/training.html

APPENDIX A

CODE LISTINGS

MINHLP Source Code

Java code listing for NHPlateReader class

```java
package licenseReader;

import com.googlecode.javacpp.Loader;
import com.googlecode.javacv.CanvasFrame;

import static com.googlecode.javacv.cpp.opencv_core.*;
import static com.googlecode.javacv.cpp.opencv_highgui.cvLoadImage;
import static com.googlecode.javacv.cpp.opencv_highgui.cvSaveImage;
import static com.googlecode.javacv.cpp.opencv_imgproc.*;

import java.awt.Color;
import java.util.Random;

public class NHPlateReader {
    static String path = "C:/Users/sunitha_general/imageanalysis/images/NHIMAGES/";
    static CanvasFrame canvas;

    public static void main(String[] args) {
        canvas = new CanvasFrame("output");
        canvas.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);

        IplImage nplt = cvLoadImage(path + "image5.jpg");
        System.out.println("nplt before resizing" + path);
        System.out.println("nplt before resizing" + nplt.width() + nplt.height());

        IplImage output = cvCreateImage(cvSize(nplt.width(), nplt.height()),
                                         IPL_DEPTH_8U, 1);

        cvCvtColor(nplt, output, CV_RGB2GRAY);
        cvSaveImage(path + "GrayImage.jpg", output);
    }
}
```
cvSmooth( output, output, CV_GAUSSIAN, 7 );
cvSaveImage( path + "Smooth.jpg", output );

cvAdaptiveThreshold( output, output, 255, 
    CV_ADAPTIVE_THRESH_MEAN_C, 
    CV_THRESH_BINARY_INV, 11, 5 );
cvSaveImage( path + "AdaptiveThreshold.jpg", output );

CvMemStorage storage = CvMemStorage.create(); 
CvSeq contours = new CvContour( null );

int noOfContors = cvFindContours( output, storage, contours, 
    Loader.sizeof(CvContour.class), 
    CV_RETR_CCOMP, 
    CV_CHAIN_APPROX_NONE, 
    new CvPoint(0,0) );

System.out.println( "Contors Detected ="+noOfContors+" \n" );
CvSeq ptr = new CvSeq();

int count =1;
Random rand = new Random();
CvPoint p1 = new CvPoint(0,0), p2 = new CvPoint(0,0);

for (ptr = contours; ptr != null; ptr = ptr.h_next())
{
    Color randomColor = new Color( rand.nextFloat(),
        rand.nextFloat(),
        rand.nextFloat() );

    CvScalar color = CV_RGB( randomColor.getRed(),
        randomColor.getGreen(),
        randomColor.getBlue() );

    CvRect sq = cvBoundingRect( ptr, 0 );

    if ( ( ( sq.height() * sq.width() > 550 &&
            sq.height() * sq.width() < 20000 ) &&
        ( sq.height() > 65 &&
            sq.height() < 90 ) &&
        ( sq.width() > 8 && sq.width() < 55 ) )
    {
        System.out.println( "Contour No ="+count );
        System.out.println( "X ="+ sq.x()+" Y="+ sq.y() );
        System.out.println( "Height =" + sq.height()
            + " Width =" + sq.width() );

        p1.x( sq.x() );
p2.x( sq.x() + sq.width() );
p1.y( sq.y() );
p2.y( sq.y() + sq.height() );
cvRectangle( nplt, pl,p2, CV_RGB(255, 0, 0), 2, 8, 0 );
cvDrawContours( nplt, ptr, color, CV_RGB(0,0,0), -1,
                 CV_FILLED, 8, cvPoint(0,0) );

    count++;
}
}
System.out.println( "Count ="+(count-1) );
cvSaveImage( path + "Final.jpg", nplt );
}
}

Java code listing for ImageReader class

package licenseReader;

import java.io.File;
import javax.swing.JFrame;
import javax.swing.JLabel;

import net.sourceforge.tess4j.Tesseract;
import net.sourceforge.tess4j.TesseractException;

public class ImageReader extends JFrame
{
    public static void main(String args[])
    {
        new ImageReader();
    }

    ImageReader()
    {
        String result= tessRead();
        JLabel jResultLabel = new JLabel( result );

        //Set the position of the text, relative to the icon:
        jResultLabel.setVerticalTextPosition( JLabel.BOTTOM );
        jResultLabel.setHorizontalTextPosition( JLabel.CENTER );
        jResultLabel.setToolTipText( "Tesseract Result-------->");

        add(jResultLabel);
        this.setSize(300, 300);
        setVisible(true);
    }

    public static String tessRead()
    {
        String result= null;
        File imageFile = new File("C:\NHIMAGES/result/image10.jpg");

        System.out.println( "Count ="+(count-1) );
        cvSaveImage( path + "Final.jpg", nplt );
    }

    ImageReader()
    {
        String result= tessRead();
        JLabel jResultLabel = new JLabel( result );

        //Set the position of the text, relative to the icon:
        jResultLabel.setVerticalTextPosition( JLabel.BOTTOM );
        jResultLabel.setHorizontalTextPosition( JLabel.CENTER );
        jResultLabel.setToolTipText( "Tesseract Result-------->");

        add(jResultLabel);
        this.setSize(300, 300);
        setVisible(true);
    }

    public static String tessRead()
    {
        String result= null;
        File imageFile = new File("C:\NHIMAGES/result/image10.jpg");

        System.out.println( "Count ="+(count-1) );
        cvSaveImage( path + "Final.jpg", nplt );
    }

    ImageReader()
    {
        String result= tessRead();
        JLabel jResultLabel = new JLabel( result );

        //Set the position of the text, relative to the icon:
        jResultLabel.setVerticalTextPosition( JLabel.BOTTOM );
        jResultLabel.setHorizontalTextPosition( JLabel.CENTER );
        jResultLabel.setToolTipText( "Tesseract Result-------->");

        add(jResultLabel);
        this.setSize(300, 300);
        setVisible(true);
    }

    public static String tessRead()
    {
        String result= null;
        File imageFile = new File("C:\NHIMAGES/result/image10.jpg");

        System.out.println( "Count ="+(count-1) );
        cvSaveImage( path + "Final.jpg", nplt );
    }

    ImageReader()
    {..
try
{
    result = instance.doOCR(imageFile);
    System.out.println(result);
}
catch (TesseractException e)
{
    System.err.println(e.getMessage());
}
return result;