Mobile Real-Time License Plate Recognition

Author: Ahmad Gull Liaqat
Date: 2011-12-05
Subject: Software Technology
Level: Master
Course code: 5DV01E
Acknowledgement

I feel a great sense of accomplishment to be able to complete this thesis, as part of the requirement for our Master’s degree. I would have not been able to do this on my own without the supports of the people whom are important to me.

First, I would like to thank my supervisor, Rüdiger Lincke, for his continuous support and guidance. His advice and comments are crucial in my progress to complete this thesis.

I would also like to thank my parents and family for their words of encouragement and their supports throughout my study period. My friends are also important to me, for without them this experience would be less meaningful.
Abstract

License plate recognition (LPR) system plays an important role in numerous applications, such as parking accounting systems, traffic law enforcement, road monitoring, expressway toll system, electronic-police system, and security systems.

In recent years, there has been a lot of research in license plate recognition, and many recognition systems have been proposed and used. But these systems have been developed for computers.

In this project, we developed a mobile LPR system for Android Operating System (OS). LPR involves three main components: license plate detection, character segmentation and Optical Character Recognition (OCR). For License Plate Detection and character segmentation, we used JavaCV and OpenCV libraries. And for OCR, we used tesseract-ocr. We obtained very good results by using these libraries. We also stored records of license numbers in database and for that purpose SQLite has been used.

Keywords

License Plate, LP, License Plate Recognition, LPR, Optical Character Recognition, OCR, Open Source Computer Vision, OpenCV, JavaCV, Tesseract-ocr, ABBY-OCR, Online OCR API, ASPIRE-OCR.
# Table of Contents

1. **Introduction** ........................................................................................................................................ 1  
   1.1 Problem Description and Motivation .................................................................................................... 1  
   1.2 Goals and Criteria ................................................................................................................................ 2  
   1.3 Outline .................................................................................................................................................. 2  

2. **Background** ........................................................................................................................................ 3  
   2.1 Background on Android ........................................................................................................................ 3  
      2.1.1 History of Android .......................................................................................................................... 3  
      2.1.2 Architecture of Android .................................................................................................................. 3  
      2.1.3 Android NDK .................................................................................................................................. 4  
      2.2 Optical Character Recognition ........................................................................................................... 5  
      2.2.1 ABBYY-OCR .................................................................................................................................. 5  
      2.2.2 Tesseract-OCR ................................................................................................................................. 5  
      2.2.3 Aspire-OCR ..................................................................................................................................... 5  
      2.2.4 Online OCR API ................................................................................................................................ 6  
      2.2.5 Selection .......................................................................................................................................... 6  
      2.3 Image Processing ................................................................................................................................. 6  

3. **Features and Requirements** .................................................................................................................... 8  
   3.1 Requirements ........................................................................................................................................ 8  
   3.2 Features ................................................................................................................................................. 8  

4. **Design and Architecture** ....................................................................................................................... 9  
   4.1 Introduction ........................................................................................................................................... 9  
   4.2 Components of Proposed License Plate Recognition System .............................................................. 9  
      4.2.1 Image Acquisition ............................................................................................................................. 10  
      4.2.2 License Plate Extractor ..................................................................................................................... 10  
      4.2.3 Character Segmentation .................................................................................................................... 11  
      4.2.4 Optical Character Recognition ......................................................................................................... 11  
      4.2.5 Database Access Model .................................................................................................................... 11  
   4.3 Flow Chart of License Plate Recognition System .................................................................................. 12  

5. **Implementation** .................................................................................................................................... 13  
   5.1 Data Preparation ................................................................................................................................... 13  
      5.1.1 Haar-Training of License Plates ....................................................................................................... 13  
      5.1.2 Compilation of Tesseract-OCR ......................................................................................................... 14  
   5.2 System Implementation ........................................................................................................................... 15  
      5.2.1 Implementation of Image Acquisition ............................................................................................... 15  
      5.2.2 License Plate Extraction Implementation .......................................................................................... 15
5.2.2.1 License Plate Detection using Haar-like features .................................................. 15
5.2.2.2 License Plate Detection through Edge Detection .................................................... 16
5.2.3 Character Segmentation Implementation ....................................................................... 17
5.2.4 Optical Character Recognition .................................................................................... 17
5.2.5 Data Access Model ....................................................................................................... 18

6. Case Study .......................................................................................................................... 19
   6.1 Benchmarks .................................................................................................................... 19
   6.2 License Plate Detection .................................................................................................. 19
   6.2.1 License Plate Detection through Haar-Training ....................................................... 19
   6.2.2 License Plate Detection through Edge Detection ...................................................... 19
   6.3 Precision of Tesseract-OCR ......................................................................................... 19
   6.4 License Plate Recognition in Practice, Real Time ......................................................... 21
   6.5 Comparison with Other OCR Libraries .......................................................................... 23

7. Conclusion and Future Work ............................................................................................... 26
   7.1 Conclusion ...................................................................................................................... 26
   7.2 Future Work .................................................................................................................... 26

References .................................................................................................................................. 27
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>License Plate</td>
</tr>
<tr>
<td>LPR</td>
<td>License Plate Recognition</td>
</tr>
<tr>
<td>LPD</td>
<td>License Plate Detection</td>
</tr>
<tr>
<td>OCR</td>
<td>Optical Character Recognition</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Source Computer Vision</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2.1 Android Architecture ................................................................. 4
Figure 2.2 Overview of OpenCV Functionality ............................................. 7
Figure 4.1 Typical License Plate Recognition System ................................. 9
Figure 4.2 Components of License Plate Recognition System ...................... 10
Figure 4.3 Flow chart of LRP System .......................................................... 12
Figure 5.1 Positive Image .............................................................................. 13
Figure 5.2 Negative Image ........................................................................... 14
Figure 5.3 Grayscale Image .......................................................................... 15
Figure 5.4 Components of License Plate Recognition System ...................... 16
Figure 5.5 Binary Image ............................................................................... 16
Figure 5.6 License Plate ............................................................................... 16
Figure 5.7 Rotated License Plate ................................................................. 17
Figure 5.8 Canny Image ............................................................................... 17
Figure 5.9 Segmented Character ................................................................. 17
Figure 5.10 Masked Image ............................................................................ 17
Figure 6.1 Results after 1 try ....................................................................... 17
Figure 6.2 Results after 2 tries ..................................................................... 17
Figure 6.3 Results after 3 tries ..................................................................... 17

List of Tables

Table 6.1: License Plate Detection through Haar-Training .......................... 19
Table 6.2: License Plate Detection Results through Edge Detection .......... 19
Table 6.3: Tesseract-OCR Results ............................................................... 21
Table 6.4: License Plate Recognition in Practice ......................................... 22
Table 6.5: Comparison between Tesseract-ocr and Online API OCR .......... 25
1. Introduction

A vehicle registration plate, or license plate, is attached to vehicles for official identification purposes. The identifier, often numerical or alphanumerical, can be used for uniquely identifying a vehicle within the issuing regions database.

There are numerous reasons why it is necessary for individuals or organizations to identify a vehicle and thus its owner. Examples include law/police enforcement, traffic control, and access to restricted areas, electronic toll collection or checking parking permissions purposes.

In some of the applications like traffic law enforcement, road monitoring and expressway toll system, where license plate recognition is used, it is necessary to process a large number of vehicles in a short time. In daily life there is huge traffic on roads, in this scenario application has to do very fast processing. Otherwise, violators and criminals can escape. The detection of a single license plate and the recognition of its characters in a reliable way is an expensive task, since it relies on computation intensive algorithms. Dedicated systems have been developed for this purpose delivering the necessary computational power.

Once a license plate has been recognized it needs to be submitted to an, often remote, IT system to match it against a database finally identifying the vehicle and possibly its driver. Then this information can be processed and used for dedicated purposes.

1.1 Problem Description and Motivation

Besides the successful uses of the existing systems there are a number of areas where these are not practical, too heavy, not mobile enough, or too expensive to be used. These areas include private and public parking areas on permanent or temporary parking places.

In this thesis we want to explore the possibility of developing a highly mobile license plate recognition system that could be used to complement or replace some of the existing systems. In particular the use of Smartphone allows nowadays the instant connection to the Internet and associated databases. Yet, the biggest limitations of these devices is that due to their small size, they are often more suitable to provide information than generating it.

Since it is often time consuming typing in even so short alphanumeric combinations as license plate registration numbers and time is a factor, we want to improve this limitation by using existing detection and recognition techniques by adapting them to modern Smartphones like iPhone or Android devices. This means we want to replace the need for manually typing information by using the built-in camera, letting the device do the job. This might allow for saving a lot of time, in particular if license plates have to be recognized repeatedly, e.g., on a parking place.

Therefore the problem to be solved by this thesis is:

“Implement a mobile and real time detection and recognition system which allows the fast and accurate processing of license plate information without the need of manual input and which is running under the limited constraints (processing power and space) of modern smartphones.”
1.2 Goals and Criteria
The main goal of this thesis is to develop a mobile and real time detection and recognition system that allows to “read” license plate information in an automated way. This goal is met if we have an application that runs on at least one of the currently leading mobile devices with a normal startup time (less than 5 seconds). This includes Smartphones like iOS or Android OS powered devices with a built-in camera. The application should recognize license plate information with an accuracy of over 70% virtually instantly by simply pointing the device at a car.

Another goal is to have a good usability of the application. This goal is met if the application provides visual (real time camera picture and display of recognized data) and audio (e.g., beep) feedback when a license plate has been recognized, failed to recognize, to keep a history of the recognized license plates.

The last goal is to provide the application as a module that could be the core component of a general license plate recognition system. This goal is met if we provide a simple check of the recognized information against a test database.

Additional constrains are that the implementation should follow good coding and design practices, that if available, standard libraries should be used, and that the application runs on a large selection of potential mobile devices or platforms.

One initial limitation to the thesis is that it is limited to using Android OS as implementation platform. The reason is that we have only a development and test environment (PC and Android phone) for this platform available.

1.3 Outline
Chapter 2 presents the background information about Android, Image Processing and OCR libraries.
Chapter 3 defines requirements and features of our system in more detail.
Chapters 4 and 5 are describing in detail the architecture and implementation of our proposed system.
Chapter 6 presents a brief case study of our application. In this chapter, we tested different parts of license plate recognition (LPR) system and verify them with results.
Chapter 7 concludes the thesis and suggests future work.
2. Background
In this chapter, we provide background information about the Android platform. Furthermore, we give an overview of different optical recognition and image-processing libraries; we compare the alternatives, and discuss in particular the libraries that we selected for use in implementation.

2.1 Background on Android
Android is an operating system that is used in mobile phones and tablets. In other words, “Android is a software stack for mobile devices that includes an operating system, middleware and key applications” [1].

This OS uses modified Linux kernel internally, and its internal implementation language is Java. Android Software Development kit (SDK) is used for the development of Android-based applications. The SDK provides the tools and libraries necessary to begin developing applications that run on Android-powered devices.

2.1.1 History of Android
In October 2003, a small company named “Android Inc” was established by group of four people in California USA. The names of group members were Andy Rubin, Rich Miner, Nick Sears and Chris White [2]. The main purpose of this company was to develop software for mobiles. Later, in July 2005 “Google” purchased this company and started more research on Android to add more features and libraries. The first Android version for mobile phones launched in market for use in 2008, it was very basic and not very enriched. Until now, different Android versions have been released. The latest version is released with new features and bug free from the previous one. The latest Android version for mobile phone is v2.3.4. In February 2011; Google also released their first Android v3.0 based tablet [2].

2.1.2 Architecture of Android
Figure 2.1 shows Android operating system consists of five components.

- Application
- Application Framework
- Libraries
- Android Runtime
- Linux Kernel

Application
This is upper most component of Android Platform which contains default and inbuilt applications like home, contacts, alarm, maps and others [1].

Application Framework
This component provides a lot of APIs to developer. Using these APIs, developers can develop extremely rich and innovative applications. Developers can also access device hardware, location information and other functionalities throughout this framework [1].

Libraries
This component contains the libraries that are developed in C/C++. These libraries are provided to developers through application framework. Developers can develop 2D, 3D and database oriented applications using these libraries [1].

![Android Architecture](image)

**Figure 2.1: Android Architecture [1]**

**Android Runtime**
This component is further divided in to two parts.
- Core Libraries
- Dalvik Virtual Machine

**Core Libraries**
These libraries provide core functionality of java programming.

**Dalvik Virtual Machine**
This is software that is used to run java based applications on mobile phones. Java programs are compiled into byte code (.class). The byte code is compatible to java virtual machine but cannot be run directly on dalvik virtual machine (DVM). So, byte code (.class) is converted in .dex extension which is compatible to DVM [1].

**Linux Kernel**
This component act as separation layer between software and hardware layer. The Linux version 2.6 is used for the core system services like security, memory and security management etc.

**2.1.3 Android NDK**
Android NDK is software or a tool that is used to compile native code languages, e.g., C and C++, etc. [3]. The shared libraries are generated from native code. Then Android
developer can use these shared libraries in their applications. That would be beneficial for certain classes of applications, in the form of reuse of existing code. And in certain cases, it also increases the speed.

2.2 Optical Character Recognition
Optical Character Recognition is used to extract text from an image or a scanned document. This text is used for further processing such as it can be edited, formatted, searched, indexed and automatically translated or converted to speech [4].

Our thesis research also involved to study some of existing OCR libraries and selected one of them for our thesis. We studied following libraries:

- ABBYY-OCR
- Tesseract-OCR
- Aspire-OCR
- Online OCR API

2.2.1 ABBYY-OCR
ABBYY is very famous company that provides document recognition, data capture and linguistic software [5]. They also provide OCR SDK for mobile platforms. Using their SDK, developers can develop OCR based applications for mobiles. They provide SDKs for following mobile platforms.

- Android
- Windows Mobile
- Mobile Linux (Moblin)
- iPhone
- Symbians

ABBYY OCR takes an image as input in all known format like .png, .gif, .jpg, etc. and makes the image text editable and searchable. We can use this text for further processing and develop lot of applications.

2.2.2 Tesseract-OCR
Tesseract-ocr is an open source engine. It is written in C/C++ and developed at Hewlett-Packard (HP) lab between 1985 and 1996. But they never used it in their products. It was one of the top 3 engines in the 1995 UNLV Accuracy test [7]. In 2005, they released it as open source engine. From 2007, Google has taken the supervision of tesseract-ocr for further development and maintenance [7].

It takes gray or color image as input and gives output in text format. At the start, only .tiff type of images was supported but now it also supports other types of images like .png, .jpg, etc. It can read data in any language from image like English, Swedish, Danish, etc. and developers can train their own language if the support for a specific language is not available [8]. Google Developer tested it on Ubuntu and Windows operating system but it also works on other Linux platforms and Mac, etc. We can also use this library on mobile platforms like Android and iPhone etc.

2.2.3 Aspire-OCR
for Barcode reading [6]. SDK is used to develop Aspire-OCR oriented applications and it has support for following platforms:

- Windows
- Mac
- Linux
- Solaris

It takes almost every format of images like .png, .jpg, .bmp, etc. as input and returns an image text in string format.

2.2.4 Online OCR API

As the name indicates, it is an online service that provides APIs for OCR. By using these APIs, it is possible to develop different kinds of mobile and web based applications. [9]. It supports .png, .jpg, .gif and .tiff type of images and can read text from every well known language like English, Swedish, German, Danish, etc.

2.2.5 Selection

We select tesseract-ocr engine for the thesis. Because it is open source and also gives the choice to the developer add his own language. Also it helped us to develop a standalone/offline application that is not possible in the case of Online OCR API. ABBY and ASPIRE engines are not open source. If we want to use these OCR libraries in our application then we have to purchase license of them.

2.3 Image Processing

Image Processing is technique in which different operations are performed on image to manipulate image data [10]. In our thesis lot of image processing is involved. So, our research also includes finding suitable library for it. We found Open Source Computer Vision (OpenCV) library that is used in those projects which requires efficient and real time image processing.

“OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. OpenCV is released under a BSD license; it is free for both academic and commercial use. It has C++, C, and Python and soon Java interfaces running on Windows, Linux, Android and Mac. The library has >2500 optimized algorithms (see Figure 2.2). It is used around the world, has >2.5M downloads and >40K people in the user group. Uses range from interactive art, to mine inspection, stitching maps on the web on through advanced robotics” [11].

JavaCV provides wrapper classes for OpenCV [12]. By using JavaCV; we can develop real time applications in Java and Android.
Figure 2.2: Overview of OpenCV Functionality [11]
3. Features and Requirements

This chapter presents the features and requirements of our application. First, we will discuss the software and hardware specification that is needed to build and run the application and in the second part, we will give overview of different features of our application.

3.1 Requirements

The objective of our thesis is to detect and recognize license plate in real time using Android-based mobile phones. Our application should fulfill the following requirements:

- Read the standard Swedish cars license plate.
- Maximum expected distance between car and camera: 5 meters.
- Minimum Camera angle: 90 degree (looking straight at the license plate).
- Works properly in daylight.
- Minimum Camera resolution: 3 Mega Pixel.

It is expected that it would not work efficiently during nighttime, rainy and cloudy days because mobiles cameras are not equipped with proper lightning. It is also expected that it will give results with decreasing accuracy with angles deviating significantly from the 90-degree (ideal) angle.

3.2 Features

The application will start in landscape mode and work as LPR system. It would also work as license plate detector and draw a rectangle on license plate area. User will be able to see currently recognized license number on his mobile screen. Additionally, the color encodes if the license plate has been recorded in a database. Green color means that our application has already record of specific license plate in database. Red Color indicates that our system has no record against that specific license number.

If user rotates the mobile in portrait mode then he will be able to see the records of previously stored license numbers. Every stored license number will contain the information about the timing and location of stored license number and user will have the choice to email these license numbers list to anyone.
4. Design and Architecture
In this chapter the design and architecture of our proposed system is discussed.

4.1 Introduction
The typical components of the LPR system include video cameras, image grabber card, computer and the corresponding license plate recognition software [13] as we can see in Figure 4.1.

![Figure 4.1: Typical License Plate Recognition System [19]](image)

Our LPR system is designed to detect and recognize license plates in real time through Android OS-based mobile phones. Our proposed system works in following steps:
1. Image is captured through mobile phone’s camera.
2. License plate is detected on and extracted from the image.
3. Character segmentation is done on that extracted license plate.
4. Segmented characters are passed to OCR engine.
5. OCR engine returns these segmented characters in text format.
6. Data in text format is matched in database.
7. Result is displayed on mobile screen.

4.2 Components of Proposed License Plate Recognition System
In this section we are going to discuss different components of our proposed system. Our proposed License plate recognition system consists of five components as we can see in Figure 4.2.
4.2.1 Image Acquisition
This is main component of LPR system. In this component following steps are performed:
1. Image Capturing
2. Image Resizing
3. Image Color Conversion
4. Image Smoothing
After performing the steps 1-4, the image is passed to next component.

4.2.2 License Plate Extractor
This is most critical process in License Plate Recognition System. In this process we apply different techniques on image to detect and extract license plate. This process is divided in two parts.

4.2.2.1 License Plate Detection through Haar-like features
In image processing techniques, Haar-like features are used to recognize objects from image [14]. If our proposed system is selected to detect only license plates then the Haar-like features are used for this purpose and no further processing is done.
4.2.2.2 License Plate Detection through Edge Detection
In the other case, if our proposed system has to recognize license plates, then the binary image is created from the image. After that following steps are performed to extract license plate from binary image:

1. Four Connected Points are searched from binary image.
2. Width/Height ratio is matched against those connected points.
3. License Plate region is extracted from image.
4. Transformation of extracted license plate is performed.

Then the extracted license plate is passed to next component for further processing.

4.2.3 Character Segmentation
In this part further image processing is done on extracted license plate to remove unnecessary data. After character segmentation, the extracted license plate has only those characters that belong to license number.

4.2.4 Optical Character Recognition
In this part, character segmented license plate is passed to optical character recognition engine. OCR engine returns license plate in text format.

4.2.5 Database Access Model
Database access model uses OCR results for data insertion and searching in the database.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LicenseDataDB</td>
<td>This is main database</td>
</tr>
<tr>
<td>LicenseDbTable</td>
<td>LicenseDbTable is database table, in which LPR record is stored</td>
</tr>
</tbody>
</table>

Table 4.1: Database Access Model

Table 4.1 and 4.2 contains the information of database. LicenseDbTable have three fields as we can see in Table 4.2. In this table StoredDate and Location contain the license number storage date and location.

<table>
<thead>
<tr>
<th>Fields Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>License No</td>
<td>String</td>
</tr>
<tr>
<td>StoredDate</td>
<td>Date</td>
</tr>
<tr>
<td>Location</td>
<td>String</td>
</tr>
</tbody>
</table>

Table 4.2: Database Table Fields
4.3 Flow Chart of License Plate Recognition System

Figure 4.3 shows flow chart of license recognition system. We can observe in this flowchart that our license recognition system has two modes. These two modes performed different functionality. Like license plate detector mode detect only license plates and other mode also reads license numbers from license plates.
5. Implementation
In this chapter, we discuss the implementation of the different implemented components of the license plate recognition system.

5.1 Data Preparation
In this section we will give the overview of preparing the data. It is an essential step prior to the implementation.

5.1.1 Haar-Training of License Plates
OpenCV provides us tools and functions to implement Haar-like features [15]. In OpenCV, Haar-like features are implemented through Haar-Training [16].

Haar-Training is the process to train objects for detection. OpenCV provides us with tools for this purpose [16]. For Haar-Training, we need two kinds of images positive and negative.

Positive Images: This kind of images contain objects that we want to detect as shown in Figure 5.1. In our case it is license plate.

![Figure 5.1: Positive Image](image)

Negative Images: Negative Images are those images that do not contain the target object as shown in Figure 5.2. We did Haar-Training of License Plates in following steps.

Data Collection
We needed almost 5000 positive images (license plates) for training. It is almost impossible to find such amount of images from Internet. So for training, we captured 2500 images of Swedish cars having license plates. This process took almost 2 days. Then we cropped 2000 license plates from the images. This cropping process also took
2 days. Then we created 5000 license plate images from the 2000 images using OpenCV utility `createsamples` [16]. This utility also packed these images into the `positive.vec` file. This file is used in training.

For training, we also needed 12,000 negative images. We collected these images from different sources like friends, colleagues etc.

**Training**
The OpenCV utility `haartraining` is used for the training [16]. During training, negative images are used as background of positive images.

The training took almost 7 days to complete. A file with .xml extension is created at the end of training. In our case, we got `haartraining.xml`. We used system for training with the following specification:

- CPU: 2.34 GHz
- CPU Cores: 4
- RAM: 8 GB
- Operating System: CentOS Linux

**5.1.2 Compilation of Tesseract-OCR**
We selected tesseract-ocr library for optical character recognition [7]. This library is written in C++ language. We are implementing this thesis for Android based mobiles. Java is the basic language used for Android development. That's why tesseract-ocr cannot be used directly in Android based projects. To use tesseract-ocr in our thesis, we created shared library (`tesslib.so`) using Android NDK [3].
5.2 System Implementation
We are implementing our solution for Android enabled platforms using Android SDK version 2.3.4 and Eclipse IDE. In our thesis, lot of image processing is involved and for image processing, we used JavaCV [12].

As we discussed in Design and Architecture chapter that license plate recognition system consists of five components.

1. Image Acquisition.
2. License Plate Extraction.
3. Character Segmentation.
4. Optical Character Recognition.
5. Data Access Model.

5.2.1 Implementation of Image Acquisition
In this component, an image is captured using Android’s built-in class Camera.java. After image capturing, sub sampling of image is done by factor four. Subsampled image is converted into gray scale image using JavaCV.cvColor function as shown in Figure 5.3. This grayscale image is used in next component for further processing.

5.2.2 License Plate Extraction Implementation
As we discussed in previous chapter, this module has two parts:

- License Plate Detection using Haar-like features.
- License Plate Detection through Edge Detection.

5.2.2.1 License Plate Detection using Haar-like features
The grayscale image from previous stage and haartraining.xml file that we obtained through training as discussed in Section 5.1.1 is passed to JavaCV.cvHaarDetectObjects function. This function detects license plate from the image using haartraining.xml file as we can see in Figure 5.4.

![Figure 5.3: Grayscale Image](image_url)
5.2.2.2 License Plate Detection through Edge Detection

In this part, grayscale image is converted to binary image as we can see in Figure 5.5. In our implementation, we convert grayscale image to binary image by using JavaCV.cvThreshold function. In Sweden, standard license plates are usually in rectangle shape. We used binary image to detect rectangle from the image. The rectangle is detected from binary image using JavaCV.cvFindCountour function.

Figure 5.5: Binary Image

This function returns arrays of connected points. If the length of connected points is four then we check height and width ratio of the area between these points. If the height and width ratio is between 3 and 8 we suppose, this area has license plate and we extract that area from grayscale image as we can see in Figure 5.6.

Figure 5.6: License Plate
License plate should be along x-axis; if it is not then we have to rotate at specific angle. As we can see in Figure 6.3, license plate is not along x-axis and as we rotate this license plate to -1.5 angles with x-axis, we get license plate at x-axis as it is shown in Figure 5.7.

![BZN846](image)

Figure 5.7: Rotated License Plate

The rotated plate is passed to character segmentation module for further processing.

### 5.2.3 Character Segmentation Implementation

Canny edge detection algorithm is used to find edges in the image [17]. We used `JavaCV.cvCanny` function for this purpose as shown in Figure 5.8. After finding edges, connected edges are found using `JavaCV.cvFindCountour` function. Then bounding rectangles are found against these connected edges. The height of bounding rectangle is compared with the height of license plate.

![BZN846](image)

Figure 5.8: Canny Image

If the bounding rectangle height is greater than the ½ times height of the license plate, then it is supposed that rectangle area belongs to license number and it is copied to new image as we can see in Figure 5.9.

![BZN846](image)

Figure 5.9: Segmented Characters

After character segmentation, masking is done on that image as shown in Figure 5.10. Image masking is the operation of image processing which is used to remove noise from image [18].

![BZN846](image)

Figure 5.10: Masked Image

### 5.2.4 Optical Character Recognition

We used tesseract-ocr (`tesslib.so`) for this module that we discussed in Section 5.1.2. The masked license plate as in Figure 5.10 is passed to it. This returned license plate number in text format, e.g., “BZN846”. The standard Swedish license plate number consists of 6 characters, three capital letter followed by three digits as we can see in Figure 5.10. We checked the validity of license plate number; if it is proper license number then it is sent to data access model for further processing otherwise no further processing is done.
5.2.5 Data Access Model
In this module, we handle database operations like insertion and searching. We used SQLite as database. The license number is searched in database; if it is present in database then it is shown to user in green color. Otherwise it is stored in database with the current date and location and that license number is shown to user in red color.
6. Case Study
In this chapter, we discuss benchmarks that we used to test our system and also give overview of their results.

6.1 Benchmarks
We collected 30 images of Swedish cars having license plates in different light conditions. We divided these images in to 3 groups and each group has 10 images. We gave name these groups A, B and C respectively. We run our test cases on these groups which are further discussed in following sections.

6.2 License Plate Detection
As we discussed previously in Architecture and Implementation chapters, we implemented two algorithms for license plate detection: Haar-Training and Edge Detection. We run the test cases on these methods separately to compare their performance.

6.2.1 License Plate Detection through Haar-Training

<table>
<thead>
<tr>
<th>Groups</th>
<th>Total License Plates</th>
<th>No. of Detected License Plates</th>
<th>Detection Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group A</td>
<td>10</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>Group B</td>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Group C</td>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Combine Results of 3 Groups</td>
<td>30</td>
<td>29</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 6.1: License Plate Detection through Haar-Training

Table 6.1 shows the results of license plate detection through haar-training. We can see in this table that Group A has less detection as compared to other groups but detection rate of haar-training for 3 groups is 0.96.

6.2.2 License Plate Detection through Edge Detection

<table>
<thead>
<tr>
<th>Groups</th>
<th>Total License Plates</th>
<th>No. of Detected License Plates</th>
<th>Detection Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group A</td>
<td>10</td>
<td>8</td>
<td>0.8</td>
</tr>
<tr>
<td>Group B</td>
<td>10</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>Group C</td>
<td>10</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>Combine Results of 3 Groups</td>
<td>30</td>
<td>26</td>
<td>0.86</td>
</tr>
</tbody>
</table>

Table 6.2: License Plate Detection Results through Edge Detection

Table 6.2 shows us results that we got through edge detection algorithm. Group A has lower detection rate as compared to other groups but detection rate of edge detection algorithm against 3 groups is 0.86 which is quite satisfactory.

6.3 Precision of Tesseract-OCR
As we discussed in Chapter 5, we used the edge detection algorithm and tesseract-ocr library in combination for LPR. Table 6.3 shows OCR results. The second column contains those license plates that were extracted in previous section using edge detection algorithm.

19
<table>
<thead>
<tr>
<th>Detected LP</th>
<th>Masked LP used for OCR</th>
<th>OCR Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Srazier 363</td>
<td>Srazier 363</td>
</tr>
<tr>
<td>2</td>
<td>FUJ 524</td>
<td>FUJ 524</td>
</tr>
<tr>
<td>3</td>
<td>APE 978</td>
<td>APE 978</td>
</tr>
<tr>
<td>4</td>
<td>HRX 369</td>
<td>HRX 369</td>
</tr>
<tr>
<td>5</td>
<td>TRT 438</td>
<td>TRT 438</td>
</tr>
<tr>
<td>6</td>
<td>RWG 088</td>
<td>RWG 088</td>
</tr>
<tr>
<td>7</td>
<td>PYN 322</td>
<td>PYN 322</td>
</tr>
<tr>
<td>8</td>
<td>CEX 414</td>
<td>CEX 414</td>
</tr>
<tr>
<td>9</td>
<td>ORJ 211</td>
<td>ORJ 211</td>
</tr>
<tr>
<td>10</td>
<td>PBT 536</td>
<td>PBT 536</td>
</tr>
<tr>
<td>11</td>
<td>KCS 404</td>
<td>KCS 404</td>
</tr>
<tr>
<td>12</td>
<td>SZP 201</td>
<td>SZP 201</td>
</tr>
<tr>
<td>13</td>
<td>ALR 064</td>
<td>ALR 064</td>
</tr>
<tr>
<td>14</td>
<td>DYF 678</td>
<td>DYF 678</td>
</tr>
<tr>
<td>15</td>
<td>JOF 393</td>
<td>JOF 393</td>
</tr>
<tr>
<td>16</td>
<td>CDG 328</td>
<td>CDG 328</td>
</tr>
<tr>
<td>17</td>
<td>RZH 629</td>
<td>RZH 629</td>
</tr>
<tr>
<td>18</td>
<td>KJL 611</td>
<td>KJL 611</td>
</tr>
<tr>
<td>19</td>
<td>NEZ 258</td>
<td>NEZ 258</td>
</tr>
<tr>
<td>20</td>
<td>TST 708</td>
<td>TST 708</td>
</tr>
<tr>
<td>21</td>
<td>TKY 143</td>
<td>TKY 143</td>
</tr>
<tr>
<td>22</td>
<td>FSA 265</td>
<td>FSA 265</td>
</tr>
<tr>
<td>23</td>
<td>STD 347</td>
<td>STD 347</td>
</tr>
<tr>
<td>24</td>
<td>JWG 790</td>
<td>JWG 790</td>
</tr>
<tr>
<td>25</td>
<td>KJJ 344</td>
<td>KJJ 344</td>
</tr>
</tbody>
</table>
We already know that masked license plates are passed to tesseract-ocr engine that are shown in third column. Last column contains the OCR results that were obtained through tesseract-ocr engine. The numbers in red color represent the incorrect results. The OCR precision can be calculated on the bases of correct OCR results and numbers of extracted plates.

Total Numbers of Extracted License Plates=26  
Correct OCR Results=22  
Precision of OCR= (22/26)*100= 84.6%

We can see from the calculation that the precision of tesseract-ocr results is almost 85 percent, which is quite acceptable.

In conclusion, we see from the above results that haartraining algorithm has higher detection rate as compared to edge detection algorithm but edge detection algorithm also shown good results. Our test cases results for tesseract-ocr also have shown us that it has 85% accuracy for character recognition. Now we can easily calculate the precision of our LPR system.

Correct OCR Results=22  
No. of License Plate used in test cases=30  
Precision of LRP System= (22/30)*100= 73%

The calculation shows that the precision of our LPR system is 73%.

### 6.4 License Plate Recognition in Practice, Real Time

We select 20 cars from university area having standard Swedish license plates and test our license plate recognition application. We tested to find that how many tries it needed to recognize license plate. We gave maximum 3 tries to recognize license plate.

<table>
<thead>
<tr>
<th>No. of Tries</th>
<th>Results</th>
<th>Beep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6.4: License Plate Recognition in Practice

Table 6.4 shows the results that we got on the basis that we discussed at the start of this section. We can see in table that it happened 3 times, our application was unable to recognize license plate. The Figure 6.1 shows the result that we got after one try and the Figure 6.2 shows the result of second try. In both cases, our application was able to recognize the license plate. Similarly, results in Table 6.4 shows that when our application recognize license plate it generate good beep and in the other case bad beep is generated.

![Figure 6.1: Results after 1 try.](image1)

![Figure 6.2 Results after 2 tries.](image2)
The Figure 6.3 shows that after 3 tries; our system was unable to recognize the license plate.

Total No. of Cars=20
Detected License Plates=17
Precision of LPR in Real Time= (17/20)*100=85%

The calculation shows that the precision of our LPR system in real time is 85%.

Figure 6.4 shows the store record of license plate numbers that were correctly recognized during experiment.

6.5 Comparison with Other OCR Libraries
Here we presented other OCR results that we got using other OCR engines. We have already mentioned the reason in section 2.2.5, why we selected tesseract-ocr but in this
section, we want to see whether the selected one gives good results or not as compared to others.

**Abby-OCR**
We have no access to this library because this engine requires license.

**Aspire-OCR**
This library also requires license to use.

**Online OCR API**

<table>
<thead>
<tr>
<th>Masked LP Used for OCR</th>
<th>Tesseract-OCR Results</th>
<th>Online OCR API Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SZR 363</td>
<td>SZR 363</td>
<td>SZR 363</td>
</tr>
<tr>
<td>2 FUJ 524</td>
<td>FUJ 524</td>
<td>FUJ 524</td>
</tr>
<tr>
<td>3 APE 978</td>
<td>APE 978</td>
<td>APE 978</td>
</tr>
<tr>
<td>4 HRX 369</td>
<td>HRX 369</td>
<td>HRX 369</td>
</tr>
<tr>
<td>5 TRT0438</td>
<td>TRTEA38</td>
<td>TRTD438</td>
</tr>
<tr>
<td>6 RWG 088</td>
<td>RWG 088</td>
<td>RWG 0881</td>
</tr>
<tr>
<td>7 PYN 322</td>
<td>PYN 322</td>
<td>PYN 322</td>
</tr>
<tr>
<td>8 CEX 414</td>
<td>CEX 414</td>
<td>CEX 414</td>
</tr>
<tr>
<td>9 ORJ 211</td>
<td>ORJ 211</td>
<td>ORJ 211</td>
</tr>
<tr>
<td>10 PBT 536</td>
<td>PBT 536</td>
<td>PBT 536</td>
</tr>
<tr>
<td>11 KCS 404</td>
<td>KCS 404</td>
<td>KCS L04</td>
</tr>
<tr>
<td>12 SZPc201</td>
<td>SZPE20</td>
<td>SZPE201</td>
</tr>
<tr>
<td>13 ALR 064</td>
<td>ALR 064</td>
<td>ALR064</td>
</tr>
<tr>
<td>14 DYF 678</td>
<td>DYF 678</td>
<td>DYF 678</td>
</tr>
<tr>
<td>15 JOF 393</td>
<td>JOF 393</td>
<td>JOFT 393</td>
</tr>
<tr>
<td>16 CDG 328</td>
<td>CDG 328</td>
<td>CD6 328</td>
</tr>
<tr>
<td>17 RZH 629</td>
<td>RZH 629</td>
<td>RZH 629</td>
</tr>
<tr>
<td>18 KJL 611</td>
<td>KJL 611</td>
<td>KJL 611</td>
</tr>
<tr>
<td>19 NEZ 258</td>
<td>NEZ 258</td>
<td>NEZ 258</td>
</tr>
<tr>
<td>20 TST 708</td>
<td>TST 708</td>
<td>TST 708</td>
</tr>
</tbody>
</table>
Table 6.5: Comparison between Tesseract-ocr and Online API OCR.

Table 6.5 shows result of tesseract-ocr and Online API OCR engines. We can see in this table that tesseract-ocr has better accuracy as compared to Online OCR API. The data in red color shows incorrect result. The below calculation results show the percentage of correct results.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>TKY 143</td>
<td>TKY143</td>
</tr>
<tr>
<td>22</td>
<td>FSA 265</td>
<td>FSA 265</td>
</tr>
<tr>
<td>23</td>
<td>STD 341</td>
<td>STD 341</td>
</tr>
<tr>
<td>24</td>
<td>JWG 790</td>
<td>JWG 790</td>
</tr>
<tr>
<td>25</td>
<td>KKJ 344</td>
<td>KKJ 344</td>
</tr>
<tr>
<td>26</td>
<td>KAW 820</td>
<td>KAW 820</td>
</tr>
</tbody>
</table>

Total no. of license plates =26
No. of correct results=19
Precision= (19/26)*100= 73%

The precision of Online API OCR is 73% which is less than tesseract-ocr (85%).
7. Conclusion and Future Work

7.1 Conclusion
As we discussed in chapter 1 that our thesis main goal was to develop mobile based real time detection and recognition system that allows to “read” license place information in an automated way and recognize license plate information with an accuracy of over 70%, virtually instantly by simply pointing the device at a car. We can see from the results in chapter 6 that we have achieved these goals. We implemented LPR system for android using JavaCV and Tesseract-OCR. Our LPR system consists of 5 components, i.e., Image Acquisition, license plate extraction, character segmentation, and OCR and data access model. Our first goal was to develop such LPR system that should have precision over 70%. We tested precision of our system against images stored in database and real time. The results from case study chapter show that precision of LPR system using images from database is 73 % which is quite satisfactory. The real time experiment also shows that precision of our LPR system is 85%. Its means that we have achieved our first goal. Our second goal was to have a good usability of the application. This goal is achieved if the application provides visual (real time camera picture and display of recognized data) and audio (e.g., beep) feedback when a license plate has been recognized, failed to recognize, to keep a history of the recognized license plates. The real time experiment result in section 6.4 from case study shows that we have achieved this goal as well.

We used two algorithms for license plate detection, the results shows that haar-training has better detection rate (96%) as compared to edge detection algorithm (83%) but the statistics of edge detection algorithm in Table 6.2 also show that detection rate of this algorithm is not bad. Other objective was to use standard libraries, so we used tesseract-ocr for Optical Character Recognition and the results from experiments in Table 6.3 show that it has 85% accuracy. The comparison between tesseract-ocr and other OCR engines in Table 6.5 shows that tesseract-ocr has better accuracy.

As we see from the results that license plate extraction and OCR components affect the results of LPR system. We can improve our system by using efficient algorithms for these components.

7.2 Future Work
Currently our LPR system is using edge detection algorithm for license plate extraction. We can use haartraining algorithm instead of this algorithm as results from case study shows that it has better detection rate.

We are using local database for data storage. For practical implementation remote data base server can be used through web services.
References
