Automatic License Plate Recognition Mobile App: Parking System

Nandan Grover
UG Student
Department of Computer Engineering
Bharati Vidyapeeth College of Engineering

Rohan Gavhankar
UG Student
Department of Computer Engineering
Bharati Vidyapeeth College of Engineering

Meghana Yadav
UG Student
Department of Computer Engineering
Bharati Vidyapeeth College of Engineering

Prof. Sheetal Thakare
Assistant Professor
Department of Computer Engineering
Bharati Vidyapeeth College of Engineering

Abstract

Automatic license plate recognition (LPR) plays an important role in numerous applications and a number of techniques have been proposed regarding the same. LPR is the extraction of vehicle license plate information from a captured image.[1] The extracted information can be used with or without databases in many applications such as electronic payment systems (parking fee payment) and monitoring system for traffic surveillance. LPR uses the camera from a smartphone to capture the image. The quality of the acquired images is a major factor in the success of the LPR. LPR as a real life application has to quickly and successfully process license plates under different environmental conditions such as day, night, and outdoors.[2] Image processing techniques that will be used are image detection, image binarization, character analysis, identification of plate edges, deskewing the image, character segmentation, optical character recognition, post processing for calculation of confidence.

Keywords- Open ALPR, Cloud ALPR, Android Studio, Internet and Java

I. INTRODUCTION

In recent years, Android Platform has achieved so much popularity in terms of the wide range of application we use in our day to day life, making us forget about our old traditional ways and the complications along with them. The system which we are designing, it can be employed to replace the existing LPR as a cheap alternative in offices, colleges for better space utilization and efficiency. The system will also reduce man power allotted for the same and will not require any qualification as the images are being simply captured from a smart-phone.

It can also be used to give out parking tickets by calculating the time a vehicle was parked. Traditional system approach used expensive CCTVs cameras, video monitoring which required skills as well as man power to monitor them. Even for the parking system, the man power as well as hardware, is required to give the directions for parking instead of allotting the space details on the token provided. Our proposed system uses OpenCV library, OpenALPR to do the license plate detection in a cheap and effective way. The system will be able to detect license plates and give out parking tickets.

II. METHODOLOGY

Mobile based app is the sole focus of this project. We use android OS for this purpose. Android studio is the development tool used to develop the interface with the code being written in Java. Maven repository is used to store the openalpr libraries locally. The OpenALPR Cloud API is a web service running in the cloud that analyzes images of vehicles and responds with license plate data, as well as vehicle color, make, model, and body type. This web service has been used to a great effect in developing the project.

III. PROPOSED SYSTEM

In India, basically, there are two kinds of license-plates, black characters in white plate and black characters in yellow plate. The former for private vehicles and latter for commercial, public service vehicles. The system tries to address these two categories of plates. The system consists of six stages, where Android mobile application is designed using the Number plate recognition, Segmentation and Optical character recognition and these are helpful in extracting and displaying the number plate on the screen.

Fig. 1:

Input image using camera, this steps acquires the image to be clicked in order for further processing. The camera used is of 16 MP, with 1920 x 1080 pixels.
Number plate detection performs detects the plate the main focus area for processing. Character segmentation segments the character to obtain the required result. OCR is a process that converts the images to machine encoded text.

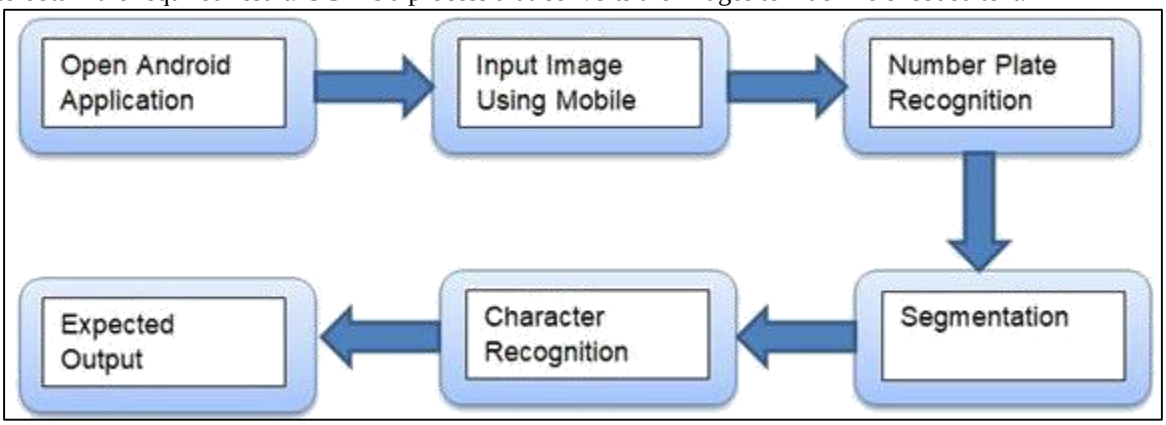

Fig. 2:

IV. WORKING

A. Image Processing Module

The images processing module makes use of the API called cloudapi, which uses the openalpr library. Cloudapi makes it easier for us to use certain functions from openalpr without directly interacting with it. It makes it much easier to implement a faster, reliable and efficient license plate detection system. Data thus flow from the smartphone to the cludapi and then to openalpr library. Image is then processed and is passed back to the smartphone.

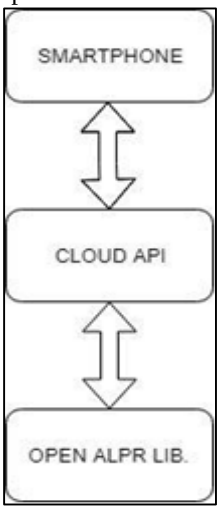

Fig. 3:

OpenALPR operates as a pipeline. The input is an image, various processing occurs in stages, and the output is the possible plate numbers in the image.
The pipeline stages occur in the following order [5]:

Pipeline Phase	*C++ class*	*Description*
Detection	*regiondetector.cpp*	*Finds potential license plate regions*
Binarization	*binarizewolf.cpp*	*Converts the plate region image into black and white*
Char Analysis	*Characteranalysis .cpp*	*Finds character-sized "blobs" in the plate region*
Plate Edges	*platelines.cpp and platecorners.cpp*	*Finds the edges/shape of the license plate*
Deskew	*Licenseplatecandi date.cpp*	*Transforms the perspective to a straight-on view based on the ideal license plate size.*
Character Segmentation	*charactersegmenter.cpp*	*Isolates and cleans up the characters so that they can be processed individually*
OCR	*ocr.cpp*	*Analyzes each character image and provides multiple possible letters/confidences*
Post Processing	*postprocess.cpp*	*Creates a top n list of plate possibilities based on OCR confidences. Also performs a Regex match against region templates if requested*

B. Detection
For every input image detection phase happens. The LBP algorithm is used to find possible license plate regions(x, y, width, height).each of these regions are directed to the subsequent pipeline phases for further processing.

The most processing intensive phase is always considered to be the detection phase. To improve the performance it can be GPU accelerated. The GPU provides far superior application performance by removing processing-intensive application sections to GPU.[5]

C. Binarization
All subsequent phases including this one occur several times at least once for every probable license plate region.
Multiple binary images are generated in this phase for each plate region. The best possible chance of finding all the characters is by using multiple binary images. A single binarized image may miss characters in case that the image is too dark or too light. Binarization uses the Wolf-Jolien method as well as the Sauovola method with various parameters. All the sequential phases processes each of the binary images.[5]

D. Character Analysis
Character analysis tries to search character sized region in the plate region. To do the following it first finds all connected blobs in the license plate region. Then it searches for blobs that are approximately the width and height of a license plate character and have tops/bottoms that are in a straight line with other blobs of similar width/height. This analysis is performed multiple times in the region. First it looks for smaller characters then serially looks for larger characters.

If the search result is negative, then the region is thrown out and no further processing takes place. If it finds some potential characters, then the character region is saved and further processing takes place.[5]

E. Plate Edges
The next phase is to find the edges of the license plate. Keep in mind that the detection phase is only responsible for identifying a possible region where a license plate may exist. It often is going to provide a region that is a little larger or smaller than the actual plate.

The plate edges tries to find the precise top/bottom/left/right edges of the license plate. The first step is to find all of the hough lines for the license plate region. Platelines.cpp method in the OpenALPR library processes the plate image and computes a list of horizontal and vertical lines.

Platecorners uses this list as well as the character height (computed in Character Analysis) to find the likeliest plate line edges. It uses a number of configurable weights to determine which edge makes the most sense. It will try using a default edge (based on the ideal width/height of the plate) to see if that makes a good match.[5]

F. Deskew
Given the plate edges, the deskew stage remaps the plate region to a standard size and orientation. Ideally this will give us a correctly oriented plate image (no rotation or skew).[5]

G. Character Segmentation
The character segmentation phase tries to isolate all the characters that make up the plate image. It uses a vertical histogram to find gaps in the plate characters. This phase also cleans up the character boxes by removing small, disconnected speckles and disqualifying character regions that are not tall enough. It also tries to remove "edge" regions so that the edge of the license plate doesn't inappropriately get classified as a '1' or an 'I'.[5]

H. OCR

The OCR phase analyzes each character independently. For each character image, it computes all possible characters and their confidences.[5]

I. Post Processing

Given a list of all possible OCR characters and confidences, post processing determines the best possible plate letter combinations. It is organized as a top N list. Post processing disqualifies all characters below a particular threshold. It also has a "soft" thresholds -- characters that are below this threshold will still be added to the possible list, but they also add a possible blank character -- since it's possible that the low confidence character is not really part of the plate.[5]

The post processing also handles region validation if requested. For example, if I tell OpenALPR that this is a "Maharashtra" plate, then it will try and match the results against a template that matches the Maharashtra format (e.g., [char][char][number]-[char][number]). So, for example, if the top 3 list was:

- MHOCI
- MH0CI
- MH0C1

The third entry matches the template, but the other two do not. So, post processing will signal that the third entry is our best match.

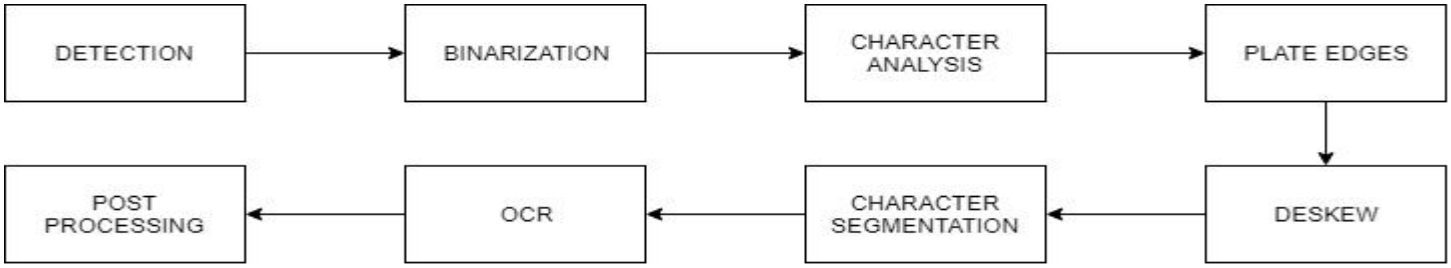

Fig. 4:

J. Parking Management

We have built a parking system on top of the image processing module. This parking system works in two ways. First, when a vehicle enters the parking lot it scans the number plate and compares it with existing number plates in the database. If it doesn't find any it saves the plate number in the database. Timestamp is recorded along with this. Secondly, when the same vehicle exits the parking lot, it scans the number plate again. This time while comparing the number plate with the existing number plates in the database, it finds a match. Our system then calculates the time elapsed and the amount owed by the customer to the parking lot.

V. RESULTS

Below are the screen displays on an android smartphone, when we open our Parking Management Application. It provides a login page for the user to fill in the credentials. The next screen displays a screen with Mark Vehicle and Clear Parking option. On tapping Mark Vehicle, the smart-phone's camera open up to capture the image, after getting captured its saved in the phone for the further processing and we get our desired output the number plate along with the time since it's parked. The user can even see a Parking status to find which spaces aren't available for parking currently, which can be cleared once the car leaves using Clear Parking option.

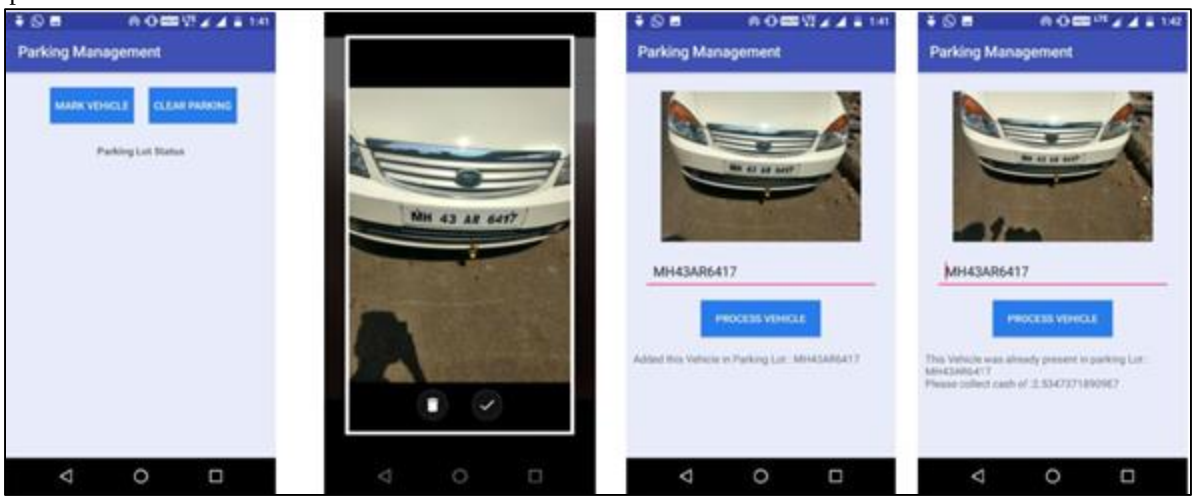

Fig. 5:

VI. CONCLUSION

In this project we proposed a new system for cheap and effective parking management. In real time scenarios our parking system has low false positives and a confidence percentage of up to 93%. This is better than a lot of home built algorithms. Parking management system does not bog down the database with excessive and unwanted data. The future work is to increase the confidence percentage up to 98 and make the plate detection faster. This can be achieved by training the OpenAlpr library with more dataset. Poor lighting conditions may affect the system and indirectly degrade the performance of the system as a whole. Our system provides an optimum solution but not a perfect solution.

ACKNOWLEDGMENT

We would like to thank the people who have given us continuous support and encouragement.
1) Prof. Sheetal Thakare, Project Guide
2) Prof. Rahul Patil, Project Coordinator
3) Dr.D.R.Ingle, Head of Department
4) Dr.M.Z.Shaikh, Principal

REFERENCES

[1] Automatic License Plate Recognition Shyang-Lih Chang, Li-Shien Chen, Yun-Chung Chung, and Sei-Wan Chen, Senior Member, IEEE.
[2] Automatic License Plate Recognition (ALPR): A State-of-the-Art Review Shan Du, Member, IEEE, Mahmoud Ibrahim, Mohamed Shehata, Senior Member, IEEE, and Wael Badawy, Senior Member, IEEE.
[3] Automatic OCR based control and analysis of vehicle license plates Ashwini Patil1 and Sachin Ruikar1 1 Department of Electronics Engineering Walchand College of Engineering, Sangli, Maharashtra, India.
[4] D. S. Kim and S. I. Chien, "Automatic vehicle license plate extraction using modified generalized symmetry transform and image warping," in Proc. IEEE Int. Symp. Ind. Electron. June. 2001, vol. 3,pp. 2022–2027.
[5] https://github.com/openalpr/openalpr/wiki/OpenALPR-Design