LICENSE PLATE RECOGNITION USING OPENCV IN PYTHON

A Project report submitted for the award of the

Degree of Bachelor of Technology

In

Computer Science & Engineering (CSE)

By

NISHRITH SAINI (15011M2204)

DHARMA TEJA B (15011M2205)

NYNESH REDDY G (15011M2208)

Under the esteemed Guidance of

Dr. V KAMAKSHI PRASAD

Professor & Director of Evaluation JNTUH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KUKATPALLY, HYDERABAD – 500 085.

DECEMBER 2018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KUKATPALLY, HYDERABAD – 500 085.

DECLARATION BY THE CANDIDATES

We, **NISHRITH SAINI (15011M2204), DHARMA TEJA B (15011M2205), and NYNESH REDDY G (15011M2208),** hereby declare that the project report entitled "**LICENSE PLATE RECOGNITION USING OPENCV IN PYTHON**", carried out by us under the guidance of **Dr. V Kamakshi Prasad, Professor and Director of Evaluation, JNTUH**, is submitted for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*. This is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced /copied from any source.

NISHRITH SAINI (15011M2204)

DHARMA TEJA B (15011M2205)

NYNESH REDDY G (15011M2208)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KUKATPALLY, HYDERABAD – 500 085.

CERTIFICATE BY THE SUPERVISOR

This is to certify that the project report entitled "**LICENSE PLATE RECOGNITION USING OPENCV IN PYTHON**", being submitted by, **Nishrith Saini (15011M2204), Dharma Teja B (15011M2205), and Nynesh Reddy G (15011M2208)** for the award of the degree of **Bachelor of Technology in Computer Science and Engineering from JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, COLLEGE OF ENGINEERING HYDERABAD (Autonomous)** is a record of bonafide work carried out by them. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Dr. V Kamakshi Prasad,

Professor and Director of Evaluation, JNTUH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KUKATPALLY, HYDERABAD – 500 085.

CERTIFICATE BY THE HEAD OF THE DEPARTMENT

This is to certify that the project report entitled "**LICENSE PLATE RECOGNITION USING OPENCV IN PYTHON**", being submitted by, **Nishrith Saini (15011M2204), Dharma Teja B (15011M2205), and Nynesh Reddy G (15011M2208),** for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by them.

Dr. R. Sridevi,

Professor & Head of the CSE Department

JNTUH College of Engineering

ACKNOWLEDGEMENT

We would like to express our profound gratitude to our esteemed guide and mentor, **Dr. V Kamakshi Prasad**, Professor and Director of Evaluation, JNTUH, for his timely advice and guidance as well as his encouragement and motivation throughout the completion of this project. His technical knowledge has been valuable asset, but more importantly sir has helped to keep us focused and pointed in the right direction throughout the Bachelor of Technology in Computer Science and Engineering course.

We are also thankful to **Dr. R Sridevi**, Professor and Head of department for the motivation and inspiration that triggered us for the project work.

We also thank our friends and family members who have supported us all the way during the project and parents for their blessings which made us stand despite of many obstacles and successful complete the project.

Regards,

NISHRITH SAINI (15011M2204)

DHARMA TEJA B (15011M2205)

NYNESH REDDY G (15011M2208)

TABLE OF CONTENTS

1. ABSTRACT **1**

2. INTRODUCTION **2**
- **2.1** Introduction 2
- **2.2** Aim 2
- **2.3** Objective 2
- **2.4** Approach 3

3. TOOLS AND METHODOLOGIES USED **4**
- **3.1** Python 4
- **3.2** Open CV 4
- **3.3** *k*-nearest neighbors algorithm(*k*-NN) 5

4. PROCESSING STEPS FOR LICENSE PLATE RECOGNISTION **7**
- **4.1** Image Acquisition 7
- **4.2** Pre-Processing 8
- **4.3** Feature Extraction 10
- **4.4** Detection/Segmentation 11
- **4.5** High Level Processing 12
- **4.6** Decision Making 13

5. PERFORMANCE EVALUATION **14**
- **5.1** Test Case 14
- **5.2** Consolidated results Table 16
- **5.3** Description 17

6. CONCLUSION & FUTURE SCOPE **19**

7.REFERENCES **20**

8.APPENDIX-A SOURCE CODE **21**
- **8.1** Main.py 21
- **8.2** Preprocess.py 25
- **8.3** PossiblePlate.py 27
- **8.4** PossibleChar.py 28
- **8.5** DetectPlates.py 29
- **8.6** DetectChars.py 36

9.APPENDIX-B **51**

1. ABSTRACT

Along with the progress of the society, the rhythm of people's life becomes quick, which makes the need of automobile inevitable. Also, the fast development of computers, communication and computer network techniques, makes the transportation management automation an inevitable goal. License Plate Recognition is an essential stage in intelligent traffic systems. The use of vehicles has been increasing because of population growth and human needs in recent years. Therefore, control of vehicles is becoming a big problem and much more difficult to solve. The License Plate Recognition (LPR) technology applies image processing and character recognition techniques to identify vehicles by automatically reading their license plates. The automatic license plate recognition, which is an important technique to obtain traffic information, mixes computer vision, image processing techniques and pattern recognition techniques. In a vehicle license plate recognition system, plate region detection is the key step before the final recognition.

In addition to this additional care taken up in this paper is to extract license plate of motorcycle (size of plate is small and double row plate), car (single as well as double row type), transport system such as bus, truck, (dirty plates) as well as multiple license plates present in an image frame under consideration. Disparity of aspect ratios is a typical feature of Indian traffic. Proposed method aims at identifying region of interest by performing a sequence of directional segmentation and morphological processing.

There are many applications ranging from complex security systems to common areas and from parking admission to urban traffic control. License plate recognition (LPR) has complex characteristics due to diverse effects such as of light and speed. This paper presents a method of implementing LPR systems using Software including Python and the Open Computer Vision Library.

2. INTRODUCTION

2.1 Introduction

The scientific world is deploying research in intelligent transportation systems which have a significant impact on peoples´ lives. License Plate Recognition (LPR) is a computer vision technology to extract the license number of vehicles from images. It is an embedded system which has numerous applications and challenges.

Typical LPR systems are implemented using proprietary technologies and hence are costly. This closed approach also prevents further research and development of the system. With the rise of free and open source technologies the computing world is lifted to new heights. People from different communities interact in a multi-cultural environment to develop solutions for mans never ending problems. One of the notable contributions of the open source community to the scientific world is Python. Intel's researches in Computer Vision bore the fruit called Open Computer Vision (OpenCV) library, which can support computer vision development.

2.2 Aim

Main aim of the project is to develop a License Plate Recognition Program which recognizes different types of characters in an image and gives specific extracted output in text format.

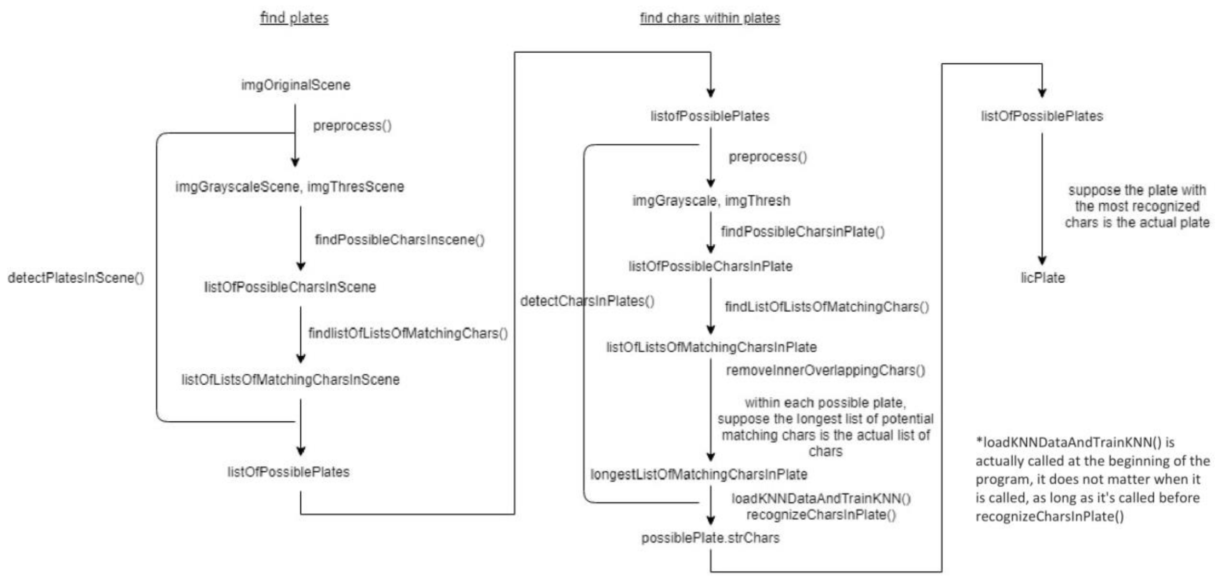
2.3 Objectives

The objective of this project is to extract characters from a License Plate with the use of Computer Vision libraries and algorithms. A suitable algorithm needs to be constructed and trained for different sets of images. The program code must be written in Python with the inclusion of suitable OpenCV libraries. The program should be able to extract the characters and print the output in text format for training purpose.

2.4 Approach

To solve the defined character recognition, process the python coded algorithm with OpenCV libraries is used. The processing is divided into the next categories.

- Image Acquisition
- Pre-Processing
- Feature Extraction
- Detection/Segmentation
- High-Level Processing
- Decision Making


find plates
find chars within plates
imgOriginalScene
preprocess()
imgGrayscaleScene, imgThresScene
findPossibleCharsInscene()
listOfPossibleCharsInScene
findlistOfListsOfMatchingChars()
listOfListsOfMatchingCharsInScene
listOfPossiblePlates
detectPlatesInScene()
listofPossiblePlates
preprocess()
imgGrayscale, imgThresh
findPossibleCharsinPlate()
listOfPossibleCharsInPlate
findListOfListsOfMatchingChars()
listOfListsOfMatchingCharsInPlate
removeInnerOverlappingChars()
within each possible plate, suppose the longest list of potential matching chars is the actual list of chars
longestListOfMatchingCharsInPlate
loadKNNDataAndTrainKNN()
recognizeCharsInPlate()
possiblePlate.strChars
detectCharsInPlates()
listOfPossiblePlates
suppose the plate with the most recognized chars is the actual plate
licPlate
*loadKNNDataAndTrainKNN() is actually called at the beginning of the program, it does not matter when it is called, as long as it's called before recognizeCharsInPlate()

3. TOOLS AND METHODOLOGIES USED

3.1 Python

Python is a remarkably powerful dynamic, object-oriented programming language that is used in a wide variety of application domains. It offers strong support for integration with other languages and tools and comes with extensive standard libraries. To be precise, the following are some distinguishing features of Python:

- Very clear, readable syntax.
- Strong introspection capabilities
- Full modularity.
- Exception-based error handling.
- High level dynamic data types
- Supports object oriented, imperative and functional programming styles.
- Embeddable.
- Scalable
- Mature

With so much of freedom, Python helps the user to think problem centric rather than language centric as in other cases. These features make Python a best option for scientific computing.

3.2 OpenCV

OpenCV is a library of programming functions for real time computer vision originally developed by Intel and now supported by Will garage. It is free for use under the open source BSD license. The library has more than five hundred optimized algorithms. It is used around the world, with forty thousand people in the user group. Uses range from interactive art, to mine inspection, and advanced robotics. The library is mainly written in C, which makes it portable to some specific platforms such as Digital Signal Processor. Wrappers for languages such as C, Python, Ruby and Java (using JavaCV) have been developed to encourage adoption by a wider audience. The recent releases have interfaces for C++. It focuses mainly on real-time image processing. OpenCV is a cross-platform library, which can run on Linux, Mac OS and Windows. To date, OpenCV is the best open source computer vision library that developers and researchers can think of.

3.3 *k*-Nearest Neighbors Algorithm(*k*-NN)

Character recognition is the final stage in vehicle license plate detection and recognition, where it reads an individual characters and numbers. And this step is considered as a significant step, for instance: in the entrance of car-park or for searching the stolen cars, to help police. Single elements on license plate should be classified and examined. This examination is termed as Optical Character Recognition (OCR) using KNN.

The k-nearest neighbor algorithm (k-NN) is a non-parametric method in the pattern recognition, which is utilized for classification and regression. In both these scenarios, the input has the k closest training samples in the feature space. The output works on whether k-NN is utilized for classification or regression:

- In k-NN classification, the output assumed as a class membership. An object is segregated by a majority vote of its neighbors, with the object is being allotted to the class most common between its its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply allotted to the lass of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

Need to allot the weight to the contributions of the neighbors, both for classification and regression process, so that the nearer neighbors contribute more to the average than the more distant ones. For instance: a common weighting scheme where every neighbor has a weight of 1/d, where d is the distance to the neighbor.

The neighbors were considered from a group of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This brings a training set for the algorithm, even though we do not require an explicit training step.

3.3.1 Concept of the Algorithm

The training examples are vectors in a multidimensional feature space, each with a class label. The training Phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidian distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming Distance). In the context of gene expression microarray data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman. Often, the classification accuracy of *k*-NN can be improved significantly if the distance metric is learned with specified algorithms such as Large Margin Nearest Neighbor or Neighborhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class trend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbors due to their large number. One way to overcome this problem is to weight the classification, considering the distance from the test point to each of its *k* nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. *K*-NN can then be applied to the SOM.

4. PROCESSING STEPS FOR LICENSE PLATE RECOGNISION

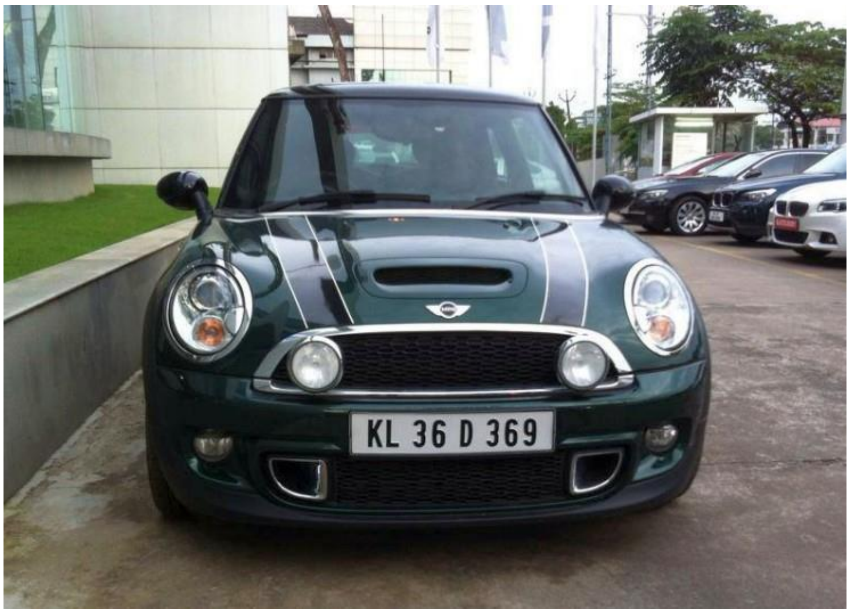
The Organization of a computer vision system is highly application dependent. Some systems are standalone applications which solve specific measurement or detection problem. The specific implementation of a computer vision system also depends on if its functionality is pre-specified or if some part or it can be learned or modified during operation. Many functions are unique to the application. There are, however, typical functions which are found in many computer vision systems.

The following are the steps involved in extraction of characters from license plate.

4.1 Image Acquisition

A digital image is produced by one or several image sensor i.e. light sensitive cameras. The pixel values typically correspond to light intensity in one or several spectral bands (grey images or colour images), but can also be related to various physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves.

Original Image (Fig 4.1)



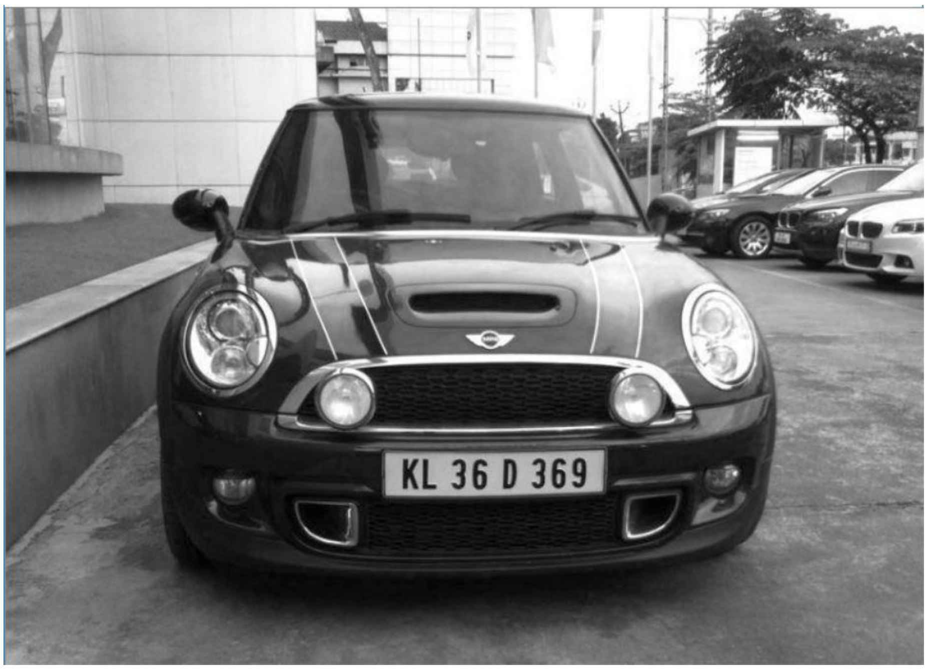
4.2 Pre-Processing

Before a computer vision method can be applied to image data in order to extract some specific of information, its is usually necessary to process the data in order to assure that it satisfies certain assumptions implied by the following methods:

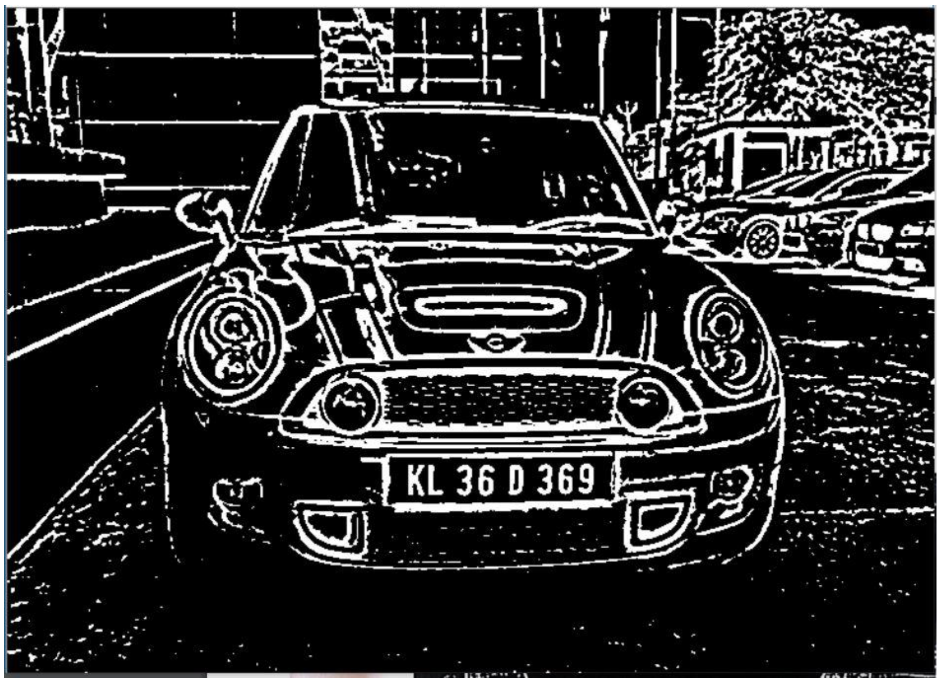
- Noise reduction in order to assure that sensor noise does not introduce false information.
- Contrast enhancement to assure that relevant information can be detected.
- Re-sampling in order to assure that the image coordinate system is correct.
- Scale space representation to enhance image structures at locally appropriate scales.

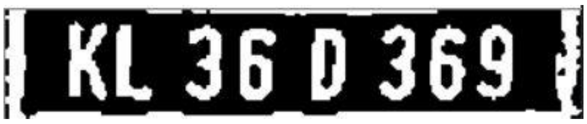
From the previous image, the image is converted to Greyscale and further to Threshold to reduce noise.

GreyScale Image (Fig 4.2)

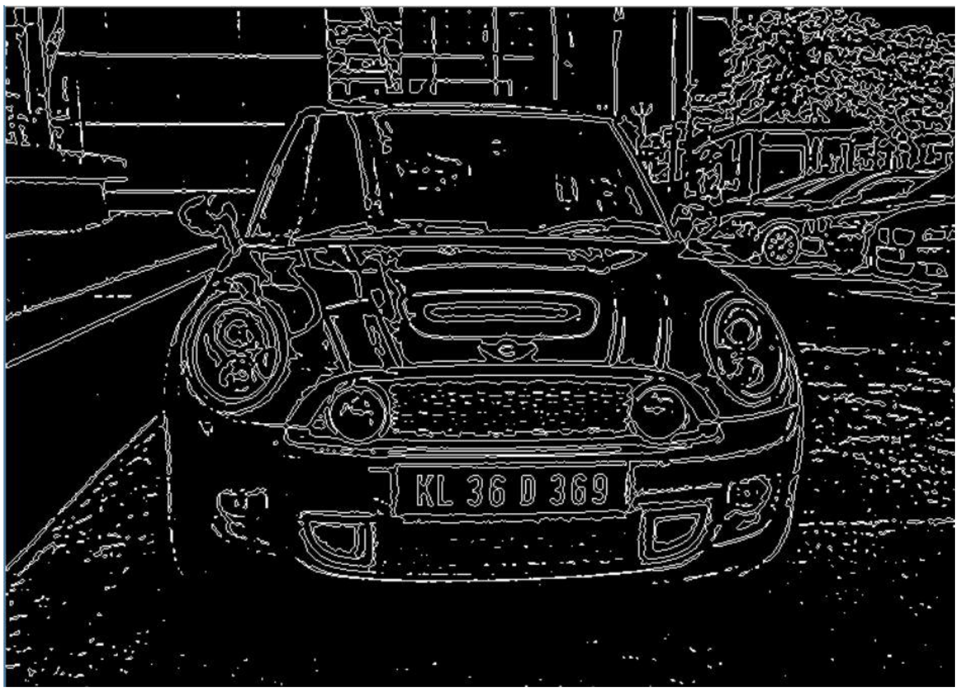


Threshold Image 1 (Fig 4.3)





Threshold Image 2 (Fig 4.4)



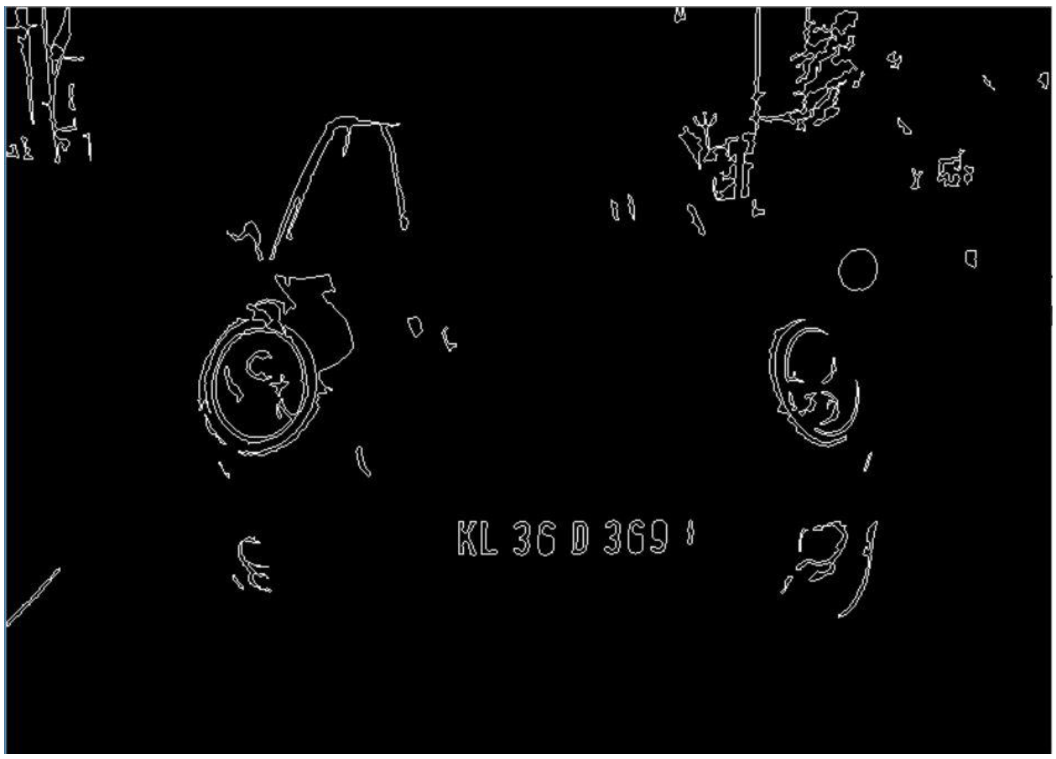
4.3 Feature Extraction

Image features at various levels of complexity are extracted from the image data. Typical examples of such features are:

- Lines, edges and ridges.
- Localized interest points such as corners, points, etc.
- More complex features may be related to texture, shape or motion.

From Threshold image, features like lines, edges, ridges etc. are separated to identify possible characters.

Possible Characters (Fig 4.5)

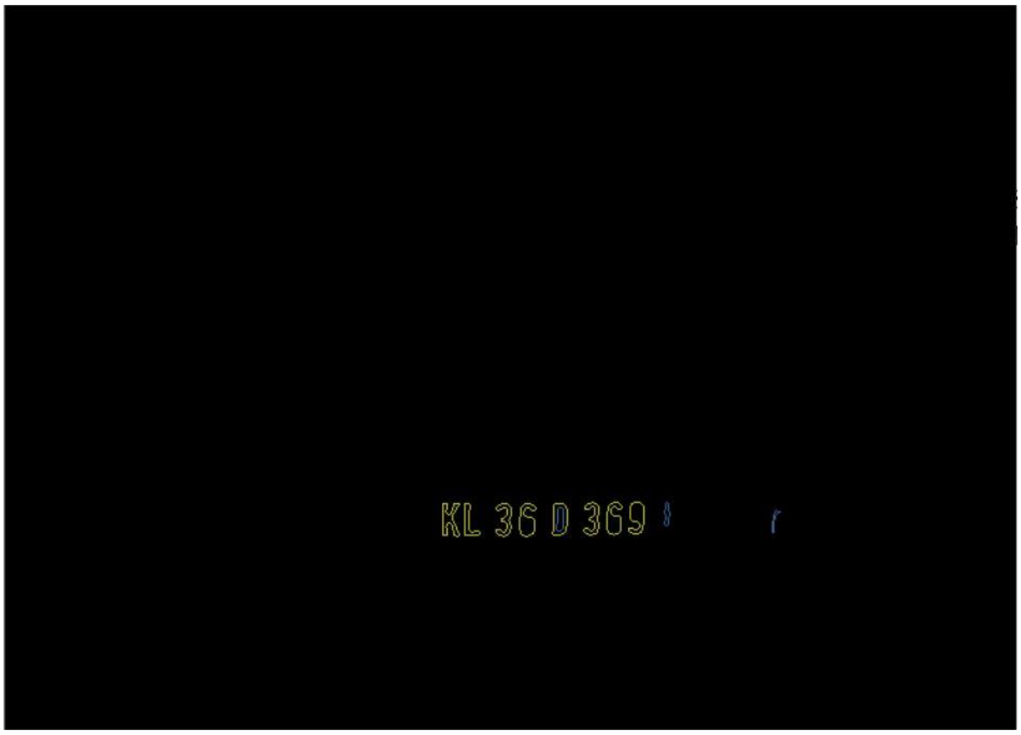


4.4 Detection/Segmentation

At some point the processing a decision is made about which image points or regions of the image are relevant for further processing.

- Selection of a specific set of interest points
- Segmentation of one or multiple image regions which contain a specific object of interest.
- Possible characters are matched with trained characters.

Matching Characters in the Scene (Fig 4.6)



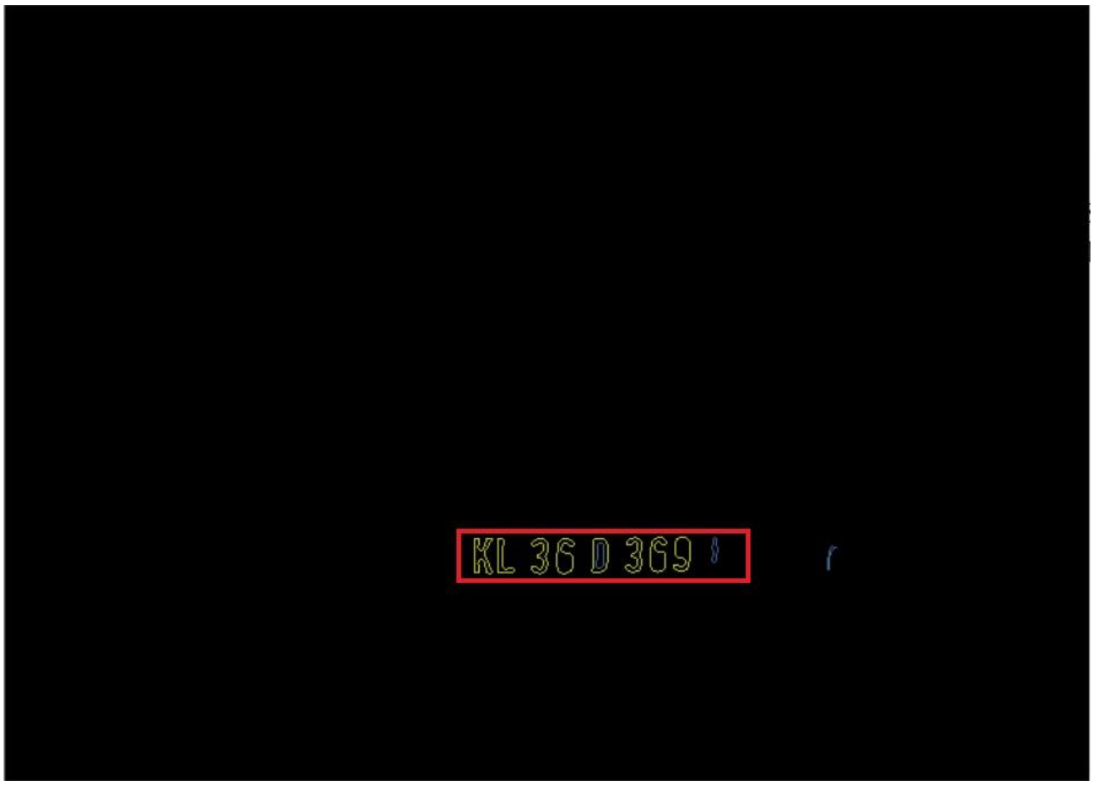
4.5 High-Level Processing

At this step the input is typically a small set of data, a set of points or an image region which is assumed to contain a specific object. The remaining processing deals with

- Verification of data that satisfy model based and application specific assumptions.
- Estimation of application specific parameters, such as object pose or object size.
- Image recognition-classifying a detected object into different categories.
- Image registration-comparing and combining two different views of the same object.

In all the possible plates are identified and treated as rectangles, the rectangles which does not contain accurate characters are discarded.

Possible Plates in the Scene (Fig 4.7)



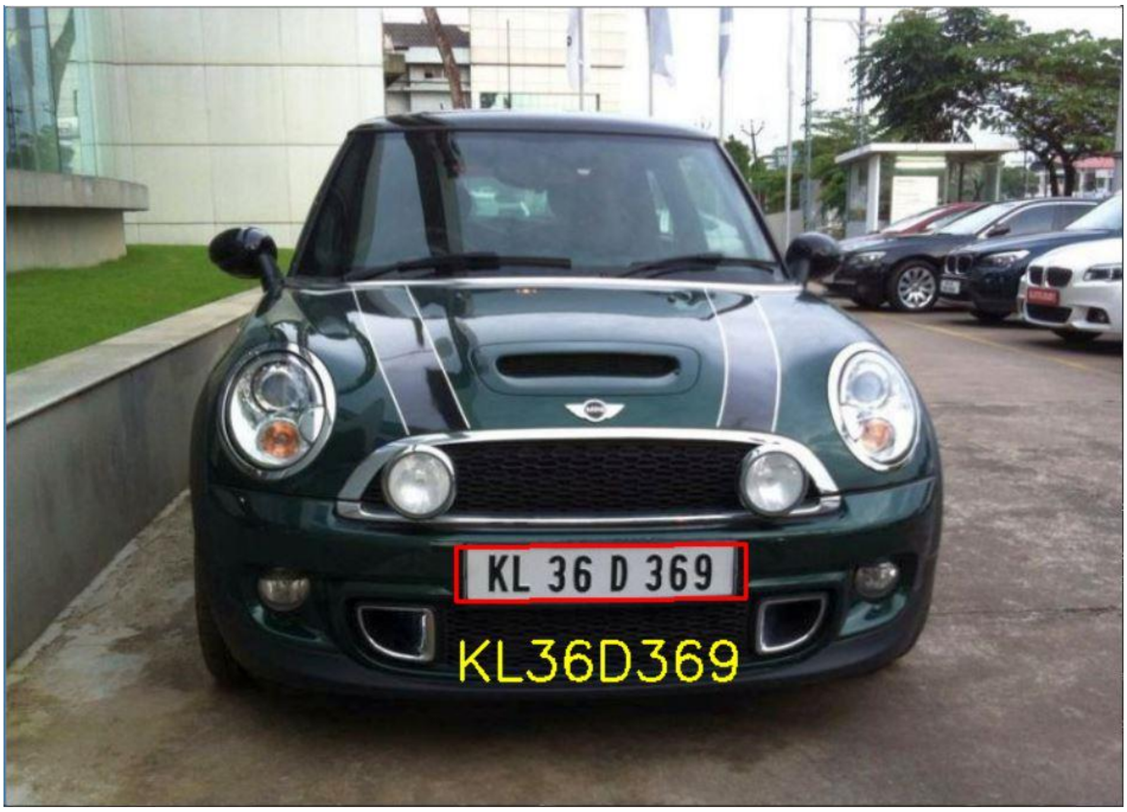
4.6 Decision Making

Making the final decision required for the following applications:

- Pass/Fail on automatic inspection
- Match/no-match in recognition

The final rectangle with appropriate characters from previous stage is matched with trained characters and printed on the original image.

Recognized characters are printed on the original image (Fig 4.8)

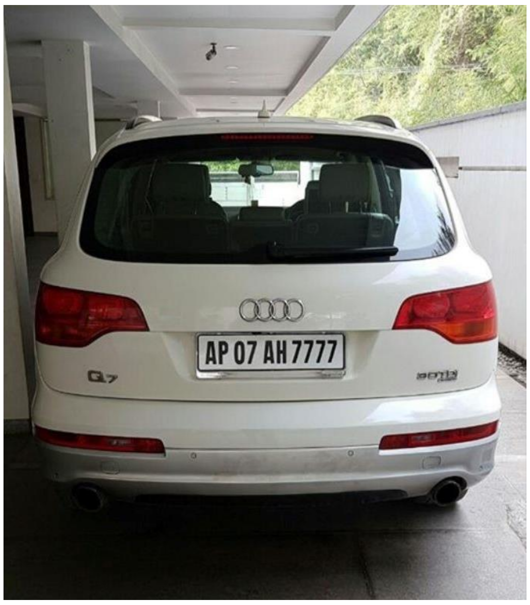


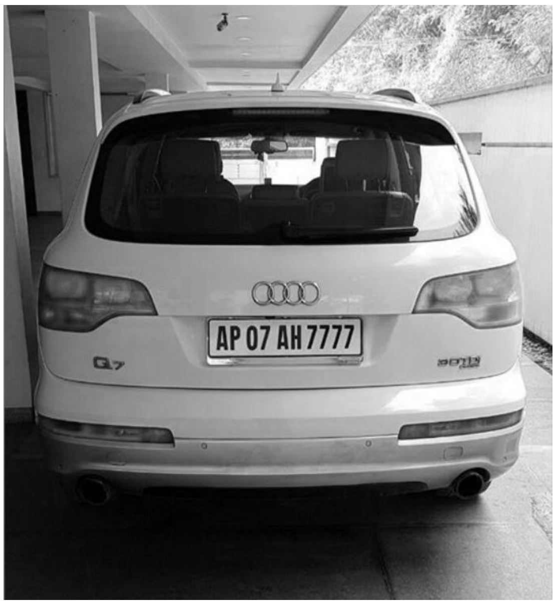
5. PERFORMANCE EVALUATION

5.1 Test Case

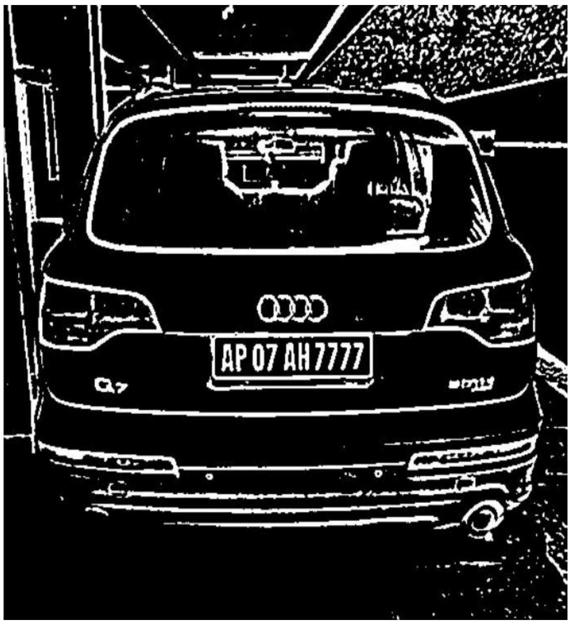
Some Random tests are performed on different images containing License Plate, results are shown below.

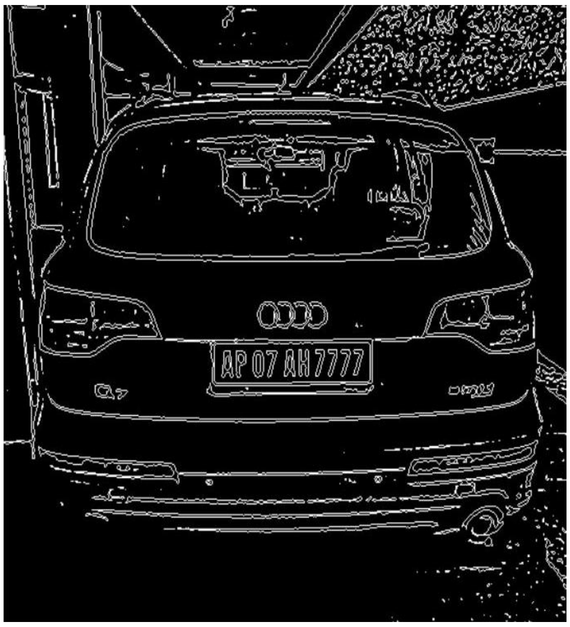
Original image converted into Greyscale image (Fig 5.1)



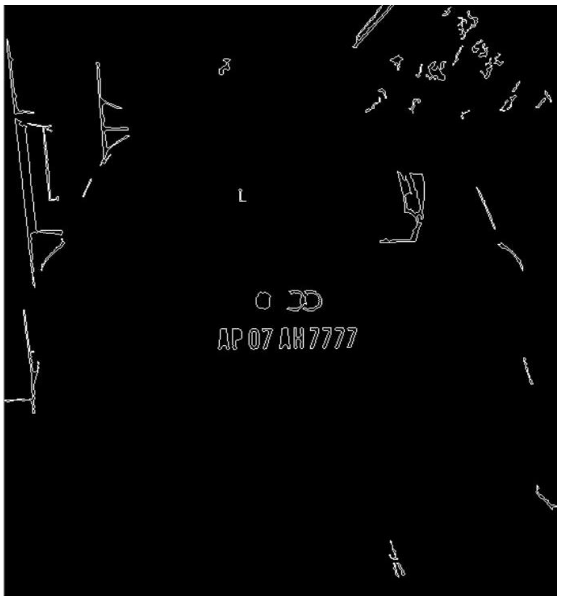


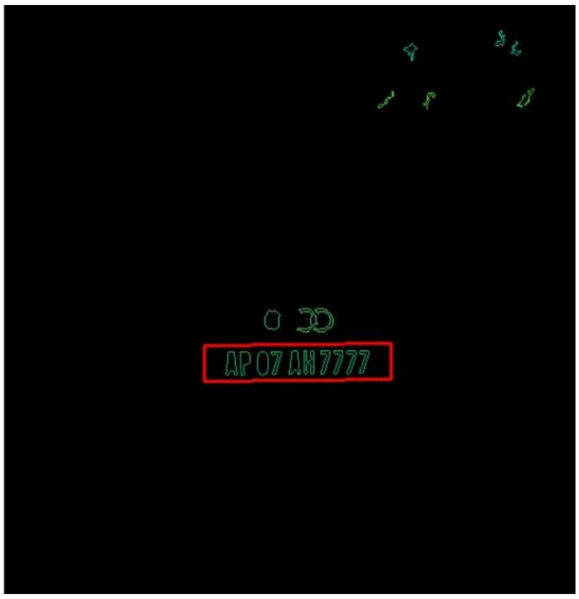
Greyscale image converted into Threshold images (Fig 5.2)



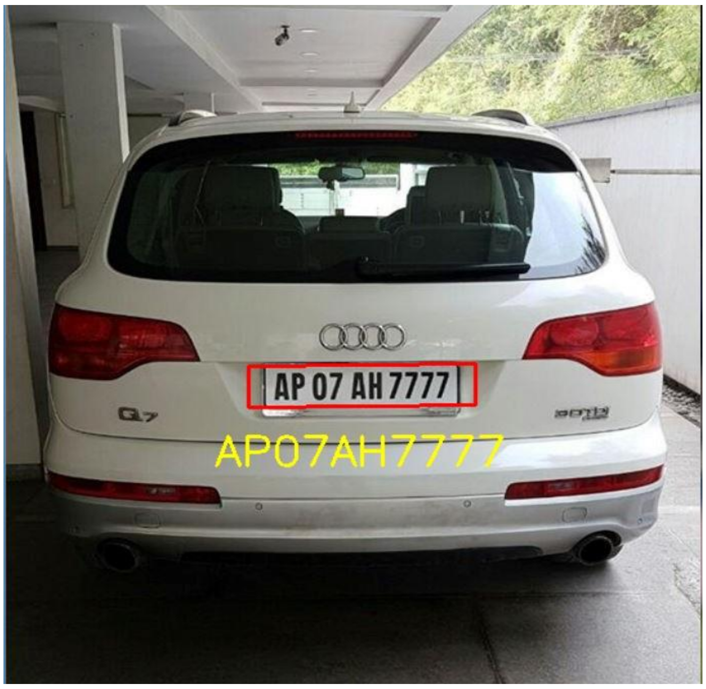


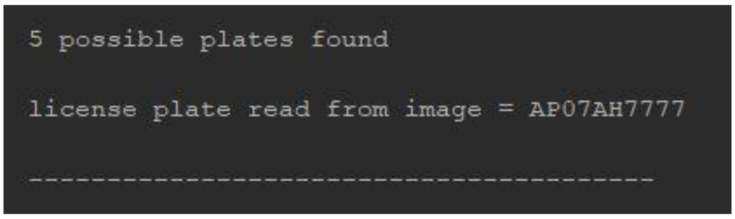
From the Threshold images, features like lines, edges, ridges etc. are separated and possible characters are found (Fig 5.3)





Extracted text is printed on the original Image (Fig 5.4)





```
5 possible plates found

license plate read from image = AP07AH7777

-----------------------------------------
```

5.2 Consolidate Results Table

Table 1: Results of the test

Units of License Plate Recognition System	**Number of Accurate Results**	**Percentage of Accuracy**
Extraction of Plate Region	322/340	94.71%
Segmentation	327/340	96.17%
Recognition of Characters	261/340	76.76%

Table 2: Capture and Read Rates for pictures of Vehicles

Type of Picture	**Capture Rate**	**Correct Read Rate**	**Overall Capture and Correct Read Rates**
Picture of a vehicle when it is stationary	85%	80%	80%
Picture of a vehicle when it is moving.	75%	80%	70%

5.3 Description

The results obtained show that the ALPR system manages to detect and recognize characters from the license plates well (up to 75 % accurately) while using the *k*-NN algorithm.

The accuracy of results at each phase are also determined quite well because of the usage of a dataset of images consisting of about 350 different pictures of license plates of vehicles from across the globe.

This recognition system is also capable of identifying and recognizing characters from a picture of a moving vehicle for up to 70% and can even be improved to handle such images. The recognition of characters from motion-based pictures is difficult because of blurriness and many other factors which are induced into the picture due to the motion of vehicle (subject) in the picture.

On the other hand, for the pictures of the vehicles which are stationary the license plate detection works extraordinarily well and character recognition has an accuracy of about 80%. This is because the preprocessing and higher-level processing (along with other intermediate steps in between) work well because of the clarity, clearness and fairly decent image of a stationary vehicle as compared to a picture of a moving vehicle.

Some of the problems we faced during the development of this project were:

- What programming platform to choose: We wanted to get the maximum capability out of our systems at the same time it should be platform independent and shouldn't deteriorate end user's system's performance. The solution was "Python" simply because it was a scripting language which was easy to work around with, platform independent, and provided with many useful image processing modules which have been very useful in aiding the system for the detection of characters from the images.

- The Source Code: We had to figure out each and every step involved in detection of characters in a license plate, progressively from start to finish and assign each one of them to a function which serves as an input to next function. The main goal of this type of function-based coding is easier and better management of code as well as have a clear idea about what function does what at any point in the code and if any discrepancy was found we could just change the function to produce the correct result rather than to meddle with the main function of the code.

6. CONCLUSION AND FUTURE SCOPE

The project provides an implementation of License Plate Recognition using Optical Character Recognition (OCR) which is an electronic conversion of images of typed, handwritten or printed text to machine-encoded text. The project was able to study and resolve algorithm and mathematical aspects of the License Plate Recognition systems, such as computer vision, pattern recognition, OCR & neural networks.

License Plate Recognition solution has been tested on static snapshots of vehicles, which has been divided into several sets according to difficulty. Sets of blurry and skewed snapshots give worse recognition rates than a set of snapshots which has been captured clearly. The main objective of this project was not to find a one hundred percent recognizable set of snapshots, but to test the invariance of the algorithm on random snapshots systematically classified to the sets according to their properties.

Using new techniques and packages the LPR can be implemented in the real time cameras and devices.

7. REFERENCES

- Prathamesh Kulkarni, Ashish Khatri, Prateek Banga, Kushal Shah, Automatic Number Plate Recognition (ANPR) System for Indian conditions.
- S.V. Rice, F.R. Jenkins, T.A. Nartker, The Fourth Annual Test of OCR Accuracy, Technical Report 95-03. Information Science Research Institute, University of Nevada, Las Vegas, (1995).
- Nobuyuki Otsu (1979). A threshold selection method from gray-level histograms. IEEE Trans. Sys., Man., Cyber. 9: 62-66.
- Chih-Hai Fana, Yu-Hang Peng, Vehicle License Plate Recognition System Design, Chung Hua Journal of Science and Engineering, Vol. 7, No. 2, pp. 47-52 (2009)
- K.M Sajjad, "ALPR Using Python and Open CV" Dept of CSE, M.E.S College of Engineering Kuttipuram, Kerala.
- Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, volume 1, pages 886–893. IEEE, 2005.
- Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. International Journal of Computer Vision, 59(2):167–181, 2004.
- OpenCV: http://docs.opencv.org/3.1.0/df/d9d/tutorialpycolorspaces.html
- *K*-NN algorithm: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- ANPR: https://en.wikipedia.org/wiki/Automatic_number-plate_recognition
- Pattern Recognition: https://en.wikipedia.org/wiki/Pattern_recognition

8. APPENDIX-A

SOURCE CODE

8.1 Main.py

```
# Main.py

import cv2
import numpy as np
import os

import DetectChars
import DetectPlates
import PossiblePlate

# module level variables
SCALAR_BLACK = (0.0, 0.0, 0.0)
SCALAR_WHITE = (255.0, 255.0, 255.0)
SCALAR_YELLOW = (0.0, 255.0, 255.0)
SCALAR_GREEN = (0.0, 255.0, 0.0)
SCALAR_RED = (0.0, 0.0, 255.0)

showSteps = False


def main():

    blnKNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN()
# attempt KNN training

    if blnKNNTrainingSuccessful == False:  # if KNN training was not successful
        print("\nerror: KNN traning was not successful\n")  # show error message
        return  # and exit program
    # end if

    imgOriginalScene = cv2.imread("LicPlateImages/1.jpg")  # open image

    if imgOriginalScene is None:  # if image was not read successfully
        print("\nerror: image not read from file \n\n")  # print error message to std out
        os.system("pause")  # pause so user can see error message
        return  # and exit program
    # end if

    listOfPossiblePlates = DetectPlates.detectPlatesInScene(imgOriginalScene)  # detect
plates
```

```
    listOfPossiblePlates = DetectChars.detectCharsInPlates(listOfPossiblePlates)  #
detect chars in plates


    cv2.imshow("imgOriginalScene",imgOriginalScene)  # show scene image

    if len(listOfPossiblePlates) == 0:  # if no plates were found
        print("\nno license plates were detected\n")  # inform user no plates were found
    else:  # else
        # if we get in here list of possible plates has at leat one plate

        # sort the list of possible plates in DESCENDING order (most number of chars to
least number of chars)
        listOfPossiblePlates.sort(key=lambda possiblePlate: len(possiblePlate.strChars),
reverse=True)

        # suppose the plate with the most recognized chars (the first plate in sorted by
string length descending order) is the actual plate
        licPlate = listOfPossiblePlates[0]

        cv2.imshow("imgPlate", licPlate.imgPlate)
# show crop of plate and threshold of plate
        cv2.imshow("imgThresh", licPlate.imgThresh)

        if len(licPlate.strChars) == 0:  # if no chars were found in the plate
            print("\nno characters were detected\n\n")  # show message
            return  # and exit program
        # end if

        drawRedRectangleAroundPlate(imgOriginalScene, licPlate)
 # draw red rectangle around plate

        print("\nlicense plate read from image = " + licPlate.strChars + "\n")
# write license plate text to std out
        print("----------------------------------------")

        writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)  # write license plate
text on the image

        cv2.imshow("imgOriginalScene", imgOriginalScene)  # re-show scene image

        cv2.imwrite("imgOriginalScene.png", imgOriginalScene)  # write image out to file

    # end if else

    cv2.waitKey(0)  # hold windows open until user presses a key

    return
```

```
# end main

def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):
   p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)
# get 4 vertices of rotated rect

   cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]),
SCALAR_RED, 2)  # draw 4 red lines
   cv2.line(imgOriginalScene, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]),
SCALAR_RED, 2)
   cv2.line(imgOriginalScene, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]),
SCALAR_RED, 2)
   cv2.line(imgOriginalScene, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]),
SCALAR_RED, 2)


# end function

def writeLicensePlateCharsOnImage(imgOriginalScene, licPlate):
   ptCenterOfTextAreaX = 0  # this will be the center of the area the text will be written
to
   ptCenterOfTextAreaY = 0

   ptLowerLeftTextOriginX = 0  # this will be the bottom left of the area that the text
will be written to
   ptLowerLeftTextOriginY = 0

   sceneHeight, sceneWidth, sceneNumChannels = imgOriginalScene.shape
   plateHeight, plateWidth, plateNumChannels = licPlate.imgPlate.shape

   intFontFace = cv2.FONT_HERSHEY_SIMPLEX  # choose a plain jane font
   fltFontScale = float(plateHeight) / 30.0  # base font scale on height of plate area
   intFontThickness = int(round(fltFontScale * 1.5))  # base font thickness on font scale

   textSize, baseline = cv2.getTextSize(licPlate.strChars, intFontFace, fltFontScale,
                                        intFontThickness)  # call getTextSize

   # unpack roatated rect into center point, width and height, and angle
   ((intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight),
    fltCorrectionAngleInDeg) = licPlate.rrLocationOfPlateInScene

   intPlateCenterX = int(intPlateCenterX)  # make sure center is an integer
   intPlateCenterY = int(intPlateCenterY)

   ptCenterOfTextAreaX = int(intPlateCenterX)
# the horizontal location of the text area is the same as the plate
```

```
    if intPlateCenterY < (sceneHeight * 0.75):
# if the license plate is in the upper 3/4 of the image
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(
            round(plateHeight * 1.6))  # write the chars in below the plate
    else:  # else if the license plate is in the lower 1/4 of the image
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(
            round(plateHeight * 1.6))  # write the chars in above the plate
    # end if

    textSizeWidth, textSizeHeight = textSize  # unpack text size width and height

    ptLowerLeftTextOriginX = int(
        ptCenterOfTextAreaX - (textSizeWidth / 2))
# calculate the lower left origin of the text area
    ptLowerLeftTextOriginY = int(
        ptCenterOfTextAreaY + (textSizeHeight / 2))
# based on the text area center, width, and height

    # write the text on the image
    cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX,
ptLowerLeftTextOriginY), intFontFace,
                fltFontScale, SCALAR_YELLOW, intFontThickness)


# end function

if __name__ == "__main__":
    main()
```

8.2 Preprocess.py

```
# Preprocess.py

import cv2

import numpy as np

import math

# module level variables

GAUSSIAN_SMOOTH_FILTER_SIZE = (5, 5)

ADAPTIVE_THRESH_BLOCK_SIZE = 19

ADAPTIVE_THRESH_WEIGHT = 9

def preprocess(imgOriginal):

    imgGrayscale = extractValue(imgOriginal)

    imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)0

    height, width = imgGrayscale.shape

    imgBlurred = np.zeros((height, width, 1), np.uint8)

    imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale,
GAUSSIAN_SMOOTH_FILTER_SIZE, 0)

   imgThresh = cv2.adaptiveThreshold(imgBlurred, 255.0,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
ADAPTIVE_THRESH_BLOCK_SIZE, ADAPTIVE_THRESH_WEIGHT)

    return imgGrayscale, imgThresh

# end function

def extractValue(imgOriginal):

    height, width, numChannels = imgOriginal.shape

    imgHSV = np.zeros((height, width, 3), np.uint8)

    imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2HSV)

    imgHue, imgSaturation, imgValue = cv2.split(imgHSV)

   return imgValue

# end function

def maximizeContrast(imgGrayscale):
    height, width = imgGrayscale.shape

    imgTopHat = np.zeros((height, width, 1), np.uint8)

    imgBlackHat = np.zeros((height, width, 1), np.uint8)

    structuringElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

    imgTopHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH_TOPHAT,
structuringElement)

    imgBlackHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH_BLACKHAT,
structuringElement)

    imgGrayscalePlusTopHat = cv2.add (imgGrayscale, imgTopHat)

    imgGrayscalePlusTopHatMinusBlackHat = cv2.subtract(imgGrayscalePlusTopHat,
imgBlackHat)

    return imgGrayscalePlusTopHatMinusBlackHat

# end function
```

8.3 PossiblePlate.py

```
# PossibleChar.py

import cv2

import numpy as np

import math

class PossibleChar:

    # constructor

    def __init__(self, _contour):

        self.contour = _contour

        self.boundingRect = cv2.boundingRect(self.contour)

        [intX, intY, intWidth, intHeight] = self.boundingRect

        self.intBoundingRectX = intX

        self.intBoundingRectY = intY

        self.intBoundingRectWidth = intWidth

        self.intBoundingRectHeight = intHeight

        self.intBoundingRectArea = self.intBoundingRectWidth *
self.intBoundingRectHeight

        self.intCenterX = (self.intBoundingRectX + self.intBoundingRectX +
self.intBoundingRectWidth) / 2

        self.intCenterY = (self.intBoundingRectY + self.intBoundingRectY +
self.intBoundingRectHeight) / 2

        self.fltDiagonalSize = math.sqrt((self.intBoundingRectWidth ** 2) +
(self.intBoundingRectHeight ** 2))

        self.fltAspectRatio = float(self.intBoundingRectWidth) /
float(self.intBoundingRectHeight)

    # end constructor

# end class
```

8.4 PossibleChar.py

```
# PossibleChar.py

import cv2

import numpy as np

import math

class PossibleChar:

    # constructor

    def __init__(self, _contour):

        self.contour = _contour

        self.boundingRect = cv2.boundingRect(self.contour)

        [intX, intY, intWidth, intHeight] = self.boundingRect

        self.intBoundingRectX = intX

        self.intBoundingRectY = intY

        self.intBoundingRectWidth = intWidth

        self.intBoundingRectHeight = intHeight

        self.intBoundingRectArea = self.intBoundingRectWidth *
self.intBoundingRectHeight

        self.intCenterX = (self.intBoundingRectX + self.intBoundingRectX +
self.intBoundingRectWidth) / 2

        self.intCenterY = (self.intBoundingRectY + self.intBoundingRectY +
self.intBoundingRectHeight) / 2

        self.fltDiagonalSize = math.sqrt((self.intBoundingRectWidth ** 2) +
(self.intBoundingRectHeight ** 2))

        self.fltAspectRatio = float(self.intBoundingRectWidth) /
float(self.intBoundingRectHeight)

    # end constructor

# end class
```

8.5 DetectPlates.py

```
# DetectPlates.py
import cv2
import numpy as np
import math
import Main
import random
import Preprocess
import DetectChars
import PossiblePlate
import PossibleChar
# module level variables
PLATE_WIDTH_PADDING_FACTOR = 1.3
PLATE_HEIGHT_PADDING_FACTOR = 1.5
def detectPlatesInScene(imgOriginalScene):
    listOfPossiblePlates = []                   # this will be the return value
    height, width, numChannels = imgOriginalScene.shape
    imgGrayscaleScene = np.zeros((height, width, 1), np.uint8)
    imgThreshScene = np.zeros((height, width, 1), np.uint8)
    imgContours = np.zeros((height, width, 3), np.uint8)
    cv2.destroyAllWindows()
    if Main.showSteps == True: # show steps
        cv2.imshow("0", imgOriginalScene)
    # end if # show steps
    imgGrayscaleScene, imgThreshScene = Preprocess.preprocess(imgOriginalScene)
# preprocess to get grayscale and threshold images
```

```
    if Main.showSteps == True: # show steps

        cv2.imshow("1a", imgGrayscaleScene)

        cv2.imshow("1b", imgThreshScene)

    # end if # show steps

            # find all possible chars in the scene,

            # this function first finds all contours, then only includes contours that could be
chars (without comparison to other chars yet)

    listOfPossibleCharsInScene = findPossibleCharsInScene(imgThreshScene)

    if Main.showSteps == True: # show steps

        print("step 2 - len(listOfPossibleCharsInScene) = " + str(

            len(listOfPossibleCharsInScene)))  # 131 with MCLRNF1 image

        imgContours = np.zeros((height, width, 3), np.uint8)

        contours = []

        for possibleChar in listOfPossibleCharsInScene:

            contours.append(possibleChar.contour)

        # end for

        cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)

        cv2.imshow("2b", imgContours)

    # end if # show steps

            # given a list of all possible chars, find groups of matching chars

            # in the next steps each group of matching chars will attempt to be recognized as
a plate

    listOfListsOfMatchingCharsInScene =
DetectChars.findListOfListsOfMatchingChars(listOfPossibleCharsInScene)

    if Main.showSteps == True: # show steps

        print("step 3 - listOfListsOfMatchingCharsInScene.Count = " + str(

            len(listOfListsOfMatchingCharsInScene)))  # 13 with MCLRNF1 image
        imgContours = np.zeros((height, width, 3), np.uint8)

        for listOfMatchingChars in listOfListsOfMatchingCharsInScene:

            intRandomBlue = random.randint(0, 255)

            intRandomGreen = random.randint(0, 255)

            intRandomRed = random.randint(0, 255)

            contours = []

            for matchingChar in listOfMatchingChars:

                contours.append(matchingChar.contour)

            # end for

            cv2.drawContours(imgContours, contours, -1, (intRandomBlue,
intRandomGreen, intRandomRed))

        # end for

        cv2.imshow("3", imgContours)

    # end if # show steps

    for listOfMatchingChars in listOfListsOfMatchingCharsInScene:                   # for
each group of matching chars

        possiblePlate = extractPlate(imgOriginalScene, listOfMatchingChars)         #
attempt to extract plate

        if possiblePlate.imgPlate is not None:                          # if plate was found

            listOfPossiblePlates.append(possiblePlate)                  # add to list of possible
plates

        # end if

    # end for

    print("\n" + str(len(listOfPossiblePlates)) + " possible plates found")  # 13 with
MCLRNF1 image

    if Main.showSteps == True: # show steps

        print("\n")

        cv2.imshow("4a", imgContours)
        for i in range(0, len(listOfPossiblePlates)):

            p2fRectPoints = 
cv2.boxPoints(listOfPossiblePlates[i].rrLocationOfPlateInScene)

            cv2.line(imgContours, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), 
Main.SCALAR_RED, 2)

            cv2.line(imgContours, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), 
Main.SCALAR_RED, 2)

            cv2.line(imgContours, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), 
Main.SCALAR_RED, 2)

            cv2.line(imgContours, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), 
Main.SCALAR_RED, 2)

            cv2.imshow("4a", imgContours)

            print("possible plate " + str(i) + ", click on any image and press a key to continue 
. . .")

            cv2.imshow("4b", listOfPossiblePlates[i].imgPlate)

            cv2.waitKey(0)

        # end for

        print("\nplate detection complete, click on any image and press a key to begin char 
recognition . . .\n")

        cv2.waitKey(0)

    # end if # show steps

    return listOfPossiblePlates

# end function

def findPossibleCharsInScene(imgThresh):

    listOfPossibleChars = []                # this will be the return value

    intCountOfPossibleChars = 0

    imgThreshCopy = imgThresh.copy()

    imgContours, contours, npaHierarchy = cv2.findContours(imgThreshCopy, 
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)   # find all contours
```

```
    height, width = imgThresh.shape

    imgContours = np.zeros((height, width, 3), np.uint8)

    for i in range(0, len(contours)):                    # for each contour

        if Main.showSteps == True: # show steps

            cv2.drawContours(imgContours, contours, i, Main.SCALAR_WHITE)

        # end if # show steps

        possibleChar = PossibleChar.PossibleChar(contours[i])

        if DetectChars.checkIfPossibleChar(possibleChar):                  # if contour is a
possible char, note this does not compare to other chars (yet) . . .

            intCountOfPossibleChars = intCountOfPossibleChars + 1           # increment
count of possible chars

            listOfPossibleChars.append(possibleChar)                       # and add to list of
possible chars

        # end if

    # end for

    if Main.showSteps == True: # show steps

        print("\nstep 2 - len(contours) = " + str(len(contours)))  # 2362 with MCLRNF1
image

        print("step 2 - intCountOfPossibleChars = " + str(intCountOfPossibleChars))  #
131 with MCLRNF1 image

        cv2.imshow("2a", imgContours)

    # end if # show steps

    return listOfPossibleChars

# end function

def extractPlate(imgOriginal, listOfMatchingChars):

    possiblePlate = PossiblePlate.PossiblePlate()          # this will be the return value

    listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX)
# sort chars from left to right based on x position
```

```
            # calculate the center point of the plate

    fltPlateCenterX = (listOfMatchingChars[0].intCenterX +
listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterX) / 2.0

    fltPlateCenterY = (listOfMatchingChars[0].intCenterY +
listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterY) / 2.0

    ptPlateCenter = fltPlateCenterX, fltPlateCenterY

            # calculate plate width and height

    intPlateWidth = int((listOfMatchingChars[len(listOfMatchingChars) -
1].intBoundingRectX + listOfMatchingChars[len(listOfMatchingChars) -
1].intBoundingRectWidth - listOfMatchingChars[0].intBoundingRectX) *
PLATE_WIDTH_PADDING_FACTOR)

    intTotalOfCharHeights = 0

    for matchingChar in listOfMatchingChars:

        intTotalOfCharHeights = intTotalOfCharHeights +
matchingChar.intBoundingRectHeight

    # end for

    fltAverageCharHeight = intTotalOfCharHeights / len(listOfMatchingChars)

    intPlateHeight = int(fltAverageCharHeight *
PLATE_HEIGHT_PADDING_FACTOR)

            # calculate correction angle of plate region

    fltOpposite = listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterY -
listOfMatchingChars[0].intCenterY

    fltHypotenuse = DetectChars.distanceBetweenChars(listOfMatchingChars[0],
listOfMatchingChars[len(listOfMatchingChars) - 1])

    fltCorrectionAngleInRad = math.asin(fltOpposite / fltHypotenuse)

    fltCorrectionAngleInDeg = fltCorrectionAngleInRad * (180.0 / math.pi)

            # pack plate region center point, width and height, and correction angle into
rotated rect member variable of plate

    possiblePlate.rrLocationOfPlateInScene = ( tuple(ptPlateCenter), (intPlateWidth,
intPlateHeight), fltCorrectionAngleInDeg )
```

```
        # final steps are to perform the actual rotation

        # get the rotation matrix for our calculated correction angle

    rotationMatrix = cv2.getRotationMatrix2D(tuple(ptPlateCenter),
fltCorrectionAngleInDeg, 1.0)

    height, width, numChannels = imgOriginal.shape      # unpack original image width
and height

    imgRotated = cv2.warpAffine(imgOriginal, rotationMatrix, (width, height))       #
rotate the entire image

    imgCropped = cv2.getRectSubPix(imgRotated, (intPlateWidth, intPlateHeight),
tuple(ptPlateCenter))

    possiblePlate.imgPlate = imgCropped         # copy the cropped plate image into the
applicable member variable of the possible plate

    return possiblePlate

# end function
```

8.6 DetectChars.py

```
# DetectChars.py

import os

import cv2

import numpy as np

import math

import random

import Main

import Preprocess

import PossibleChar

# module level variables


kNearest = cv2.ml.KNearest_create()


        # constants for checkIfPossibleChar, this checks one possible char only (does not
compare to another char)

MIN_PIXEL_WIDTH = 2

MIN_PIXEL_HEIGHT = 8

MIN_ASPECT_RATIO = 0.25

MAX_ASPECT_RATIO = 1.0

MIN_PIXEL_AREA = 80

        # constants for comparing two chars

MIN_DIAG_SIZE_MULTIPLE_AWAY = 0.3

MAX_DIAG_SIZE_MULTIPLE_AWAY = 5.0

MAX_CHANGE_IN_AREA = 0.5

MAX_CHANGE_IN_WIDTH = 0.8
```

```
MAX_CHANGE_IN_HEIGHT = 0.2

MAX_ANGLE_BETWEEN_CHARS = 12.0

        # other constants

MIN_NUMBER_OF_MATCHING_CHARS = 3

RESIZED_CHAR_IMAGE_WIDTH = 20

RESIZED_CHAR_IMAGE_HEIGHT = 30

MIN_CONTOUR_AREA = 100

def loadKNNDataAndTrainKNN():

    allContoursWithData = []                # declare empty lists,

    validContoursWithData = []              # we will fill these shortly

    try:

        npaClassifications = np.loadtxt("classifications.txt", np.float32)

 # read in training classifications

    except:                                                  # if file could not be opened

        print("error, unable to open classifications.txt, exiting program\n")  # show error
message

        os.system("pause")

        return False                                          # and return False

    # end try

    try:

        npaFlattenedImages = np.loadtxt("flattened_images.txt", np.float32)                 #
read in training images

    except:                                                  # if file could not be opened

        print("error, unable to open flattened_images.txt, exiting program\n")  # show error
message

        os.system("pause")

        return False                                          # and return False
    # end try

    npaClassifications = npaClassifications.reshape((npaClassifications.size, 1))

  # reshape numpy array to 1d, necessary to pass to call to train

    kNearest.setDefaultK(1)                                          # set default K to 1

    kNearest.train(npaFlattenedImages, cv2.ml.ROW_SAMPLE, npaClassifications)
# train KNN object

    return True                       # if we got here training was successful so return true

# end function

def detectCharsInPlates(listOfPossiblePlates):

    intPlateCounter = 0

    imgContours = None

    contours = []

    if len(listOfPossiblePlates) == 0:          # if list of possible plates is empty

        return listOfPossiblePlates            # return

   # end if

            # at this point we can be sure the list of possible plates has at least one plate

    for possiblePlate in listOfPossiblePlates:          # for each possible plate, this is a big
for loop that takes up most of the function

        possiblePlate.imgGrayscale, possiblePlate.imgThresh =
Preprocess.preprocess(possiblePlate.imgPlate)     # preprocess to get grayscale and
threshold images

        if Main.showSteps == True: # show steps

            cv2.imshow("5a", possiblePlate.imgPlate)

            cv2.imshow("5b", possiblePlate.imgGrayscale)

            cv2.imshow("5c", possiblePlate.imgThresh)

        # end if # show steps

                # increase size of plate image for easier viewing and char detection
    possiblePlate.imgThresh = cv2.resize(possiblePlate.imgThresh, (0, 0), fx = 1.6, fy
= 1.6)

        # threshold again to eliminate any gray areas

    thresholdValue, possiblePlate.imgThresh = cv2.threshold(possiblePlate.imgThresh,
0.0, 255.0, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

    if Main.showSteps == True: # show steps

      cv2.imshow("5d", possiblePlate.imgThresh)

    # end if # show steps

        # find all possible chars in the plate,

        # this function first finds all contours, then only includes contours that could
be chars (without comparison to other chars yet)

    listOfPossibleCharsInPlate =
findPossibleCharsInPlate(possiblePlate.imgGrayscale, possiblePlate.imgThresh)

    if Main.showSteps == True: # show steps

      height, width, numChannels = possiblePlate.imgPlate.shape

      imgContours = np.zeros((height, width, 3), np.uint8)

      del contours[:]                                # clear the contours list

      for possibleChar in listOfPossibleCharsInPlate:

        contours.append(possibleChar.contour)

      # end for

      cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)

      cv2.imshow("6", imgContours)

    # end if # show steps

        # given a list of all possible chars, find groups of matching chars within the
plate

    listOfListsOfMatchingCharsInPlate =
findListOfListsOfMatchingChars(listOfPossibleCharsInPlate)

    if Main.showSteps == True: # show steps
```

```
            imgContours = np.zeros((height, width, 3), np.uint8)

            del contours[:]

            for listOfMatchingChars in listOfListsOfMatchingCharsInPlate:

                intRandomBlue = random.randint(0, 255)

                intRandomGreen = random.randint(0, 255)

                intRandomRed = random.randint(0, 255)

                for matchingChar in listOfMatchingChars:

                    contours.append(matchingChar.contour)

                # end for

                cv2.drawContours(imgContours, contours, -1, (intRandomBlue,
intRandomGreen, intRandomRed))

            # end for

            cv2.imshow("7", imgContours)

        # end if # show steps


        if (len(listOfListsOfMatchingCharsInPlate) == 0):                  # if no
groups of matching chars were found in the plate

            if Main.showSteps == True: # show steps

                print("chars found in plate number " + str(

                    intPlateCounter) + " = (none), click on any image and press a key to
continue . . .")

                intPlateCounter = intPlateCounter + 1

                cv2.destroyWindow("8")

                cv2.destroyWindow("9")

                cv2.destroyWindow("10")

                cv2.waitKey(0)

            # end if # show steps
            possiblePlate.strChars = ""

            continue                                                    # go back to top of for loop

        # end if

        for i in range(0, len(listOfListsOfMatchingCharsInPlate)):                              # 
within each list of matching chars

            listOfListsOfMatchingCharsInPlate[i].sort(key = lambda matchingChar: 
matchingChar.intCenterX)        # sort chars from left to right

            listOfListsOfMatchingCharsInPlate[i] = 
removeInnerOverlappingChars(listOfListsOfMatchingCharsInPlate[i])              # and 
remove inner overlapping chars

        # end for

        if Main.showSteps == True: # show steps

            imgContours = np.zeros((height, width, 3), np.uint8)

            for listOfMatchingChars in listOfListsOfMatchingCharsInPlate:

                intRandomBlue = random.randint(0, 255)

                intRandomGreen = random.randint(0, 255)

                intRandomRed = random.randint(0, 255)

               del contours[:]

                for matchingChar in listOfMatchingChars:

                    contours.append(matchingChar.contour)

                # end for

                cv2.drawContours(imgContours, contours, -1, (intRandomBlue, 
intRandomGreen, intRandomRed))

            # end for

            cv2.imshow("8", imgContours)

        # end if # show steps

                # within each possible plate, suppose the longest list of potential matching 
chars is the actual list of chars
```

```
        intLenOfLongestListOfChars = 0

        intIndexOfLongestListOfChars = 0

                # loop through all the vectors of matching chars, get the index of the one with
the most chars

        for i in range(0, len(listOfListsOfMatchingCharsInPlate)):

            if len(listOfListsOfMatchingCharsInPlate[i]) > intLenOfLongestListOfChars:

                intLenOfLongestListOfChars = len(listOfListsOfMatchingCharsInPlate[i])

                intIndexOfLongestListOfChars = i

            # end if

        # end for

                # suppose that the longest list of matching chars within the plate is the actual
list of chars

        longestListOfMatchingCharsInPlate =
listOfListsOfMatchingCharsInPlate[intIndexOfLongestListOfChars]

        if Main.showSteps == True: # show steps

            imgContours = np.zeros((height, width, 3), np.uint8)

            del contours[:]

            for matchingChar in longestListOfMatchingCharsInPlate:

                contours.append(matchingChar.contour)

            # end for

            cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)

            cv2.imshow("9", imgContours)

        # end if # show steps

        possiblePlate.strChars = recognizeCharsInPlate(possiblePlate.imgThresh,
longestListOfMatchingCharsInPlate)

        if Main.showSteps == True: # show steps

            print("chars found in plate number " + str(
                intPlateCounter) + " = " + possiblePlate.strChars + ", click on any image and
press a key to continue . . .")

            intPlateCounter = intPlateCounter + 1

            cv2.waitKey(0)

        # end if # show steps

    # end of big for loop that takes up most of the function

    if Main.showSteps == True:

        print("\nchar detection complete, click on any image and press a key to continue . .
.\n")

        cv2.waitKey(0)

    # end if

    return listOfPossiblePlates

# end function

def findPossibleCharsInPlate(imgGrayscale, imgThresh):

    listOfPossibleChars = []                        # this will be the return value

    contours = []

    imgThreshCopy = imgThresh.copy()

            # find all contours in plate

    imgContours, contours, npaHierarchy = cv2.findContours(imgThreshCopy,
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:                        # for each contour

        possibleChar = PossibleChar.PossibleChar(contour)

        if checkIfPossibleChar(possibleChar):              # if contour is a possible char, note
this does not compare to other chars (yet) . . .

            listOfPossibleChars.append(possibleChar)       # add to list of possible chars

        # end if

    # end if
    return listOfPossibleChars

# end function

def checkIfPossibleChar(possibleChar):

            # this function is a 'first pass' that does a rough check on a contour to see if it
could be a char,

            # note that we are not (yet) comparing the char to other chars to look for a group

    if (possibleChar.intBoundingRectArea > MIN_PIXEL_AREA and

        possibleChar.intBoundingRectWidth > MIN_PIXEL_WIDTH and
possibleChar.intBoundingRectHeight > MIN_PIXEL_HEIGHT and

        MIN_ASPECT_RATIO < possibleChar.fltAspectRatio and
possibleChar.fltAspectRatio < MAX_ASPECT_RATIO):

        return True

    else:

        return False

    # end if

# end function


def findListOfListsOfMatchingChars(listOfPossibleChars):

            # with this function, we start off with all the possible chars in one big list

            # the purpose of this function is to re-arrange the one big list of chars into a list
of lists of matching chars,

            # note that chars that are not found to be in a group of matches do not need to be
considered further

    listOfListsOfMatchingChars = []                  # this will be the return value

    for possibleChar in listOfPossibleChars:                        # for each possible char in the
one big list of chars

        listOfMatchingChars = findListOfMatchingChars(possibleChar,
listOfPossibleChars)        # find all chars in the big list that match the current char
```

```
        listOfMatchingChars.append(possibleChar)                # also add the current char to
current possible list of matching chars

        if len(listOfMatchingChars) < MIN_NUMBER_OF_MATCHING_CHARS:     # if
current possible list of matching chars is not long enough to constitute a possible plate

            continue                            # jump back to the top of the for loop and try again
with next char, note that it's not necessary

                                                # to save the list in any way since it did not have enough
chars to be a possible plate

        # end if

   # if we get here, the current list passed test as a "group" or "cluster" of matching chars

        listOfListsOfMatchingChars.append(listOfMatchingChars)

# so add to our list of lists of matching chars

        listOfPossibleCharsWithCurrentMatchesRemoved = []

                                                # remove the current list of matching chars from the big
list so we don't use those same chars twice,

                                                # make sure to make a new big list for this since we don't
want to change the original big list

        listOfPossibleCharsWithCurrentMatchesRemoved = list(set(listOfPossibleChars) -
set(listOfMatchingChars))

        recursiveListOfListsOfMatchingChars =
findListOfListsOfMatchingChars(listOfPossibleCharsWithCurrentMatchesRemoved)
# recursive call

        for recursiveListOfMatchingChars in recursiveListOfListsOfMatchingChars:        #
for each list of matching chars found by recursive call

            listOfListsOfMatchingChars.append(recursiveListOfMatchingChars)            #
add to our original list of lists of matching chars

        # end for

        break       # exit for

    # end for

    return listOfListsOfMatchingChars
```

```
# end function

def findListOfMatchingChars(possibleChar, listOfChars):

   # the purpose of this function is, given a possible char and a big list of possible chars,

         # find all chars in the big list that are a match for the single possible char, and
return those matching chars as a list

   listOfMatchingChars = []              # this will be the return value

   for possibleMatchingChar in listOfChars:              # for each char in big list

      if possibleMatchingChar == possibleChar:    # if the char we attempting to find
matches for is the exact same char as the char in the big list we are currently checking

                                        # then we should not include it in the list of matches b/c
that would end up double including the current char

         continue                         # so do not add to list of matches and jump back to
top of for loop

      # end if

               # compute stuff to see if chars are a match

      fltDistanceBetweenChars = distanceBetweenChars(possibleChar,
possibleMatchingChar)


      fltAngleBetweenChars = angleBetweenChars(possibleChar,
possibleMatchingChar)

      fltChangeInArea = float(abs(possibleMatchingChar.intBoundingRectArea -
possibleChar.intBoundingRectArea)) / float(possibleChar.intBoundingRectArea)

      fltChangeInWidth = float(abs(possibleMatchingChar.intBoundingRectWidth -
possibleChar.intBoundingRectWidth)) / float(possibleChar.intBoundingRectWidth)

      fltChangeInHeight = float(abs(possibleMatchingChar.intBoundingRectHeight -
possibleChar.intBoundingRectHeight)) / float(possibleChar.intBoundingRectHeight)

            # check if chars match

      if (fltDistanceBetweenChars < (possibleChar.fltDiagonalSize *
MAX_DIAG_SIZE_MULTIPLE_AWAY) and

         fltAngleBetweenChars < MAX_ANGLE_BETWEEN_CHARS and
```

```
            fltChangeInArea < MAX_CHANGE_IN_AREA and

            fltChangeInWidth < MAX_CHANGE_IN_WIDTH and

            fltChangeInHeight < MAX_CHANGE_IN_HEIGHT):

            listOfMatchingChars.append(possibleMatchingChar)        # if the chars are a
match, add the current char to list of matching chars

        # end if

    # end for

    return listOfMatchingChars                  # return result

# end function

# use Pythagorean theorem to calculate distance between two chars

def distanceBetweenChars(firstChar, secondChar):

    intX = abs(firstChar.intCenterX - secondChar.intCenterX)

    intY = abs(firstChar.intCenterY - secondChar.intCenterY)

    return math.sqrt((intX ** 2) + (intY ** 2))

# end function


# use basic trigonometry (SOH CAH TOA) to calculate angle between chars

def angleBetweenChars(firstChar, secondChar):

    fltAdj = float(abs(firstChar.intCenterX - secondChar.intCenterX))

    fltOpp = float(abs(firstChar.intCenterY - secondChar.intCenterY))

    if fltAdj != 0.0:                         # check to make sure we do not divide by zero if the
center X positions are equal, float division by zero will cause a crash in Python

        fltAngleInRad = math.atan(fltOpp / fltAdj)      # if adjacent is not zero, calculate
angle

    else:

        fltAngleInRad = 1.5708                        # if adjacent is zero, use this as the angle,
this is to be consistent with the C++ version of this program
```

```
    # end if

    fltAngleInDeg = fltAngleInRad * (180.0 / math.pi)       # calculate angle in degrees

    return fltAngleInDeg

# end function

# if we have two chars overlapping or to close to each other to possibly be separate
chars, remove the inner (smaller) char,

# this is to prevent including the same char twice if two contours are found for the same
char,

# for example for the letter 'O' both the inner ring and the outer ring may be found as
contours, but we should only include the char once

def removeInnerOverlappingChars(listOfMatchingChars):

    listOfMatchingCharsWithInnerCharRemoved = list(listOfMatchingChars)                #
this will be the return value

    for currentChar in listOfMatchingChars:

        for otherChar in listOfMatchingChars:

            if currentChar != otherChar:        # if current char and other char are not the
same char . . .

                                                            # if current char and other char have
center points at almost the same location . . .

                if distanceBetweenChars(currentChar, otherChar) <
(currentChar.fltDiagonalSize * MIN_DIAG_SIZE_MULTIPLE_AWAY):

                                # if we get in here we have found overlapping chars

                                # next we identify which char is smaller, then if that char was not
already removed on a previous pass, remove it

                    if currentChar.intBoundingRectArea < otherChar.intBoundingRectArea:
# if current char is smaller than other char

                        if currentChar in listOfMatchingCharsWithInnerCharRemoved:
# if current char was not already removed on a previous pass . . .

                            listOfMatchingCharsWithInnerCharRemoved.remove(currentChar)
# then remove current char
```

```
                        # end if

                    else:                                                    # else if other char is
smaller than current char

                        if otherChar in listOfMatchingCharsWithInnerCharRemoved:                #
if other char was not already removed on a previous pass . . .

                            listOfMatchingCharsWithInnerCharRemoved.remove(otherChar)
# then remove other char

                        # end if

                    # end if

                # end if

            # end if

        # end for

    # end for

    return listOfMatchingCharsWithInnerCharRemoved

# end function


# this is where we apply the actual char recognition

def recognizeCharsInPlate(imgThresh, listOfMatchingChars):

    strChars = ""               # this will be the return value, the chars in the lic plate

    height, width = imgThresh.shape

    imgThreshColor = np.zeros((height, width, 3), np.uint8)

    listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX)
# sort chars from left to right

    cv2.cvtColor(imgThresh, cv2.COLOR_GRAY2BGR, imgThreshColor)
# make color version of threshold image so we can draw contours in color on it

    for currentChar in listOfMatchingChars:                               # for each char in
plate

        pt1 = (currentChar.intBoundingRectX, currentChar.intBoundingRectY)
        pt2 = ((currentChar.intBoundingRectX + currentChar.intBoundingRectWidth),
(currentChar.intBoundingRectY + currentChar.intBoundingRectHeight))

        cv2.rectangle(imgThreshColor, pt1, pt2, Main.SCALAR_GREEN, 2)           # draw
green box around the char

                # crop char out of threshold image

        imgROI = imgThresh[currentChar.intBoundingRectY :
currentChar.intBoundingRectY + currentChar.intBoundingRectHeight,

                            currentChar.intBoundingRectX : currentChar.intBoundingRectX +
currentChar.intBoundingRectWidth]

        imgROIResized = cv2.resize(imgROI, (RESIZED_CHAR_IMAGE_WIDTH,
RESIZED_CHAR_IMAGE_HEIGHT))          # resize image, this is necessary for char
recognition

        npaROIResized = imgROIResized.reshape((1,
RESIZED_CHAR_IMAGE_WIDTH * RESIZED_CHAR_IMAGE_HEIGHT))        #
flatten image into 1d numpy array

        npaROIResized = np.float32(npaROIResized)              # convert from 1d numpy
array of ints to 1d numpy array of floats

        retval, npaResults, neigh_resp, dists = kNearest.findNearest(npaROIResized, k =
1)             # finally we can call findNearest !!!

        strCurrentChar = str(chr(int(npaResults[0][0])))           # get character from results

        strChars = strChars + strCurrentChar                       # append current char to full
string

    # end for

    if Main.showSteps == True: # show steps

        cv2.imshow("10", imgThreshColor)

    # end if # show steps

    return strChars

# end function
```

9. APPENDIX-B

Steps for installing OpenCV 3.0 and Python 2.7(Ubuntu):

- **Open a terminal and update the apt-get package manager followed by upgrading any pre-installed packages:**
  ```
  $ sudo apt-get update
  $ sudo apt-get upgrade
  ```

- **Install developer tools:**
  ```
  $ sudo apt-get install build-essential cmake git pkg-config
  ```

- **OpenCV needs to be able to load various image file formats from disk, including JPEG, PNG, TIFF, etc. In order to load these image formats from disk, we'll need our image I/O packages:**
  ```
  $ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev
  ```

- **GTK development library, which the highgui module of OpenCV depends on to guild Graphical User Interfaces (GUIs):**
  ```
  $ sudo apt-get install libgtk2.0-dev
  ```

- **We can load images using OpenCV, but what about processing video streams and accessing individual frames we need the following:**
  ```
  $ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
  ```

- **Install libraries that are used to optimize various routines inside of OpenCV:**
  ```
  $ sudo apt-get install libatlas-base-dev gfortran
  ```

- **Install pip, a Python package manager:**
  ```
  $ wget https://bootstrap.pypa.io/get-pip.py
  $ sudo python get-pip.py
  ```

- **Install virtualenv and virtualenvwrapper, these two packages allow us to create *separate Python environments* for each project:**
  ```
  $ sudo pip install virtualenv virtualenvwrapper
  $ sudo rm -rf ~/. cache/pip
  ```

- **Now virtualenv and virtualenvwrapper installed, we need to update our ~/. bashrc file:**
```
export WORKON_HOME=$HOME/. virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
$ source ~/. bashrc
```

- **create our cv virtual environment where we'll be doing our computer vision development and OpenCV 3.0 + Python 2.7+ installation:**
```
$ mkvirtualenv cv
```

- **install Python 2.7+**
```
$ sudo apt-get install python2.7-dev
```

- **Since OpenCV represents images as multi-dimensional NumPy arrays, we better install NumPy into our cv virtual environment:**
```
$ pip install numpy
```

- **environment is now all setup — we can proceed to change to our home directory, pull down OpenCV from GitHub, and checkout the 3.0.0 version:**
```
$ cd ~
$ git clone https://github.com/Itseez/opencv.git
$ cd opencv
$ git checkout 3.0.0
```

- **opencv_contrib repo as well. Without this repository, we won't have access to standard keypoint detectors and local invariant descriptors (such as SIFT, SURF, etc.) that were available in the OpenCV 2.4.X version. We'll also be missing out on some of the newer OpenCV 3.0 features like text detection in natural images:**
```
$ cd ~
$ git clone https://github.com/Itseez/opencv_contrib.git
$ cd opencv_contrib
$ git checkout 3.0.0
```

- **Time to setup the build:**
```
$ cd ~/opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D INSTALL_C_EXAMPLES=ON \
        -D INSTALL_PYTHON_EXAMPLES=ON \
        -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
        -D BUILD_EXAMPLES=ON.
```

- **finally compile OpenCV:**

```
$ make -j4
```

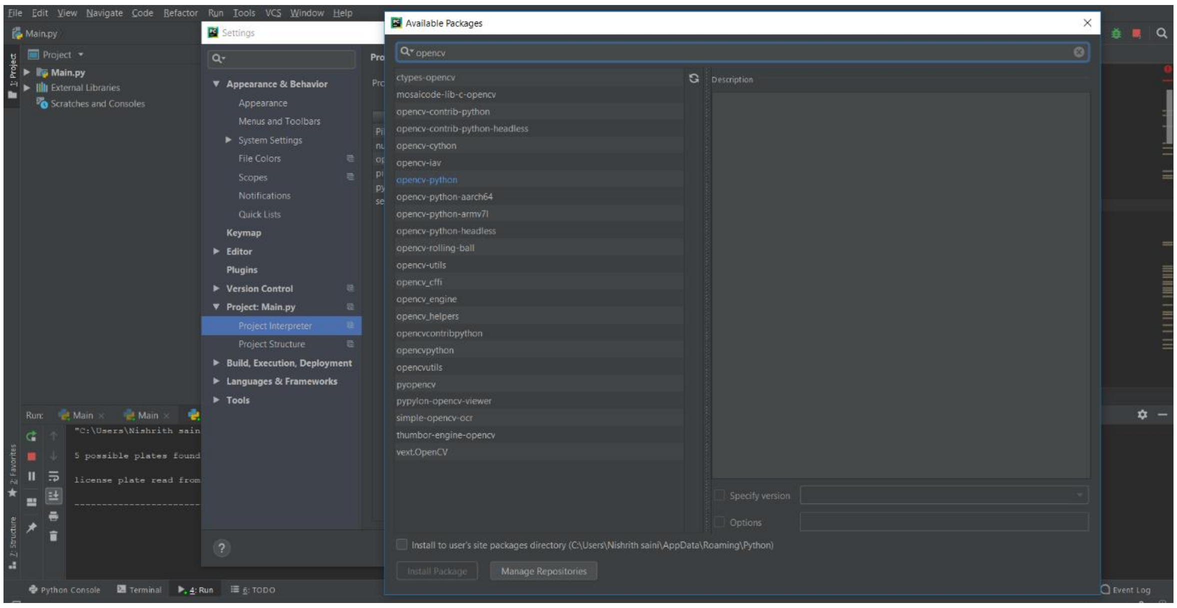
- **Assuming that OpenCV compiled without error, you can now install it on your Ubuntu system:**

```
$ sudo make install
$ sudo ldconfig
```

- **Installation is Finished.**

Steps for installing OpenCV 3.0 and Python 2.7(Windows):

- **Install Phycharm open source IDE**
- **Install OpenCV packages in Phycharm**
    - **Go to settings**
    - **Select project interpreter**
    - **Install OpenCV-python package**
    - **Install Numpy package**
    - **Install Pillow package**



```
"C:\Users\Nishrith main
5 possible plates found
license plate read from
-----------------------
```

Fig (9.1)