**Abstract**

Automatic License Plate Recognition (ALPR) is a mass surveillance system that captures the image of vehicles and recognizes their license number. ALPR can be assisted in the detection of stolen vehicles. The detection of stolen vehicles can be done in an efficient manner by using the ALPR systems located in the highways. This paper presents a recognition method in which the vehicle plate image is obtained by the digital cameras and the image is processed to get the number plate information. A rear image of a vehicle is captured and processed using various algorithms. In this context, the number plate area is localized using a novel „feature-based number plate localization‟ method which consists of many algorithms. But our study mainly focusing on the two fast algorithms i.e., Edge Finding Method and Window Filtering Method for the better development of the number plate detection system. It is observed from the experiment that the developed system successfully detects and recognize the vehicle number plate on real images.



**CONTENTS**

**1. INTRODUCTION** ........................................................................................................................................ 9

**1.1 BACKGROUND AND MOTIVATION**..................................................................................................... 9

**1.2 AIM OF THE PROJECT** ........................................................................................................................10

**1.3 THESIS STRUCTURE**............................................................................................................................10

**2. An overview of Basic Image Processing Concepts** ................................................................................11

**2.1 Image Representation**.......................................................................................................................11

**2.2 Pixel**....................................................................................................................................................11

**2.3 Colorspaces**....................................................................................................................................... 12

**2.3.1 RGB Color Model................................................................................................................12**

**2.3.2 YCbCr models.....................................................................................................................12**

**2.3.3 Grayscale model.................................................................................................................13**

**2.3.4 Binary image......................................................................................................................13**

**2.4 Thresholding**......................................................................................................................................13

**2.4.1 Global thresholding............................................................................................................13**

**2.4.2 Adaptive Thresholding .......................................................................................................13**

**2.5 Edge Detection** ................................................................................................................................14

**2.5.1 Kernel convolution .............................................................................................................14**

**2.5.2 Sobel edge detection..........................................................................................................14**

**2.5.3 Canny edge detection.........................................................................................................15**

**2.6 Morphological image processing**.....................................................................................................18

**2.6.1 Erosion ..............................................................................................................................18**

**2.6.2 Dilation..............................................................................................................................18**

**2.6.3 Opening of an image ........................................................................................................19**

**2.6.4 Closing of an image ..........................................................................................................19**

**2.6.5 Morphological Gradient ...................................................................................................20**

**2.6.6 Top Hat............................................................................................................................20**

**2.6.7 Black Hat..........................................................................................................................20**

**3. METHODOLOGY**......................................................................................................................................21

**3.0 Development of the ANPR system** ..................................................................................................21

**3.1 Image acquisition** .............................................................................................................................21

**3.2 Image pre-processing** .......................................................................................................................22

**3.3 License Plate Localization and Segmentation** .................................................................................26

**3.4 Characters Recognition** ....................................................................................................................29

**4. RESULTS AND OBSERVATIONS** ..............................................................................................................30

**5. CONCLUSION** ..........................................................................................................................................31

**REFERENCES**................................................................................................................................................32

**APPENDIX A** ................................................................................................................................................33

**APPENDIX B.**….……………………………………………………………………………………………………………………………………….34

**1**. **INTRODUCTION**

**1.1 BACKGROUND AND MOTIVATION**

Automatic License Plate Recognition (ALPR) is a computer vision technology to extract the license number of vehicles from images without human intervention. It is one of the necessary systems designed to detect the vehicle number plate. In today’s world with the increasing number of vehicle day by day it’s not possible to manually keep a record of the entire vehicle. With the development of this system it becomes easy to keep a record and use it whenever required. Some of the typical applications of ANPR is also used by police forces around the world for law enforcement purposes, including to check if a vehicle is registered or licensed.

Typical ALPR 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 it became easy for people to develop solutions for many problems. One of the notable contribution of the open source community to the scientiﬁc world is Python. Intel’s researches in Computer Vision developed open source library called Open Computer Vision (OpenCV) library, which can support computer vision development. However, the development of ANPR systems is no easy task, since it faces numerous challenges due to environmental and number plate variations. Number Plates in different countries follows different rules and fonts so it becomes a difficult task to develop a universal number plate recognition system. License plate recognition depends on the image quality. Noises in the image plate make the recognition difficult. Many countries uses ALPR specific to their countries number plate and these ALPR systems uses properties of the plates to improve their accuracy.

**1.2 AIM OF THE PROJECT**

The aim of this project is to be able to develop a system for automatically recognizing number plates from High Resolution digital images by using image processing techniques and optical character recognition technique. The main objective is to develop a efficient, fast and reliable algorithm. The system developed is supposed to work on different country plates.

**1.3 THESIS STRUCTURE**

This Thesis is divided into five different chapters. Chapter 1 explains the motivation for the project and establishes the aim of the project. Chapter2 is an overview on basic image processing techniques Chapter3 describes in detail the methodology carried out for developing the ALPR application. Chapter 4 shows the experimental results obtained with the developed system. Finally chapter 5 summarizes the most important conclusions extracted from the development of the project.

**2. An overview of Basic Image Processing Concepts**

2.1 Image Representation

An image is a type of data representation in which any still real-life physical object or scene is depicted on a 2 dimensional screen or as a 3 dimensional hologram. In the context of signal processing an image is a distributed amplitude of color.

The digital image is formed as a result of process which is converts analog signals into digital signals. This process is possible with the subject of energy spread (analog signal) predicted by an electromagnetic sensor to the digital signal into the detection range.

A digital camera captures the image with the help of sensors that measure the intensity of light falling on various regions of the filter and thereby assigning the respective intensity values accordingly.

**2.2 Pixel**

A pixel is a basic unit of an image. In digital imaging, a pixel, or picture element is a physical point in a raster image, or the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen.

There are two basic features of creating a pixel.

 Radiometric feature: It is a gray value which creates pixel detection on the electromagnetic spectrum.

 Geometric feature: It is a matrix coordinate which indicates the position of the image matrix.

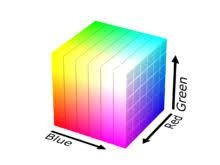


**2.3 Colorspaces**

A color space is a specific organization of colors. In combination with physical device profiling, it allows for reproducible representations of color, in both analog and digital representations. A color space may be arbitrary, with particular colors assigned to a set of physical color swatches and corresponding assigned color names or numbers, or structured mathematically.

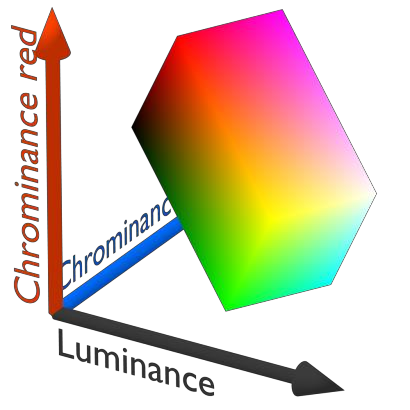
**2.3.1 RGB Color Model**

The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue. In this the intensity levels of individual colors red, green and blue in individual pixels are stored in 3 different matrices.



**2.3.2 YCbCr models**

YCbCr, Y′CbCr also written as YCBCR or Y'CBCR, is a family of color spaces used as a part of the color image pipeline in video and digital photography systems. Y′ is the luma component and CB and CR are the blue-difference and red-difference chroma components.



**2.3.3 Grayscale model**

A grayscale image is one in which the value of each pixel is a single sample representing only an amount of light, that is, it carries only intensity information. Grayscale images, a kind of black- and-white or gray monochrome, are composed exclusively of shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest. In general the intensity ranges from a value of 0 to 255. For image processing the image is often converted to its grayscale. New Greyscale image = ((0.3\*R)+(0.59\*G)+(0.11\*B))

**2.3.4 Binary image**

A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The color used for the object in the image is the foreground color while the rest of the image is the background color.

**2.4 Thresholding**

Thresholding is the processing of a grayscale image in which the pixel value is set to zero if its initial value is below a certain limit and is set to the maximum possible value otherwise. Thresholding converts a grayscale image to a binary image.

**2.4.1 Global thresholding**

In this thresholding the threshold limit is set by the user. All the pixel values above this value

are assigned a binary 1 and all the pixel values below this threshold are assigned a value 0. Thus creating a binary image.

The function **cv.threshold** is used to apply the thresholding. The first argument is the source image, which should be a grayscale image. The second argument is the threshold value which is used to classify the pixel values. The third argument is the maximum value which is assigned to pixel values exceeding the threshold. OpenCV provides different types of thresholding which is given by the fourth parameter of the function.

**2.4.2 Adaptive Thresholding**

We used one global value as a threshold. But this might not be good in all cases, e.g. if an image has different lighting conditions in different areas. In that case, adaptive thresholding can help. Here, the algorithm determines the threshold for a pixel based on a small region around it. So we get different thresholds for different regions of the same image which gives better results

for images with varying illumination.

**2.5 Edge Detection**

Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction.

 Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness.

 The edge representation of an image significantly reduces the quantity of data to be processed, yet it retains essential information regarding the shapes of objects in the scene.

Different techniques used:

 Sobel edge detection

 Canny edge detection

**2.5.1 Kernel convolution**

A matrix usually smaller than the image which transforms the image to a required format based on the contents of the kernel

Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel. This is related to a form of mathematical convolution. It should be noted that the matrix operation being performed - convolution - is not traditional matrix multiplication,

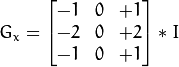
despite being similarly denoted by \*.

**2.5.2 Sobel edge detection**

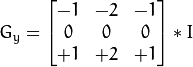
Sobel edge detection is a technique in which the edges are computed by taking the gradient in vertical and horizontal directions. It is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

We calculate two derivatives:

a. Horizontal changes: This is computed by convolving with a kernel  with odd size. For example for a kernel size of 3,  would be computed as:



b. Vertical changes: This is computed by convolving with a kernel  with odd size. For example for a kernel size of 3,  would be computed as:



At each point of the image we calculate an approximation of the gradient in that point by combining both results above:



**2.5.3 Canny edge detection**

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in

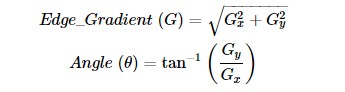
1. It is a multi-stage algorithm and we will go through each stages.

**2.Noise Reduction**

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

1. **Finding Intensity Gradient of the Image**

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction ( Gx) and vertical direction ( Gy). From these two images, we can find edge gradient and direction for each pixel as follows:



Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

1. **Non-maximum Suppression**

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:



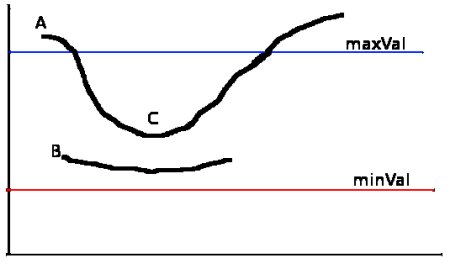
**image**

Point A is on the edge ( in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed ( put to zero).

In short, the result you get is a binary image with "thin edges".

1. **Hysteresis Thresholding**

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:



**image**

The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

This stage also removes small pixels noises on the assumption that edges are long lines.

So what we finally get is strong edges in the image.

**2.6 Morphological image processing**

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image.

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, and second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc. also comes into play. We will see them one-by-one with help of following image:



**2.6.1 Erosion**

The erosion of a binary image *f* by a structuring element *s* (denoted *f*  *s*) produces a new binary image *g* = *f*  *s* with ones in all locations (x, y) of a structuring element's origin at which that structuring element *s* fits the input image *f*, i.e. *g*(x, y) = 1 is *s* fits *f* and 0 otherwise, repeating for all pixel coordinates (x, y).



**2.6.2 Dilation**

The dilation of an image *f* by a structuring element *s* (denoted *f*  *s*) produces a new binary image *g* = *f*  *s* with ones in all locations (*x, y*) of a structuring element's origin at which that structuring element *s* hits the input image *f*, i.e. *g(x, y)* = 1 if *s* hits *f* and 0 otherwise, repeating for all pixel coordinates (*x, y*). Dilation has the opposite effect to erosion -- it adds a layer of pixels to both the inner and outer boundaries of regions.



**2.6.3 Opening of an image**

Opening is just another name of erosion followed by dilation. It is useful in removing noise.



**2.6.4 Closing of an image**

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.



**2.6.5 Morphological Gradient**

It is the difference between dilation and erosion of an image



**2.6.6 Top Hat**

It is the difference between input image and Opening of the image. Below example is done for a

9x9 kernel.



**2.6.7 Black Hat**

It is the difference between the closing of the input image and input image.



**3. METHODOLOGY**

This chapter explains the Methodology that has been followed throughout the development of this project.

**3.0 Development of the ALPR system**

The Automatic License Plate Recognition System designed in this project has been developed using python in the Integrated Development Environment (IDE) of anaconda with openCV library to be able to use computer vision sources , pytesseract ocr for character recognition. All ANPR system shares four common stages they are

1. Image Acquisition

2. Pre-Processing

3. License Plate Localization and Segmentation

4. Recognition of Characters

**3.1 Image acquisition**

Image acquisition is the ﬁrst stage of every ALPR system and it consists in acquiring the image from which the number plate is going to be recognized. Thus, unlike the rest of the stages, it does not depend on software, but it does require a good hardware that is a camera that captures high quality pictures of cars. In this project we considered the captured image to be from a digital camera which captures a high quality color image.

*Implementation*

After obtaining the image we must load this image into our program. For this purpose the user must specify the location of the image file. We load the image by the means openCV command cv2.imread(),and we display the image by the means openCV command cv2.imshow().This function loads images in many formats.

cv2.imread(“source\_image”,cv2.IMREAD\_COLOR)



**Original image**

**3.2 Image pre-processing**

Image pre-processing forms the second stage of our system. It is a crucial stage which processes the image and prepares the image for the remaining stages so that they can be carried out successfully. The pixel values of the grayscale image range from 0-255. Working with greyscale images is comparatively easier than colour images the computation time required for performing operations is also less for greyscale images due to their simplicity.

*Implementation*

OpenCV provides a function cv2.cvtcolor() that carries out different colour transformations. We use this function to convert our RGB image to greyscale image. The first argument is the input image file and the second one is the flag to convert to grayscale. The result obtained after the conversion is

cv2.cvtcolor(input\_image,cv2.COLOR\_BGR2GRAY)



**grayscale image**

The next step in preprocess is to blur the image so as to remove the noise. In this project we used two kinds of filters median blur and Gaussian blur. Median blur is a simple but effective method to remove noise. It is a nonlinear digital filter technique and under certain conditions (small window size) it preserves edges while removing noise. Median filter is mainly used to remove shot noises and defective pixels in the image. Gaussian blur is used to smoothen the image. We use Gaussian kernel to blur the image. Gaussian blurring is highly effective in removing Gaussian noise from the image. We can also use bilateral filter but the operation is slow compare to other filters.

*Implementation*

OpenCV provides functions for both median filter and Gaussian blur. Gaussian blurring is highly effective in removing gaussian noise from the image. We must specify the kernel size for both the commands and standard deviation in x and y directions for Gaussian blur.

Blurred image = cv2.GaussianBlur(gray,(5,5),0.0)

The result obtained from the above filter operations is shown below



**Blurred Image**

The next step is to equalize histogram.Histogram Equalization is a computer image processing

technique used to improve contrast in images.It accomplishes this by effectivey spreading out the

most frequent intensity values,i.e., stretching out the intensity range of the image.This method

usually increases the global contrast of images when its usable data is represented by close

contrast values.This allows for areas of lower local contrast to gain a higher contrast.



**Histogram Equalized image**

We then perform Opening which is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function, **cv2.morphologyEx()**

cv2.morphologyEx(equal\_histogram, cv2.MORPH\_OPEN, kernel)

After performing opening we perform thresholding,It aims at identifying the the strong pixels,weak pixels and non-relevant pixels of the image.



The next step is to find the edges in the image. In order to perform edge detection canny edge detection method is used. This is done with the help of openCV function cv2.Canny(). Image

gradients and hysteresis thresholding are used to find edges. The output from the canny

operation is a binary image. The result from canny edge detection is shown below

Canny\_edge = cv2.Canny(img,lower,upper)



**Output of Canny Edge Operator**

Finally in the last step we apply dilation. Here, a pixel element is ‘1’ if atleast one pixel under the kernel is ‘1’. So it increases the white region in the image or size of foreground object increases It helps to highlight even more the area of the image where number plate is present.



**Dilated image**

**3.3LICENSE PLATE LOCALIZATION**

After preprocessing the image we proceed to image segmentation. This stage consists in trying to locate and extract possible number plate regions within the image. It is an important stage because if this stage fails, the whole ANPR system will fail. For this reason, it is very important to use the appropriate image processing techniques in order to create a robust image segmentation stage that prevents this from happening. The number plate is in rectangular shape, so in order to find the number plate we need to look for rectangles in the image that nearly matches the dimension of the number plate. This number plate contour search provides the ‘possible’ location

of the number plate within the image and thus, it enables its extraction from the input image.

*Implementation*

OpenCV library provides a command to find contours in the image, which is cv2.findContours().This function retrieves contours from binary images using an algorithm based on border following and it stores each of these contours as a vector of points. There are three arguments, first one is source image, second is contour retrieval mode, and third is contour approximation method.

Cv2.findContours(edge, cv2.RET\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

Once the contours found are stored, we will sort the contours based on their area in descending order. In order to save time we consider only the first twenty contours. This forms the limitation of the algorithm, if the number plates contour doesn’t appear in first twenty contours then the

algorithm fails which occurs rarely as most of the time the number plate contour has large area compared to most other contours. We can overcome this limitation by taking all the contours but this takes more time compared to alternate one. It’s a tradeoff between time and accuracy. We use openCV functions cv2.contourArea() and cv2.arcLength(cnt,True) to find area and perimeter

of the contours.

Area = cv2.contourArea(cnt)

Peri = cv2.arcLength(cnt, True)

In the above command true represents that the contour is closed shape. We proceed to our next steps with the twenty contours. We approximate the contour shape to another shape with less number of vertices depending upon the precision using openCV cv2.approxPolyDP() command.

It uses Douglas-Peucker algorithm.

Approx. = cv2.approxPolyDP(cnt, eplision, True)

First argument is the contour, second epsilon, which is maximum distance from contour

to approximated contour. It is an accuracy parameter. True represents that contour is closed shape.

We check the vertices of the approximated contour polygon and we consider only four vertex polygons. A dimensions check of each of them is carried out in order to be able to remove the ones that do not ﬁt into the dimensions of a number plate. This dimensions check is carried out

by find the width and height of the polygon by drawing a bounding box using openCV command

rect = cv2.minAreaRect(cnt)

Only those which pass the dimensionality test are checked further. In this project we used aspect ratio as a parameter to classily the contour as a number plate or not. More parameters

cv2.drawContours(image, contours, -1, (0,255,0), 3)

The first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness .The detected plate

region obtained from above procedure for a sample image is shown below.

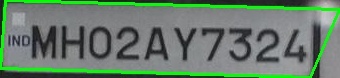


**Detected Plate Region**

**3.3.2 LICENSE PLATE SEGMENTATION**

The next stage of the developed ANPR system is number plate detection. In the previous stage we obtained the vertices of the bounding rectangle, using these vertices we save the number plate region into a new image file as “plate.jpg”.

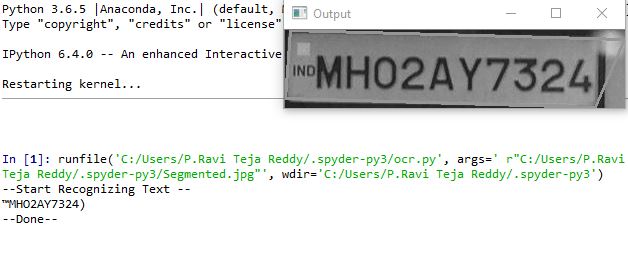
Our region of interest may be aligned in a different orientation which might not allow for the correct detection of the characters in the license plate. We use cv2.boundingRect() to find the 4 corners of the required contour .With these we segment the required contour from the original image.The transformed image is saved as plate.jpg which is rewritten. The figure below shows how perspective transform is used to align the plate region.



**Separated Plate Region**

**3.4 .Characters Recognition**

The last stage of the ALPR system is characters recognition. In this stage we try to successfully identify all the characters present in the plate image obtained from the previous stage. The process of recognizing characters from digital images is called Optical Character Recognition (OCR). In this project we used open source OCR library tesseract to recognize the characters.

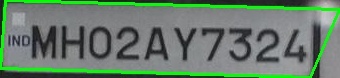


**Final result**

**4. RESULTS AND OBSERVATIONS**

The following images show test cases where the algorithms works well and where it fails. These

are just few

** **

** Segmented2**

**5. CONCLUSION**

In this project, an Automatic License Plate Recognition (ALPR) system that makes use of image processing and computer vision techniques has been developed. This system receives a car image, processes it and analyses it by means of several Computer Vision and, ﬁnally, it identiﬁes the number plate of the car appearing in the image. This system has achieved its objective. Nevertheless, some limitations have also been found in the developed ALPR system. The main one is in the detection of the plate region where we have assumed the contour to be in first twenty of the area based descending ordered contours. The performance of OCR can also be increased using Machine Learning based classifiers.

**REFERENCES**

[1] R.C. Gonzalez, R.E. Woods, Digital Image Processing, Addison-Wesley.

[2] OpenCV python tutorials https://docs.opencv.org/3.1.0/d6/d00/tutorial\_py\_root.html

[3]. Image Processing in OpenCV

https://docs.opencv.org/3.1.0/d2/d96/tutorial\_py\_table\_of\_contents\_imgproc.html

[4] Numpy array Object

https://scipy-lectures.org/intro/numpy/array\_object.html#creating-arrays

[5] Scikit learn Documentary

https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label

[6] An Improved License Plate Location Method Based On Edge Detection. Rongbao Chen, Yunfei Luo. 2012 International Conference on Applied Physics and Industrial Engineering

[7] Feng Yang, Zheng Ma “Vehicle License Plate location Based on Histogramming and

Mathematical Morphology “, 2005.

APPENDIX- A

-------------------------

FLOW CHART:

-------------------------

INPUT IMAGE

GRAY SCALE

GUASSIAN BLUR

HISTOGRAM EQUALIZATION

THRESHOLDING

CANNY EDGE DETECTION

LICENSE PLATE SEGMENTATION

LICENSE PLATE LOCALIZATION

OUTPUT TEXT

**APPENDIX-B**

localizePlate.py

####################################################

import cv2

import numpy as np

img = cv2.imread("Car\_Image\_6.jpg")

img\_gray = cv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)

noise\_removal = cv2.bilateralFilter(img\_gray,9,75,75)

cv2.imshow("Noise Removed Image",noise\_removal)

equal\_histogram = cv2.equalizeHist(noise\_removal)

cv2.imshow("After Histogram equalisation",equal\_histogram)

cv2.imwrite("C:/Users\Mahaveer\.spyder-py3\Histogram.jpg",equal\_histogram)

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT,(5,5))

morph\_image =

cv2.morphologyEx(equal\_histogram,cv2.MORPH\_OPEN,kernel,iterations=15)

sub\_morp\_image = cv2.subtract(equal\_histogram,morph\_image)

ret,thresh\_image = cv2.threshold(sub\_morp\_image,0,255,cv2.THRESH\_OTSU)

canny\_image = cv2.Canny(thresh\_image,250,255) cv2.imshow("Image after applying Canny",canny\_image) cv2.imwrite("C:/Users\Mahaveer\.spyder-py3\Canny.jpg",canny\_image); canny\_image = cv2.convertScaleAbs(canny\_image) kernel = np.ones((3,3), np.uint8)

dilated\_image = cv2.dilate(canny\_image,kernel,iterations=1)

cv2.imshow("Dilation", dilated\_image)

cv2.imwrite("C:/Users\Mahaveer\.spyder-py3\Dilation.jpg",dilated\_image); new,contours, hierarchy = cv2.findContours(dilated\_image, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

contours= sorted(contours, key = cv2.contourArea, reverse = True)[:10]

screenCnt = None

for c in contours:

peri = cv2.arcLength(c, True)

approx = cv2.approxPolyDP(c, 0.06 \* peri, True)

if len(approx) == 4:

screenCnt = approx

break

final = cv2.drawContours(img, [screenCnt], -1, (0, 255, 0), 2)

x,y,w,h = cv2.boundingRect(screenCnt)

crop\_img = final[y:y+h, x:x+w]

cv2.imwrite("C:/Users\Mahaveer\.spyder-py3\Segmented.jpg",crop\_img)

cv2.namedWindow("Image with Selected Contour",cv2.WINDOW\_NORMAL)

cv2.imshow("Image with Selected Contour",final)

cv2.imshow("Segment",crop\_img)

cv2.waitKey(0)

#############################################

OCR.py

#############################################

from PIL import Image

import pytesseract

import cv2

import os

image = cv2.imread('Segmented.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

cv2.imshow("Image", gray)

filename = "{}.jpg".format(os.getpid())

cv2.imwrite(filename, gray)

print("--Start Recognizing Text --")

text = pytesseract.image\_to\_string(Image.open(filename))

print(text)

print("--Done--")

cv2.imshow("Output", gray)

cv2.waitKey(0)

cv2.destroyAllWindows()