Automatic License/Number Plate Recognition (ANPR) using image processing

Group-41 | Project-40
Jhalak Gupta - 170001024
Ashwini Jha - 170001012

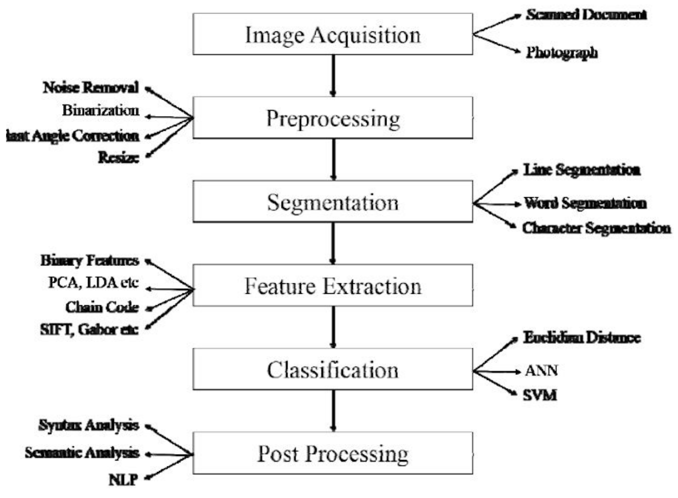
Under Supervision of
Dr. Surya Prakash
Associate Professor, Discipline of CSE

Introduction

The prime objective is to detect the vehicle number from the vehicle number plate with high accuracy using various image processing techniques. Image Processing and OCR are important parts of this project. Optical Character Recognition involves the detection of text content on images and translation of the images to encoded text that the computer can easily understand. An image containing text is scanned and analyzed in order to identify the characters in it. Upon identification, the character is converted to machine-encoded text.

Technologies Involved

- K-Nearest Neighbours (KNN)
- Optical Character Recognition (OCR)
- Tesseract
- OpenCV
- Image Processing



(Steps of OCR Process)

Background

Automatic number-plate recognition (ANPR) is a technology that uses optical character recognition on images to read vehicle registration plates to create vehicle location data. It can use existing closed-circuit television, road-rule enforcement cameras, or cameras specifically designed for the task. ANPR is used by police forces around the world for law enforcement purposes, including to check if a vehicle is registered or licensed. It is also used for electronic toll collection on pay-per-use roads and as a method of cataloguing the movements of traffic, for example by highways agencies.

Goals

1. Detect and localize a license plate in an input image/frame

2. Extract the characters from the license plate
3. Apply some form of Optical Character Recognition (OCR) to recognize the extracted characters
4. Do the same for videos as well

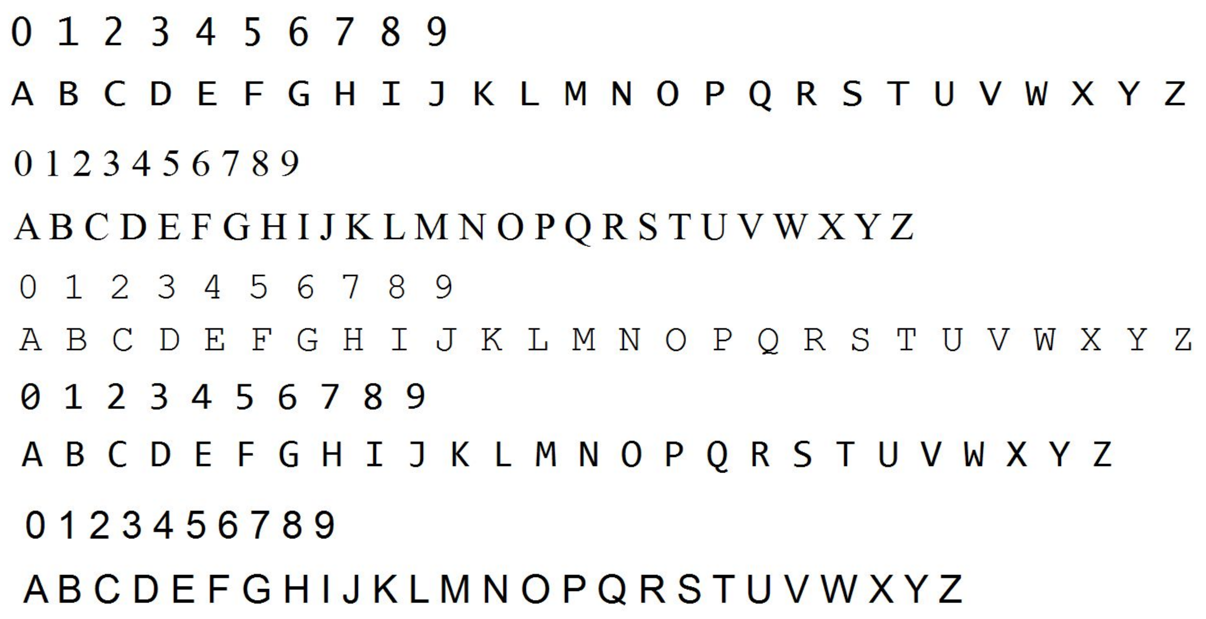
Dataset

The dataset contains of images of vehicles at various locations, angles, weather (including rain) and lighting conditions. We used the dataset provided in this Github repository: https://github.com/NanoNets/nanonets-ocr-sample-python/tree/master/images

Some of the images and videos of the vehicles are also stored in the code repository shared with this document.

All the images we used were of high quality (minimum of 480 pixels in height/width). The videos which we used were 1280-pixel * 720-pixel resolution.

We have used png images to train the KNN model to detect alphanumeric characters from the image.



0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Methodology / Work Done

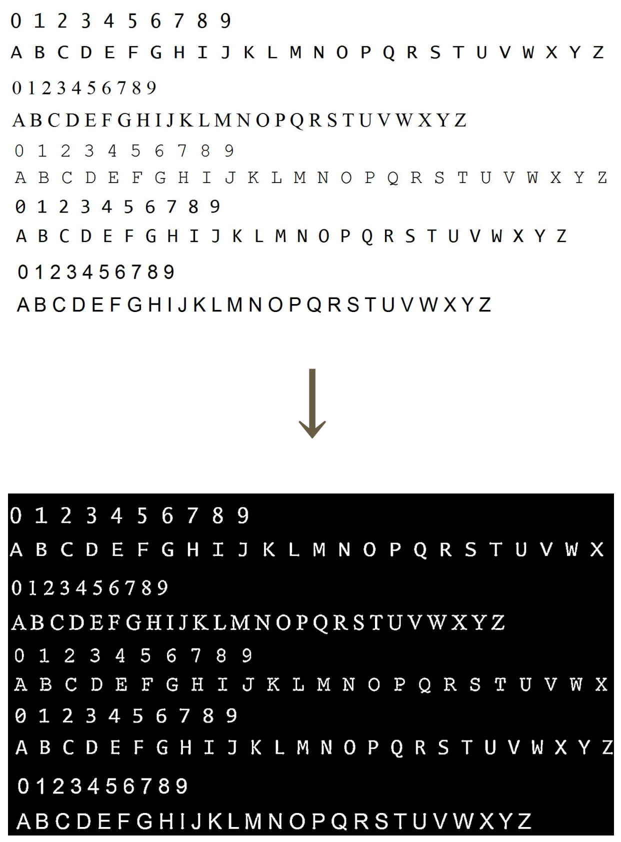
We will use various image processing techniques to detect and extract the number plate from the image of the vehicle and then separate the characters present inside the number plate. Then we use machine learning to identify the extracted characters and ultimately recognize the number plate.

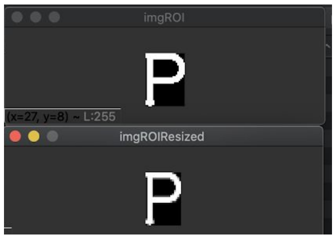
To detect the alphanumeric characters, there are multiple ways to do so. Tesseract OCR is a very popular tool developed by Google. We used it as well but applying Google's Tesseract resulted in low accurate digits recognition despite using Tesseract's options to recognize an image as a single text line and to OCR digits only. Adding to it, Tesseract was running considerably slower than our requirement as we required an OCR model to run at a good enough speed to detect number plates in videos as well.

So we implemented code to train a KNN model to detect alphanumeric characters which ran fast enough to give results at 6-10 frames per second for videos. Moreover, the accuracy was better than Tesseract. A comparison table is given in the "*Results and Analysis*" section.

Training KNN Model

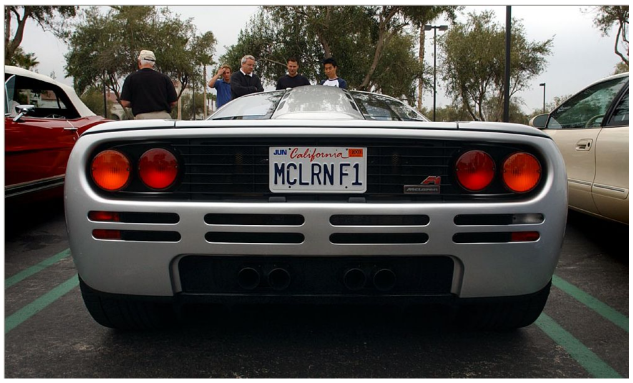
The dataset for training the model was a png image with alphanumeric characters. Here first the image was processed to single out the digits for labelling. Once we labelled all the data, training was completed.


0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

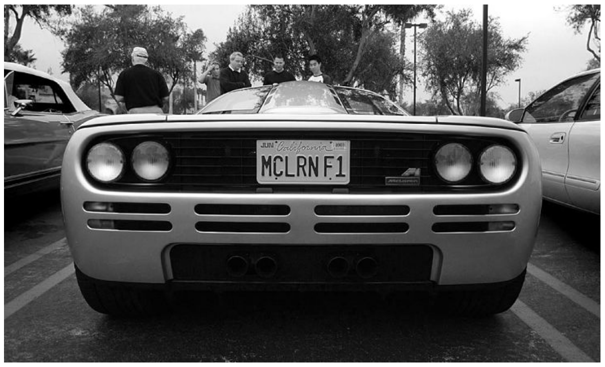


Labelled as P while training

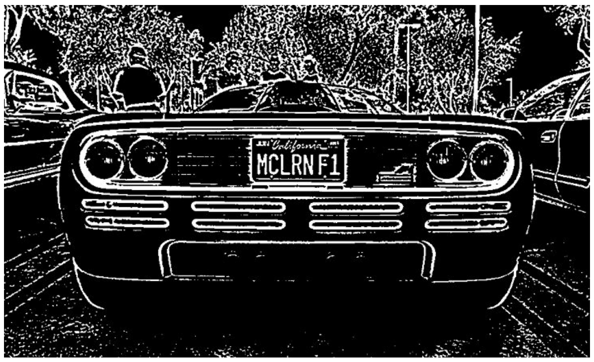
Localize License Plate in Input Image



(Input Image)

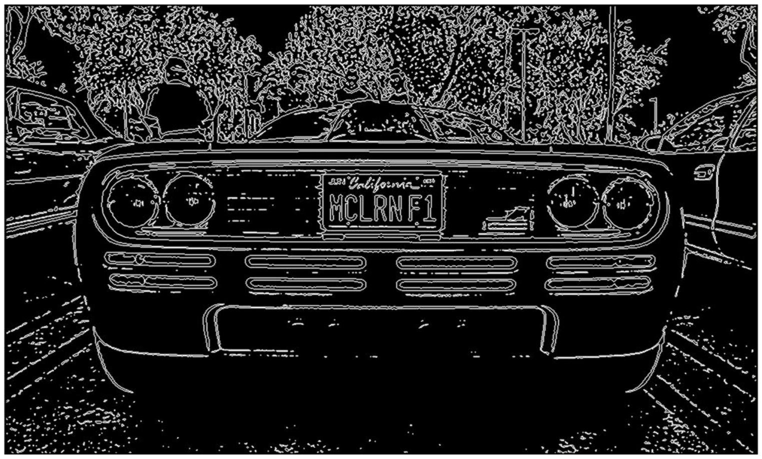


(Converted to Grayscale)



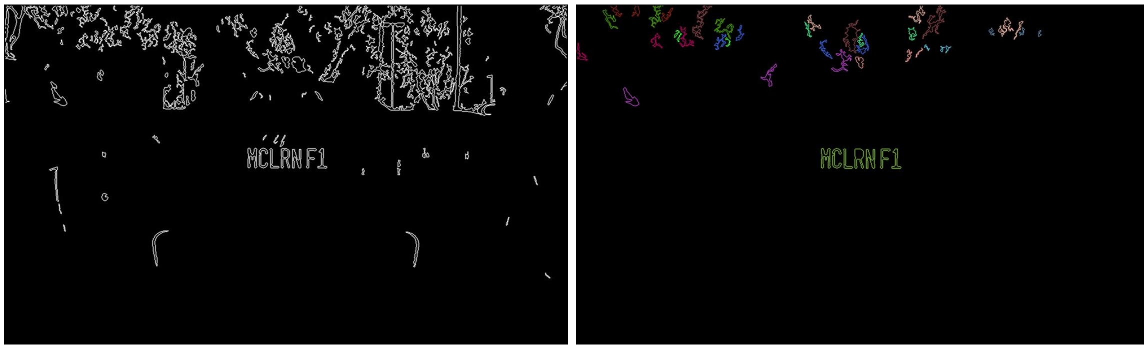
(Threshold)

Converting the image into grayscale facilitates better edge detection. We increase the contrast and add threshold too for creating a better distinction between the regions.

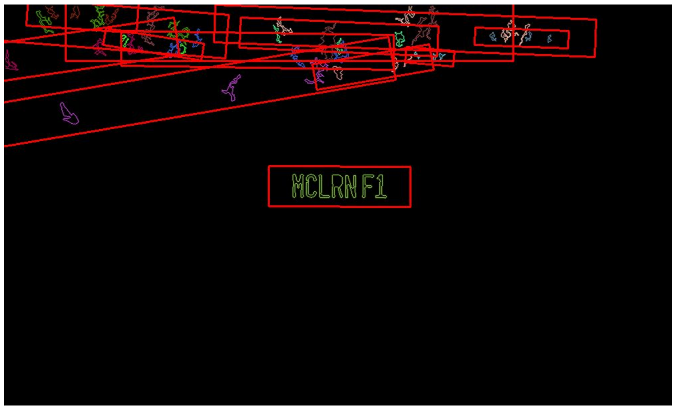


(Find Contours)

Edge Detection from the grayscale image: This makes it easier to plot contours and extract information from the contour structures.



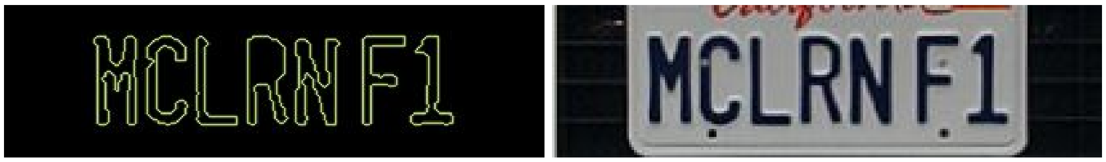
(Possible Characters)



(Possible Plates)

Area-based segmentation. Done by plotting the contours found in the image, then constructing box points around the contour area. We apply restrictions on this rectangle area to classify it as a possible number plate candidate or not.

$\downarrow$

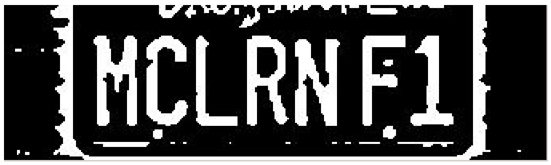


(Plate with the most number of characters of almost uniform size)

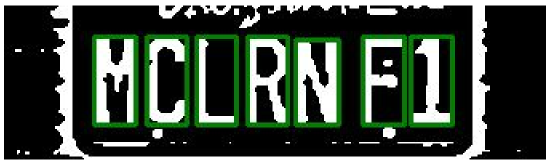
Extract Characters and Apply OCR

We convert the image to grayscale and apply a small Gaussian blur to smooth it out.

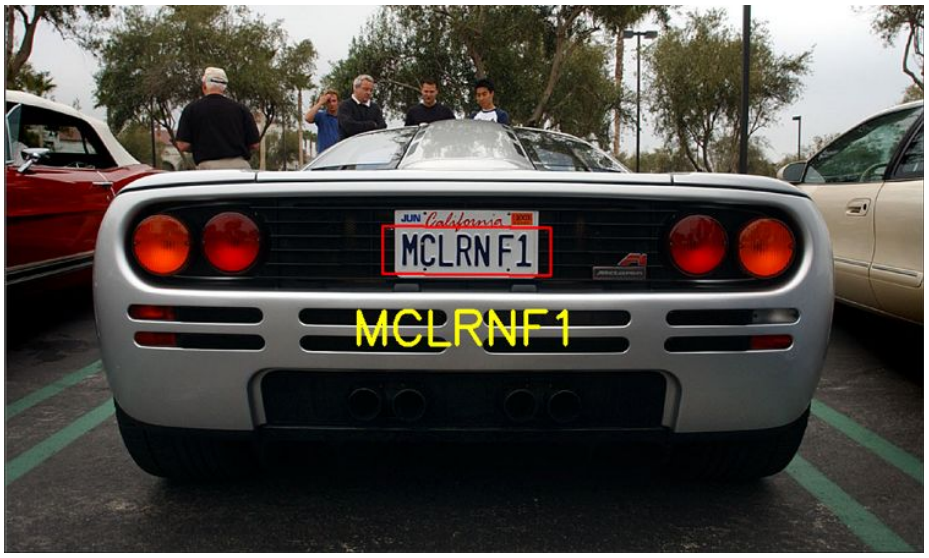
Following this, the image is thresholded to white text with a black background and has Otsu's method also applied. This white text on black background helps to find contours of the image. The image is then dilated using OpenCV in order to make contours more visible and be picked up in future step.



Next, we use OpenCV to find all the rectangular-shaped contours on the image and sort them left to right.



The individual characters of the license plate number are now the only regions of interest left. We segment each sub-image and use the trained KNN model to recognize the character. And finally, we get the following result:



MCLRNF1

Apply ANPR in Videos

To apply ANPR in videos, we used OpenCV to read the video frame by frame and used each frame to be processed as per the above algorithm.

Vehicle Data

As stated in the “Background” section, ANPR is used to detect stolen vehicles and vehicles crossing speed limit among many other purposes, we also tried to make something on the top of our algorithm. Vehicle data is not publicly and easily available by the government. We tried to scrap https://vahan.nic.in/nrservices/faces/user/searchstatus.xhtml using python. We even used OpenCV to solve the captcha successfully but some other regulations didn't allow us to process the website via code. So we just created a database in form of CSV file and used it to get the vehicle details for the vehicle number derived from the ANPR algorithm.

Result & Analysis

We trained KNN on just 130 training samples. Since numberplates on cars are not handwritten, so the test samples were pretty simple and hence with high-quality images, we got an accuracy of about 85%. To find this accuracy, we used OpenCV to read image files and the model to predict the characters.

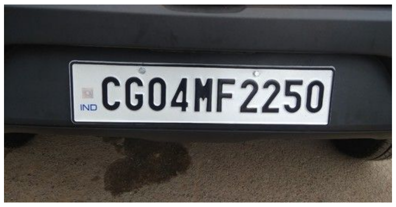
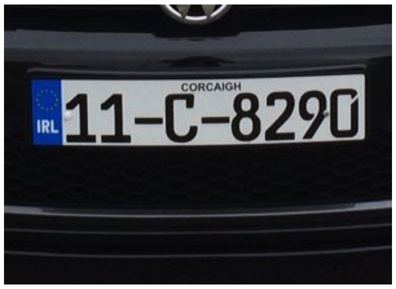
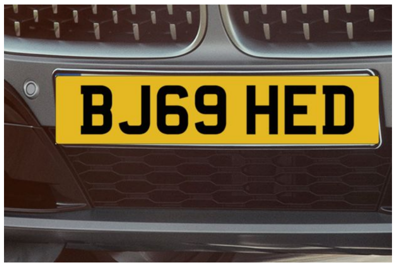
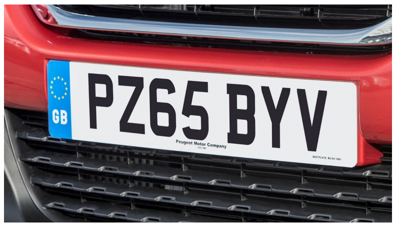
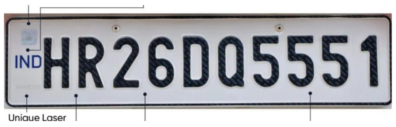
The following table shows the accuracy of our ANPR model:

Criteria	**Accuracy**
Number Plates Detected	92%
Number Plates Recognized	65%
Characters Recognized in Number Plates	85%

The rain or hot/cold climate did not show much effect on the model, but the angle at the which the image/video was captured does affect the accuracy. Following is the variation of the accuracy of the ANPR model based on angles of the vehicle to the perpendicular of the plane of the paper. The dataset to test this was very small (25 images) as it was quite difficult for us to manually find the area. The angles mentioned are also approximate ranges.

Angle (in degrees)	**Accuracy (approx, based on a small dataset)**
0 - 10	90
10 - 30	60
30+	40

We also implemented both KNN based OCR as well as Tesseract OCR for our code and compared the results. KNN performed much better than Tesseract OCR. The following table contains some examples for comparison.

Number Plate	**KNN based OCR**	**Tesseract OCR**
	CG04MF2250	CG04HF2250
	11 C 8290	11 C B29O
	BJ69HED	8J69HED
	PZ6SBYV	PZ65BVV
	HR26DQ5551	HR26O0551

The above comparison was done for the images. In the case of videos, a comparison was not possible as Tesseract took a lot of time per frame. Using KNN, we got a decent output of 6-10 frames per second, while Tesseract took around 1 second to render per frame.

Conclusion

From the test results, we conclude that the code works very well for all the scenarios mentioned (day, night, slightly inclined etc.) when the angle of the vehicle to the perpendicular of the plane of paper does not exceed 10 degrees approximately. After this angle, the accuracy and prediction fall drastically and is not reliable anymore. Also, the images and frames should be high resolution. For our work, KNN based OCR performed better than Google's Tesseract OCR.

References

- USING K-NEAREST NEIGHBOR IN OPTICAL CHARACTER RECOGNITION
- (PDF) Accuracy of automatic number plate recognition (ANPR) and real world UK number plate problems
- (PDF) Performance Evaluation of Automatic Number Plate Recognition on Android Smartphone Platform
- OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python
- NanoNets/nanonets-ocr-sample-python: NanoNets OCR API Example for Python