

**King Fahd University of Petroleum and Minerals**

*College of Engineering and Physics*

*Control and Instrumentation Engineering Department*

**CISE - 483**

**Final Project Report**

**“***License Plate Recognition System***”**

**November 27, 2021**

**Group members:**

1. **Mohammed Alghamdi 201354170**
2. **Zed Ali 201757290**
3. **Ahmed Balhareth 201648540**

**Instructor:** *Khaled Alshehri*

**Table of Contents**

[**I. Abstract 2**](#_Toc88878100)

[**II. Methodology 2**](#_Toc88878101)

[Plate Recognition 2](#_Toc88878102)

[Heuristic Method 2](#_Toc88878103)

[Edge Detection 4](#_Toc88878104)

[Character Segmentation 5](#_Toc88878105)

[Heuristic Method (Extended): 6](#_Toc88878106)

[Using Tesseract (Open Source OCR) 6](#_Toc88878107)

[Character Recognition 7](#_Toc88878108)

[Support Vector Machine Classifier 7](#_Toc88878109)

[**III. Code Index 9**](#_Toc88878110)

[Libraries 9](#_Toc88878111)

[Data Loading 10](#_Toc88878112)

[Classifier Training 11](#_Toc88878113)

[Classifier Evaluation 11](#_Toc88878114)

[Model Persistence 12](#_Toc88878115)

[Image Preparation 13](#_Toc88878116)

[Plate Detection 13](#_Toc88878117)

[Character Segmentation 14](#_Toc88878118)

[Inference 16](#_Toc88878119)

[**IV. Conclusion 17**](#_Toc88878120)

With this project we aim to create a License Plate Recognition System which can identify the License Plate number of the vehicles on the road. This system can be expanded on further and be used for various other systems e.g. a parking system, vehicle tracking in a metropolitan city etc.

# Abstract

We live in a world where vehicles are a crucial pillar of society and thus they have been attached a special identification number which can be used to achieve a myriad number of goals. This project is part of our course in which we aim to recognize license plates. The input to our software is an image of a car and the output is the car’s license plate number in a text form.

# Methodology

Although a complicated system like this can be dealt with in an infinite number of ways, we chose to simplify it into 3 simple steps (simple in its wording but complex in implementation!): Plate detection, character extraction and character recognition. All of these steps will be pipelined in our final code. In a more elaborated way we are going to recognize the number plate in an image on a car. After that we will perform segmentation. A way to separate the whole license plate into separate images of individual characters. Once we get an array of images in the specific order, we will simply run each image through our classifier of choice and detect the characters. We will discuss our methodology further.

## Plate Recognition

The general idea is to perform feature extraction from the input image so that we can reduce the amount of data present in the image. We can use several methods for this like region segmentation, fuzzy sets, edge features, etc. We have used two different methods to perform recognition i.e. Heuristic and Edge Detection method.

### Heuristic Method

In the heuristic method, a few assumptions are made to extract the image containing the number plate. These assumptions are made while considering input pictures and the camera positioning.

##### Assumptions

1. The height of the plate is 5 to 20 percent of the image height.

2. The width of the plate is 15 to 60 percent of the image width.

3. Plate height is greater than 20 percent of plate width.

##### Steps

1. Take the image as input.

The front of a car

Description automatically generated with low confidence

2. Converts the image to grayscale

3. Converts grayscale to binary using OTSU threshold method.

3. Label all the connected components in the binary image.

4. Selected corresponding bounding-boxes

5. Select the listed bounding boxes

The license plate of a car

Description automatically generated with medium confidence

### Edge Detection

The basic idea is to identify the number plate and then use this number plate image with an OCR to get the numbers and alphabets in the license plate.

1. First, we take the given image containing the license plate of the car and read it as a NumPy array. Now since the license plate number remains unaffected whether the image is colored or black and white, we convert images to grayscale.

The front of a car

Description automatically generated with medium confidence

1. The advantage of the image to grayscale is that it has only 1 layer whereas the colored image has 3 layers, hence the processing speed is greatly increased by converting the image to grayscale.
2. The next step is to remove noise from the image not affected by this step so it is better to remove the noise especially from the background. Fig: Removing unwanted noise from the image.

The front of a car

Description automatically generated with medium confidence

1. Now we perform the edge detection as the edges are sufficient to identify the number plate and the text (numbers and alphabets) inside it. The whiteness of the number plate nor the blackness (color) of the car is not important for us so we run filters horizontally and vertically to get the image that has only edges.

A close up of a car

Description automatically generated with low confidence

1. After getting the edges we perform contour identification. Contours are curves joining continuous points having the same color/intensity. This contour is generated using the edge detected image of the car and then We pass a copy of the image to the contour identification as the function alters the image.
2. After identifying all the contours, we know that one of the contours is the number plate. We also know that the contour corresponding to the number plate is one of the largest in terms of area and perimeter. Another characteristic of the number plate is that it will always be a polygon of four sides (irrelative of different angles, scale, and location of the number plate).

A picture containing text, car

Description automatically generated

1. Hence, we can infer that we can get the number by sorting the contours by area in decreasing order and taking the contours with 4 closed edges and drawing the rectangular bounding box around it. This gives us the number plate.

## Character Segmentation

After the license plate is recognized, we need to perform character segmentation. In character segmentation, we identify each character present on the license number plate and later these characters are used for character recognition. For character segmentation, there are various methods available to do this like using contours, projection, transforms, etc. We have done the following:

### Heuristic Method (Extended):

A license plate on a car

Description automatically generated with medium confidence

##### Assumptions

1. The plate contains only 10 characters.
2. The height of each character is within 35%-90% of the total plate height.
3. The width of each character is within 2%-10% of the total plate width.

##### Steps

1. Labels all the connected components inside the plate area selected before. It then selects a bounding box to represent the connected components.
2. These selected boxes are checked against the assumption made for characters.
3. In the end, assumption number 1 is checked, and the corresponding segmented set of characters is returned.

A picture containing text, slot machine, close

Description automatically generated

### Using Tesseract (Open-Source OCR)

Tesseract is an open-source OCR (optical character recognition) that can work on more than 100 languages and find the text from the input image. It was the first OCR engine to be able to handle black and white texts efficiently.

The Tesseract Pipeline:

1. First, it performs connected component analysis in which it stores the outlines of the connected components.
2. Next, it creates Blobs by gathering the outlines together.
3. These blobs are organized in the form of text lines and then analyzed for a fixed proportion of text. This proportional text is then divided into words.
4. Next, the recognition of characters takes place using an adaptive classifier.

All this is wrapped into the python library “pytesseract” which is imported when we want to use the Tesseract OCR. Once we have our license plate recognized, extraction of license plate number from the plate image is done using pytesseract image\_to\_string() function.

## Character Recognition

After we perform character segmentation, we need to recognize each of the segmented characters. In other words, we need to create an OCR (Optical Character Recognition). For this, we can use either a statistical classifier or a computational classifier. Statistical classifiers include Hidden Markov Models and Support Vector Machines. Computational classifiers include Self-organized neural networks and probabilistic neural networks. We used the following:

### Support Vector Machine Classifier

The main objective of the support vector machine algorithm is to find a hyperplane in N-dimensional space (N — the number of features) that uniquely classifies the information points into different predefined groups or segments. To separate the 2 classes of information points, there are many possible hyperplanes that would be chosen. Our objective is to seek out a plane that has the most margin, i.e., the most distance between data points of both classes. Maximizing the margin distance provides some reinforcement in order that future data points are often classified with more confidence. There are cases when data points are not always linearly separable. In those cases, SVM uses different kernel modes that measure the closeness (or closeness) of the data points in order to make them linearly separable.

We use a support vector machine classifier to classify each alphanumeric character (characters obtained in the character segmentation step). Support Vector Machine is a linear classifier. To perform a non-linear classification kernel trick is used. Support Vector Machine is used for binary classification to distinguish each alphanumeric character. We get the binary image of each alphanumeric character by using the Otsu threshold value. The training set consists of 20pX20p images of all alphanumeric characters. We are using a sci-kit learn package for implementing our model and cross-validation with four folds (that is 3/4th of the dataset is used for training and 1/4th is used for testing). Once the model is trained it is saved for our later use i.e. to make predictions on the character segmented image of license plate.

# Challenges faced

By using the three plate recognition methods, it seems impossible to recognize Saudi Arabia Plates. One of the tested Saudi Arabia Plates was N U J 1236, which is owned by our partner Ahmed; the reason behind this issue could be having Arabic Letters on the same plate, which confuses and affects the recognition process done by the system. Therefore, we decided to crop the Arabic numbers and letters; in order to avoid the interference while testing that image of plate. See the following figure:

Graphical user interface, text, application, email

Description automatically generated

# Code Index

For our project we used **python 3.8.8**. The development of this project was all done on a ***Jupyter notebook***.

## Libraries

import os

import numpy as np

from skimage.io import imread

from skimage.filters import threshold\_otsu

from skimage import measure

from skimage.transform import resize

import matplotlib.patches as patches

import numpy as np

import joblib

from sklearn.decomposition import PCA

from sklearn.svm import SVC

import pickle

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import accuracy\_score

from operator import itemgetter

import zipfile

import matplotlib.pyplot as plt

## Data Loading

training\_20X20\_dir = os.path.join('C:\\Users\\balha\\Desktop\\ CISE-483-Final Report \\training\_data', 'train20X20')

training\_10X20\_dir = os.path.join('C:\\Users\\balha\\Desktop\\ CISE-483-Final Report \\training\_data', 'train10X20')

letters = [

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D',

'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T',

'U', 'V', 'W', 'X', 'Y', 'Z'

]

image\_data = []

target\_data = []

for each\_letter in letters:

for each in range(10):

#training\_data[1] is for 10X20 training data images

img\_details = imread(training\_20X20\_dir+'/'+each\_letter+'/'+each\_letter+'\_'+str(each)+'.jpg', as\_gray=True)

binary\_image = img\_details < threshold\_otsu(img\_details)

flat\_bin\_image = binary\_image.reshape(-1)

image\_data.append(flat\_bin\_image)

target\_data.append(each\_letter)

image\_data, target\_data = (np.array(image\_data), np.array(target\_data))

## Classifier Training

# kernel can be linear, rbf e.t.c

svc\_model = SVC(kernel='linear', probability=True)

svc\_model.fit(image\_data, target\_data)

## Classifier Evaluation

wrong\_predictions = True

img\_train, img\_test, target\_train, target\_test = train\_test\_split(image\_data, target\_data)

svc\_model.fit(img\_train, target\_train)

prediction = svc\_model.predict(img\_test)

accuracy = (float(np.sum(prediction == target\_test)) / len(target\_test))

print(str(round(accuracy \* 100, 2))+ "% accuracy was recorded")

def top\_predictions(probabilities\_prediction):

predictions = probabilities\_prediction.reshape(-1).tolist()

predictions\_label = []

for index in range(len(predictions)):

predictions\_label.append((letters[index], predictions[index]))

predictions\_label = sorted(predictions\_label, key=itemgetter(1), reverse=True)

print(predictions\_label[:5])

if wrong\_predictions:

print('Here are the wrong predictions')

print('Prediction\tCorrect Label')

print('------------------------------')

for i in range(len(prediction)):

if prediction[i] != target\_test[i]:

probabilities = svc\_model.predict\_proba(img\_test[i].reshape(1, -1))

print('Predicted: '+prediction[i]+'\t\t Actual:'+target\_test[i])

print('Probability Distribution')

top\_predictions(probabilities)

print('------------------------')

print('------------------------------')

# self.print\_wrong\_predictions(prediction, target\_test, img\_test, model)

## Model Persistence

filename = "model.sav"

pickle.dump(svc\_model, open(filename, 'wb'))

print("model saved")

## Image Preparation

image\_path = "test\_image/car4.jpg"

car\_image = imread(image\_path, as\_gray=True)\*255

threshold\_value = threshold\_otsu(car\_image)

binary\_car\_image = car\_image > threshold\_value

fig, (axis, axis1) = plt.subplots(2, 1)

## Plate Detection

label\_image = measure.label(binary\_car\_image)

image\_height, image\_width=label\_image.shape

plate\_dim=(0.05\*image\_height,0.2\*image\_height,0.15\*image\_width,0.6\*image\_width)

## Plot car image

axis.imshow(car\_image,cmap="gray")

lp\_cands=[]

lp\_cand\_dimension=[]

for region in measure.regionprops(label\_image):

minRow, minCol, maxRow, maxCol = region.bbox

(region\_height,region\_width)=(maxRow-minRow,maxCol-minCol)

if(region.area < 50 or region\_height<0.2\*region\_width ):

continue

candidate=np.invert(binary\_car\_image[minRow:maxRow,minCol:maxCol])

if(region\_height>=plate\_dim[0] and region\_height <=plate\_dim[1] and region\_width>=plate\_dim[2] and region\_width<= plate\_dim[3]):

r, c=candidate.shape

if np.sum(candidate) > 0.3\*r\*c:

continue

rectBorder = patches.Rectangle((minCol, minRow), maxCol-minCol, maxRow-minRow, edgecolor="red", linewidth=2, fill=False)

lp\_cands.append(candidate)

lp\_cand\_dimension.append(((minRow,minCol),(maxRow-minRow,maxCol-minCol)))

# plotting.add\_borders(rectBorder,fig,axis)

## Add borders to license plate

axis.add\_patch(rectBorder)

## Character Segmentation

segmented\_characters=[]

idx=0

for idx in range(len(lp\_cands)):

cand = lp\_cands[idx]

# plotting.plot\_car\_image(cand, fig, axis1)

axis1.imshow(cand,cmap="gray")

char\_dim = (0.30\*cand.shape[0], 0.90\*cand.shape[0], 0.02\*cand.shape[1], 0.1\*cand.shape[1])

labelled\_cand = measure.label(cand)

cnt=0

border=[]

temp\_chars=[]

for region in measure.regionprops(labelled\_cand):

minRow, minCol, maxRow, maxCol = region.bbox

(region\_height,region\_width)=(maxRow-minRow,maxCol-minCol)

if(maxRow==lp\_cand\_dimension[idx][1][0]):

continue

#print(region\_height,region\_width)

if(region\_height>=char\_dim[0] and region\_height <=char\_dim[1] and region\_width>=char\_dim[2] and region\_width<= char\_dim[3]):

rectBorder = patches.Rectangle((minCol, minRow), maxCol-minCol, maxRow-minRow, edgecolor="red", linewidth=2, fill=False)

border.append(rectBorder)

temp\_chars.append((minRow,maxRow,minCol,maxCol))

# plotting.add\_borders(rectBorder, fig, axis1)

axis1.add\_patch(rectBorder)

if(len(border)==10):

for borders in border:

# plotting.add\_borders(borders, fig, axis1)

axis1.add\_patch(borders)

dim= lp\_cand\_dimension[idx]

for val in temp\_chars:

r1=dim[0][0]+val[0]

r2=dim[0][0]+val[1]

c1=dim[0][1]+val[2]

c2=dim[0][1]+val[3]

segmented\_characters.append((val[2],resize(np.invert(binary\_car\_image[r1:r2,c1:c2]),(20,20))))

## Show everything

plt.show()

## Inference

ans=[]

for char in segmented\_characters:

# print(plt.imshow(char[1]))

ans.append(svc\_model.predict(char[1].reshape(1,-1)))

license\_plate= []

for val in ans:

license\_plate.append(val[0])

print("Detected License Plate Number:", "".join(license\_plate))

# Conclusion

There are many ways to develop a License Plate Recognition. Each method requires different algorithms at different stages (stages refers to Plate Recognition, Character Segmentation, and Character Recognition). Each has its own pros and cons.

In this project we have developed LPR using two different methods:

1. In the first method, we have used the above mentioned edge feature detection for plate recognition and Tesseract for character segmentation and recognition. Since Tesseract is a State of the Art OCR the results obtained using this method were very good. For all car images in which license plates are readable this method returns the license plate number.
2. In the second method, we have used the above mentioned Heuristic method for plate recognition and character segmentation and SVM classifiers to recognize each segmented character. Since this heuristic method requires car images to fulfill few criteria therefore this method worked on only a selective type of image. But the results obtained from this method even for the images in which the license plate was small were very good.

• ∼ **End of Final Report** ∼ •