DIGITAL IMAGE PROCESSING

LICENSE PLATE RECOGNITION

Submitted By -

Aman Jaswani (2K16/SE/008)

Raghuvansh Raj (2K16/MCE/062)

Contents

1. Introduction
2. Abstract
3. K Nearest Neighbours
4. Proposed System
    a. System Overview
    b. Algorithm
    c. Pre processing
    d. Localization
    e. KNN based recognition
5. Snapshots
6. Code

Introduction

License Plate Recognition Systems use the concept of optical character recognition to read the characters on a vehicle license plate. In other words, LPR takes the image of a vehicle as the input and outputs the characters written on its license plate.

LPR sometimes called ALPR (Automatic License Plate Recognition) has 3 major stages.

1. License Plate Detection: This is the first and probably the most important stage of the system. It is at this stage that the position of the license plate is determined. The input at this stage is an image of the vehicle and the output is the license plate.
2. Character Segmentation: It's at this stage the characters on the license plate are mapped out and segmented into individual images.
3. Character Recognition: This is where we wrap things up. The characters earlier segmented are identified here. We'll be using machine learning for this.

Abstract

In the traffic control and security management framework, License plate recognition methodology plays a vital role, which manages more responsibility for high security. Identifying the vehicle's number plate is a complex task, because of the existence of noise and differing illumination and angles. For this purpose, in our work, Automated Vehicle License Plate Detection using KNN method is done, which identifies the license plate accurately. Here, pre-processing is the initial step, which is done with the help of contrast enhancement and Gaussian blur approach. After pre-processing, the system will extract the license plate, from the image according to the characteristics of license. The license numbers were recognized from the extracted license plate with the help of character segmentation approach, further it is learned and recognized accurately by making use of the machine learning technique, which is termed as KNN classifier.

K-Nearest Neighbour

One of the useful supervised based learning and a non-parametric techniques is K-Nearest Neighbour or KNN algorithm. The output result of the algorithm depends on K- nearest neighbour category which implemented by finding K- number of training points closest to the required character and consider the votes among the K object. The algorithm is very simple. However, is capable of learning highly-complex non-linear decision boundaries and regression functions. The intuition of KNN that similar instances should have similar class labels (in classification) or similar target values (regression). On the downside, the algorithm is computationally expensive, and is prone to overfitting.

PROPOSED SYSTEM

System Overview

Given an image, using the procedure of finding and detecting the license plate is based on the following steps:

Step 1: Convert the original image that taken by camera to grayscale image.

Step 2: Convert grayscale image as result of step 1 to binary image and apply contrast enhancement using OpenCV library to easy detect LP location.

Step 3: Apply Gaussian blur filter to smooth the image.

Step 4: List all contour in image. The contour with maximum number of characters (detected as maximum area of similar regions) is assumed to be the License Plate

Step 5 Using KNN algorithm to list all matching character in image.

Step 6 List all possible plate that can be found in image.

Algorithm: Finding License Plate Using K-NN

```
1: Begin
2: Input: Original Image
3: Output: Characters
4: Method: K-Nearest Neighbours
5: LP: License Plate
6: Convert RGB image to Grayscale
7: Filter Morphological Transformation
8: Transform Grayscale image to binary image and apply contrast enhancement
9: Filter Gaussian for Blurs image
10: Finding all contours in image
11: Search & recognize all possible character in image using KNN
12: Crop part of image with highest candidate LP
13: Crop the LP from original image
14: Apply steps from 6 to 11 again on crop image
15: Print the characters in LP
16: End
```

1) Pre-processes Involved

- **Maximize Contrast** - to maximize the image contrast on the basis of the intensity histogram.

- **Gaussian blur** (also known as **Gaussian smoothing**) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2) License Plate Localization

Find contours in the image. For each contour:

- Find bounding box of the contour
- Find the area of the bounding box
- Detect the maximum area of similar region contour. This area is assumed to be the License Plate
- Store the bounding box and the corresponding image

3) KNN based character recognition

Character recognition is the final stage in vehicle license plate detection and recognition, where it reads an individual characters and numbers. Single elements on license plate should be classified and examined. This examination is termed as Optical Character Recognition (OCR) using KNN.

The k-nearest neighbour 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 it’s 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.

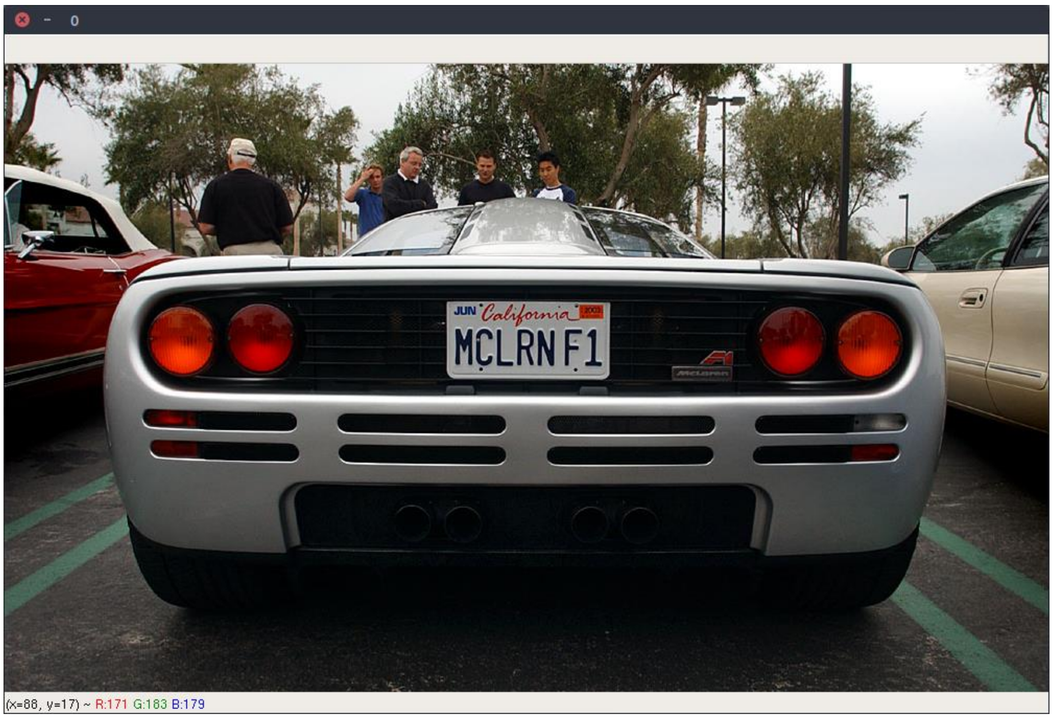
Need to allot the weight to the contributions of the neighbors, both for classification and regression process, so that the nearer neighbors contribute more to the average than the more distant ones. For instance: a common weighting scheme where every neighbor has a weight of 1/d, where d is the distance 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 does not require an explicit training step. A peculiarity of the k-NN algorithm is very sensitive to the local structure of the data. The algorithm doesn’t get mixed or confused with k-means, moreover, the algorithm of KNN classifier make use of backward elimination, which is explained further:

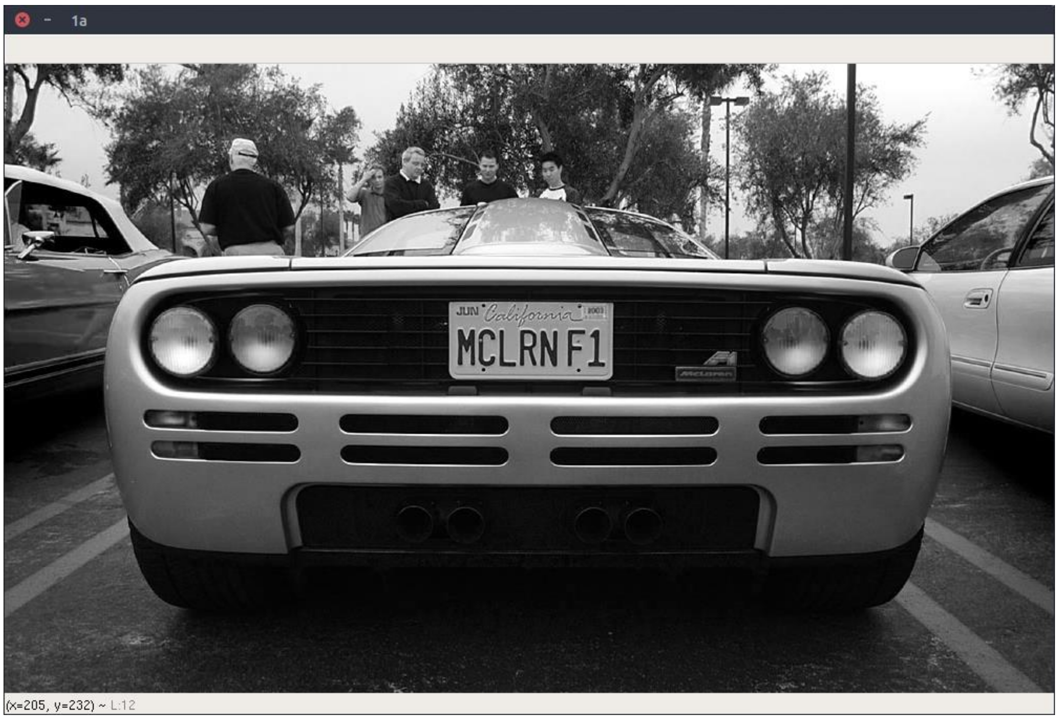
```
- Read the training data from a file
- Read the testing data from a file
- Set K to some value Normalize the attribute values in the range 0 to 1.
        Value = Value / (1+Value);

//Apply Backward Elimination
- For each testing example in the testing data set
- Find the K nearest neighbors in the training data set based on the Euclidean distance
- Predict the class value by finding the maximum class represented in the K nearest
neighbors
- Calculate the accuracy as
        Accuracy = (# of correctly classified examples / # of testing examples) X 100
```

SNAPSHOTS

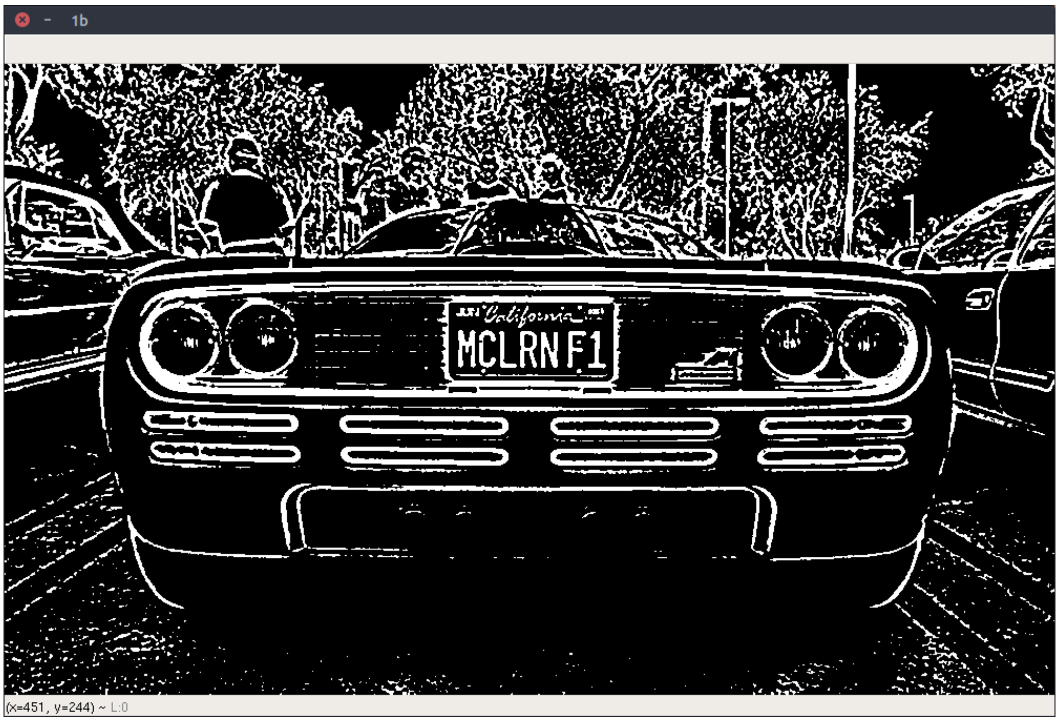
1. Pre-processing
   - Load image from the project directory



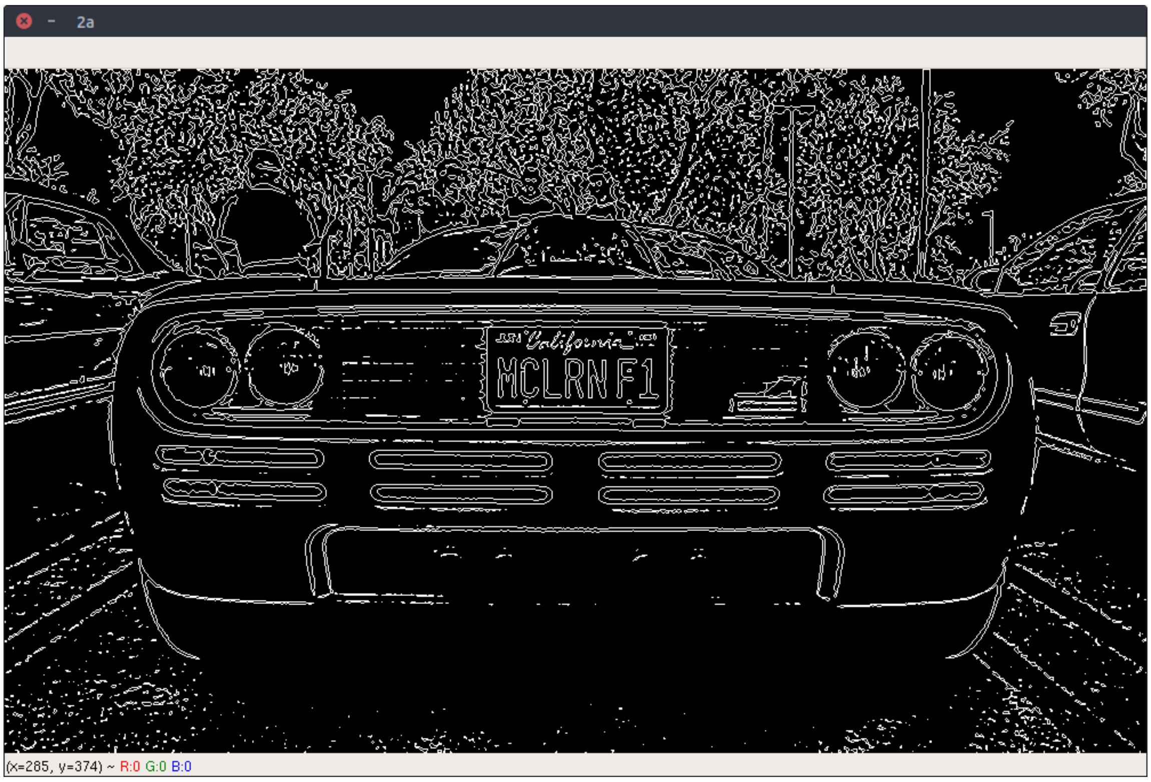
   - Convert image to grayscale and perform contrast enhancement



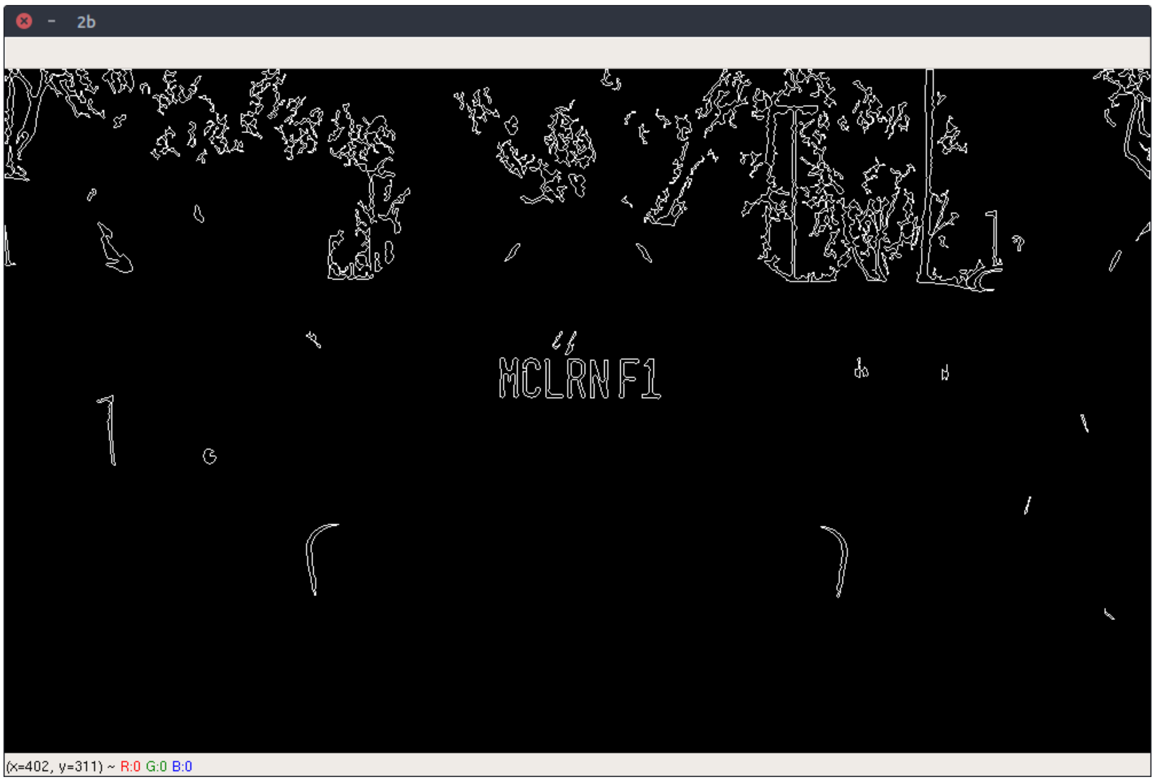
- Apply Gaussian Blur to the grayscale image using a filter size of (5x5)
- Threshold the grayscale image using adaptive thresholding with a block size of 19 and a weight of 9



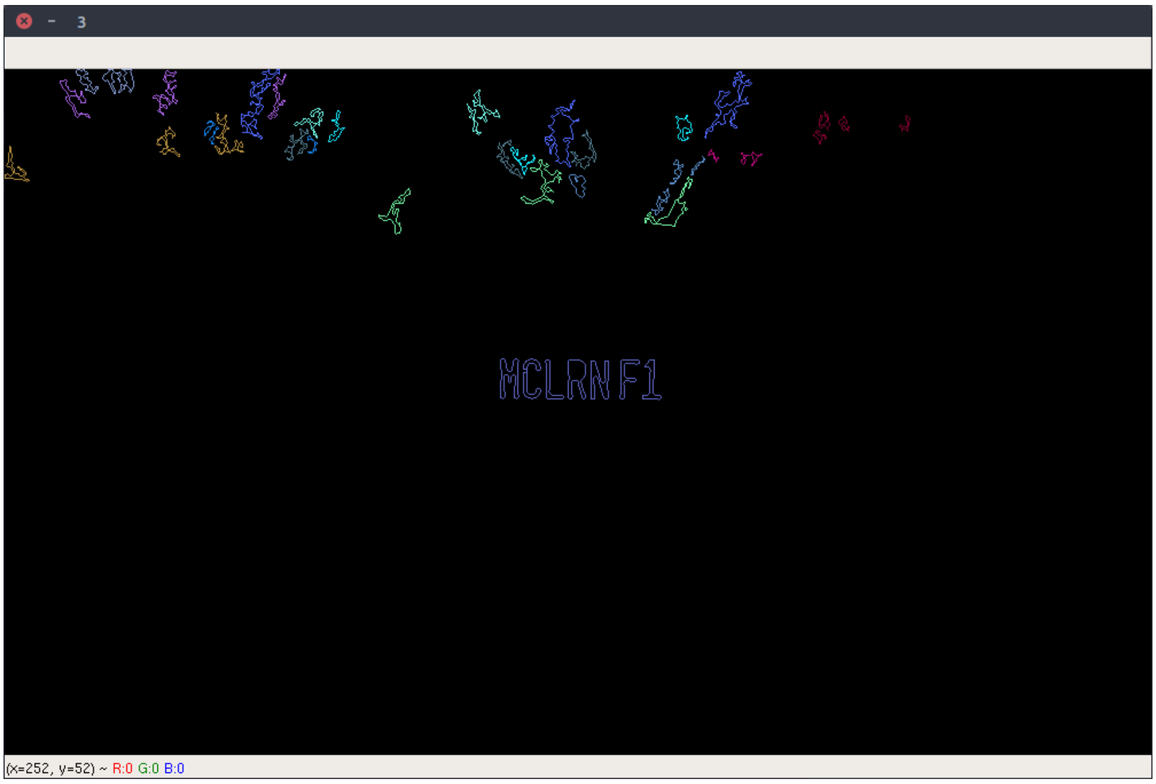
2. Detect contours in the final thresholded image



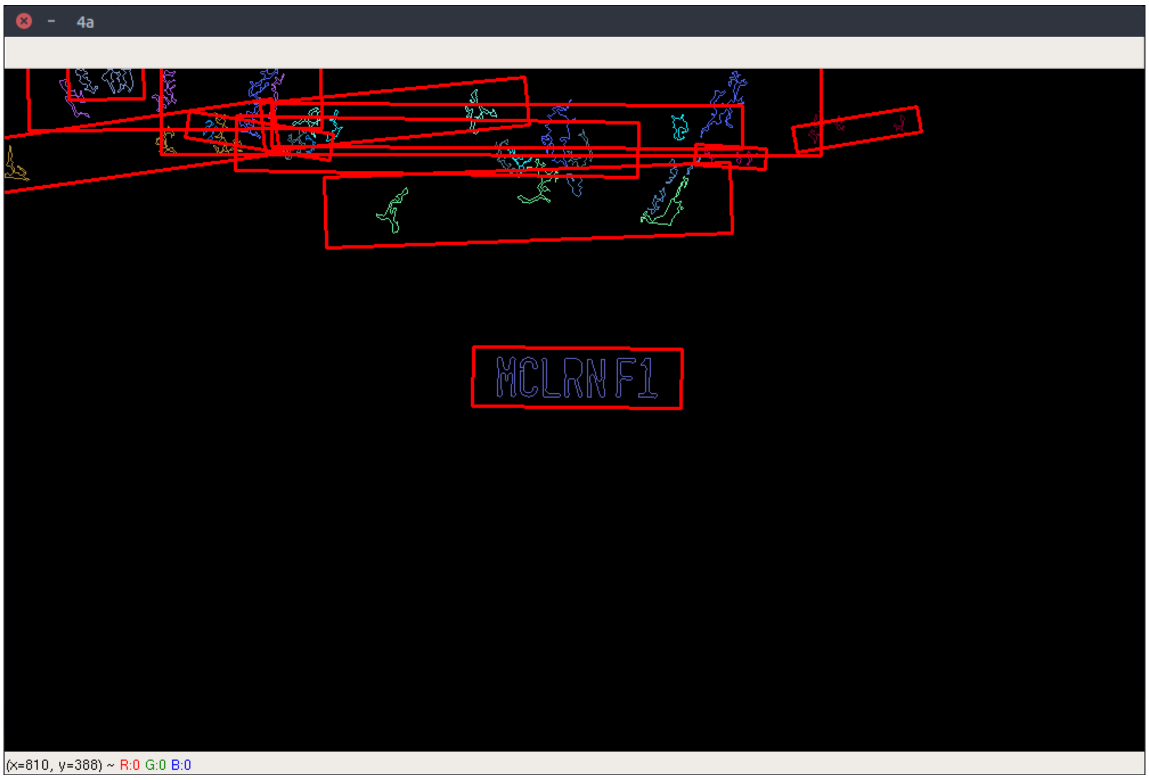
3. Extract contours which could be possible characters



4. Group those contours together which seem connected using hardcoded parameters like distance, change in area, angle between characters, etc.

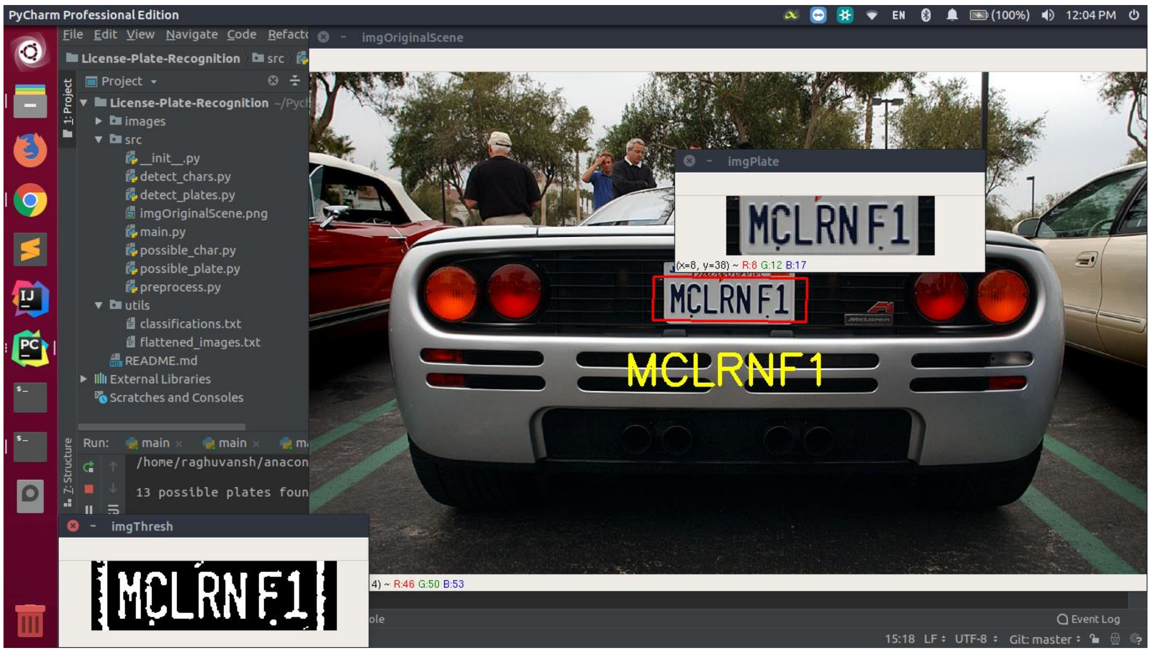


5. Generate bounding boxes for matching contours



```
(x=810, y=388) ~ R:0 G:0 B:0
```

6. Extract the box with the maximum number of characters, assumed to be the license plate



```
/home/raghuvansh/anacon
13 possible plates foun
```

```
(x=8, y=38) ~ R:8 G:12 B:17
4) ~ R:46 G:50 B:53
```

CODE

```
import cv2
import os

from src import detect_chars
from src import detect_plates

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)

CURR_IMG_DIR = '/home/raghuvansh/PycharmProjects/License-Plate-Recognition/images/'

showSteps = False


def main():
    blnKNNTrainingSuccessful = detect_chars.loadKNNDataAndTrainKNN()

    if blnKNNTrainingSuccessful == False:
        print("\nerror: KNN traning was not successful\n")
        return

    imgOriginalScene = cv2.imread(CURR_IMG_DIR + "1.png")

    if imgOriginalScene is None:
        print("\nerror: image not read from file \n\n")
        os.system("pause")
        return

    listOfPossiblePlates = detect_plates.detectPlatesInScene(imgOriginalScene)

    listOfPossiblePlates = detect_chars.detectCharsInPlates(listOfPossiblePlates)

    cv2.imshow("imgOriginalScene", imgOriginalScene)

    if len(listOfPossiblePlates) == 0:
        print("\nno license plates were detected\n")
```

```python
    else:

        listOfPossiblePlates.sort(key=lambda possiblePlate: len(possiblePlate.strChars), reverse=True)

        licPlate = listOfPossiblePlates[0]

        cv2.imshow("imgPlate", licPlate.imgPlate)
        cv2.imshow("imgThresh", licPlate.imgThresh)

        if len(licPlate.strChars) == 0:
            print("\nno characters were detected\n\n")
            return

        drawRedRectangleAroundPlate(imgOriginalScene, licPlate)

        print("\nlicense plate read from image = " + licPlate.strChars + "\n")
        print("----------------------------------------")

        writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)

        cv2.imshow("imgOriginalScene", imgOriginalScene)

        cv2.imwrite("imgOriginalScene.png", imgOriginalScene)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return

def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):
    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)

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


def writeLicensePlateCharsOnImage(imgOriginalScene, licPlate):
    ptCenterOfTextAreaX = 0
    ptCenterOfTextAreaY = 0

    ptLowerLeftTextOriginX = 0
    ptLowerLeftTextOriginY = 0

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

    intFontFace = cv2.FONT_HERSHEY_SIMPLEX
    fltFontScale = float(plateHeight) / 30.0
    intFontThickness = int(round(fltFontScale * 1.5))

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

    ((intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight),
     fltCorrectionAngleInDeg) = licPlate.rrLocationOfPlateInScene

    intPlateCenterX = int(intPlateCenterX)
    intPlateCenterY = int(intPlateCenterY)

    ptCenterOfTextAreaX = int(intPlateCenterX)

    if intPlateCenterY < (sceneHeight * 0.75):
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight * 1.6))
    else:
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight * 1.6))

    textSizeWidth, textSizeHeight = textSize

    ptLowerLeftTextOriginX = int(ptCenterOfTextAreaX - (textSizeWidth / 2))
    ptLowerLeftTextOriginY = int(ptCenterOfTextAreaY + (textSizeHeight / 2))

    cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX, ptLowerLeftTextOriginY), intFontFace,
                fltFontScale, SCALAR_YELLOW, intFontThickness)


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