Computer Vision for Human-Computer Interaction
CSE 5524 Final Project Report

License Plate Character Recognition

by
Sarthak Awasthi and Vritangi Kansal

Introduction

Character recognition in license plates plays a crucial role in road safety. It helps in tracing stolen cars, finding parking slots, etc. This final project aims to detect the vehicle license plate, segment the characters on that plate, and identify each using a KNN algorithm. All these steps involve some traditional computer vision techniques. The main function takes an image as input and displays the corresponding characters on the car plate as the output.

Our license plate character recognition model has three major parts: license plate detection, character segmentation, and recognition of these characters.

Methodology/Framework

Dataset Description: The data set being used here are images of vehicles with their number plates. These images, 35 in total, are a random assortment of publicly available images containing vehicles with license plates in different online locations. Most of these license plates are US-based, while some belong to different European countries.

Pre-processing: The first step involved was pre-processing the images in the form of gray-scaling and detecting the canny edges to enhance the quality and contrast of our detected plate. We used morphological operators such as opening and closing to remove the apparent noise in the image. The final step in pre-processing was to threshold the image to help us get appropriate contours. An example of the thresholded image is shown below:

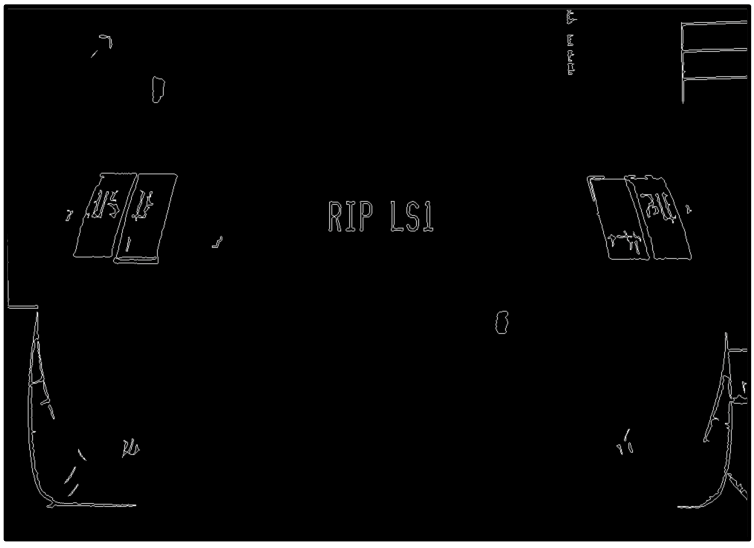
Original Image

Thresholding results after pre-processing

Find appropriate contours: The next step after thresholding the image was to take this thresholded image and use it to find appropriate closed components or contours in the image. For example, for the vehicle image displayed on the previous page, we found a count of 1082 possible contours in the entire image.

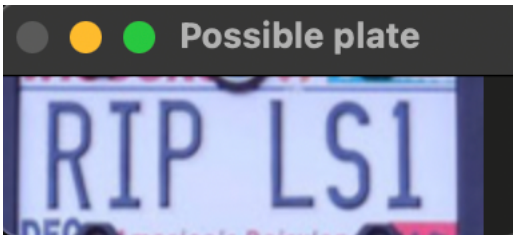
Once we have all the possible contours for the given image, the next task is to identify the contours that may be possible characters. For the example image, we found 49 possible contours that could be characterized as possible characters later for segmentation. This is done by hyper-tuning the parameters, such as setting min-max values for pixel width, height, aspect ratio, and area. The contours detected are displayed below:

1082 possible contours



49 contours identified as possible characters

Detect the license plates: The final step in the first part i.e., license plate detection, is to extract the potential license plates. For this purpose, we find the string of characters closer to each other and group them. For the example image we have been following, we found 2 possible contours that resemble something on the lines of a license plate. The two extracted license plates are shown below:



Possible plate 1

Possible plate 2

Character segmentation: Once we have the extracted license plate image with us, we start pre-processing this image by converting it into a grayscale image. We apply Gaussian blurring to it, having a kernel size of 5x5 and a sigma value of 1. The final pre-processing step is to threshold this image. We obtained some white blobs with a dark background, as displayed below:

Thresholded plate image

In the next step, we find the connected components with character-like features for every possible plate by setting dimensional constraints for possible characters. We set a dimensional constraint on each connected component's pixel width, height, aspect ratio, and area. Once we have only white blobs containing possible characters, we segment these characters using the bounding box.

Segmented possible characters

The cv2.boundingRect() function returns the parameters of height and width, which we used to crop each character out of the image. Each character, being an image, acts as a region of interest. This region of interest is resized before feeding it to the K-Nearest neighbor classifier.

K-Nearest Neighbor classification: The KNN classifier is trained using five font templates of 180 characters (36 unique alphanumeric characters consisting of 26 letters and 10 digits). All these characters of different contours are used as a training set of standard font. Each character image has a unique label value corresponding to a character. We found the best possible value of K equal to 1 for the best possible accuracy in the 35 test images we had. Each segmented character is an input into the KNN classifier (K=1) to find the nearest character. The classifier outputs the best possible character for each segment one by one, giving a list or string of characters that could be license plate characters.

Results/Analysis

The image below is an output image produced by our code when the image was given to the model as input.


```
The detected characters on the license plate: RIPLS1
```

Evaluation of results: Out of the 35 total test images, we had 28 accurate detections of license plate plates as well as their corresponding characters. Among the test images, there was 31 accurate license plate detection. 28 of the 35 test images had accurate character recognition.

Problems encountered: The thresholding fails for images where the contrast difference is not high between the background and foreground as displayed below:

Original image

Thresholded image

Another problem we faced was that the contour detection failed for certain characters on the license plate, as shown below:

This resulted in the wrong character recognition for the above license plate image.

In a few cases when the contour detection was correctly executed, some characters still were not recognized correctly:

Lessons learned: The image quality needs to be sufficiently high to detect characters. Applying preprocessing techniques solves a lot of issues regarding noise and background. Hyperparameter tuning (optimization) for a dataset can really improve results, but at the same time, it might not work for all the use cases.

Future Work

Our current implementation does not work on tilted images with varied camera angles. We could potentially improve plate detection, including other factors like plate background/texture/natural light, etc. We could also include motion-blurred effects due to fast-paced vehicles. Another thing we could try is extracting frames from a video to perform the same algorithms we did for the test images. There are better techniques to solve the above problems, which could be explored to see if those methods improve the accuracy of the license plate character recognition system.

Contribution

Vritangi: Preprocessing functions and detection of license plate.
Sarthak: Segmentation of characters and classification using KNN.

Attached source code:

Main.py

```
# Main.py

import cv2
import numpy as np
import os

import DetectChars
import DetectPlates
import PossiblePlate

def main():

    blnKNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN()

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

    imgOriginalScene  = cv2.imread("LicPlateImages/14.png")

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

```
    listOfPossiblePlates = DetectPlates.detectPlatesInScene(imgOriginalScene)

    listOfPossiblePlates = DetectChars.detectCharsInPlates(listOfPossiblePlates)

    cv2.imshow("imgOriginalScene", imgOriginalScene)

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

        drawRedRectangleAroundPlate(imgOriginalScene, licPlate)
        writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)
        cv2.imshow("imgOriginalScene", imgOriginalScene)
        cv2.imwrite("imgOriginalScene.png", imgOriginalScene)
    cv2.waitKey(0)

    return licPlate

def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):

    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)
    p2fRectPoints = p2fRectPoints.astype(int)

    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)
# end function

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

DetectPlates.py

```python
# DetectPlates.py

import cv2
import numpy as np
import math
import Main
import random

import DetectChars

PLATE_WIDTH_PADDING_FACTOR = 1.3
PLATE_HEIGHT_PADDING_FACTOR = 1.5

def detectPlatesInScene(imgOriginalScene):
    listOfPossiblePlates = []

    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()

    imgGrayscaleScene, imgThreshScene = Preprocess.preprocess(imgOriginalScene)
    listOfPossibleCharsInScene = findPossibleCharsInScene(imgThreshScene)

    if Main.showSteps == True:
        print("step 2 - len(listOfPossibleCharsInScene) = " + str(
            len(listOfPossibleCharsInScene)))

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

        contours = []

        for possibleChar in listOfPossibleCharsInScene:
            contours.append(possibleChar.contour)

        cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)
        cv2.imshow("Contours_0", imgContours)


    listOfListsOfMatchingCharsInScene = DetectChars.findListOfListsOfMatchingChars(listOfPossibleCharsInScene)


        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)

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

        cv2.imshow("Contours_color", imgContours)

    for listOfMatchingChars in listOfListsOfMatchingCharsInScene:
        possiblePlate = extractPlate(imgOriginalScene, listOfMatchingChars)

        if possiblePlate.imgPlate is not None:
            listOfPossiblePlates.append(possiblePlate)

    print("\n" + str(len(listOfPossiblePlates)) + " possible plates found")


        for i in range(0, len(listOfPossiblePlates)):
            p2fRectPoints = cv2.boxPoints(listOfPossiblePlates[i].rrLocationOfPlateInScene)
            cv2.imshow("Contours_reds1", imgContours)

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

            cv2.imshow("Possible plate", listOfPossiblePlates[i].imgPlate)
            cv2.waitKey(0)
        # end for
        cv2.waitKey(0)
    return listOfPossiblePlates
```

DetectChars.py

```
# DetectChars.py
import os

import cv2
import numpy as np
import math
import random
import Main

kNearest = cv2.ml.KNearest_create()

MIN_PIXEL_WIDTH = 2
```

```
MIN_PIXEL_HEIGHT = 8

MIN_ASPECT_RATIO = 0.25
MAX_ASPECT_RATIO = 1.0

MIN_PIXEL_AREA = 80

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

MIN_NUMBER_OF_MATCHING_CHARS = 3

RESIZED_CHAR_IMAGE_WIDTH = 20
RESIZED_CHAR_IMAGE_HEIGHT = 30

MIN_CONTOUR_AREA = 100

def loadKNNDataAndTrainKNN():
    allContoursWithData = []
    validContoursWithData = []
    npaClassifications = npaClassifications.reshape((npaClassifications.size, 1
    kNearest.setDefaultK(1)

    kNearest.train(npaFlattenedImages, cv2.ml.ROW_SAMPLE, npaClassifications
    return True

def detectCharsInPlates(listOfPossiblePlates):
    intPlateCounter = 0
    imgContours = None
    contours = []

    if len(listOfPossiblePlates) == 0:
        return listOfPossiblePlates


    for possiblePlate in listOfPossiblePlates:
        possiblePlate.imgGrayscale, possiblePlate.imgThresh = Preprocess.preprocess(possiblePlate.imgPlate)

      # 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)

        listOfPossibleCharsInPlate = findPossibleCharsInPlate(possiblePlate.imgGrayscale,
possiblePlate.imgThresh)
            del contours[:]                                         # clear the contours list

            for possibleChar in listOfPossibleCharsInPlate:
                contours.append(possibleChar.contour)

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

            cv2.imshow("Contours", imgContours)
        listOfListsOfMatchingCharsInPlate = findListOfListsOfMatchingChars(listOfPossibleCharsInPlate)



            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("Contours_1", imgContours)

        if (len(listOfListsOfMatchingCharsInPlate) == 0):
            possiblePlate.strChars = ""
            continue

        for i in range(0, len(listOfListsOfMatchingCharsInPlate)):# within each list of matching chars
            listOfListsOfMatchingCharsInPlate[i].sort(key = lambda matchingChar: matchingChar.intCenterX)
            listOfListsOfMatchingCharsInPlate[i] =
removeInnerOverlappingChars(listOfListsOfMatchingCharsInPlate[i])

            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)
```

```
                cv2.drawContours(imgContours, contours, -1, (intRandomBlue, intRandomGreen, intRandomRed))
            cv2.imshow("Contours_2", imgContours)
        intIndexOfLongestListOfChars = 0

        for i in range(0, len(listOfListsOfMatchingCharsInPlate)):
            if len(listOfListsOfMatchingCharsInPlate[i]) > intLenOfLongestListOfChars:
                intLenOfLongestListOfChars = len(listOfListsOfMatchingCharsInPlate[i])
                intIndexOfLongestListOfChars = i

        longestListOfMatchingCharsInPlate = listOfListsOfMatchingCharsInPlate[intIndexOfLongestListOfChars]

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

        cv2.imshow("Contours_3", imgContours)
     cv2.waitKey(0)

    return listOfPossiblePlates

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 =

    for possibleMatchingChar in listOfChars:
        if possibleMatchingChar == possibleChar:
        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

# 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

# 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)

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

        cv2.rectangle(imgThreshColor, pt1, pt2, Main.SCALAR_GREEN, 2)
        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))           #

        npaROIResized = imgROIResized.reshape((1, RESIZED_CHAR_IMAGE_WIDTH * RESIZED_CHAR_IMAGE_HEIGHT))

        npaROIResized = np.float32(npaROIResized)
        retval, npaResults, neigh_resp, dists = kNearest.findNearest(npaROIResized, k = 1)
        strCurrentChar = str(chr(int(npaResults[0][0])))            # get character from results
        strChars = strChars + strCurrentChar                        # append current char to full string

    return strChars
```