Highway Speed Sensor with Numberplate Detection: Documentation

Aaryen Mehta

Abstract

This paper documents a method on how to calculate speed of a car and then identify it's numberplate using simple Python libraries. The source code for this project is in Python 3.X.

Technologies Used

These are the main libraries used in this project.

- NumPy
- OpenCV
- SciPy
- Pytesseract

OpenCV, NumPy and SciPy libraries are used for simple image processing applications. And the text detection from the numberplate is done using a library called Pytesseract. The working of this library is based on Leptonica.

Methodology

Installing Required Libraries

To directly install tesseract-ocr using terminal, type the following command after opening terminal in Linux based operating system.

```
$ sudo apt-get install tesseract-ocr
```

This only works for some latest versions of Ubuntu. For example, in Ubuntu 16.04, by the above mentioned command, Leptonica 1.7.0 and Tesseract 3.X is installed. If you want to install latest versions of Tesseract like 4.0 and 5.0-alpha, you have to download Leptonica and Tesseract and then compile them manually. To do this, follow this procedure.

Installing Dependencies

If not installed, then install these following libraries first.

```
$ sudo apt-get install g++
$ sudo apt-get install autoconf automake libtool
$ sudo apt-get install autoconf-archive
$ sudo apt-get install pkg-config
$ sudo apt-get install libpng-dev
$ sudo apt-get install libjpeg8-dev
$ sudo apt-get install libtiff5-dev
$ sudo apt-get install zlib1g-dev
$ sudo apt-get install libicu-dev
$ sudo apt-get install libpango1.0-dev
$ sudo apt-get install libcairo2-dev
```

Installing Leptonica >= 1.74.0

Download desired version of Leptonica from this link : `http://www.leptonica.org/download.html`. Now go to Download directory and then install it with terminal using following commands. This procedure is shown for Leptonica 1.74.

```
$ sudo tar xf leptonica-1.74.tar.gz
$ cd leptonica-1.74
$ sudo ./configure
$ sudo make
$ sudo make install
```

Installing Tesseract

Download or clone desired version of Tesseract from this github repository: `https://github.com/tesseract-ocr/tesseract/`. Install tesseract by following commands in terminal.

```
$ sudo sh autogen.sh
$ ./configure
$ LDFLAGS="-L/usr/local/lib" CFLAGS="-I/usr/local/include" make
$ sudo make install
$ sudo make install -langs
$ sudo ldconfig
```

To verify the installation use this.

```
$ tesseract --version
```

Source Code

```
from cv2 import cv2
from pylab import *
from scipy.ndimage import filters
import pytesseract

def resize(image,percent_width=100,percent_height=100): #function to resize
#image
    width = int32(image.shape[1] * percent_width / 100)
    height = int32(image.shape[0] * percent_height / 100)
    resized = cv2.resize(image,(width,height))
    return resized

def printf(text): #function to print detected numberplate in a nice way
    for i in text:
        if i == '_':
            pass
        else :
            print(i,end='')
    print()

pytesseract.tesseract_cmd = '/usr/local/bin/tesseract' #specify path to your
#tesseract executable file
image = cv2.imread('ind_2.jpg')
cv2.imshow("Original", image)

im2 = filters.gaussian_filter(image,0.2) #applying gaussian blur to image
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #grayscaling image
edged = cv2.Canny(im2, 170, 200) #canny edge detection

contours,heirarchy = cv2.findContours(edged.copy(), cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE) #finding contours
contours = sorted(contours, key = cv2.contourArea, reverse = True)[:30]
#sorting contours more than area of 30 units

for c in contours:
    length = cv2.arcLength(c,closed=True)
    approx = cv2.approxPolyDP(c, 0.02 * length, closed=True)
    #approx polygon covering contour
    if len(approx) == 4:  # Select the contour with 4 corners
        numberplate = approx #This is our approx Number Plate Contour
        break
    else :
        numberplate = None
```

```
# Drawing the selected contour on the original image
#cv2.drawContours(image, [numberplate], -1, (0,255,0), 3)

# the following method is to straighten a numberplate that is slant or skewed

x = numberplate
a = min(x[0][0][1],x[2][0][1])
b = max(x[0][0][1],x[2][0][1])
c = min(x[2][0][0],x[0][0][0])
d = max(x[2][0][0],x[0][0][0])
roi = image[a:b,c:d]
width = roi.shape[1]
height = roi.shape[0]
pts1 = float32(x)
for i in range(4):
    if (x[i][0][1] < x[(i+1)%4][0][1] and x[i][0][0] < x[(i-1)%4][0][0]):
        flag = 1
        break

for j in range(4):
    if flag == 1 :
        break
    if (x[j][0][1] < x[(j-1)%4][0][1] and x[j][0][0] < x[(j+1)%4][0][0]):
        flag = -1
        break

if flag == 1 :
    if i == 0 :
        pts2 = float32([[0,0],[0,height],[width,height],[width,0]])
    elif i == 1 :
        pts2 = float32([[width,0],[0,0],[0,height],[width,height]])
    elif i == 2 :
        pts2 = float32([[width,height],[width,0],[0,0],[0,height]])
    elif i == 3 :
        pts2 = float32([[0,height],[width,height],[width,0],[0,0]])

elif flag == -1 :
    if j == 0 :
        pts2 = float32([[0,0],[width,0],[width,height],[0,height]])
    elif j == 1 :
        pts2 = float32([[0,height],[0,0],[width,0],[width,height]])
    elif j == 2 :
        pts2 = float32([[width,height],[0,height],[0,0],[width,0]])
    elif j == 3 :
        pts2 = float32([[width,0],[width,height],[0,height],[0,0]])
```

```
matrix = cv2.getPerspectiveTransform(pts1,pts2)
result = cv2.warpPerspective(image,matrix,(image.shape[0],image.shape[1]))
roi = result[0:height,0:width]
cv2.imshow("numerplate",roi)

#text detection starts here

gray_ = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY) #preparing image for thresholding
ret, roi = cv2.threshold(gray_, 200, 255, cv2.THRESH_BINARY) #thresholding
#image
roi = resize(roi,100,89) #using resize function for better text detection
cv2.imshow("threshold",roi)

rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))
#creating rectangular kernel
dilation = cv2.dilate(roi, rect_kernel) #dilating thresholded image
cv2.waitKey(500)

contours_, hierarchy = cv2.findContours(dilation, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE) #fincing contours
im3 = roi.copy() #creating copy of the image

for cnt in contours_:
    x, y, w, h = cv2.boundingRect(cnt)

    rect = cv2.rectangle(im3, (x, y), (x + w, y + h), (0, 255, 0), 2)
    #drawing rectangle on copied image
    cropped = im3[y:y + h, x:x + w] #cropping the image for input to ocr
    # applying tesseract-ocr
    text = pytesseract.image_to_string(cropped,lang='eng', config=-c
    tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 --psm 7)
    if text == '':
        continue
    printf(text)
```

Results

Here is an image that gives accurate output. Otherwise, in many images, some letters and numbers are not detected accurately. Sometimes the resolution is low and sometimes lighting is not adequate. This is the original image.

And this is the numberplate detected by the above Python code.

And this is the numberplate after thresholding.

And this is the output on the terminal of this code.

```
$ INDHR26DK8337
```

Future Developements

Only a limited amount of images are detected with perfect accuracy. Such as images which have proper lighting on numberplate and images that are high resolution. The first aim is to improve methodology in the code so as to use it to detect a wide variety of images which is more generalised. This code here is just a preface. Main aim is to apply this methodology on video of traffic and find out the speed and detect the numberplate of cars. Further modifications to the code will be done once a proper video feed is obtained after lockdown due to covid-19 is over and we all return to IITK campus. Video feed will be preferably obtained in IITK campus. As the current code is for images only, speed detection parts will be added later. And the method used in that will be covered under Technology Used section.