Minor Project Report
Submitted for

ARTIFICIAL INTELLIGENCE (UCS411)

Submitted by:

(102003449) LAKSHAY BHARDWAJ

BE Second Year

Submitted to-

Niyaz Wani

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

TIET, Patiala

Feb-June 2022

- PROJECT OVERVIEW:

  Self-Driving/Autonomous Cars are becoming a rather common phenomena these days with companies like apple, google, etc investing huge amounts in their R&D. It's one step towards a technologically advanced world. As computers are getting better at understanding images due to advances in computer vision, the concept of an autonomous vehicle that can manoeuvre independently and detect objects is becoming increasingly realistic. In this project, we created an Artificial Intelligence tool capable of detecting lane and give directions.

- PROBLEM STATEMENT:

  The goal is to detect lanes on a road; the tasks involved are the following:
  1. Download and pre-processing a suitable video(dataset)
  2. Detect lanes using open-cv library functions
  3. Track position of car using TrackerMedianFlow
  4. Display commands to keep car lane bound.

  The final application is expected to be useful for detecting lanes on highways, city roads and manoeuvre vehicle.

- PRE-PROCESSING:

First of all we are importing 2 libraries, which are OpenCV(cv2) for image processing and numpy for calculation of data. We are taking input in the form of video by using the function cv2.videocapture() and we are dividing the video into multiple frames and read each frame as input image via function cv.read(). This will read the image and return it as a multi dimension numpy array containing the relative intensity of each pixel in the image. We will create copy of every frame and work on the copy because any change we make in original frame will also be reflected in the original mutable array.

```
def canny(lane_image):
    #convert pic to grayscale
    gray = cv2.cvtColor(lane_image,cv2.COLOR_RGB2GRAY)
    #blurring image(smoothening) using a 5x5 matrix and weighted average of pixels
    blur = cv2.GaussianBlur(gray,(5,5),0)
    #finding edges(large gradient/drastic change in intensity)
    edge = cv2.Canny(blur,50,150)
    return edge
```

- Def canny()-

After conversion we will give each frame as a input to this function. This function is used for edge detection in an image. The goal of edge detection is to identify boundaries of an object in images. We will do this by detecting sharp change in intensities, sharp change in color. Image is stored as an array of pixels and therefore we will check difference in intensity(steep change in values of adjacent pixel) to identify boundary. This is a multistep process,

step 1 being converting the image to grey scale because single channel is processed faster than 3 channel color image(RGB) . We will convert each frame into grey scale by using the function cv2.cvtcolor(frame, cv2.colorRGB2GRAY)(for conversion from RGB to GRAY).

Step2 Now for faster and accurate processing we will reduce noise and smoothen our image. For this we will use gaussian blur function that is Cv2.gaussainblur()()using 5x5 kernel). With this we will apply gaussian blur to our greyscale image.

Step3 As an image can be represented a 2D coordinate space, X (image width) and Y (image height) representing no. of rows and column in an image. By this we can look at our image as a continuous function of X and Y. What cv2.canny(blurred_image, Low_threshold. High_threshold) does is perform a derivative on our function in both x and y direction(measuring change in intensity w r t adjacent pixels). It computes gradient in all direction of our blurred image and is then going to trace our strongest gradients as a series of our white pixels(line). It returns the rendered frames with traced edges.

- IMPLEMENTATION:

```
def area_of_interest(lane_image):
    height = lane_image.shape[0]
    width = lane_image.shape[1]
    area = np.array([[(-30,height-100),(width-75,height-100),(650,300)]])   #marking region of interest using x,y coordinates
    mask = np.zeros_like(lane_image)          #reference black image
    cv2.fillPoly(mask,area,255)               #area of interest super-imposed in black image
    masked = cv2.bitwise_and(lane_image,mask) #takes bitwise & of each bit of pixel to retain area of interest in canned image
    return masked
```

- Def area_of_interest()-

This function will take an image as an input and will mark the area of interest. For this we will first find out the height and width of image using image.shape[0] and image.shape[1]. Now we will mark the region of interest using x,y coordinates using np.array() function. We will create a reference black image called mask with np.zeros_like(image) which has the same no. of pixels and dimensions as our original image. After that we have to fill this mask with our ROI by using cv2.fillPoly(mask,area,255) function. The fill poly function fills the area bounded by several ROI's. We now are going to apply this mask onto our canny image to ultimately only show the ROI traced by the polygonal contour. We do this by applying the bitwise AND operation between the 2 Images. It occurs elementwise between the 2 images, between the 2 array of pixels(both images have same no. of pixels and dimensions). Since it occurs elementwise, we are taking bitwise AND of each homologous pixel on both arrays by using cv2.bitwise_and(image,mask) function.


```
lines = cv2.HoughLinesP(cropped,2,np.pi/180,100,np.array([]),minLineLength=40,maxLineGap=5)  #finding best fit line for given points
#using hough transform(rho,theta and each bin) :arg 2 & 3 define size of each bin(rho,theta); arg 4 defines resolution or no. of
# intersection per bin to detect a line;arg 5 is a placeholder ; arg 6 defines min. length of line in pixel to be detected;
# arg 7 defines min gap in pixel between segment to be considered as a single line
```

- Hough Lines:

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. The main advantage of the Hough transform technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise. We use Hough technique to identify the lines in our image using cv2.HoughLinesP(cropped,2,np.pi/180,100,np.array([]),minLineLength=40,maxLineGap=5) function. The final result of the linear Hough transform is a two-dimensional array (matrix)—one dimension of this matrix is the quantized angle θ, and the other dimension is

the quantized distance *r*. Each element of the matrix has a value equal to the sum of the points or pixels that are positioned on the line represented by quantized parameters (*r*, *θ*). So the element with the highest value indicates the straight line that is most represented in the input image.

Cv2.imshow() function is used to render the image and show the output. This is followed by cv2.waitkey function which displays the image for a specified amount of milliseconds.

```
def average_slope(lane_image,lines):     #taking average of slope interecpt to obtain single line
    left_fit = []        #coordinates of line on left
    right_fit = []       #coordinated of line on right
    for line in lines:
        x1,y1,x2,y2 = line.reshape(4)
        parameters = np.polyfit((x1,x2),(y1,y2),1)    #finds slope and y intercept of each line
        slope = parameters[0]
        intercept = parameters[1]
        if slope<0:       #categorizing lines as left and right on the basis of sign of slope
            left_fit.append((slope,intercept))
        else:
            right_fit.append((slope,intercept))
    left_fit_average = np.average(left_fit,axis=0)   #taking average of all lines on left hand side to obtain a single line
    right_fit_average = np.average(right_fit,axis=0)
    left_line = coordinates(lane_image,left_fit_average)     #obtaining coordinates for 2 lines
    right_line = coordinates(lane_image,right_fit_average)
    return np.array([left_line,right_line])
```

- Def Average Slope()-

After obtaining all the lines in the image using Hough line function in variable “lines,” we know categorize the lines into left and right lines on the basis of the sign of their slope. After which the average of the slope and y intercept of the respective lines is obtained in order to get one line on both left- and right-hand side using the np.average function. Using the values of slope and y intercept, two coordinates each of both lines is obtained using the function y=mx + b.

```
def reference(frame,lines):                          #determing the midpoint of the lane to reference car's motion
    left,right = average_slope(frame,lines)
    x1,y1,x3,y3 = left.reshape(4)
    x2,y2,x4,y4 = right.reshape(4)
    mid_x,mid_y = (x3+x4)/2,(y3+y4)/2
    limit = mid_x - x3                               #width of the lane to make sure car stays in-bound
    return mid_x,mid_y,limit
```

- Def Reference()-

This function is defined in order to obtain the midpoint of the lane. Here we get the coordinates of left and right lines from function Average Slope and using the formula x=(x1+x2)/2 determine the midpoint of the lane in order to direct the car in the right direction.

```
def marker(lane_image):                              #uses library function TrackerMedianFlow to select a region of interest(car here)
    mark = cv2.legacy.TrackerMedianFlow_create()     #and track it in further frames
    box = cv2.selectROI(lane_image,False)
    mark.init(lane_image,box)
    return mark,box
```

- Def Marker()-

This function uses a library function namely TrackerMedianFlow that accepts a bounding box and an image. A number of points within the bounding box are tracked and thus estimate the motion of the bounding box. Using the function selectROI, a region of image is selected which is initialized once and later updated for each frame giving us the vital information to track the very region in the form of coordinates of the top left corner of the bounding box, its height and width.

```
def command(lane_image,dis,limit):                   #using the horizontal distance between the reference point of the car
    if(dis>limit):                                   #and midpoint of line, suitbale command is given
        if (dis>=-575 and dis<=575):
            cv2.putText(final_image, "CONTINUE STRAIGHT", (100,80), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
        elif (dis>575):
            cv2.putText(final_image, "SLIGHT RIGHT", (100,80), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
        else:
            cv2.putText(final_image, "SLIGHT LEFT", (100,80), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
    else:
        cv2.putText(final_image, "OUT OF LANE", (100,80), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
```

- Def Command()-

Using the relative position of the bounding box and the reference point of the lane or its midpoint, an alert is displayed on the screen regarding the direction in which the driver needs to steer in order to remain in bounds of the lane.

➢ RESULT:

Therefore, on a given dataset we are able to successfully detect the lane with the help of various library functions mentioned above and also flash a message for the driver to steer in the right direction so as to stay in lane bounds.

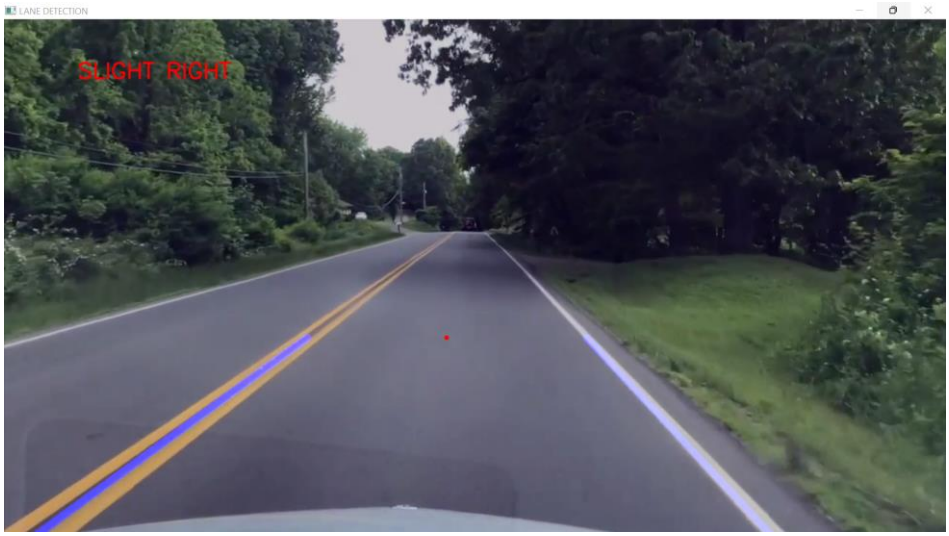
➢ IMPROVEMNETS AND FUTURE APPLICATIONS:

The basic model of lane detection has variety of real-life applications, some of them are to maneuver a self-driving and aid blind people by reading out the direction to move using an audio output device taking an input live feed from one’s spectacles to navigate and detect obstacles.

- CONCLUSION:

Figure below depicts the process of selecting region of interest using a bounding box which is used to give the position of car.

Figure below is a snapshot of the output window where the lane boundaries are depicted with blue lines and the relative position of the car to the midpoint of the lane is calculated and suitable steering command is displayed on screen.



-x-x-x-x-x-x-x-x-x-x-