International Journal for Modern Trends in Science and Technology, 6(12): 176-180, 2020

ISSN: 2455-3778 online
DOI: https://doi.org/10.46501/IJMTST061234
Available online at: http://www.ijmtst.com/vol6issue12.html

Lane Detection for Autonomous Vehicles using Open CV Library

Aarushi Mittal[1] | Narinder Kaur[2]

[1]Information Technology (B.Tech Scholar), Maharaja Agrasen Institute of Technology.
[2]Information Technology (Assistant Professor), Maharaja Agrasen Institute of Technology.

To Cite this Article
Aarushi Mittal and Narinder Kaur, "Lane Detection for Autonomous Vehicles using Open CV Library", *International Journal for Modern Trends in Science and Technology*, 6(12): 176-180, 2020.

Article Info
Received on 08-November-2020, Revised on 28-November-2020, Accepted on 02-December-2020, Published on 05-December-2020.

ABSTRACT

For vehicles to have the option to drive without anyone else, they have to comprehend their encompassing world like human drivers, so they can explore their way in roads, pause at stop signs and traffic signals, and try not to hit impediments, for example, different vehicles and pedestrians. In view of the issues experienced in identifying objects via self-governing vehicles an exertion has been made to show path discovery utilizing OpenCV library. The explanation and method for picking grayscale rather than shading, distinguishing and detecting edges in an image, selecting region of interest, applying Hough Transform and choosing polar coordinates over Cartesian coordinates has been discussed.

KEYWORDS: *Numpy , OpenCV , Canny , Lane-Detection , Hough Transform.*

I. INTRODUCTION

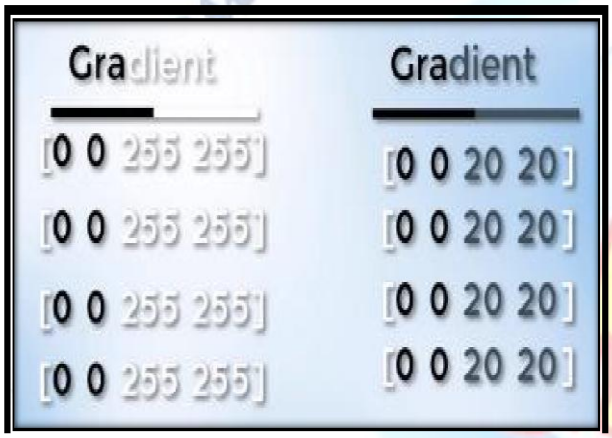
During the driving activity, people utilize their optical vision for vehicle moving. The street path stamping, go about as a consistent reference for vehicle route. One of the requirements to have in a self-driving vehicle is the advancement of an Automatic Lane Recognition framework utilizing an algorithm. PC vision is an innovation that can empower vehicles to sort out their environmental factors. It is a part of man-made consciousness that empowers programming to comprehend the substance of picture and video. Current PC vision has progressed significantly due to the propels in profound realizing, which empowers it to perceive various items in pictures by looking at and contrasting great many models and cleaning the visual examples that characterize each item. While particularly proficient for arrangement assignments, profound taking in experiences genuine constraints and can fall flat in flighty manners. [1] This implies that a driverless vehicle may collide with a truck without trying to hide, or more regrettable, inadvertently hit a passer-by. The current PC vision innovation utilized in self-ruling vehicles is likewise powerless against antagonistic assaults, by controlling the AI's input channels to constrain it to commit errors. For example, analysts have indicated they can deceive a self-driving vehicle to evade perceiving stop signs by staying high contrast marks on them.[2]

II. METHODOLOGY

The undertaken project is majorly based on the Open CV also known as the Open Source Computer Vision. It is in a package in python and has various tools for analysing images and making various interpretations.

2.1 Canny Edge Detection Technique

The objective of edge recognition is to distinguish the limits of items inside pictures. A discovery is utilized to attempt to discover areas in a picture where there is a sharp change in force. We can perceive a picture as a lattice or a variety of pixels. A pixel contains the light force at some area in the picture. Each pixel's power is meant by a numeric worth that goes from 0 to 255, a force estimation of zero shows no power if something is totally dark though 255 speaks to greatest force something being totally white. An inclination is the adjustment in brilliance over a progression of pixels. A solid inclination shows a steep change while a little angle speaks to a shallow change.



On the right-hand side there is a figure of the angle of the football. The framework of white pixels relates to the irregularity in splendour and the focuses the fortify slope. This causes distinguished edges in our picture since an edge is characterized by the distinction in power esteems in neighbouring pixels

Other than that, any place where there is a sharp change in power (quick change in brightness) i.e., any place there is a solid inclination, there is a relating bright pixel in the angle picture. By following out every one of these pixels, we get the edges. We will utilize this idea to recognize the edges in our road picture.

The image is read and loaded into an array .

image = cv2.imread(test_image.jpg)

In order to obtain the image in grayscale we will first make a copy of the original image using Numpy:

lane_image = np.copy(image)

gray=cv2.cvtColor(lane_image,cv2.COLOR_RGB2GRAY)

Hence , an image in grayscale is obtained.

2.2 Gaussian Blur

Every one of the pixels for a grayscale picture are portrayed by a solitary number that depicts the brightness of the pixel. To smoothen a picture, the commonplace answer is changing the estimation of a pixel with the normal estimation of the pixel powers around it. Averaging out the pixels to diminish the commotion will be finished by a part. This bit of regularly conveyed numbers (np.array([[1,2,3],[4,5,6],[7,8,9]]) is stumbled into our whole picture and sets every pixel esteem equivalent to the weighted normal of its neighbouring pixels, in this manner smoothening our picture. For our situation we will apply a 5x5 Gaussian part.

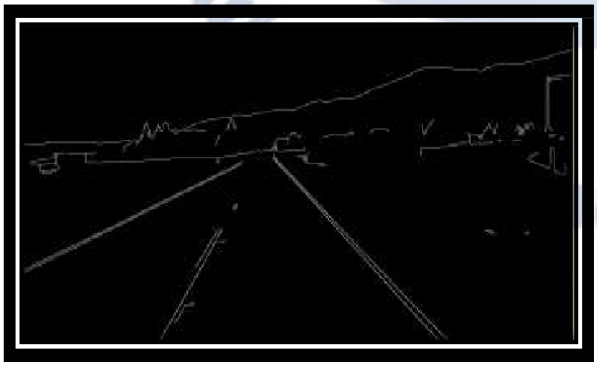
blur = cv2.GaussianBlur(gray,(5,5), 0)

The results that are obtained by applying the gaussian blur are depicted in the image shown above. This has reduced noise.

2.3 Edge Detection

An edge compares to a locale in a picture where there is a sharp change in the power/shading between nearby pixels in the picture. A solid inclination is a precarious change and the other way around is a shallow change. So, in a manner we can say a picture is a pile of lattice with lines and segments of powers. This implies that we can likewise speak to a picture in 2D facilitate space, x hub navigates the width (sections) and y pivot comes the picture tallness (lines). Vigilant capacity plays out a subordinate on the x and y hub in this way estimating the adjustment in powers regarding adjoining pixels. All in all, we are figuring the inclination (which is change in light) every which way. It at that point follows the most grounded slopes with a progression of white pixels.

```
canny = cv2.Canny(blur, 50, 150)
```

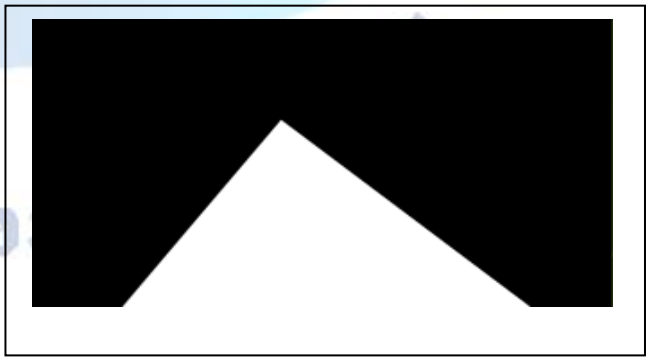


The low threshold, high threshold permit us to segregate the adjoining pixels that follow the most grounded angle. In the event that the angle is bigger than the upper limit then it is acknowledged as an edge pixel, in the event that it's underneath the low edge, at that point it is dismissed. On the off chance that the slope is between the limits then it is acknowledged just if it's associated with a solid edge. Zones where it's totally dark relate to low changes in force between neighbouring pixels though the white line speaks to a district in the picture where there is a high change in force surpassing the edge.

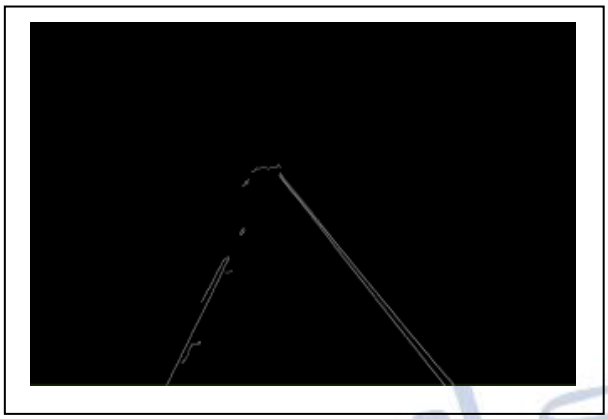
2.4 Region of Interest

The elements of the picture are picked which will contain the street paths and imprint it as our area of interest or the triangle. At that point a veil is made which is same as the element of the picture which would basically be a variety, all things considered. Presently we fill the triangle measurement in this veil with the power of 255 so our district of interest measurements is white. Presently I will do a bitwise AND activity with the vigilant picture and the mask which will bring about our last locale of interest.

```
def region(image):

height = image.shape[0]

poly=np.array([[(200,height),(1100,height),(550,25
0)]])

mask = np.zeros_like(image)

cv2.fillPoly(mask, poly,255)

masked_image= cv2.bitwise_and(image,mask)

return masked_image
```



In the above image , we have the image of the mask.



In the above image , we have masked_image which is a canny image with the bitwise operation and masking performed on it.

2.5 Hough Transform

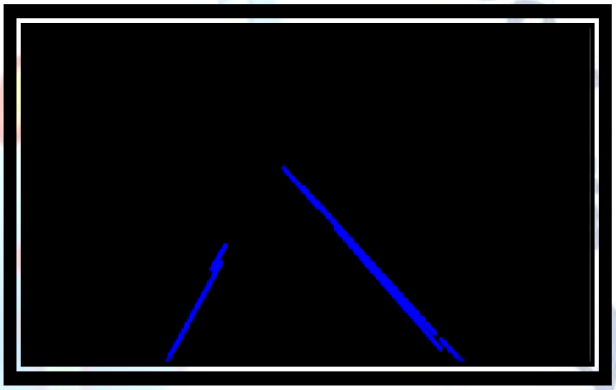
By implementing hough transform technique we will be looking for straight lines which are our lane lines. Furthermore, the slope of the line is essentially an ascent over run. On the off chance that the y intercept and slope is given, at that point the line can be plotted in the Hough Space as a solitary spot. There are numerous potential lines that can go through this spot each line with various qualities for M and B. There are numerous potential lines that can cross each point independently, each line with various slope and y intercept esteems Anyway there is one line that is reliable with the two focuses. We can verify that by taking a gander at the purpose of convergence enough space since that purpose of convergence in Hough Space and that purpose of crossing point speaks to the M and B estimations of a line predictable with intersection both the focuses. Presently to recognize the lines, we will initially part our Hough space into a matrix. Each receptacle inside the network comparing to the slope and y intercept estimation of the line. For each purpose of crossing point in a Hough Space receptacle we will make a choice within the container that it has a place with. The container with greatest number of votes will be our line. However, as we realize that the slope of a vertical line is limitlessness. So to communicate vertical

lines, we will utilize polar arranges rather than cartesian directions. Therefore the condition of our line becomes:

roh= xcos(theta) + ysin(theta)

```
def display_lines(image, lines):
  line_image= np.zeros_like(image)
  if lines is not None:
  for  line in lines:
  x1,y1,x2,y2 = line.reshape(4)
  cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
  return line_image
```

lines= cv2.HoughLinesP(cropped,2,np.pi/180, 100, np.array([]), minLineLength=40, maxLineGap=5)



Now we will combine all the results obtained, this would be given by:

combo_image = cv2.addWeighted(lane_image, 0.8, line_image, 1, 1)

III. CONCLUSION

In the approach, we utilized the OpenCV library and its capacities, for example, the Canny Function through which we accomplished edge recognition. At that point we arranged a mask of zero force and planned our locale of interest by playing out the bitwise activity. At that point we utilized the Hough Transform procedure that recognized the straight lines in the picture and distinguished the path lines. We utilized the polar directions since the Cartesian directions don't give us a proper slope of vertical also, even lines. At last, we joined the path picture with our zero- power picture to show path lines.

REFERENCES

[1] Jay Hoon Jung, Yousun Shin, YoungMin Kwon (2019). "Extension of Convolutional Neural Network with General Image Processing Kernels".

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (2016). "You Only Look Once: Unified, Real-Time Object Detection".

[3] Mayank Singh Chauhan, Arshdeep Singh, Mansi Khemka, Arneish Prateek, Rijurekha Sen (2019). "Embedded CNN based vehicle classification and counting in non-laned road traffic".

[4] Tejas Mahale, Chaoran Chen, Wenhui Zhang (2018). "End to End Video Segmentation for Driving: Lane Detection for Autonomous Car".