ACADEMIA

Accelerating the world's research.

IRJET-Lane Detection for Autonomous Vehicles using Opencv Library

IRJET Journal

Related papers

Download a PDF Pack of the best related papers

Lane Detection for Autonomous Vehicles using OpenCV Library

Aditya Singh Rathore

B.Tech, J.K. Lakshmipat University

---***---

Abstract - *For vehicles to be able to drive by themselves, they need to understand their surrounding world like human drivers, so they can navigate their way in streets, pause at stop signs and traffic lights, and avoid hitting obstacles such as other cars and pedestrians. Based on the problems encountered in detecting objects by autonomous vehicles an effort has been made to demonstrate lane detection using OpenCV library. The reason and procedure for choosing grayscale instead of colour, detecting edges in an image, selecting region of interest, applying Hough Transform and choosing polar coordinates over Cartesian coordinates has been discussed.*

Key Words: Numpy, OpenCV, Canny, Lane-Detection, Hough Transform

1. INTRODUCTION

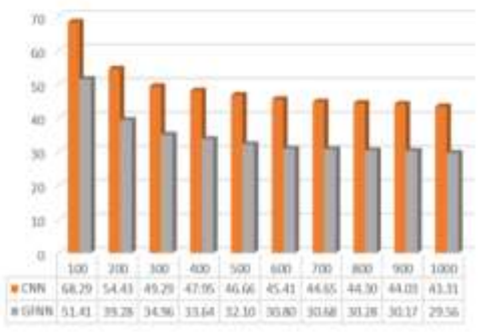
During the driving operation, humans use their optical vision for vehicle maneuvering. The road lane marking, act as a constant reference for vehicle navigation. One of the prerequisites to have in a self-driving car is the development of an Automatic Lane Detection system using an algorithm.

Computer vision is a technology that can enable cars to make sense of their surroundings. It is a branch of artificial intelligence that enables software to understand the content of image and video. Modern computer vision has come a long way due to the advances in deep learning, which enables it to recognize different objects in images by examining and comparing millions of examples and cleaning the visual patterns that define each object. While especially efficient for classification tasks, deep learning suffers from serious limitations and can fail in unpredictable ways.

This means that a driverless car might crash into a truck in broad daylight, or worse, accidentally hit a pedestrian. The current computer vision technology used in autonomous vehicles is also vulnerable to adversarial attacks, by manipulating the AI's input channels to force it to make mistakes. For instance, researchers have shown they can trick a self-driving car to avoid recognizing stop signs by sticking black and white labels on them.

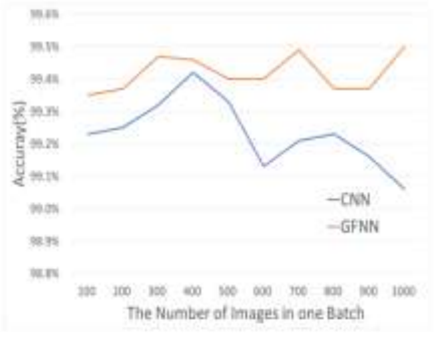
1.1 Use of general image processing kernels

Instead of letting Convolutional neural network choose its kernel, some predefined kernels used for image processing such as sharpening, edge-detecting, discrete cosine transformation and blurring were applied to the first layer of CNN. This concept was named as the General Filter Convolutional Neural Network (GFNN). Under the architecture, as the processing happens on each layer of convolutional neural network, the target area in the image becomes smaller. A total of 41 3x3 matrices were applied at the first layer of CNN out of which there were 3 sharpening filters, 2 blurring filters, 1 embossing filter, 2 second order operators, 1-degree cosine transformation operator and 32 edge detecting gradient operators. The results for this model showed 30% less training time than that of CNN and the accuracy was higher compared to CNN despite the model being quicker. [1]



	100	200	300	400	500	600	700	800	900	1000
■ CNN	68.29	54.43	49.29	47.95	46.66	45.41	44.65	44.30	44.03	43.31
■ GFNN	51.41	39.28	34.96	33.64	32.10	30.80	30.68	30.28	30.17	29.56

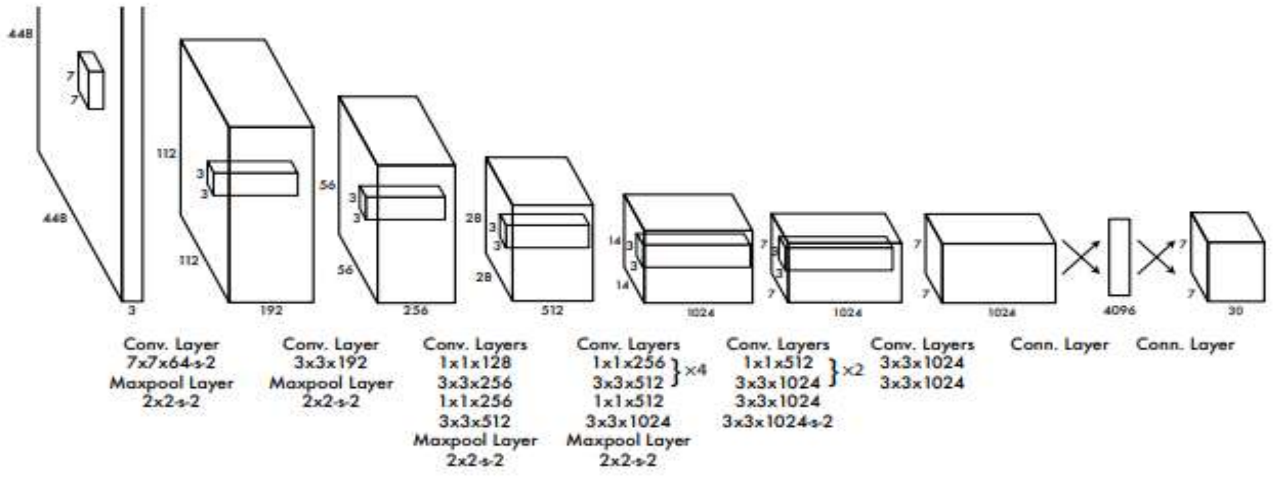
Training Time



Accuracy Comparision

1.2 You Only Look Once Object Detection Model

Separate components of object detection have been combined in a single neural network. Every information in the image is used to predict the boundaries of each object. The image is first separated in an A x A matrix. The cell where the centre of the object falls in is responsible for identifying the object along with the confidence scores. These scores explain the confidence level of the model in identifying an object. Below is the architecture of the model:



The dataset used to train the model was the ImageNet 1000-class dataset. The first 20 convolutional layers of the model were used that were followed by average-pooling layer and a fully connected layer. The major limitation of the model is that each cell can predict only 2 boxes and only 1 class can exist per cell. As a result, nearby objects do not get detected. Small objects in groups are not identifiable by this model. [2]

1.3 Classification and Counting of vehicles in Non-laned road traffic

A Convolutional Neural Network (CNN) based object detection model was trained on different vehicle classes that run on Delhi roads. A result of up to 75% MAP was achieved when the model was trained on 5562 video frames using 80-20 ratio of training-testing data. The YOLO CNN model was used, and it was fine-tuned by training the model on MS-COCO dataset, PASCAL VOC 2007 dataset, KITTI datasets and some custom tuned datasets before it was put to test. It was found that high intra-class variation such as bicycle and rickshaws and less quantity of training samples were reducing the accuracy for some classes. So, to reduce the intra- class variation, different labels were recommended for different looking objects in the same class. Training of this model to different cities traffic such as Bangalore and Mumbai will enhance the performance. The observations of installation and maintenance costs, traffic measurement effectiveness and rule violation would be shared with the traffic authorities. Below are the sample images of the model detecting objects: [3]

Sample images with object detection bounding box and class label outputs

1.4 Lane Detection for Autonomous Car via Video Segmentation

There are many steps in detecting lanes on a road, first comes the camera calibration. Cameras uses curved lenses to form an image, and light rays often bend a little too much or too little at the edges of these lenses. Images can be undistorted mapping distorted points to undistorted points such as a chessboard. This distortion correction is then applied to raw images, convert images to grayscale, apply gradients and finally apply deep leaning. Then perspective transform is applied to the binary image from bird's eye view. A Global Convolution Networks (GCN) model is used to rectify the classification and localization issues for semantic segmentation of lane. The model is evaluated using colour-based segmentation. For classification task the conical CNN have proved to be better. For object segmentation, the size of the kernel matters a lot. But a larger size results in increase in weights and the number of parameters also increases. In the model 1 D kernels were used to increase performance with less weights. The dataset was collected from Carla Simulator which contained 3000 images of road. The sky portion and car hood part from the images was cropped to increase the performance. [4]

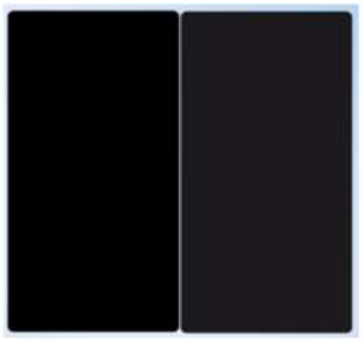
2. METHODOLOGY

The project involves detection of lane lines in an image using Python and OpenCV. OpenCV means "Open-Source Computer Vision", which is a package that has many useful tools for analyzing images.

2.1 The Canny Edge Detection Technique:

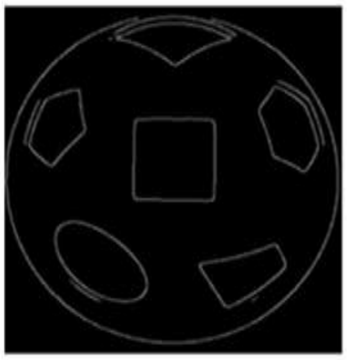
The goal of edge detection is to identify the boundaries of objects within images. A detection is used to try and find regions in an image where there is a sharp change in intensity. We can recognize an image as a matrix or an array of pixels. A pixel contains the light intensity at some location in the image. Each pixel's intensity is denoted by a numeric value that ranges from 0 to 255, an intensity value of zero indicates no intensity if something is completely black whereas 255 represents maximum intensity something being completely white. A gradient is the change in brightness over a series of pixels. A strong gradient indicates a steep change whereas a small gradient represents a shallow change.

Strong Gradient 0 → 255


Small Gradient 0 → 15

On the right hand side there is a figure of the gradient of the soccerball. The outline of white pixels corresponds to the discontinuity in brightness at the points the strengthen gradient. This helps us identify edges in our image since an edge is defined by the difference in intensity values in adjacent pixels.

Original Image


Gradient Image

And wherever there is a sharp change in intensity (rapid change in brightness) i.e., wherever there is a strong gradient, there is a corresponding bright pixel in the gradient image. By tracing out all these pixels, we obtain the edges. We're going to use this concept to detect the edges in our road image.

We will load and read our image into an array:

image = cv2.imread('test_image.jpg')

In order to convert the image to 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)

Now the image is converted to grayscale:

2.2 Gaussian blur:

Each of the pixels for a grayscale image is described by a single number that describes the brightness of the pixel. In order to smoothen an image, the typical answer would be to modify the value of a pixel with the average value of the pixel intensities around it. Averaging out the pixels to reduce the noise will be done by a kernel. This kernel of normally distributed numbers(np.array([[1,2,3],[4,5,6],[7,8,9]]) is run across our entire image and sets each pixel value equal to the weighted average of its neighboring pixels, thus smoothening our image. In our case we will apply a 5x5 Gaussian kernel:

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

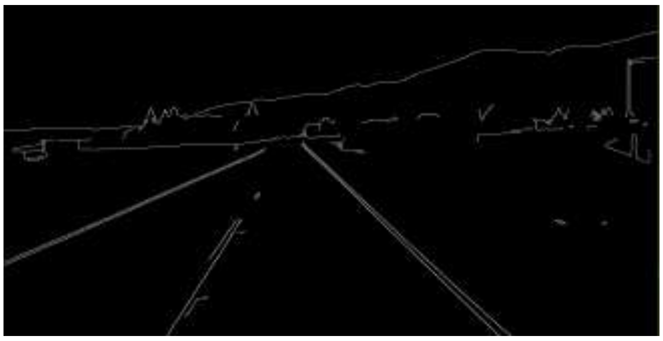
Below is the image with reduced noise:

2.3 Edge Detection:

An edge corresponds to a region in an image where there is a sharp change in the intensity/colour between adjacent pixels in the image. A strong gradient is a steep change and vice versa is a shallow change. So in a way we can say an image is a stack of matrix with rows and columns of intensities. This means that we can also represent an image in 2D coordinate space, x axis traverses the width (columns) and y axis goes along the image height (rows). Canny function performs a derivative on the x and y axis thereby measuring the change in intensities with respect to adjacent pixels. In other words we are computing the gradient (which is change in brightness) in all directions. It then traces the strongest gradients with a series of white pixels.

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

Below is the image after applying the Canny function:



The low_threshold, high_theshold allow us to isolate the adjacent pixels that follow the strongest gradient. If the gradient is larger than the upper threshold then it is accepted as an edge pixel, if it's below the low threshold then it is rejected. If the gradient is between the thresholds then it is accepted only if it's connected to a strong edge. Areas where it's completely black correspond to low changes in intensity between adjacent pixels whereas the white line represents a region in the image where there is a high change in intensity exceeding the threshold.

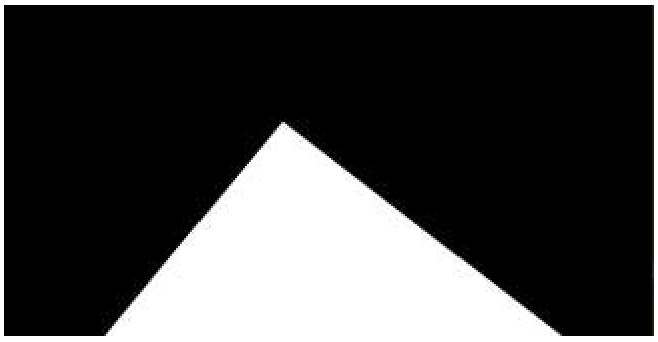
2.4 Region of Interest:

The dimensions of the image are chosen which will contain the road lanes and mark it as our region of interest or the triangle. Then a mask is created which is same as the dimension of the image which would essentially be an array of all zeros. Now we fill the triangle dimension in this mask with the intensity of 255 so that our region of interest dimensions are white. Now I will do a bitwise AND operation with the canny image and the mask which will result in our final region of interest.

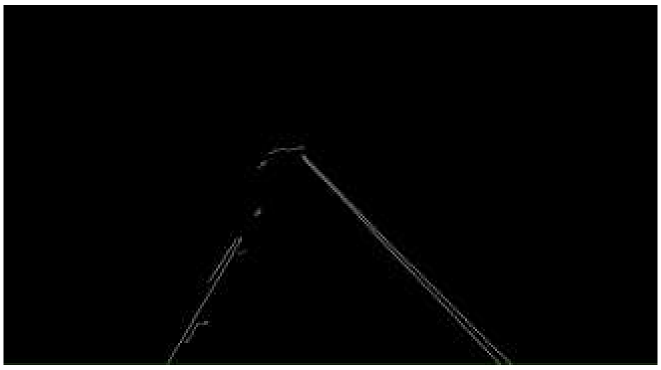
```
def region(image):

    height= image.shape[0]
    poly= np.array([[(200,height),(1100,height),(550,250)]])
    mask= np.zeros_like(image)
    cv2.fillPoly(mask, poly,255)
    masked_image= cv2.bitwise_and(image,mask)
    return masked_image
```

Below is the image of the mask:



Below is the image after doing a bitwise operation with the canny image and the mask, also called the masked_image:



2.5 Hough Transform:

Now we make use of hough transform technique that will detect straight lines in the image and thus identify the lane lines. We know that a straight line is represented by the below equation:

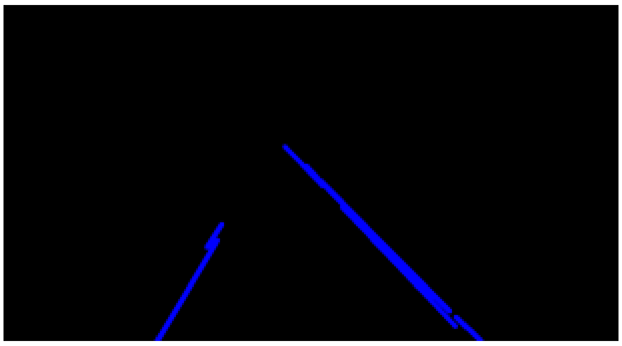
y= mx + b

And the slope of the line is simply a rise over run. If the y intercept and slope is given then the line can be plotted in the Hough Space as a single dot. There are many possible lines that can pass through this dot each line with different values for M and B. There are many possible lines that can cross each point individually, each line with different slope and y intercept values. However there is one line that is consistent with both points. We can determine that by looking at the point of intersection enough space because that point of intersection in Hough Space and that point of intersection represents the M and B values of a line consistent with crossing both the points. Now in order to identify the lines, we will first split our Hough space into a grid. Each bin inside the grid corresponding to the slope and y intercept value of the line. For every point of intersection in a Hough Space bin we're going to cast a vote inside of the bin that it belongs to. The bin with maximum number of votes will be our line. But as we know that the slope of a vertical line is infinity. So to express vertical lines, we will use polar coordinates instead of cartesian coordinates. So the equation 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)
```

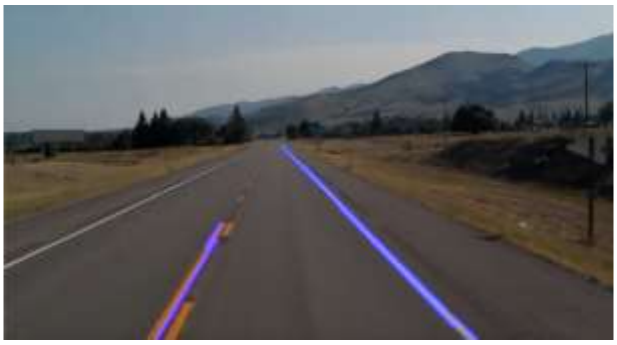
Below is the image of the blue coloured lines drawn on a zero-intensity image:



Now we will combine our zero intensity image with our lane_image:

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

Below is the combined image:



3. CONCLUSION

In the methodology, we made use of the OpenCV library and its functions such as the Canny Function through which we achieved edge detection. Then we prepared a mask of zero intensity and mapped our region of interest by performing the bitwise operation. Then we used the Hough Transform technique that detected the straight lines in the image and identified the lane lines. We made use of the polar coordinates since the Cartesian coordinates don't give us an appropriate slope of vertical and horizontal lines. Finally, we combined the lane image with our zero-intensity image to show lane 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".