ROAD LANE-LINE DETECTION FOR ADVANCED DRIVING ASSISTANCE USING PYTHON, OPENCV

A PROJECT REPORT

Submitted by

CHINTHAM REDDY LALITHADITYA [211420104049]

in the partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University,Chennai)

MARCH 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this Project report "**ROAD LANE-LINE DETECTION FOR ADVANCED DRIVING ASSISTANCE USING PYTHON, OPENCV**" is the bonafide work of **CHINTHAM REDDY LALITHADITYA [211420104049]** who carried out the project work under my supervision

Signature of the HOD with date

Dr.L.JABASHEELA M.E.,Ph.D.,

Professor and Head,

Department of Computer Science and Engineering,

Panimalar Engineering College,

Chennai – 123

Signature of the Supervisor with date

S.SOPHANA JENNIFER M.E..

Assistant Professor

Department of Computer Science and Engineering,

Panimalar Engineering College,

Chennai – 123

Submitted for the Project Viva-Voice examination held on .....

INTERNAL EXAMINER **EXTERNAL EXAMIER**

DECLARATION BY THE STUDENT

I , **CHINTHAM REDDY LALITHADITYA (211420104049)** hereby declare that this project report titled **"ROAD LANE-LINE DETECTION FOR ADVANCED DRIVING ASSISTANCE USING PYTHON, OPENCV"**, under the guidance of **Mrs.SOPHANA JENNIFER S, M.E**., is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

CHINTHAM REDDY LALITHADITYA [211420104049]

ACKNOWLEDGEMENT

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D.,**for his benevolent words and fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express my heartfelt thanks to **Dr. L. JABASHEELA,M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of the project.

We would like to express our sincere thanks to **Dr. N.PUGHAZENDI, M.E., Ph.D.,** and **Mrs**.**SOPHANA JENIFFER S, M.E** and I also thank my parents, friends and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

CHINTHAM REDDY LALITHADITYA (211420104049)

ABSTRACT

Self-driving vehicles have expanded dramatically over the last few years. The introduction of autonomous vehicles will alter human existence. Large datasets and powerful computers are needed for such systems. An emulator exists that makes it simple to produce the desired number of photos of a moving vehicle. Based on only the available photographs, the challenge tried to anticipate and detect traffic lanes. We use our eyes to choose where to go when we are driving. The lines on the road show us where the lanes are, and we can guide the car using them as a constant reference. Naturally, one of the first features we would like to build into a self-driving car is the ability to detect and recognise lane lines using an algorithm. The problem of lane detecting is difficult to tackle. It has long caught the interest of the computer vision community. Lane detection, which has proven to be challenging for computer vision and machine learning algorithms to tackle, is fundamentally a multi-feature detection problem. We offer a technique based on image processing that uses Canny Edge Detection and region masking.

A growing technology used in cars to enable autonomous navigation is lane detecting. The majority of lane-detection systems are built for properly designed roads and rely on the presence of markings. The main drawback of these methods is that they can yield incorrect results or fail to function at all when there are indistinct markings or none at all. This paper reviews one such method for spotting lanes on an unmarked road before moving on to a better method. Both methods only use data from vision or cameras and are based on digital image processing techniques. The primary goal is to acquire a real-time curve value that will let the driver or autonomous vehicle make necessary turns and stay on the road.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

ABBREVIATION	DEFINITION
LDA	Lane Detection Algorithms
ROI	Region of Interest
VioLET	Video-Based Lane Estimation and Tracking
RGB	Red, Green, Blue
SHT	Standard Hough Transform
EGFO	Evolutionary Gabor Filter Optimization
SVM	Support Vector Machines

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Finding sufficient real-world data to be fed into the potent deep learning algorithms that are needed to carry out tasks like lane detection is one of the key challenges in edge detection in self -driving automobiles. For it to develop the algorithms for self-driving vehicle applications, a significant amount of data must be gathered and labeled. Collecting and labeling real-world data takes time and money, and it is impractical to test every conceivable event, such as a car smashing at high speed into a brick wall. For self-driving automobiles, the issue of road lane detection and signal detection is to automatically identify lanes and traffic signs. The ability to recognize traffic signs and road tracks from video frames throughout the self-driving process is entirely owing to advances in image processing and deep learning. In this study, the vehicle incorporates YOLO version 1 for object detection, a polynomial regression model with thresholding for lane guidance, and a controller to coordinate data across the systems.

The suggested methods can be applied to steering suggestion, object identification, object location detection (left, front, or right), and road lane guidance. A crucial challenge is the detection and identification of objects in real time. A notable instance of a security failure involves the 2016 Tesla autopilot accident, in which the vehicle's sensors were mixed by sunlight and the system did not recognize the truck approaching from the right, resulting in the crash. Real-time identification and detection of objects is a critical task. The Tesla auto-pilot disaster in 2016 is a well-known example of a security failure. In that incident, the vehicle's sensors were masked by the sun and the algorithm failed to detect a truck approaching from the right, which resulted in the collision.

With the expansion of knowledge and communication technology as well as vehicle design, the competition to develop and commercialize intelligent and autonomous cars has become increasingly fierce. The comfort of the driver, the steadiness of the automobile, and improved traffic efficiency are all features of these vehicles. Through a human-machine interface, the advanced driving assistance system (ADAS) assists drivers by using the lane departure warning system, lane keeping assistance system (LKAS), front collision warning system, and smart parking assistant system (SPAS).

CHAPTER 2

LITERATURE REVIEW

[1] Tullimalli Sarsha Sree and Sandeep Kumar Sathapathy, In order to locate the lane lines along the road, we're employing a software programme that we are currently working on. The development of algorithms for self-driving cars depends on their capacity to distinguish and follow lanes. Here, we'll estimate the price of developing a software pipeline for tracking traffic lanes using computer vision techniques. We'll approach this task using two different approaches. They are the hough transform technique and convolutional neural networks (CNN). However, for safe driving, lane markings may be a vital point of reference. This work suggests a modified Hough transform-supported lane line recognition algorithm to increase the accuracy and speed of lane line recognition. The threshold is automatically selected for threshold processing based on edge information once the Canny operator has located the edge of the image's region of interest. Three limitations from the angles and lane width are suggested to enhance the Hough transform's recognition of lane lines. As a result, rectilinear regression is used to fit the proper lane lines.

[2] Nidhi Lakhani, Ritika Karande, Deep Manek, Vivek Ramakrishnan, The rapid societal development has led to the rise of the automobile as a mode of mobility. On the congested road, there are a rising number of different kinds of automobiles. Intelligent car systems have made use of lane detection, a hot topic in the fields of computer vision and machine learning. It is a new field that has applications in the business sector. The vehicle's position and trajectory with respect to the lane are reliably approximated by the lane detection system, which uses lane markers in a complex environment. The lane exit warning system heavily relies on lane detection at the same time. The two fundamental components of lane detection are detection of edges and line detection. In the method of lane detection, line detection is just as crucial as edge detection. The Hough transform and convolution-based methods are the most often used lane line detectors. The process of recognizing lane markings on the road and then presenting the positions to an intelligent system is known as lane detection. In intelligent transportation systems, intelligent cars work together with the infrastructure to create a more secure atmosphere and better traffic conditions. The use of a lane detection system can range from straightforward jobs like pointing out lane positions to the user on an external display through more complex ones like anticipating a lane change in an instant to avoid colliding with other vehicles.

[3] Iftikhar Ahmad , Jin Ho Lee , and Soon Ki Jung, Every day, humans make a variety of decisions, many of which are influenced by the sensory data we gather from our environment. The majority of this perception when related to driving is visual. Autonomous vehicles, commonly referred to as self -driving automobiles, are built to be able to recognise items in their environment and decide how to react to them quickly. This creates a number of computer vision issues for autonomous vehicles, including identifying pedestrians, other vehicles, lanes, and traffic signs. This study specifically addresses lane detection, which is a vital component of a vehicle's movement planning. In this approach,a lane detection method suggested for the Asphalt 8: Airborne real-time racing game utilising the Canny edge detection technique. The programming language's Python Imaging Library (PIL)'s ImageGrab module is used to access the game's screen. OpenCV is used to determine the lanes' boundaries using Canny edge detection, a popular technique in computational image processing, and a masking algorithm was employed to remove obstructions including trees, rocks, and cables. The game's lanes were marked and drawn using the Hough Transform. The gaming environment features real physics, graphics, and a range of settings, including lane-keeping aids like abrupt corners, slopes, and various weather conditions.

[4] Raja Muthalagu, Anudeep Sekhar Bolimera, Dhruv Duseja, Shaun Fernandes, In the modern world, people spend a lot of time driving or dealing with other automobiles while walking on the streets. 1.35 million people worldwide lose their lives in traffic accidents every year, and every day, up to 3,700 people die in collisions involving buses, lorries, cars, motorbikes, bicycles, or people (Singh, 2015). Worldwide, the number of cars on the road is also rising quickly (Bellis et al., 2008). Researchers have found that the majority of traffic accidents are caused by human error. A fully autonomous self-driving automobile is being developed as a result of the Advanced Driver Assistance Systems (ADAS) that have been created in recent years to improve passenger safety and comfort (Lu et al., 2005; Bengler et al., 2014; Liyong et al., 2020). Perception, planning, and control are the three main components that make up self -driving car technology. Researchers have put in a lot of effort to create new methods for increasing driving safety and lowering traffic accidents. The majority of ADAS systems use a range of sensors to detect lanes and objects on the highway. The detection of the lanes and objects is proposed using a variety of camera vision techniques.

[5] Vighnesh Devane, Ganesh Sahane, Hritish Khairmode, Gaurav Datkhile, For lane curve fitting, the method [4] proposes an effective variation of the sliding window algorithm. This enhanced variant has the benefit of allowing sliding windows to be used even on uneven lane markers. In the second way, the road borders from the binary image are used to apply this technique. The curve value and the car's offset

are then calculated using the geographic coordinates of the roadway boundaries. The overall goal of the project is to make lane detection possible on roads with limited visibility or worn-out lane markings. With the aid of image warping, thresholding, and techniques like pixel summation and sliding window algorithm, this research attempts to build lane detection for an efficient autonomous car. Finally, the benefits and drawbacks of the aforementioned techniques as well as the most appropriate application scenarios have been discussed.

[6] Jamel Baili, Mehrez Marzougui, Ameur Sboui, Samer Lahouar, Mounir Hergli, Lane detection has been the subject of numerous studies in image processing. A summary of these studies reveals that they can be divided into two main groups. In the first, markers for lanes in an input picture from a rear-view camera are recognised using the bird's-eye view transform. The second group makes advantage of the front-mounted camera. In the latter case, various image processing methods have been developed, such as the Probability of Picture Shape (LOIS) algorithm, the B-Snake technique, and others. These algorithms employ a variety of ways to extract features from an image, such as edges, using a featurebased approach. The suggested method has performed well, but it has to be tweaked to accommodate inclement weather (such as rain and snow) and poor lighting (nighttime). We intend to develop an LDW System in an integrated processor in future work and assure a robust monitoring mode using information from TLC calculation.

[7] Yan Liu , Jingwen Wang , Yujie Li , Canlin Li, This paper proposes a lane detection system to address the challenge of lane detection in hazy settings. The following list highlights the main contributions made by this study.

1. This page includes a dataset of blurred lane lines.

2. To improve the characteristics of the lanes & increase the effectiveness of blurring lane recognition in complicated road settings, an upgraded GAN is utilised.

3. The proposed method outperforms existing state-of-the-art detectors in high speed and difficult road conditions (line curves, dirty lane line, illumination change, occlusions), resulting in a significant improvement over the current state-of-the-art detectors.

[8] Zequn Qin, Huanyu Wang, and Xi Li, Applying global image features-based row-based selecting to lane detection. In other words, our approach involves leveraging the global features to choose the appropriate lanes for each predefined row. Lanes are modeled in our formulation as row anchors are a

sequence of horizontal points in predetermined rows. The initial stage in representing places is gridding. The location is separated into several cells on each row anchor. This makes it possible to compare the identification of lanes to the selection of specific cells over predetermined row anchors. According to the suggested definition, lane detection is a row-based selection problem that needs to be solved using global features. The issue of speed and no-visual-clue can be solved in this way. It is also suggested to employ structural loss to explicitly model lane prior knowledge. Both the qualitative and quantitative testing support the value of our approach and the structural loss. Particularly, our model with the Resnet-34 backbone could reach the highest levels of accuracy and speed. Even at the same resolution, a lightweight Resnet-18 version of our approach was capable of 322.5 FPS with a competitive performance.

[9] Ling Ding, Huyin Zhang, Jinsheng Xiao, Cheng Shuand Shejie Lu, The approach has its roots in a roadway segmentation and lane detection technique and incorporates the cavity convolution used in DeepLabv1 and LMD algorithms as well as the discriminant loss function used in LaneNet. To increase the receptive field, void convolution is used in place of the extracted feature part's common convolutional layer. The discriminant loss function stands out due to its simplicity of integration into various network structures and the fact that instance segmentation is accomplished through post-processing. Additionally, some works on denoising images initial processing have been discussed in earlier articles. The correctness and robustness in the suggested method are confirmed after being tested on different data sets and lanes in various weather conditions. The algorithm has a significantly slower detection speed but a much higher detection accuracy, especially when it comes to the recognition of corners and false actual lane lines. In comparison, the deep learning algorithm can produce precise detection and has none of these issues when it comes to effective detection.

[10] Y. Wang, E. K. Teoh and D. Shen, The lane model is essential for lane detecting. The lane modelling must make a few assumptions about the real structure of the road in order to fully recover 3D data from the 2D static image. As we focus on creating the 2D lane model in this work, both sides of the road borders are taken to be in line on the ground plane. A fresh B-Snake-inspired lane model has been developed to describe the viewpoint effect that results from general lane borders (or markings). It can depict a wider range of lane structures than other lane models, like straight and parabolic models. Here, the challenge of finding the middle of the lane and the challenge of detecting the two ends of the lane.

S.No	Authors	Advantages	Disadvantages
1	Tullimalli Sarsha Sree and Sandeep Kumar Sathapathy	This study proposes a lane line identification algorithm supported by modified Hough transform to improve the precision and real-time efficiency of lane line detection.	More Image Focused than Video graphic Input
2	Nidhi Lakhani, Ritika Karande, Deep Manek, Vivek Ramakrishnan	It is a new field that has applications in the business sector. The vehicle's position and trajectory with respect to the lane are reliably approximated by the lane detection system, which uses lane markers in a complex environment.	Time Consuming in video graphic inputs
3	Raja Muthalagu, Anudeep Sekhar Bolimera, Dhruv Duseja, Shaun Fernandes	Accurate Results	To combat overfitting, initially the use of Dropout was made. However, the performance was much worse.
4	Vighnesh Devane, Ganesh Sahane, Hritish Khairmode	Overall purpose of this project is to enable lane detection in poor road conditions	More Image Focused than Video graphic Input
5	Jamel Baili, Mehrez Marzougui, Ameur Sboui, Samer Lahouar, Mounir Hergli	Different algorithms for image processing have been created in this latter situation, including the Likelihood of Picture Shape algorithm, the B-Snake algorithm etc.	It has to be tweaked to accommodate inclement weather (such as rain and snow) and poor lighting (nighttime).
6	Yan Liu , Jingwen Wang , Yujie Li , Canlin Li	Aiming at the difficulty of lane detection in blurred scenarios, a lane detection networks a blurred image is proposed in this article.	Time Consuming in video graphic inputs
7	Zequn Qin, Huanyu Wang, and Xi Li	Feature aggregation method for high-level semantics and lowlevel visual information is also depicted.	Proposed formulation regards lane detection as a problem of row-based selecting
8	Ling Ding, Huyin Zhang, Jinsheng Xiao, Cheng Shuand Shejie Lu	To increase the receptive field, void convolution is used in place of the extracted feature part's common convolution layer.	the algorithm is much slower in detection speed
9	Y. Wang, E. K. Teoh and D. Shen	A novel B-Snake inspired lane model has been devised that represents the viewpoint effect of parallel lines.	More Image Focused than Video graphic Input

Table 2.1 - Literature Survey

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

At present, these kinds of lane marking line detection methods based on machine vision and can be divided into two categories:

1. The traditional image processing method and

2. Semantic segmentation (includes deep learning) methods.

3.2 LIMITATIONS OF EXISTING SYSTEM

All the existing cars does not have ADAS (Advanced Driver Assistance System)

• When affected by other markers on the driveway, or when the lane is damaged or the pollution is serious, the image of the lane line collected by the system is incomplete and of poor quality, making it difficult for the system to accurately judge and analyze the data

• Lane detection in complex road conditions is susceptible to external environmental factors, such as lack of light, road tortuosity and vehicle obstruction.

• While existing lane recognition algorithms demonstrated over 90% detection rate, the validation test was usually conducted on limited scenarios.

3.3 PROPOSED SYSTEM

Vehicles equipped with intelligent transportation system we proposed Heuristic Algorithm to detect lanes. Firstly, we describe Hough-based detection and Color-based detection individually and look at their results. To do Hough-based detection, we send image to convert into gray scale, applying Region of Interest (ROI), using global thresholding, using edge detection, and finally find lines by Hough Line Transform. To do Color-based detection, we send captured images to color segmentation and noise reduction step. In Color-based detection, we extract more data from lines and their boundaries.

We have developed a simple heuristic method for detection of lanes in video. In Heuristic Method a clustering methodology is used to group the detected points and a best fit line in the Mean Square Squares sense is then fitted to the remaining points. In lane recognition and departure warning systems, accurate detection of lane roads is crucial. When a vehicle breaches a lane boundary, vehicles equipped with the predicting lane borders system direct the vehicles to avoid crashes and issue an alarm.

CHAPTER 4

REQUIREMENT SPECIFICATION

Vehicles equipped with intelligent transportation system we proposed Heuristic Algorithm to detect lanes. Firstly, we describe Hough-based detection and Color-based detection individually and look at their results. To do Hough-based detection, we send image to convert into gray scale, applying Region of Interest (ROI), using global thresholding, using edge detection, and finally find lines by Hough Line Transform. To do Color-based detection, we send captured images to color segmentation and noise reduction step. In Color-based detection, we extract more data from lines and their boundaries. Next, we describe our proposed method, which uses the extracted data from video stream and improves the efficiency of lane detection. The result of lane detection using Heuristic Algorithm is considerably better than method that is just based on Hough Line Transform. We have developed a simple heuristic method for detection of lanes in video. In Heuristic Method a clustering methodology is used to group the detected points and a best fit line in the Mean Square Squares sense is then fitted to the remaining points.

ane detection is one of the methods which uses the principle of vision-based lane detection. As the name itself indicates, it is a process of detecting as well as recognizing the lanes where the ground traffic circulates. For advanced driving assistance, lane detection is one of the essential functions. The lane detection has become very specific term that implies the utilization of certain perceptive sensors, certain processing units, and certain algorithms to perform this functionality. The lane detection is a process which must be effective with the following. There are many factors which affect lane detection. The Good quality of lane should not be affected by shadows of which can be caused by appearances of trees, buildings and other aid boards, the existences of surrounding object, the change of light condition, the dirt left on the road surface etc. We humans has still some problems in detection of road lanes marks, detection should also have to assume the curved roads instead of assuming only that the roads are straight.

A real time vision-based lane detection method is presented to find the position and type of lanes in each video frame proposed a method for lane detection effective combination of filter functions edge-Link channels. The first filter means candidates are sought in the region of interest (ROI). During the research, a broad edge linking algorithm circuit slot marginal land was used to produce the filter width for wider access board and serves to research the edge

orientation and tape are used to filter the channels marked border pair link candidates. A linear model-based method has been developed for detecting the tracking markers in real time. To estimate the linear model is robust filtering capabilities such as efficient roads and edges, color, width and direction are combined to follow the markings on the parameters of the linear model is used to represent traces are calculated. Lane position can be determined from the linear model parameters and Lane Departure can be calculated.

4.1 HARDWARE AND SOFTWARE SPECIFICATION

4.1.1 HARDWARE REQUIREMENTS

- Laptop with windows 7 (min)
- Camera
- 6 GB Ram

4.1.2 SOFTWARE REQUIREMENTS

- TensorFlow (Machine learning frameworks)
- Python
- Open CV
- Canny edge detector

4.2 TECHNOLOGIES USED

- Python Programming language.
- OpenCV stands for “Open-Source Computer Vision”, which is used for analysis of images.
- Using computer vision techniques in Python, we will identify road lane lines.
- Pre-process image using grayscale and gaussian blur.
- Detecting road lanes can be done using Canny Edge Detector and Hough Transform.

4.2.1 GAUSSIAN BLUR

A grayscale image's pixels are each defined by a single number that indicates the pixel's brightness. The common solution for smoothing an image is to change the value of a pixel with the average value of the pixel intensities around it. A kernel will average out the pixels in order to reduce noise. This kernel of normally distributed numbers (np.array ([[1,2,3],[4,5,6],

[7,8,9]]) is applied to our entire image, smoothing it out by setting each pixel value to the weighted average of its neighbors. In our case we will apply a 5x5 Gaussian kernel: blur= cv2.GaussianBlur(gray_image, (5,5),0)

4.2.2 EDGE DETECTION

An edge is a region in a picture where the intensity/color between neighboring pixels in the image changes dramatically. A steep change is a significant gradient, while a shallow change is the opposite. In this sense, an image may be thought of as a matrix with rows and columns of intensities. This means that a picture can also be represented in 2D coordinate space, with the x axis traversing the width (columns) and the y axis traversing the image height (rows). The Canny function measures the change in brightness between neighboring pixels by performing a derivative on the x and y axes. In other words, we're calculating the gradient (or brightness change) in all directions. It then traces the strongest gradients with a series of white pixels.

canny_image = cv2.Canny(blur, 100, 120)

We can separate nearby pixels that follow the strongest gradient using the low_threshold and high_theshold functions. It is allowed as an edge pixel if the gradient is more than the upper threshold; if it is less than the lower threshold, it is rejected. If the gradient falls between the criteria, it is only approved if it is linked to a strong edge. The white line depicts a location in the image where there is a high change in intensity above the threshold, whilst the fully black areas relate to low variations in intensity between adjacent pixels.

4.2.3 ADD-ONS FOR ADAS

ADAS are required because of the traffic increase and by national and international regulations. ADAS applications may avoid accidents and any concomitant injuries or possible fatalities. ADAS use surrounding sensors such as radar, infrared, video or ultrasound to monitor and analyses a vehicle's environment. Various companies such as Continental, Delphi Automotive, Bosch, Freescale, Texas Instruments and many other suppliers provide different types of ADAS solutions for End-User Applications. The SRL sensor is an infrared laser sensor that works on the principle of the pulse transit time method. The distance to the reflecting object (accuracy: ± 0.1 m ± 10%) can be determined without contact. The SRL sensor uses three independent channels and in addition to the distance also the speed (accuracy: ± 2 km/h ± 10%) of multiple objects can determine.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 METHODOLOGY

Vehicles equipped with intelligent transportation system we proposed Heuristic Algorithm to detect lanes. Firstly, we describe Hough-based detection and Color-based detection individually and look at their results. To do Hough-based detection, we send image to convert into gray scale, applying Region of Interest (ROI), using global thresholding, using edge detection, and finally find lines by Hough Line Transform. To do Color-based detection, we send captured images to color segmentation and noise reduction step. In Color-based detection, we extract more data from lines and their boundaries. Next, we describe our proposed method, which uses the extracted data from video stream and improves the efficiency of lane detection.

The result of lane detection using Heuristic Algorithm is considerably better than method that is just based on Hough Line Transform.

We have developed a simple heuristic method for detection of lanes in video. In Heuristic Method a clustering methodology is used to group the detected points and a best fit line in the Mean Square Squares sense is then fitted to the remaining points.

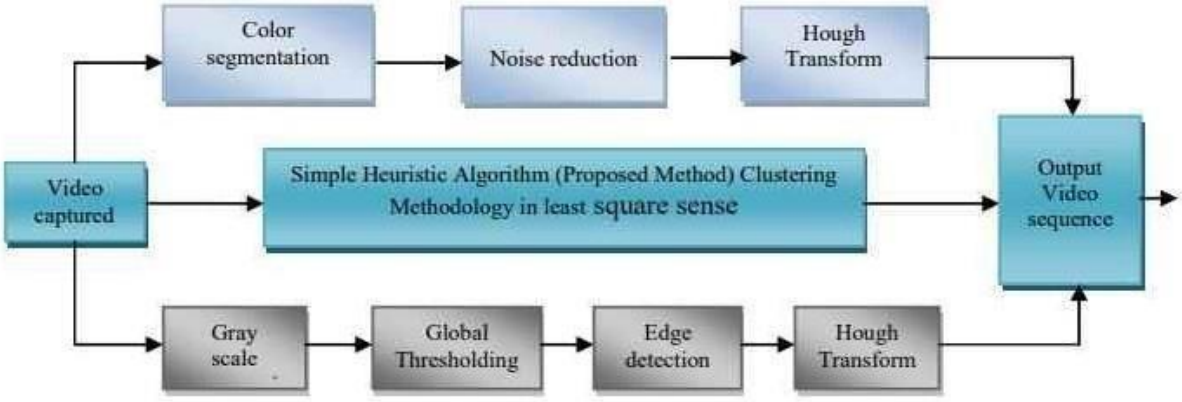


Fig 5.1.1- Block Diagram of lane detection methodology

5.1.1 Hough-based Detection

In this section we will explain the lane detection by using gray scale method and edge detection and will examine the results by using this method. The Hough based detection includes the four parts as shown in the gray block diagram.

5.1.2 Defining and Conversion of ROI into Gray Scale

In this step, the captured color image is converted to gray scale to make method faster, less computational, and less sensitive to scene condition. In our proposed method, captured series of images received from a camera on top of a car would be processed. The camera is adjusted in a way that the vanishing point of road should be placed on the top of ROI. Based on camera place adjustment, only part of the bottom of the captured image would be valuable for processing and it causes short time processing and less memory usage.

Fig 5.1.2- Converting to Gray Scale and ROI Selection

5.1.3 Global Optimum Thresholding

Single global thresholding does not effectively segment an image containing phenomenon like illumination. By using Optimum Global Thresholding with the help of Otsu's method, we could solve this problem of illumination, and make the method less sensitive to scene conditions. The main principle of Otsu's method is choosing threshold to maximize the interclass variance of the black and white pixels [46]. The interclass variance is given as Eq1.

$$\partial_{B=}^2 P_1\left(m_1 - m_g\right)^2 + P_2\left(m_2 - m_g\right)^2 \ldots\ldots\ldots(3.1)$$

Here,

$P_i = Probability\ of\ pixel\ associated\ with\ class\ C_i$

$m_i = Mean\ value\ of\ pixel\ associated\ with\ class\ C_i$

$m_g = Global\ Mean\ value$

In the optimum thresholding method, we convert an intensity image to binary image, which is fed as the input to the edge detection step. There have been many edge detectors, canny, Sobel etc., but here we will deal with Canny edge detectors.

5.1.4 Canny Edge Detection

By applying the optimum global thresholding to selected part of image, ROI, we have binary image as the input for this step. In this step, to find lane boundaries in the image we use one of the edge detection methods called Canny Edge Detection and the detected boundaries are shown in figure.

Fig 5.1.3 - Canny edge detection

Canny Edge detector is most used for step edges due to Optimal, then is corrupted by white noise. The objective is the edges detected must be as close as possible to the true edges the number of local maxima around the true edge should be minimum.

Canny edge detection basically uses gradient vector of an intensity image. Lane boundaries have high contrast in the image, and this feature yields high values of gradient vector by which we can find the edge direction, which is orthogonal to gradient vector. Many edge detection methods are based on this principle, but the efficiency levels are different. One of the best and most efficient methods is canny edge detection. The most important characteristics of canny method are that the error rate of this method is low because this algorithm uses double thresholding, hysteresis thresholding. Hysteresis threshold, double thresholding, suppresses the pixels that are not related to edges. Therefore, the detected edge is close to true place. We should also mention that canny edge detectors are very sensitive to noise; therefore, we smooth the image with a low pass filter to reduce the effect of noise.

Let $f(x,y)$ denote the input image $G(x,y)$ denote the Gaussian function than,

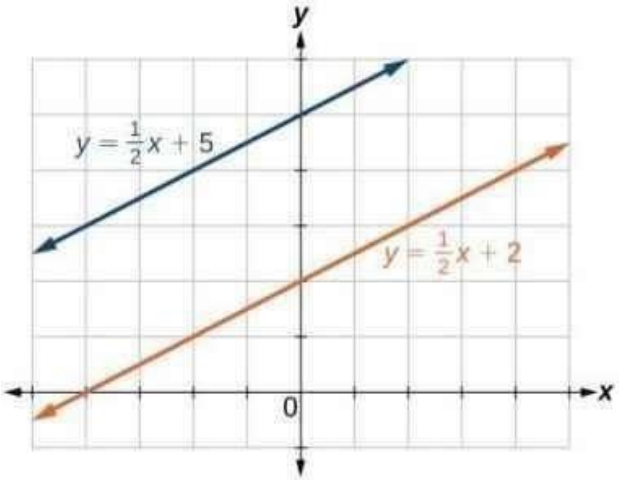
$$G(x,y) = \frac{e^{-(x^2+y^2)}}{2\partial^2} \quad \ldots\ldots\ldots (3.2)$$

We form a smooth image $fs(x,y)$ by convolving G and $f : fs(x,y) = G(x,y) * f(x,y)$

5.1.5 Hough Line Transform

"The Hough transform is a general technique for identifying the Locations and orientations of certain types of features in a digital Image. Developed by Paul Hough in 1962 and patented by IBM, the transform consists of parameterizing a description of a feature at any given location in the original image's space. A mesh in the space defined by this parameter is then generated, and at each mesh point a value is accumulated, indicating how well an object generated by the parameters defined at that point fits the given image. Mesh points that accumulate relatively larger values then that described features may be projected back onto the image, fitting to some degree the features present in the image." A method for finding global relationships between pixels, for example if we want to find straight lines in an image, we apply edge enhancement filter e.g. Laplacian set a threshold for what filter response is considered a true ''Edge Pixel'' extract the pixels that are on straight line using the Hough Transform.

A simple shape is one that can be represented by only a few parameters. For example, a line can be represented by two parameters (slope, intercept) and a circle has three parameters, the coordinates of the center and the radius (x, y, r). Hough transform does an excellent job in finding such shapes in an image. The main advantage of using the Hough transform is that it is insensitive to occlusion. Let's see how Hough transform works by way of an example. In the cartesian plane (x and y-axis), lines are defined by the formula y=mx+b, where x and y correspond to a specific point on that line and m and b correspond to the slope and y-intercept.



A regular line plotted in the cartesian plane has 2 parameters (m and b), meaning a line is defined by these values. Also, it is important to note that lines in the cartesian plane are plotted as a function of their x and y values, meaning that we are displaying the line with respect to how many (x, y) pairs make up this specific line (there is an infinite amount of x, y pairs that makeup any line, hence the reason why lines stretch to infinity). However, it is possible to plot lines as a function of its m and b values. This is done in a plane called Hough Sp.

Fig 5.1.4 - Line Detection Using Hough Line Transform

5.2 COLOR-BASED DETECTION

In Color-based detection, we extract more information about the lines and line boundaries based on their color Information to improve the efficiency of detection. We perform color-based detection on the same ROI shown in Figure. The color-based detection has two parts. The first step is Color Segmentation, and the second step is Noise Reduction.

5.2.1 Color Segmentation

The process that an image is divided into multiple segments is called segmentation. Color segmentation helps us to identify the boundaries and objects in an image based on desired color. Color images can be modeled with many colors' spaces like RGB, CMYK, and HSI. Every color space could be converted to another by using some formulas. In our proposed algorithm, the acquired image is in RGB, and we convert it to HSI color model as we use HSI color model. In lane detection using HSI model, saturation component S and intensity component I are more important than hue component H and give us more consequential information.

The reason that we use HSI color space model with ordinary threshold value in our method is that to detect lanes in lane detection using RGB color space model we should use all three

components R, G, and B, and they are all important for processing to detect the lanes. But when we use HSI color space model, just saturation S and Intensity I are important for processing to detect the lanes because of their considerable variations. This feature causes less computations and we have faster algorithms to detect the lanes. Usually, captured images from the roads contain white lines and gray background of road. High contrast between gray color of road and white color of lines causes higher values of saturation S and intensity I components rather than hue component H. This feature leads us to detect the lanes using HSI color space model based on information of saturation S and Intensity I. The segmented white lines in ROI based on color information of lines are shown in Figure.

Fig 5.2.1 Color Segmentation

5.2.2 Noise Reduction

After detecting the lanes in an image using color segmentation, the segmented image contains noise. Therefore, the proposed algorithm should be tolerant to noise. In this part, we remove noise and make our proposed algorithm noise tolerant. To remove noise from the segmented image, we use Morphological operation. We first do Opening operation to remove small objects that are responsible for the noise, and then Closing operation to make the lane boundaries clearer and softer. Then we put in some operations to remove some small and unrelated objects. The output after removing the noise from segmented image is shown in Figure.

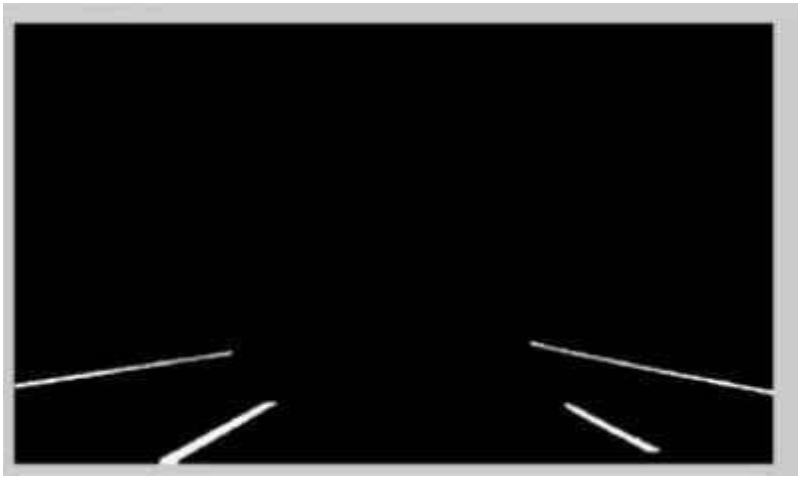


Fig 5.2.2 Noise Reduction

5.3 ADVANTAGES OF METHODOLOGY

This system demands low computational power and memory requirements, and is robust in the presence of noise, shadows, pavement, and obstacles such as cars, motorcycles and pedestrians' conditions. The result images can be used as preprocessed images for lane tracking, road following, or obstacle detection moving across lanes to overtake vehicles and avoid obstacles, searching for the correct and shortest route to a destination.

As we mentioned, Hough-based detection and Color-based detection are efficient, but there are some problems when we use them individually like detecting unwanted lines or not detecting some existing lines for Hough-based detection and sensitive to scene condition for Color -based detection. By using the proposed method, we solve the problem of sensitivity of Color-based detection and detecting missing lines in Hough-based detection. We compare two methods based on our experimental results and tabulate their detection rates in different scene conditions.

CHAPTER 6

PROJECT PURPOSE AND SCOPE

6.1PROJECT PURPOSE

The purpose of a project focusing on lane line detection using Python and OpenCV typically revolves around enhancing road safety, particularly in the context of autonomous vehicles or advanced driver assistance systems (ADAS). Here's a breakdown of its purposes:

1. Lane Keeping Assistance: The primary purpose is to assist drivers in keeping their vehicles within lanes. By detecting lane lines accurately, the system can alert the driver if they're drifting out of their lane, reducing the risk of accidents due to unintentional lane departures.
2. Autonomous Navigation: For autonomous vehicles, lane line detection is crucial for understanding the vehicle's position on the road. It helps the vehicle stay within its lane and navigate safely, especially in scenarios like highway driving.
3. Traffic Sign Detection and Recognition: Lane line detection can also be a part of a broader system that includes traffic sign detection and recognition. Integrating lane detection with traffic sign recognition enhances the vehicle's understanding of road conditions and regulatory signage.
4. Obstacle Detection and Avoidance: Lane line detection can aid in detecting obstacles on the road, such as pedestrians or vehicles. By understanding the lane boundaries, the system can identify potential hazards and take appropriate actions to avoid collisions.
5. Research and Development: Lane line detection projects are often undertaken as part of research and development efforts in the fields of computer vision, machine learning, and robotics. These projects contribute to advancements in technologies related to autonomous driving and intelligent transportation systems.
6. Education and Learning: Lane line detection projects serve as excellent educational tools for learning about computer vision techniques, image processing, and algorithm development using libraries like OpenCV and programming languages like Python. They provide hands-on experience in implementing real-world applications of these concepts.
7. Data Collection and Analysis: Lane line detection projects can also involve data collection and analysis, where real-world driving data is used to improve algorithms and fine-tune parameters for better performance and reliability.

6.2 PROJECT SCOPE

First, one obvious way to make the problem easier is to work out our solution for a single image. A video is, after all, just a series of images. We can then move on to running our pipeline on an input video frame-by-frame as a final solution to the original problem of processing an entire video for lane detection.

the lane markers are obvious to any human observer. We perform processing of this image intuitively, and after being trained to drive a human can detect the lane in which the vehicle appears to be moving. Humans also effortlessly identify many other objects in the scene, such as the other vehicles, the embankment near the right shoulder, some road signs alongside the road, and even the mountains visible on the horizon. While many of these objects are complex in visual structure, it could be said that the lane markers are some of the simplest structures in the image!

Pre-existing knowledge of driving gives us certain assumptions about the properties and structure of a lane, further simplifying the problem. One obvious assumption is that the lane is oriented to be parallel with the direction of movement. This being the case, the lines denoting the lane will tend to extend from the foreground of an image into the background along paths that are angled slightly inwards. We can also assume that the lines will never quite reach the horizon, either disappearing with distance or being obscured by some other image feature along the way.

One very simple filter on the image could be to crop out all the areas which we believe will never contain information about the lane markers. We'll kick off our project by writing the code necessary to do a simple crop of the region of interest.

6.3 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements for road lane line detection using OpenCV encompass aspects related to the system's behavior, performance, usability, reliability, and other quality attributes. Here's a list of typical non-functional requirements for such a system:

1.Performance:

- Throughput: Specify the number of frames or images processed per unit of time (e.g., frames per second).
- Latency: Define the maximum acceptable delay between capturing a frame and producing the lane detection results.
- Resource Usage: Specify CPU, memory, and other resource utilization limits to ensure efficient operation.

2. Accuracy:

- Detection Accuracy: Define the minimum acceptable accuracy level for detecting lane lines, including the tolerance for false positives and false negatives.
- Localization Accuracy: Specify the precision requirement for localizing lane boundaries relative to the vehicle's position.

3. Robustness:

- Environmental Robustness: Specify the system's ability to handle variations in lighting conditions, weather conditions (e.g., rain, fog), road surface conditions, and other environmental factors.
- Noise Robustness: Define the system's tolerance for noise in input data, such as sensor noise or image artifacts.

4. Scalability:

- Road Complexity: Specify the system's ability to handle different types of road environments (e.g., highways, urban streets) and lane configurations (e.g., straight lanes, curved lanes, multiple lanes).
- Camera Setup: Define the compatibility requirements with various camera sensors, resolutions, and configurations commonly used in automotive and surveillance applications.

5. Adaptability:

- Parameter Adaptation: Specify how the system adapts to changes in road conditions, lighting conditions, camera parameters, and other dynamic factors without manual intervention.
- Configuration Flexibility: Define the ease of configuring system parameters and thresholds to accommodate different scenarios and user preferences.

6.Usability:

- User Interface: Define requirements for the user interface, including simplicity, intuitiveness, and accessibility for operators or end-users.
- Diagnostic Information: Specify requirements for providing diagnostic information and feedback to aid in troubleshooting and system monitoring.

7. Reliability:

- Error Handling: Define mechanisms for detecting and handling errors, including error recovery strategies and fault tolerance mechanisms.
- Availability: Specify requirements for system availability, including uptime targets and provisions for maintenance downtime.

8.Security:

- Data Privacy: Define measures to protect sensitive data processed by the system, such as video feeds or vehicle telemetry data.
- Access Control: Specify access control mechanisms to restrict system access and prevent unauthorized use or tampering.

9.Compliance:

- Regulatory Compliance: Ensure compliance with relevant industry standards, regulations, and legal requirements, particularly in safety-critical applications such as autonomous vehicles.

10. Documentation:

- Documentation Standards: Define requirements for system documentation, including documentation formats, completeness, and accessibility.

11. Performance Monitoring:

- Logging and Monitoring: Specify requirements for logging system events, performance metrics, and diagnostic information for monitoring and analysis purposes.

12. Interoperability:

- Integration Compatibility: Define compatibility requirements with other systems, protocols, or interfaces commonly used in the target environment.
- By clearly defining and addressing these non-functional requirements during the design and development phases, developers can ensure that the road lane line detection system meets the necessary quality attributes and performs reliably in real-world scenarios.

CHAPTER 7

SYSTEM DESIGN

7.1 ARCHITECTURE DIAGRAM

Architecture diagramming is the process of creating visual representations of software system components. It plays a crucial role in showcasing the various functions of a software system, their implementations, and how they interact with each other. By providing a bird's-eye view of the system, architectural diagrams facilitate better decision-making and enhance the clarity of complex software structures**.**

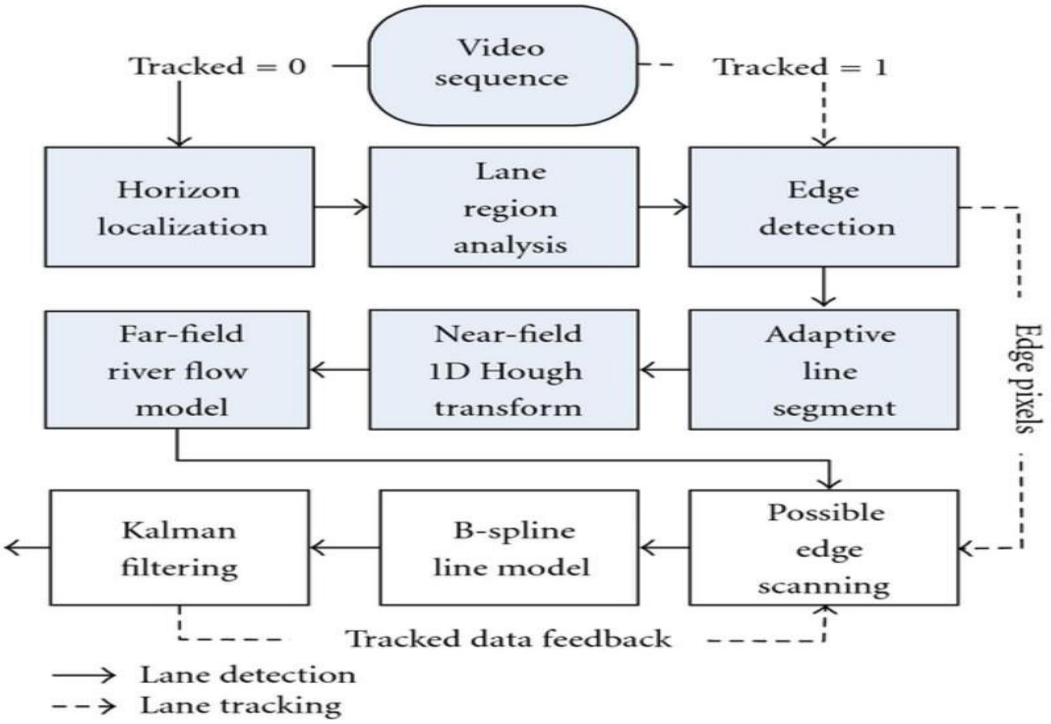


Fig 7.1.1- Architecture Diagram

An input video, collected from a camera attached to the front of the car as the car is driven by the user, is fed to the model in real-time. Frames are extracted from the video by OpenCV according to the required frames per second. The extracted frames are then pre-processed and passed through two models simultaneously: the object detection model and the lane detection model. The output consists of the detection of obstructions by instance segmentation and the highlighting of lane lines.

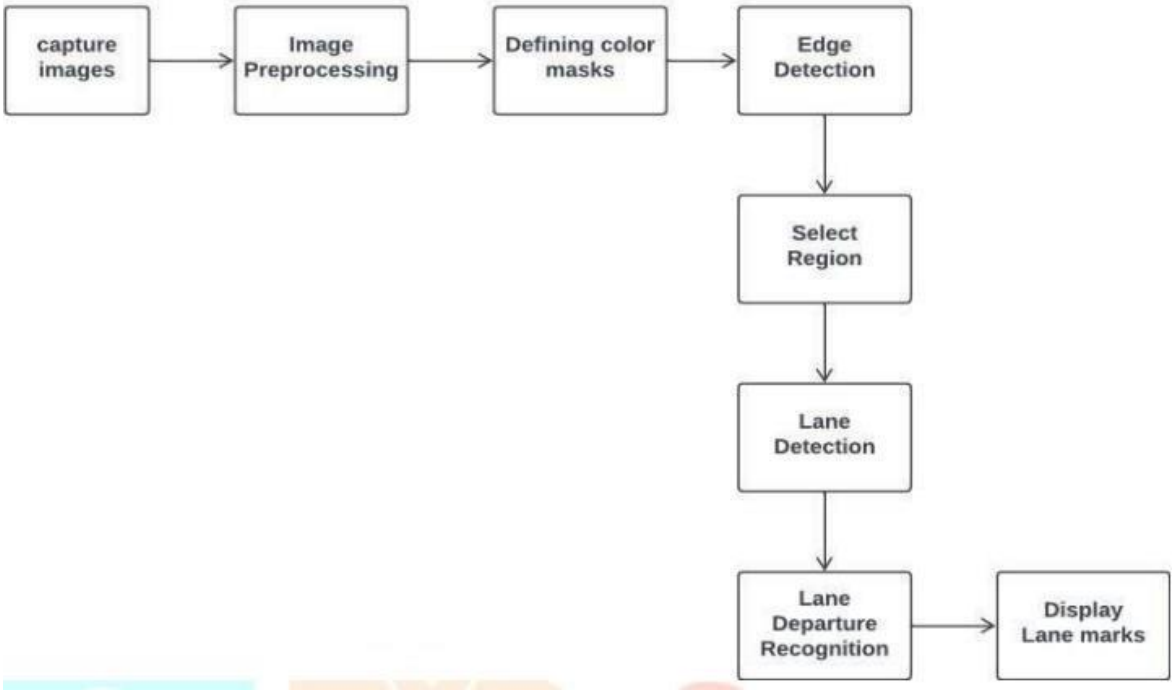


Fig 7.1.2 - Block Diagram

7.2 SEQUENCE DIAGRAM

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

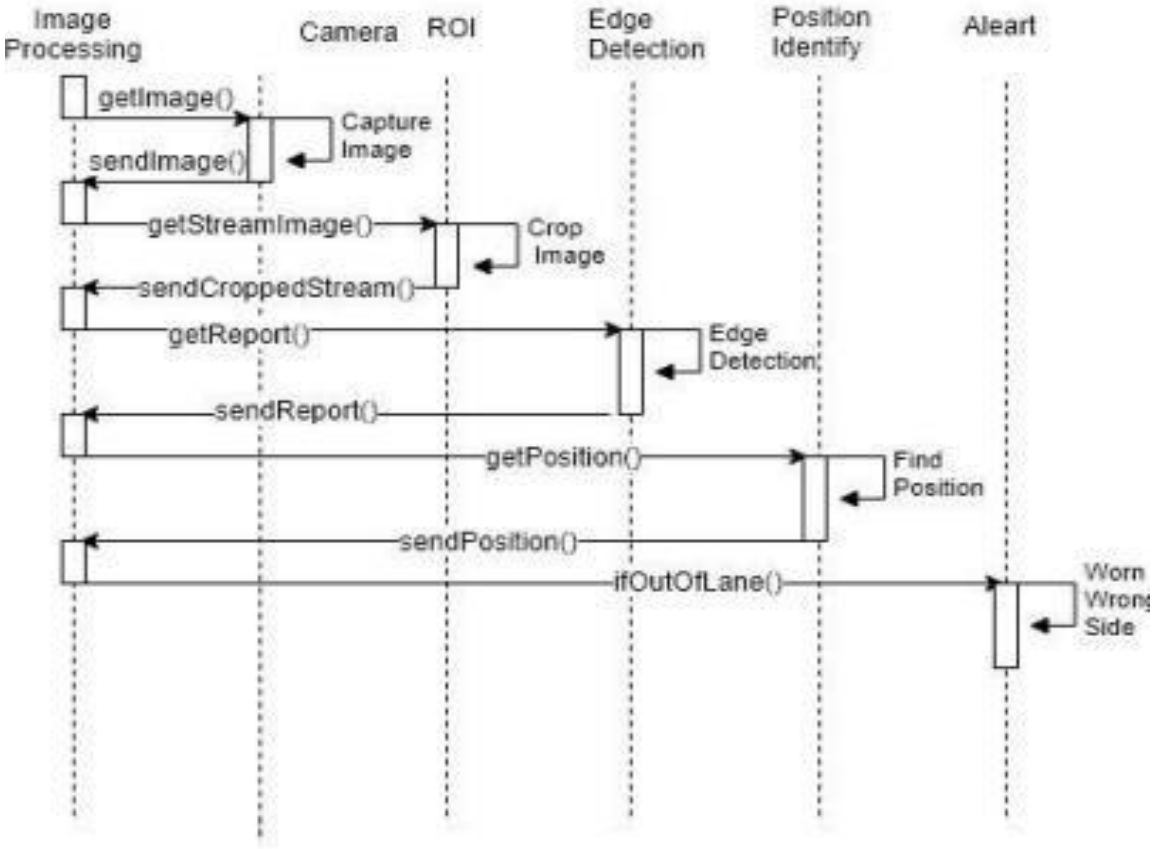


Fig 7.2.1 - Sequence diagram

7.3 USECASE DIAGRAM

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development.

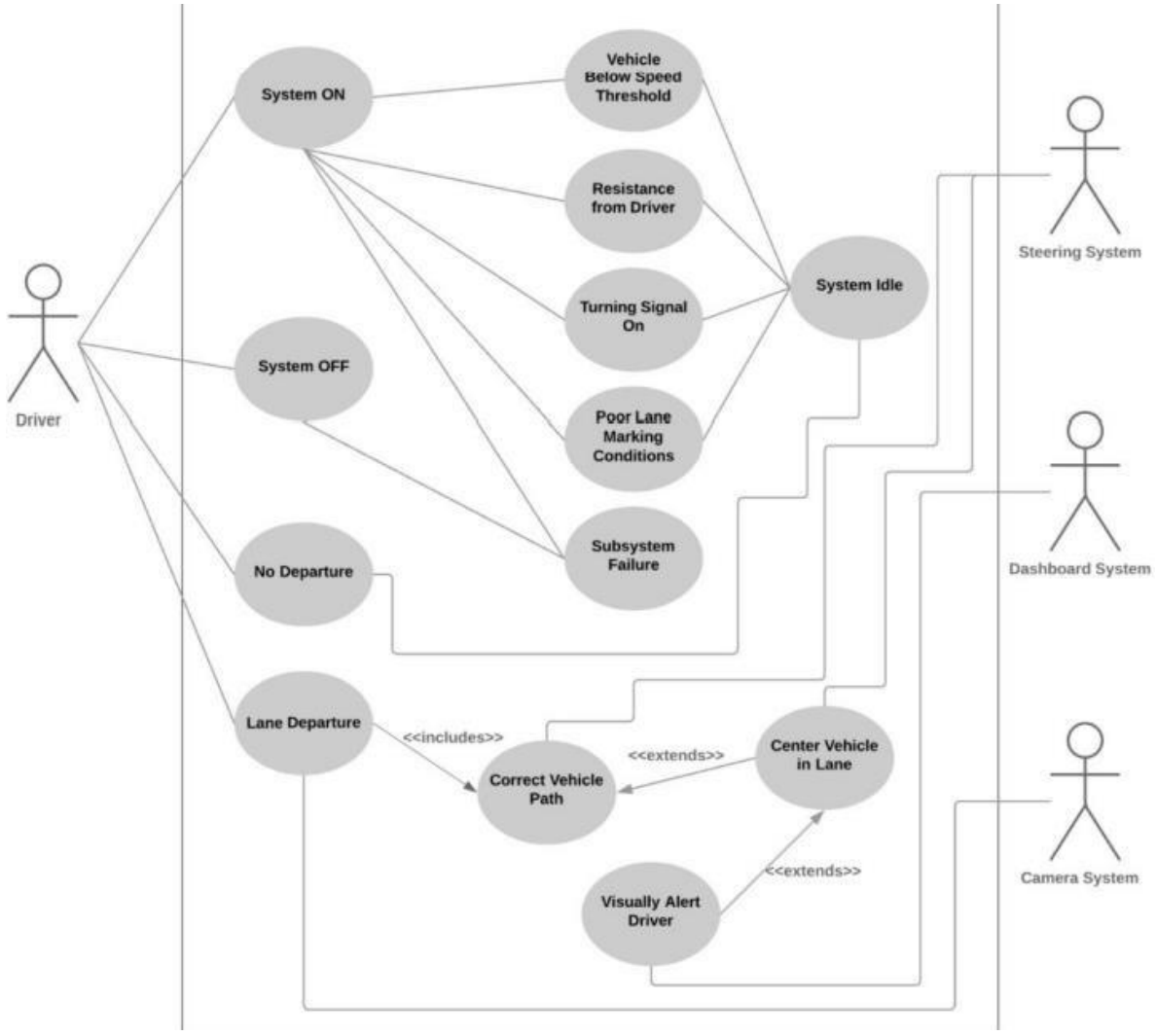


Fig 7.3.1- Use Case Diagram

7.4 ACTIVITY DIAGRAM

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.

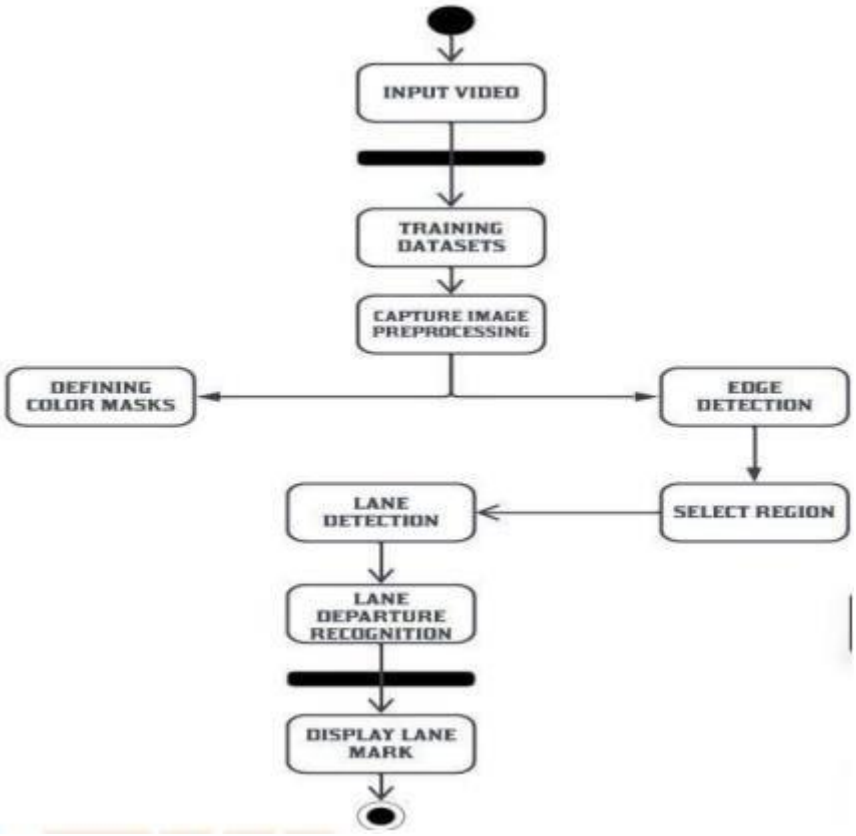


Fig 7.4.1 Activity Diagram

7.5 COLLABORATION DIAGRAM

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.

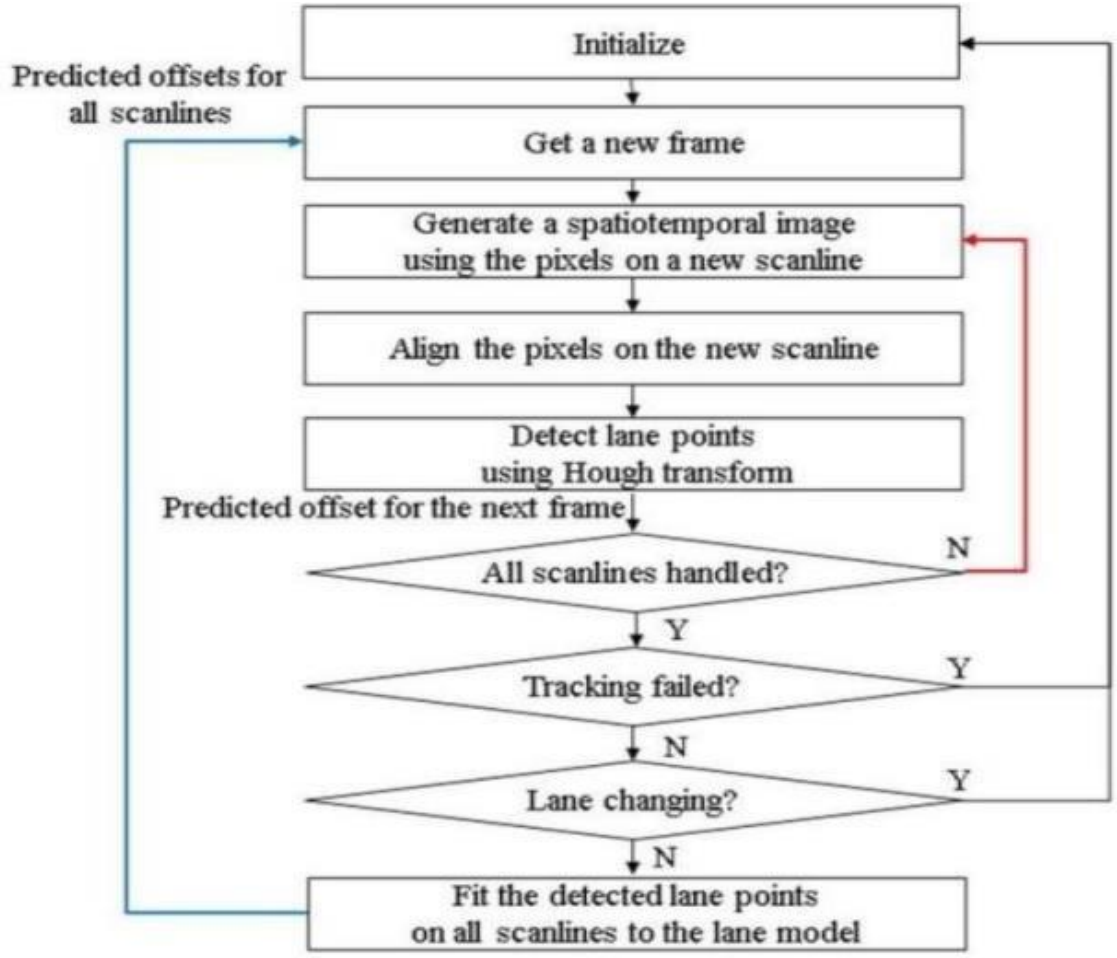


Fig 7.5.1 - Collaboration Diagram

7.6 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its aspects. It is a preliminary step used to create an overview of the system which can later be elaborated DFDs can also be used for visualization of data processing.

Level 0

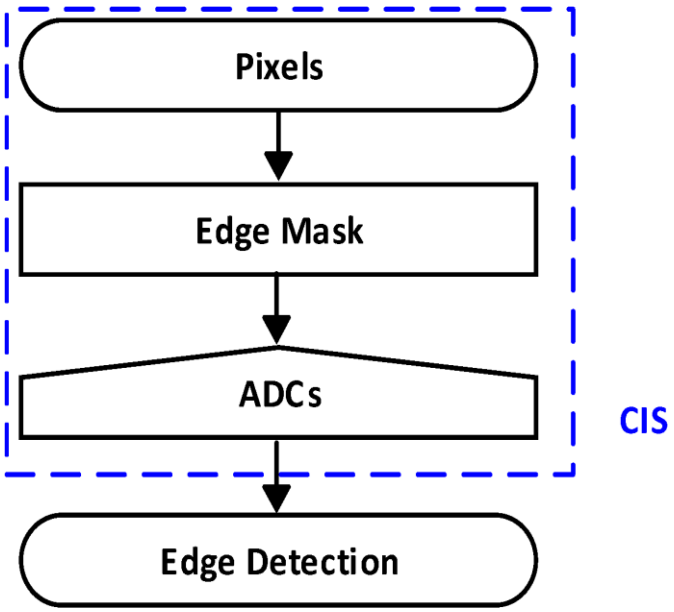


Fig 7.6.1 Level 0 data flow diagram

Level 1

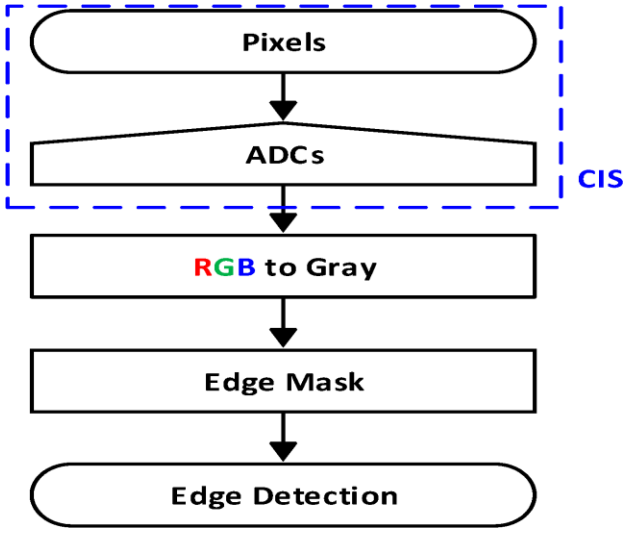


Fig 7.6.2 - Level 2 data flow diagram

7.7 CLASS DIAGRAM

A Class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

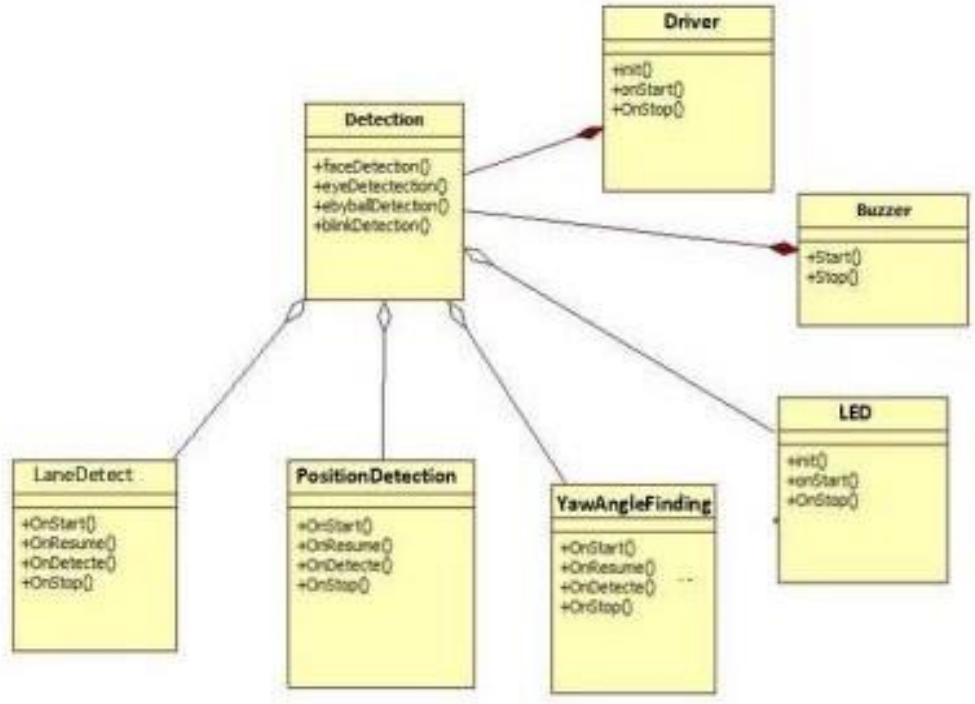


Fig 7.7.1 - Class Diagram

CHAPTER 8

CODING AND TESTING

8.1 CODING

Once the design aspect of the system is finalizing the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screwed into the system.

8.1.1 Coding Standards

Coding standards are guidelines to programming that focus on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary.

Some of the standard needed to achieve the above-mentioned objectives are as follows:

- Program should be simple, clear and easy to understand.
- Naming conventions
- Value conventions
- Script and comment procedure
- Message box format
- Exception and error handling

8.2 NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be self-descriptive. One should even get the meaning and scope of the variable by its name. The conventions are adopted for easy understanding of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letter of the new words in uppercase and the rest of letters in lowercase.

8.2.1 Value Conventions

Value conventions ensure values for variables at any point of time. This involves the following:

- Proper default values for the variables.
- Proper validation of values in the field.
- Proper documentation of flag values.

8.2.2 Script Writing and Commenting Standard

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

8.2.3 Message Box Format

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

- X – User has performed illegal operation.

8.3 TESTING

TESTCASE ID	TESTCASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/ FAIL
1	Grayscale Conversion of Image	Black and white picture	Picture is black and white	Pass
2	Noise Reduction	Thresholded image	Image is thresholded	Pass
3	Canny Edge Detection	Detect the edges in the lanes	Edges of the lanes are detected	Pass
4	Hough Line Transform	Lines are taken as the priority	Lines are prioritized	Pass
5	Region of interest	Perspective Transformation	Perspective is set to detect lanes	Pass
6	Drawing Lane Lines	Lane lines are generated	Lane lines appeared	Pass

Table 8.3.1 - Test case and Report

CHAPTER 9

RESULTS AND DISCUSSION

9.1 RESULT

A real time vision-based lane detection method was proposed. Image segmentation and remove the shadow of the road were processed. Canny operator was used to detect edges that represent road lanes or road boundaries. A hyperbola-pair road model used to deal with the occlusion and imperfect road condition. A series of experiment showed that the lanes were detected using Hough transformation with restricted search area and the projection of their intersection will form the last scan point called the horizon. Furthermore,In order to search out for the left and right vector points that represent the road lanes, the lane scan boundary phase uses the edge image and the left and right Hough lines and the horizon line as inputs, to effectively allocate the lane points. That was demonstrated by two hyperbola lines. The experimental results showed that the system is able to achieve a standard requirement to provide valuable information to the driver to ensure safety.

Many images of the road with different lanes is used for testing the system and it had resulted above 95% in detecting the road lanes correctly. The system was less efficient in detecting the lanes which are curve but when the road lanes are straight the system was effective. These are the screenshots taken at the time of testing

The Lane Detection is needed for today's everyday life. Accidents on street are the primary hassle for authorities of any country. The foremost cause of injuries is surprising alternate in lane on speedy using roads. Most of this hassle happens whilst in terrible environmental situation whilst it unsuccessful to discover or withinside the road curves in which detection is just too tedious. Actual time imaginative and prescientprimarily based totally lane detection technique become developed. This system is efficient in detecting the road lanes.

Lane detection stands as one of the foundational components enabling a fully functional autonomous driving system. Over recent years, numerous efforts have been devoted to achieving heightened accuracy coupled with increased speed in this area. Various methodologies have been proposed to detect road lanes, each offering a unique perspective. Nonetheless, the quest for a dependable system capable of handling an array of unpredictable scenarios remains a formidable challenge.

9.2 DISCUSSION

9.2.1 Advantages and Disadvantages

Based on the reviewed studies of road lane detections in the section above, recent major road lane detection methods can be distinguished into two different areas: model-based and intelligent-based approaches. Both of these approaches have achieved significant performance in autonomous vehicle driving systems. However, due to the differences in the nature of those two approaches, some advantages and disadvantages need to be recognized while applying them to applications.

9.2.2 Model-Based

Advantages: High accuracy becomes one of the most outstanding advantages of a technique that uses geometric patterns to match curves in the images. The nature of this technique allows the algorithm to find corresponding lane patterns easily, and it can be refined over time to improve accuracy. In addition, this approach has low computational requirements as they only need to fit the model to the road image. Since the model is pre-defined, the algorithm does not need to perform any additional processing on the image data, which can significantly reduce the computational requirements. Lastly, model-based methods are often more transparent and explainable than deep learning-based methods. This is because the model is based on a set of rules or principles that can be easily understood and interpreted. This makes it easier to debug and fine-tune the algorithm and identify cases where it may be making incorrect predictions.

Disadvantages: Due to the nature of the model-based method, it may struggle under different situations where the environment does not match the assumption of the established model. For example, if the lane markings are faded, damaged, or have different colors than those accounted for by the model, the algorithm may not be able to detect the lanes correctly. Moreover, this approach based on geometric patterns is likely to rely heavily on the given geometric data, which sets limits on the applied scenarios. This means that the algorithm may not be able to adapt to new situations or environments, which can limit its usefulness in real-world applications.

9.2.3 Intelligent Based

Advantages: Unlike model-based methods, intelligent-based methods can learn complex features and patterns from the input data, making them well-suited to handling complex road scenes with changing lighting and weather conditions. The iterated learning is able to train the data with various features, and it allows the algorithm to detect the lane markings even when they are partially obscured or degraded, which can improve the accuracy of the detection. Furthermore, instead of using a static pattern to detect lanes, an intelligent-based approach gives the ability to generalize to new situations and environments, which makes them more flexible than model-based methods. Lastly, deep learning models can be trained with large datasets and can process large amounts of input data in real time. This makes them well-suited to applications such as autonomous driving, where real-time performance is critical.

Disadvantages: While this approach has the ability to process large datasets, it also creates higher requirements on the scale of input data to create high-accuracy results. This can be a challenge in some applications, where obtaining a large dataset of annotated images can be time-consuming and expensive. Deep learning-based methods are also often difficult to interpret, meaning it can be challenging to understand which features or patterns the algorithm uses to make predictions. This can make it difficult to improve the algorithm or diagnose issues with its performance.

CHAPTER 10

CONCLUSION AND FUTURE WORK

10.1 CONCLUSION

When we are driving, we make decisions based on our vision. Our constant point of reference concerning where to direct the car is the lines on the roadway that the model has identified as the lane markings. Additionally, this steering is automatic. Naturally, automatically detecting lane lines with an algorithm is one of the initial tasks we would like to achieve when creating a self-driving car. The region of interest (ROI) for road detection must be adaptable. The horizon will shift when moving upward or downward along a steep incline and will no longer be determined by the frame's proportions. Additionally, this is something to consider when there are tight corners and heavy traffic.

In-depth research, designing, and planning resulted in the development of a lane detection system that autonomous driving systems can use to identify lanes, ensuring safer travel for the passengers in the car as well as for pedestrians and other vehicles in the area. The system preprocesses the image frame using the clever edge detection method before applying the Hough transform algorithm to highlight the lanes for the user's convenience. This method's key benefit is that it can anticipate the lane configurations on the road without the need for a trained model. Instead, it actively receives the video frames as input and actively recognizes the frame boundaries, which are subsequently returned to the user as lanes. As a result, no effort is wasted creating, training, and testing a dataset. With the help of this technology, users can swiftly integrate lane detection into their autonomous driving systems.

Image enhancing methods based on deep convolution neural networks are being gradually proposed, influenced by the advancement of artificial intelligence and deep learning. This kind of approach can provide the speed and accuracy requirements for digital image processing. It is a kind of signal processing where a picture serves as the input and the output can either be another image or attributes related to the input image. One of the technologies with the quickest growth rate nowadays is image processing. It also opens a significant area for research in computer science and engineering.

In summary, the technology for processing images has a significant social impact and is employed in almost every aspect of people's lives. Digital image processing continues to have a lot of new areas to investigate and will continue to advance and develop in a more positive manner as long as human requirements continue to expand

10.2 FUTURE ENHANCEMENTS

Lane line detection is a critical problem in intelligent vehicles and autonomous driving. There is a lot of potential for further research in this area, particularly in complex road conditions. Some areas that could be explored in more depth and receive further analysis are as follows:

- Improving the robustness and accuracy of lane line detection algorithms under various lighting and weather conditions, such as rain, snow, and fog.
- Developing methods for detecting and tracking lane lines in the presence of other visual distractions, such as pedestrians, vehicles, and road signs.
- Investigating machine learning techniques, such as deep learning, to improve the performance of lane line detection algorithms.
- Developing approaches for integrating lane line detection with other perception tasks, such as object detection and classification, to enable more comprehensive identification of the environment.
- Exploring the use of multiple sensors, such as cameras, lidar, and radar, to improve the accuracy and reliability of lane line detection in complex road conditions.

Because we use modular implementation, updating algorithms is simple, and work on the model can be continued in the future. We insert the model's pickle file into the relevant locations, which can then be simply transferred to goods. As a result, compiling the full big code might be avoided easily. We can also enhance the idea by creating a new future in which the road can be identified in the dark, or at night. In daylight, the color recognition and selection process are highly effective. Adding shadows will make things a little noisier, but it won't be as rigorous as driving at night or in low light (e.g., heavy fog). And this research can only recognize lanes on bitumen roads, not on the loamy soil roads that are ubiquitous in Indian villages.

As a result, this project can be improved to detect and avoid accidents on loamy soil roads found in communities.

REFERENCES

1. Hou, H.; Guo, P.; Zheng, B.; Wang, J. An Effective Method for Lane Detection in Complex Situations. In Proceedings of the 2021 9th International Symposium on Next Generation Electronics (I.S.N.E.), Changsha, China, 9–11 July 2021; IEEE: Piscataway, NJ, USA, 2021.
2. Zhang, Z.; Ma, X. Lane recognition algorithm using the hough transform based on complicated conditions. J. Comput. Commun. 2019, 7, 65.
3. Qiu, D.; Weng, M.; Yang, H.; Yu, W.; Liu, K. Research on Lane Line Detection Method Based on Improved Hough Transform. In Proceedings of the 2019 Chinese Control And Decision Conference (C.C.D.C.), Nanchang, China, 3–5 June 2019; IEEE: Piscataway, NJ, USA, 2019.
4. Gu, K.X.; Li, Z.Q.; Wang, J.J.; Zhou, Y.; Zhang, H.; Zhao, B.; Ji, W. The Effect of Cryogenic Treatment on the Microstructure and Properties of Ti-6Al-4V Titanium Alloy. In Materials Science Forum; Trans Tech Publications Ltd.: Wollerau, Switzerland, 2013.
5. Li, X.; Yin, P.; Zhi, Y.; Duan, C. Vertical Lane Line Detection Technology Based on Hough Transform. IOP Conf. Ser. Earth Environ. Sci. 2020, 440, 032126.
6. Wu, P.-C.; Chang, C.-Y.; Lin, C.H. Lane-mark extraction for automobiles under complex conditions. Pattern Recognit. 2014, 47, 2756–2767.
7. Wang, Y.; Shen, D.; Teoh, E.K. Lane detection using spline model. Pattern Recognit. Lett. 2000, 21, 677–689.
8. Savant, K.V.; Meghana, G.; Potnuru, G.; Bhavana, V. Lane Detection for Autonomous Cars Using Neural Networks. In Machine Learning and Autonomous Systems; Springer: Berlin/Heidelberg, Germany, 2022; pp. 193–207.
9. Borkar, A.; Hayes, M.; Smith, M.T. Robust Lane Detection and Tracking with Ransac and Kalman Filter. In Proceedings of the 2009 16th IEEE International Conference on Image Processing (I.C.I.P.), Cairo, Egypt, 7–10 November 2009; IEEE: Piscataway, NJ, USA, 2009.
10. Lee, J.W.; Yi, U.K. A lane-departure identification based on L.B.P.E., Hough transform, and linear regression. Comput. Vis. Image Underst. 2005, 99, 359–383.
11. Ghafoorian, M.; Nugteren, C.; Baka, N.; Booij, O.; Hofmann, M. El-Gan: Embedding Loss Driven Generative Adversarial Networks for Lane Detection. In Proceedings of the European Conference on Computer Vision (E.C.C.V.), Munich, Germany, 8–14 September 2018.
12. Gurghian, A.; Koduri, T.; Bailur, S.V.; Carey, K.J.; Murali, V.N. Deeplanes: End-to-End Lane Position Estimation using Deep Neural Networksa. In Proceedings of the IEEE Conference on

Computer Vision and Pattern Recognition Workshops, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 38–45.

13. Chougule, S.; Koznek, N.; Ismail, A.; Adam, G.; Narayan, V.; Schulze, M. Reliable Multilane Detection and Classification by Utilizing CNN as a Regression Network. In Proceedings of the European Conference on Computer Vision (E.C.C.V.), Munich, Germany, 8–14 September 2018.
14. Wang, Z.; Ren, W.; Qiu, Q. Lanenet: Real-time lane detection networks for autonomous driving. arXiv 2018, arXiv:1807.01726.
15. Van Gansbeke, W.; De Brabandere, B.; Neven, D.; Proesmans, M.; Van Gool, L. End -to-End Lane Detection Through Differentiable Least-Squares fitting. In Proceedings of the IEEE International Conference on Computer Vision Workshops, Seoul, Republic of Korea, 27–28 October 2019.
16. Khan, M.A.M.; Haque, M.F.; Hasan, K.R.; Alajmani, S.H.; Baz, M.; Masud, M.; Nahid, A.A. LLDNet: A Lightweight Lane Detection Approach for Autonomous Cars Using Deep Learning. Sensors 2022, 22, 5595.
17. Eskandarian A. (2012) Fundamentals of Driver Assistance. In: Eskandarian A. (eds) Handbook of Intelligent Vehicles. Springer, London.
18. Zhao, Z.Q., Zheng, P., Xu, S.T. & Wu, X. (2018) Object detection with deep learning: A review, arXiv e-prints, arXiv:1807.05511.
19. Bellis, Elizabeth, & Jim. (2008) National motor vehicle crash causation survey (NMVCCS) .(SASanalytical user's manual. No. HS-811 053.)
20. Hernández, D.C., Hoang, V.D. and Jo, K.H., 2013, July. Vanishing point based image segmentation and clustering for omnidirectional images. In the International Conference on Intelligent Computing (pp. 541-550). Springer, Berlin, Heidelberg.

APPENDCIES

A.1 SDG GOALS

Road lane line detection for Advanced Driving Assistance System (ADAS) in India contributes to several Sustainable Development Goals (SDGs) outlined by United Nations. Here are some of the SDGs that ADAS aligns with:

Goal 1: Good Health and Well-being: ADAS technologies such as automatic emergency braking (AEB), lane departure warning (LDW), and adaptive cruise control (ACC) contribute to reducing the number of accidents and fatalities on the roads, thus promoting good health and well-being by making transportation safer.

Goal 2: Industry, Innovation, and Infrastructure: ADAS involves technological innovations in the automotive industry, including sensors, cameras, and software algorithms. The development and deployment of ADAS contribute to advancements in infrastructure and technology, fostering innovation within the industry.

Goal 3: Sustainable Cities and Communities: ADAS technologies improve traffic management, reduce congestion, and enhance the efficiency of transportation systems in cities. By making roads safer and optimizing traffic flow, ADAS contributes to creating more sustainable urban environments.

Goal 4: Climate Action: Some ADAS features, such as adaptive cruise control and eco-driving assistance systems, can promote fuel-efficient driving behavior, leading to reduced greenhouse gas emissions from vehicles and contributing to climate action efforts.

Goal 5: Peace, Justice, and Strong Institutions: Enhancing road safety through ADAS technologies can contribute to creating more just and equitable societies by reducing accidents, injuries, and fatalities on the roads, and promoting safer road usage.

A.2 SOURCE CODE

A.2.1 Video Testing

Lane detection.py

```
import cv2
import numpy as np
import matplotlib.image as mpimg
from docopt import docopt
from IPython.display import HTML
from IPython.core.display import Video
from moviepy.editor import VideoFileClip
from CameraCalibration import CameraCalibration
from Thresholding import Thresholding
from PerspectiveTransformation import PerspectiveTransformation
from LaneLines import LaneLines


def canny(img):
    if img is None:
        cap.release()
        cv2.destroyAllWindows()
        exit()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kernel = 5
    blur = cv2.GaussianBlur(gray,(kernel, kernel),0)
    canny = cv2.Canny(gray, 50, 150)
```

```
    return canny


def region_of_interest(canny):
    height = canny.shape[0]
    width = canny.shape[1]
    mask = np.zeros_like(canny)
    triangle = np.array([[
    (200, height),
    (800, 350),
    (1200, height),]], np.int32)
    cv2.fillPoly(mask, triangle, 255)
    masked_image = cv2.bitwise_and(canny, mask)
    return masked_image


def houghLines(cropped_canny):
    return cv2.HoughLinesP(cropped_canny, 2, np.pi/180, 100,
        np.array([]), minLineLength=40, maxLineGap=5)


def addWeighted(frame, line_image):
    return cv2.addWeighted(frame, 0.8, line_image, 1, 1)


def display_lines(img,lines):
    line_image = np.zeros_like(img)
    if lines is not None:
        for line in lines:
```

```
            for x1, y1, x2, y2 in line:
                cv2.line(line_image,(x1,y1),(x2,y2),(0,0,255),10)
    return line_image


def make_points(image, line):
    slope, intercept = line
    y1 = int(image.shape[0])
    y2 = int(y1*3.0/5)
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    return [[x1, y1, x2, y2]]


def average_slope_intercept(image, lines):
    left_fit    = []
    right_fit   = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1,x2), (y1,y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0:
                left_fit.append((slope, intercept))
            else:
```

```
                right_fit.append((slope, intercept))
    left_fit_average  = np.average(left_fit, axis=0)
    right_fit_average = np.average(right_fit, axis=0)
    left_line  = make_points(image,  left_fit_average)
    right_line = make_points(image,  right_fit_average)
    averaged_lines = [left_line, right_line]
    return averaged_lines

cap = cv2.VideoCapture("test1.mp4")
while(cap.isOpened()):
    _, frame = cap.read()
    canny_image = canny(frame)
    cropped_canny = region_of_interest(canny_image)
    # cv2.imshow("cropped_canny",cropped_canny)

    lines = houghLines(cropped_canny)
    averaged_lines = average_slope_intercept(frame, lines)
    line_image = display_lines(frame, averaged_lines)
    combo_image = addWeighted(frame, line_image)
    cv2.imshow("result", combo_image)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
```

```
cv2.destroyAllWindows()
```

Camera Caliberation.py

```
import numpy as np
import cv2
import glob
import matplotlib.image as mpimg
import matplotlib.pyplot as plt


class CameraCalibration():
    """ Class that calibrate camera using chessboard images.

    Attributes:
        mtx (np.array): Camera matrix
        dist (np.array): Distortion coefficients
    """
    def __init__(self, image_dir, nx, ny, debug=False):
        """ Init CameraCalibration.

        Parameters:
            image_dir (str): path to folder contains chessboard images
            nx (int): width of chessboard (number of squares)
            ny (int): height of chessboard (number of squares)
        """
        fnames = glob.glob("{}/*".format(image_dir))
```

```
    objpoints = []
    imgpoints = []

    # Coordinates of chessboard's corners in 3D
    objp = np.zeros((nx*ny, 3), np.float32)
    objp[:,:2] = np.mgrid[0:nx, 0:ny].T.reshape(-1, 2)

    # Go through all chessboard images
    for f in fnames:
        img = mpimg.imread(f)

        # Convert to grayscale image
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

        # Find chessboard corners
        ret, corners = cv2.findChessboardCorners(img, (nx, ny))
        if ret:
            imgpoints.append(corners)
            objpoints.append(objp)

    shape = (img.shape[1], img.shape[0])
    ret, self.mtx, self.dist, _, _ = cv2.calibrateCamera(objpoints, imgpoints, shape, None, None)

    if not ret:
```

```
            raise Exception("Unable to calibrate camera")


    def undistort(self, img):
        """ Return undistort image.

        Parameters:
            img (np.array): Input image

        Returns:
            Image (np.array): Undistorted image
        """
        # Convert to grayscale image
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        return cv2.undistort(img, self.mtx, self.dist, None, self.mtx)
```

Thresholding.py

```
import cv2
import numpy as np


def threshold_rel(img, lo, hi):
    vmin = np.min(img)
    vmax = np.max(img)

    vlo = vmin + (vmax - vmin) * lo
    vhi = vmin + (vmax - vmin) * hi
```

```
    return np.uint8((img >= vlo) & (img <= vhi)) * 255


def threshold_abs(img, lo, hi):
    return np.uint8((img >= lo) & (img <= hi)) * 255


class Thresholding:
    """ This class is for extracting relevant pixels in an image.
    """
    def __init__(self):
        """ Init Thresholding."""
        pass

    def forward(self, img):
        """ Take an image and extract all relavant pixels.

        Parameters:
            img (np.array): Input image

        Returns:
            binary (np.array): A binary image represent all positions of relavant pixels.
        """
        hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
        hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
        h_channel = hls[:,:,0]
        l_channel = hls[:,:,1]
```

```
    s_channel = hls[:,:,2]
    v_channel = hsv[:,:,2]

    right_lane = threshold_rel(l_channel, 0.8, 1.0)
    right_lane[:,:750] = 0

    left_lane = threshold_abs(h_channel, 20, 30)
    left_lane &= threshold_rel(v_channel, 0.7, 1.0)
    left_lane[:,550:] = 0

    img2 = left_lane | right_lane

    return img2
```

Perspective Transformation.py

```
import cv2
import numpy as np

class PerspectiveTransformation:
    """ This a class for transforming image between front view and top view

    Attributes:
        src (np.array): Coordinates of 4 source points
        dst (np.array): Coordinates of 4 destination points
        M (np.array): Matrix to transform image from front view to top view
```

```
    M_inv (np.array): Matrix to transform image from top view to front view
"""
def __init__(self):
    """Init PerspectiveTransformation."""
    self.src = np.float32([(550, 460),     # top-left
                           (150, 720),     # bottom-left
                           (1200, 720),    # bottom-right
                           (770, 460)])    # top-right
    self.dst = np.float32([(100, 0),
                           (100, 720),
                           (1100, 720),
                           (1100, 0)])
    self.M = cv2.getPerspectiveTransform(self.src,  self.dst)
    self.M_inv = cv2.getPerspectiveTransform(self.dst, self.src)

def forward(self, img, img_size=(1280, 720), flags=cv2.INTER_LINEAR):
    """ Take a front view image and transform to top view

    Parameters:
        img (np.array): A front view image
        img_size (tuple): Size of the image (width, height)
        flags : flag to use in cv2.warpPerspective()

    Returns:
        Image (np.array): Top view image
```

```
        """
        return cv2.warpPerspective(img, self.M, img_size, flags=flags)

    def backward(self, img, img_size=(1280, 720), flags=cv2.INTER_LINEAR):
        """ Take a top view image and transform it to front view

        Parameters:
            img (np.array): A top view image
            img_size (tuple): Size of the image (width, height)
            flags (int): flag to use in cv2.warpPerspective()

        Returns:
            Image (np.array): Front view image
        """
        return cv2.warpPerspective(img, self.M_inv, img_size, flags=flags)
```

A.2.2 Image Testing

Lane Lines.py

```
import cv2
import numpy as np
import matplotlib.image as mpimg

def hist(img):
    bottom_half = img[img.shape[0]//2:,:]
    return np.sum(bottom_half, axis=0)
```

```
class LaneLines:
    """ Class containing information about detected lane lines.


    Attributes:
        left_fit (np.array): Coefficients of a polynomial that fit left lane line
        right_fit (np.array): Coefficients of a polynomial that fit right lane line
        parameters (dict): Dictionary containing all parameters needed for the pipeline
        debug (boolean): Flag for debug/normal mode
    """
    def __init__(self):
        """Init Lanelines.


        Parameters:
            left_fit (np.array): Coefficients of polynomial that fit left lane
            right_fit (np.array): Coefficients of polynomial that fit right lane
            binary (np.array): binary image
        """
        self.left_fit = None
        self.right_fit = None
        self.binary = None
        self.nonzero = None
        self.nonzerox = None
        self.nonzeroy = None
        self.clear_visibility = True
```

```
        self.dir = []
        self.left_curve_img = mpimg.imread('left_turn.png')
        self.right_curve_img = mpimg.imread('right_turn.png')
        self.keep_straight_img = mpimg.imread('straight.png')
        self.left_curve_img = cv2.normalize(src=self.left_curve_img, dst=None, alpha=0,
beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        self.right_curve_img = cv2.normalize(src=self.right_curve_img, dst=None, alpha=0,
beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        self.keep_straight_img = cv2.normalize(src=self.keep_straight_img, dst=None, alpha=0,
beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

        # HYPERPARAMETERS
        # Number of sliding windows
        self.nwindows = 9
        # Width of the the windows +/- margin
        self.margin = 100
        # Mininum number of pixels found to recenter window
        self.minpix = 50

    def forward(self, img):
        """Take a image and detect lane lines.

        Parameters:
            img (np.array): An binary image containing relevant pixels

        Returns:
```

```
        Image (np.array): An RGB image containing lane lines pixels and other details
    """
    self.extract_features(img)
    return self.fit_poly(img)

def pixels_in_window(self, center, margin, height):
    """ Return all pixel that in a specific window

    Parameters:
        center (tuple): coordinate of the center of the window
        margin (int): half width of the window
        height (int): height of the window

    Returns:
        pixelx (np.array): x coordinates of pixels that lie inside the window
        pixely (np.array): y coordinates of pixels that lie inside the window
    """
    topleft = (center[0]-margin, center[1]-height//2)
    bottomright = (center[0]+margin, center[1]+height//2)

    condx = (topleft[0] <= self.nonzerox) & (self.nonzerox <= bottomright[0])
    condy = (topleft[1] <= self.nonzeroy) & (self.nonzeroy <= bottomright[1])
    return self.nonzerox[condx&condy], self.nonzeroy[condx&condy]

def extract_features(self, img):
```

```
    """ Extract features from a binary image

    Parameters:
        img (np.array): A binary image
    """
    self.img = img
    # Height of of windows - based on nwindows and image shape
    self.window_height = np.int(img.shape[0]//self.nwindows)

    # Identify the x and y positions of all nonzero pixel in the image
    self.nonzero = img.nonzero()
    self.nonzerox = np.array(self.nonzero[1])
    self.nonzeroy = np.array(self.nonzero[0])

def find_lane_pixels(self, img):
    """Find lane pixels from a binary warped image.

    Parameters:
        img (np.array): A binary warped image

    Returns:
        leftx (np.array): x coordinates of left lane pixels
        lefty (np.array): y coordinates of left lane pixels
        rightx (np.array): x coordinates of right lane pixels
        righty (np.array): y coordinates of right lane pixels
```

```
    out_img (np.array): A RGB image that use to display result later on.
"""
assert(len(img.shape) == 2)

# Create an output image to draw on and visualize the result
out_img = np.dstack((img, img, img))

histogram = hist(img)
midpoint = histogram.shape[0]//2
leftx_base = np.argmax(histogram[:midpoint])
rightx_base = np.argmax(histogram[midpoint:]) + midpoint

# Current position to be update later for each window in nwindows
leftx_current = leftx_base
rightx_current = rightx_base
y_current = img.shape[0] + self.window_height//2

# Create empty lists to reveice left and right lane pixel
leftx, lefty, rightx, righty = [], [], [], []

# Step through the windows one by one
for _ in range(self.nwindows):
    y_current -= self.window_height
    center_left = (leftx_current, y_current)
    center_right = (rightx_current, y_current)
```

```
            good_left_x, good_left_y = self.pixels_in_window(center_left, self.margin,
self.window_height)
            good_right_x, good_right_y = self.pixels_in_window(center_right, self.margin,
self.window_height)

            # Append these indices to the lists
            leftx.extend(good_left_x)
            lefty.extend(good_left_y)
            rightx.extend(good_right_x)
            righty.extend(good_right_y)

            if len(good_left_x) > self.minpix:
                leftx_current = np.int32(np.mean(good_left_x))
            if len(good_right_x) > self.minpix:
                rightx_current = np.int32(np.mean(good_right_x))

        return leftx, lefty, rightx, righty, out_img

    def fit_poly(self, img):
        """Find the lane line from an image and draw it.

        Parameters:
            img (np.array): a binary warped image
```

```
Returns:
    out_img (np.array): a RGB image that have lane line drawn on that.
"""

leftx, lefty, rightx, righty, out_img = self.find_lane_pixels(img)

if len(lefty) > 1500:
    self.left_fit = np.polyfit(lefty, leftx, 2)
if len(righty) > 1500:
    self.right_fit = np.polyfit(righty, rightx, 2)

# Generate x and y values for plotting
maxy = img.shape[0] - 1
miny = img.shape[0] // 3
if len(lefty):
    maxy = max(maxy, np.max(lefty))
    miny = min(miny, np.min(lefty))

if len(righty):
    maxy = max(maxy, np.max(righty))
    miny = min(miny, np.min(righty))

ploty = np.linspace(miny, maxy, img.shape[0])

left_fitx = self.left_fit[0]*ploty**2 + self.left_fit[1]*ploty + self.left_fit[2]
```

```
        right_fitx = self.right_fit[0]*ploty**2 + self.right_fit[1]*ploty + self.right_fit[2]

        # Visualization
        for i, y in enumerate(ploty):
            l = int(left_fitx[i])
            r = int(right_fitx[i])
            y = int(y)
            cv2.line(out_img, (l, y), (r, y), (0, 255, 0))

        lR, rR, pos = self.measure_curvature()

        return out_img

    def plot(self, out_img):
        np.set_printoptions(precision=6, suppress=True)
        lR, rR, pos = self.measure_curvature()

        value = None
        if abs(self.left_fit[0]) > abs(self.right_fit[0]):
            value = self.left_fit[0]
        else:
            value = self.right_fit[0]

        if abs(value) <= 0.00015:
            self.dir.append('F')
```

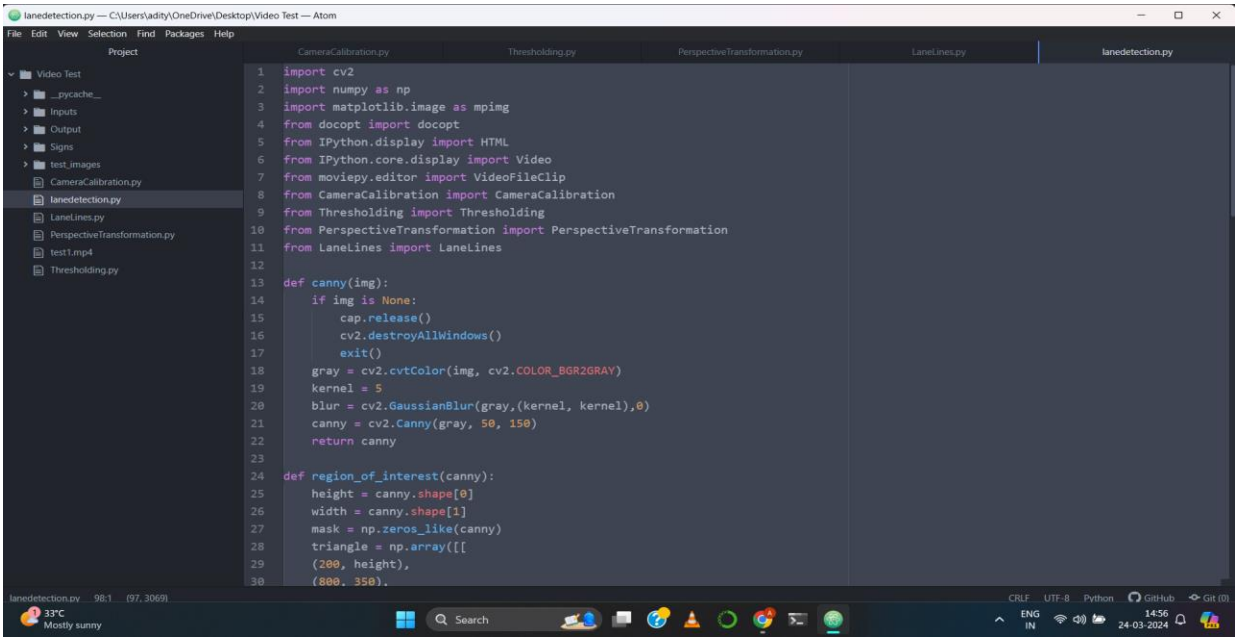
```
elif value < 0:
    self.dir.append('L')
else:
    self.dir.append('R')

if len(self.dir) > 10:
    self.dir.pop(0)

W = 400
H = 500
widget = np.copy(out_img[:H, :W])
widget //= 2
widget[0,:] = [0, 0, 255]
widget[-1,:] = [0, 0, 255]
widget[:,0] = [0, 0, 255]
widget[:,-1] = [0, 0, 255]
out_img[:H, :W] = widget

direction = max(set(self.dir), key = self.dir.count)
msg = "Keep Straight Ahead"
curvature_msg = "Curvature = {:.0f} m".format(min(lR, rR))
if direction == 'L':
    y, x = self.left_curve_img[:,:,3].nonzero()
    out_img[y, x-100+W//2] = self.left_curve_img[y, x, :3]
    msg = "Left Curve Ahead"
```

```
        if direction == 'R':
            y, x = self.right_curve_img[:,:,3].nonzero()
            out_img[y, x-100+W//2] = self.right_curve_img[y, x, :3]
            msg = "Right Curve Ahead"
        if direction == 'F':
            y, x = self.keep_straight__img[:,:,3].nonzero()
            out_img[y, x-100+W//2] = self.keep_straight_img[y, x, :3]

        cv2.putText(out_img, msg, org=(10, 240),
fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(255, 255, 255),
thickness=2)
        if direction in 'LR':
            cv2.putText(out_img, curvature_msg, org=(10, 280),
fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(255, 255, 255),
thickness=2)

        cv2.putText(
            out_img,
            "Good Lane Keeping",
            org=(10, 400),
            fontFace=cv2.FONT_HERSHEY_SIMPLEX,
            fontScale=1.2,
            color=(0, 255, 0),
            thickness=2)

        cv2.putText(
```

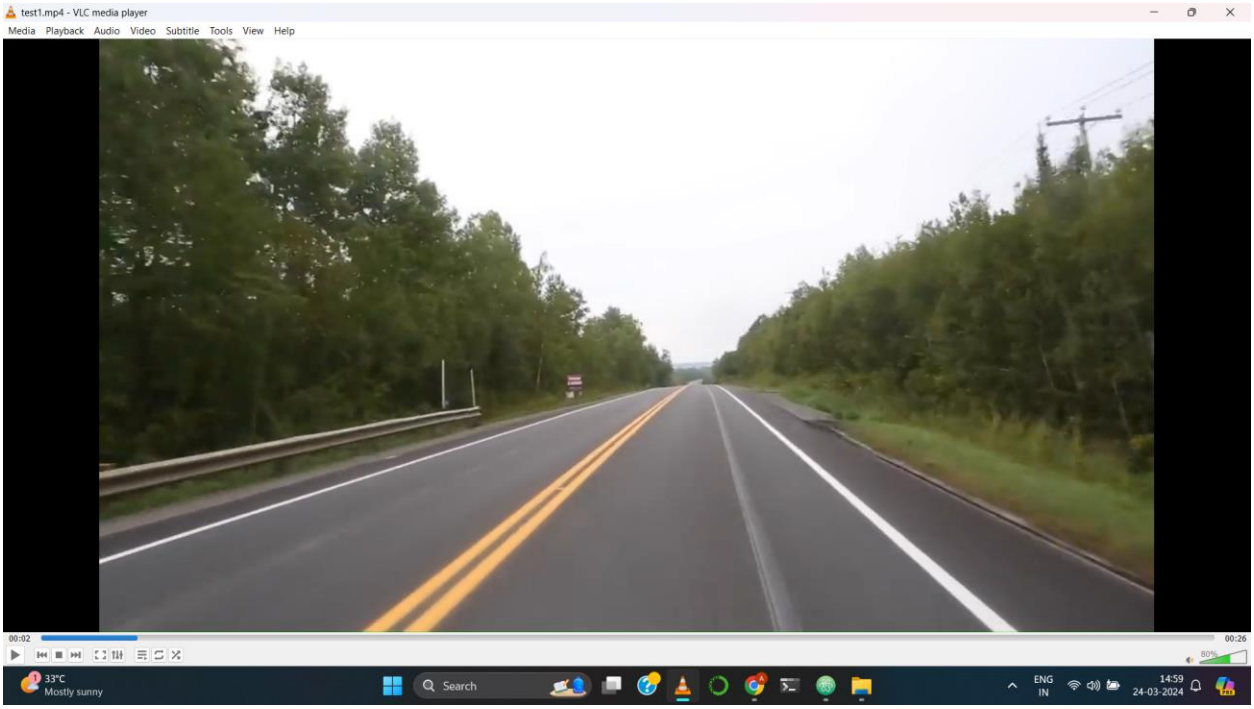
```
            out_img,
            "Vehicle is {:.2f} m away from center".format(pos),
            org=(10, 450),
            fontFace=cv2.FONT_HERSHEY_SIMPLEX,
            fontScale=0.66,
            color=(255, 255, 255),
            thickness=2)

        return out_img

    def measure_curvature(self):
        ym = 30/720
        xm = 3.7/700

        left_fit = self.left_fit.copy()
        right_fit = self.right_fit.copy()
        y_eval = 700 * ym

        # Compute R_curve (radius of curvature)
        left_curveR =  ((1 + (2*left_fit[0] *y_eval + left_fit[1])**2)**1.5)  /
np.absolute(2*left_fit[0])
        right_curveR = ((1 + (2*right_fit[0]*y_eval + right_fit[1])**2)**1.5) /
np.absolute(2*right_fit[0])

        xl = np.dot(self.left_fit, [700**2, 700, 1])
```

```
xr = np.dot(self.right_fit, [700**2, 700, 1])
pos = (1280//2  - (xl+xr)//2)*xm
return left_curveR, right_curveR, pos
```

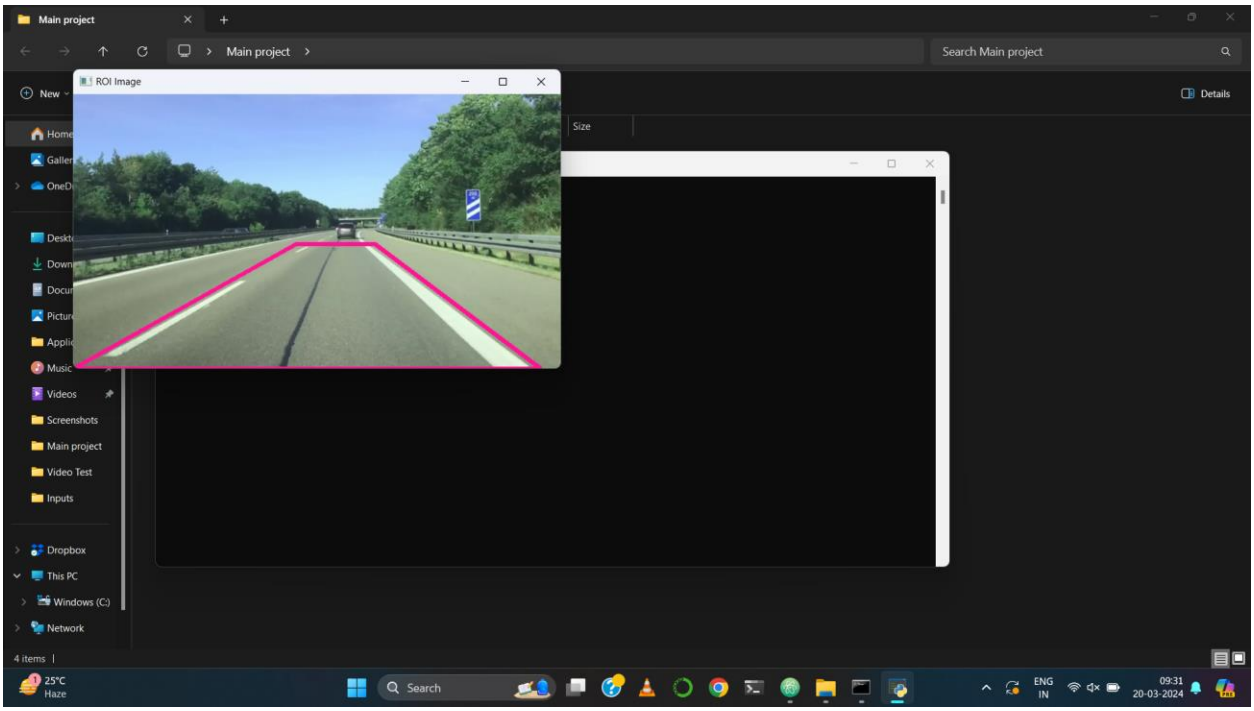
A.3 SCREEN SHOTS



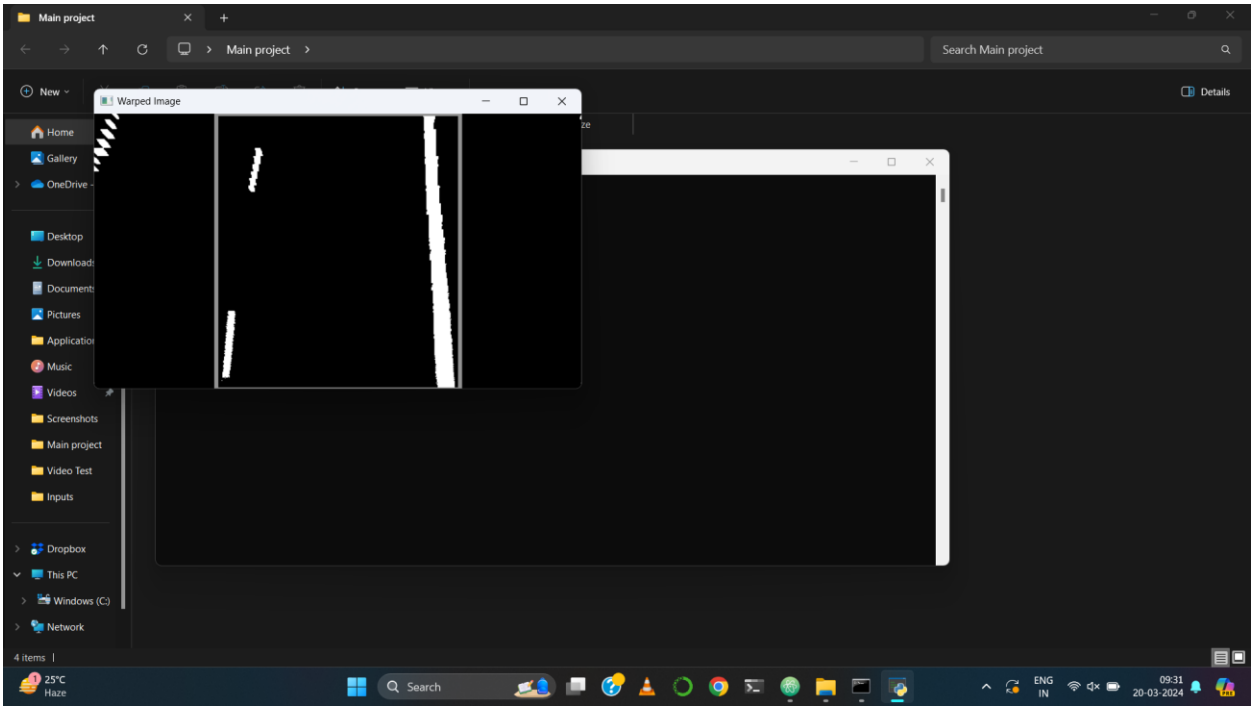
A.3.1 Atom Text Editor



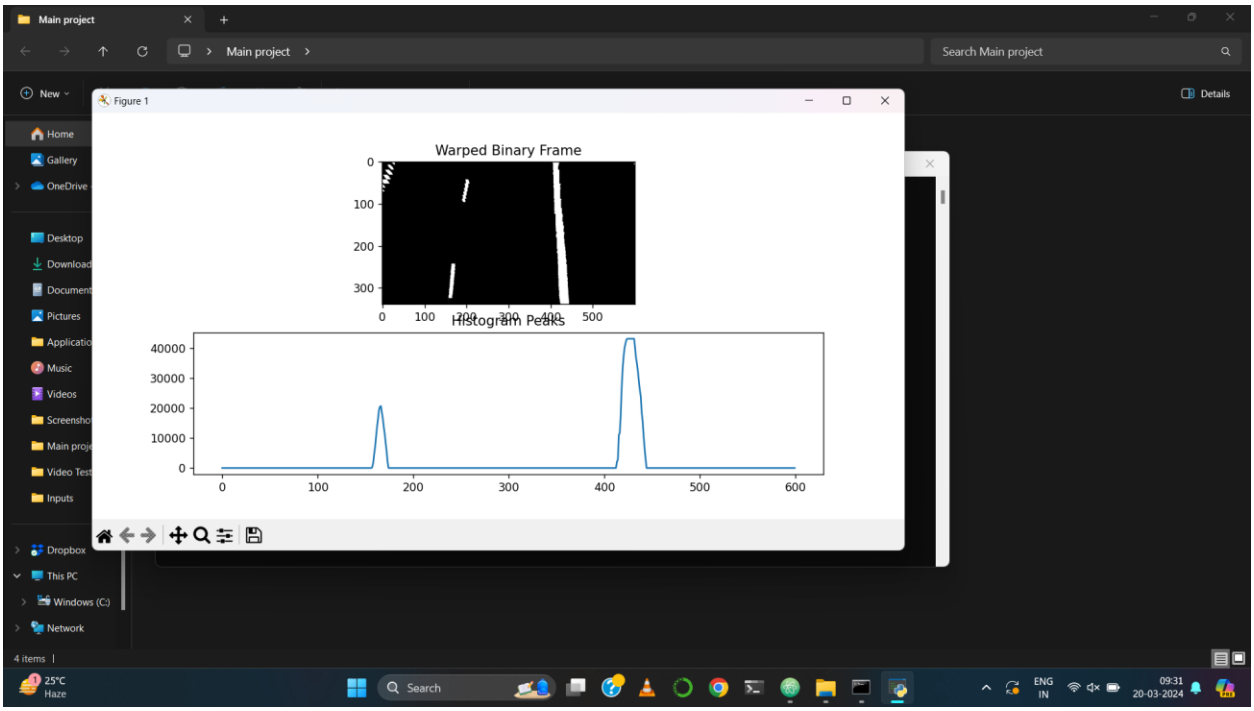
A.3.2 Input Video Frame



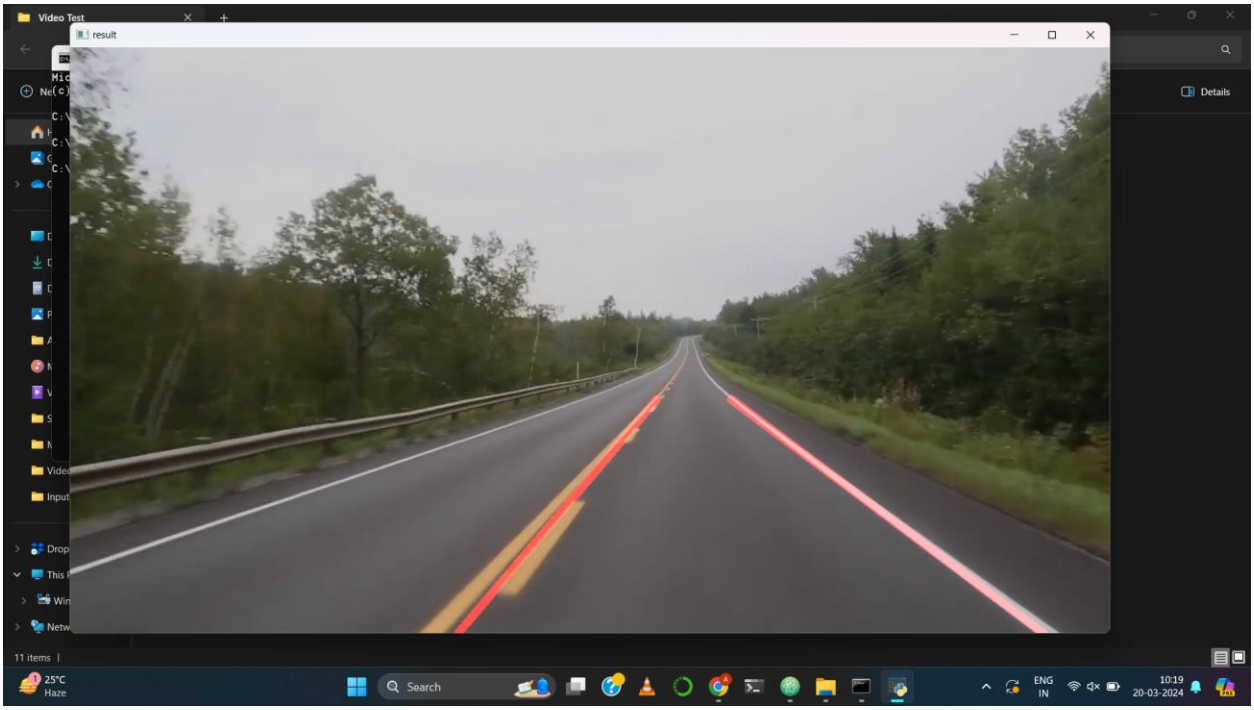
A.3.3 ROI Image



A.3.4 Noice Reduced Wrapped Image



A.3.5 Histogram



A.3.6 Output Video Frame

Result

The system found minor similarities with other sources, however, the text successfully passed the plagiarism check.

Analysis

Result	4%
Document title	ccbjournal.pdf
Content hash	f30569d7de0e259478de69130be4d68e
Date	2024-03-22 07:58:00
Check time	20 seconds
Character count	10,000
Special character count	12
Word count	1,313
Number of plagiarized words	51

Plagiarism sources

https://answers.opencv.org/question/201175/line-detection-with-opencv-python-and-hough-transform/

https://github.com/mohamedameen93/Lane-lines-detection-using-Python-and-OpenCV/blob/master/Lane-Lines-Detection.ipynb

https://stackoverflow.com/questions/77239067/line-detection-in-tables-on-pdf-using-cv-python

https://www.iosrjen.org/Papers/vol11_issue6/G1106015460.pdf

https://www.irjet.net/archives/V6/i1/IRJET-V6I1245.pdf

https://www.e3s-conferences.org/articles/e3sconf/pdf/2022/20/e3sconf_evf2021_03002.pdf

https://odr.chalmers.se/server/api/core/bitstreams/b5ed7a44-4170-4841-99ea-c371c8e7e51e/content

LANE - LINE DETECTION USING PYTHON,OPENCV

S.SOPHANA JENNIFER,M.E
Assitant Professor,
Department of Computer Science and Engineering
Panimalar Engineering College
Chennai,India
sophanajennifers@gmail.com

N.PUGHAZENDI,Ph.D,M.E
Professor,
Department of Computer Science and Engineering
Panimalar Engineering College
Chennai,India
pughazendi@panimalar.ac.in

CHINTHAM REDDY LALITHADITYA
UG Student,
Department of Computer Science and Engineering
Panimalar Engineering College
Chennai,India
adityareddy770@gmail.com

Abstract - *When we drive, we use our eyes to decide where to go. The lines on the road portray us where the lanes act as our steady reference for where to steer the vehicle. Naturally, one of the first things we would like to develop a self-driving car is to detect lane lines using an algorithm automatically. Lane detection is a difficult problem to solve. For decades, it has piqued the interest of the computer vision community. Lane detection is essentially a multi-feature detection problem that has proven to be difficult for computer vision and machine learning algorithms to solve. We present an Image Processing based method that involves Region Masking and Canny Edge Detection. The prime goal is to use the canny edge detection algorithm to extract the features and then use another method to detect the lanes on the region masked image. As the images are not perfect every time, looping through the pixels in order to find the slope and intercept would be a difficult task. This is where we looked at the Hough transform concept. It aids us in identifying prominent lines and connecting disjoint edge points in an image.*

Key Words: Lane Detection, Image Processing, OpenCV, Hough Transform, Canny Edge Detection.

I. INTRODUCTION

Automobiles have become one of the transportation tools for people to travel due to the rapid development of society. There are an increasing number of vehicles of various types on the narrow road. Lane detection is a prominent topic in the field of machine learning and computer vision and it has been utilized in intelligent vehicle systems [1]. It is an emerging field and finds its applications in the commercial world. The lane detection system comes from the lane markers in an intricate environment and is used to approximate the vehicle's position and trajectory relative to the lane reliably [2]. At the same time, lane detection plays a substantial part in the lane departure warning system. The task of lane detection is mainly split into two parts: edge detection and line detection. Line detection is equally important as edge detection in the process of lane detection.

The most popular lane line detectors are the Hough transform and convolution-based techniques. Lane detection is the course of detecting lane markers on the road and thereafter presenting these locations to an intelligent system. Intelligent vehicles cooperate with the infrastructure to achieve a safer environment and better traffic conditions in intelligent transportation systems. The utilization of a lane detecting system could be as simple as pointing out lane locations to the driver on an external display to more intricate tasks such as predicting a lane change in an instantaneous moment to avoid collision with other vehicles.

II. GENERAL DESCRIPTION

In this project we have detected lane lines in images using Python as it is one of the widely used programming languages for this purpose and OpenCV. OpenCV stands for "Open-Source Computer Vision", which is a module that has numerous useful tools for analysis of images [3]. OpenCV supports a wide variety of programming languages like Python, and Java. It can assess images and videos to recognize objects, faces, or even the handwriting of a human. In this project, we also use NumPy for a few simpler image processing techniques.

The lane detection module is divided into two steps:

(1) Gaussian Smoothing and Canny Edge Detection and

(2) ROI selection and detecting lane lines using Hough Transform.

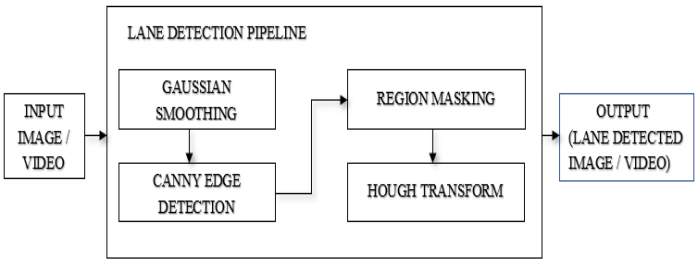


Figure-1: Block Diagram

II.A. GAUSSIAN SMOOTHING

Gaussian blur, in image processing, is also known as Gaussian smoothing. It is the outcome of blurring an image by a Gaussian function [4]. Noise reduction is gained by blurring the image with the support of a smoothing filter. Gaussian Smoothing is a widely used effect in graphics software, typically to reduce image noise. The 'kernel' for smoothing actually expresses the shape of the function that is used to compute the average of the neighboring points. A Gaussian kernel is a kernel which has the shape of a Gaussian i.e., a normal distribution curve. In order to use the Gaussian filter, we will first convert the image to grayscale so that it is easier for us to distinguish the output we get after running canny edge detector.

II.B. CANNY EDGE DETECTION

Edge detection is an image processing technique used to recognize points in a digital image with discontinuities, basically, sharp changes in the image brightness. These points where the image brightness varies sharply are called the edges (or boundaries) of the image. Canny edge [5] detector is probably the most commonly used and most effective method, because it uses first a gaussian filter, it has more noise immunity than other methods. There is a multitude of information available in the documentation of OpenCV itself. Thus, we decided to use Canny Edge Detection for this project.

To summarize, first, the algorithm will detect strong edge (strong gradient) pixels above the high threshold and reject pixels below the low threshold. Next, pixels with values between the low threshold and high threshold will be included as long as they are connected to strong edges. The output edges are a binary image with white pixels tracing out the detected edges and black everywhere else. We will also include Gaussian smoothing before running Canny, which is basically a way of suppressing noise and spurious gradients by averaging. cv2.Canny() function by OpenCV actually applies Gaussian smoothing internally, but we include it explicitly here because we can get a different result by using further smoothing and it's not a changeable parameter within cv2.Canny().

II.C. REGION MASKING

In an image, a region is a group of connected pixels that have similar properties. Because they may correlate to a items in a scene, regions are significant for image interpretation. Region Masking [6] is the process that is underneath many types of image processing, including edge detection, motion detection, and noise reduction. Masking a region is a view-specific graphic that can be used to hide certain and show certain elements in a view. It is a non-destructive process of editing images. Assuming that the camera (which is front-facing) that captures the image is mounted in a fixed spot on the car in a way that the lane lines will always appear in the same general section of the image [7]. Here, we use a triangular mask to illustrate the simplest case, but we can use a quadrilateral, and in principle, we could use any polygon. We would now use edge detection algorithm for detecting lane lines rather than relying on color. This would be a more reliable and efficient model.

II.D. HOUGH TRANSFORM

Now, to select lane lines in the edge detected image, we use the Hough Transform. The Hough Transform is a transform used to detect straight lines [8]. As per the documentation, to apply the transform, it is first desirable to have an edge detection pre-processing. In essence, a line can be spotted by finding the number of intersections between curves [9]. The more curves intersecting means that the line represented by that intersection have more points. In general, we can describe a threshold of the minimum number of intersections needed to detect a line. This is what the Hough Transform does. It keeps track of the intersection between curves of every point in the image. If the number of intersections is over a certain threshold, then it declares it as a line.

II.E. LANE DETECTION PIPELINE

The usual lane line detection method is to take an image of the road first with the aid of a camera fixed in the vehicle. Then the image taken is converted to a grayscale image to minimize the processing time. Secondly, the appearance of noise in the image will prevent the detection of the correct edge. Hence, the filters should be applied to remove noises like Gaussian filter. Then, the edge detector produces an edge image by using a canny filter with automatic thresholding to obtain the edges. Next, the edged image would be sent to the line detector after detecting the edges. Further, the Hough Transform is used to detect the required polygon on an edge detected image. Once the image goes through all these intermediate steps, a pipeline is made using these steps as functions with a single argument which is the source image. Once the testing and validation parts on images are over, we can move ahead with applying the same concept on lane videos. Here, each frame of the video is treated as an image and it goes through the lane detection pipeline. This step is repeated for all the frames in the videos and later all these frames are merged to create a single video file with the lane detected output.

Now that we have established the techniques and functions we need to use, we have created a lane detection pipeline. Through this function, our input image will go through all the required steps mentioned above.

II.F. IMPLEMENTATION

We will use Jupyter Notebook for code building purpose and to input the video and image file. Then, we will show the output file as a small GUI (Graphical User Interface) application written in Python. For this purpose, we use tkinter and moviepy library of Python.

III. RESULTS AND OUTPUT

Here, Figure 1 shows the output as a lane detected image on our test images and Figure 2 shows the output when our code was run on the test video.

Figure-2: Output images

Figure-3: Video output screenshot

IV. APPLICATIONS

It can be used for lane detection in a driverless car that isn't fully automated so that it can assist the driver.

V. ADVANTAGES

- It assists vehicular drivers as well as autonomous cars to drive safely.
- This does not involve complex and advanced machine learning, deep learning or neural networks. Rather, relies on easy-to-understand and simple image processing and computer vision techniques.
- It has minimal cost of development as it uses open source technologies such as Python, Jupyter and OpenCV.

VI. DISADVANTAGES

- Uniformity of the markings present on the lanes is an important factor in order to minimize confusion and uncertainty. Our model needs lanes where there are uniform and visible markings.
- The methods that have been developed so far work efficiently and give good results in cases where noise is not present in the image. But the problem arises when lot of noise is in the road images, as it does not give efficient results.

VII. CONCLUSION

With all the developments in the autonomous vehicle industry, lane detection systems are going to be more in demand. Through this project, we effectively tried to propose the lane detection code without having to write long codes that use machine learning or deep learning. Through the Hough transform, the robustness and adaptability of the detection results are enhanced. There is some need for good dataset for rural road images which will broaden the application area of our project. The presence of noise is just one of the drawbacks but it can be rectified using good noise minimizing algorithms. We have not used object detection and tracking but it can be used to further incorporate more features into the algorithm.

VIII. FUTURE SCOPE

To further improve the robustness of the algorithm, some other methods can be considered in the future, such as, lane detection can be implemented using guided image filtering technique, for example, Gabor filter. Another issue to be dealt with is fog removal. In the near future, one can modify the existing Hough Transformation to measure both the curved and straight roads. Various steps should be taken to improve the results in different environmental conditions like sunny, foggy, rainy, etc. Further, the system can be expanded to include not only lane lines, but also road sign recognition as well as lane detection on rural or lesser travelled roads.

REFERENCES

[1] Jianfeng Zhao, Bodong Liang, and Qiuxia Chen, "The key technology toward the self-driving car.", International Journal of Intelligent Unmanned Systems, Vol. 6, No. 1 (2018).

[2] Mohamed Aly, "Real Time Detection of Lane Markers in Urban Streets.", IEEE Intelligent Vehicles Symposium,

Eindhoven University of Technology, Eindhoven, The Netherlands (2008).

[3] https://docs.opencv.org/4.x/

[4] Hongwei Guo, "A Simple Algorithm for fitting a gaussian Function," IEEE Signal Processing Magazine, September 2011.

[5] Y. Wang, E. K. Teoha, and D. Shen., "Lane detection and tracking using B-Snake.", Image and Vision Computing 22, pp: 269-28 (2004).

[6] https://www.cse.unr.edu/~bebis/CS791E/Notes/AreaProcess.pdf

[7] Raman Shukla et al., "Lane Detection System in Python using OpenCV," IJIRT, Volume 8 Issue 1, June 2021.

[8] F. Mariut, C. Fosalau and D. Petrisor, "Lane Mark Detection Using Hough Transform.", IEEE International Conference and Exposition on Electrical and Power Engineering, pp. 871 – 875 (2012).

[9] K. Ghazali, R. Xiao, and J. Ma, "Road Lane Detection Using H-Maxima and Improved Hough Transform.", Fourth International Conference on Computational Intelligence, Modelling and Simulation, pp. 2166-8531 (2012).

A.5 PAPER PUBLICATION

International Journal of All Research Education & Scientific Methods

UGC Certified Peer-Reviewed Refereed Multi-disciplinary Journal
ISSN: 2455-6211, New Delhi, India
Impact Factor: 8.536, SJR: 5.89, UGC Journal No. : 7647

Acceptance Letter

Dated:25/03/2024

Dear Authors,

We are glad to inform you that your paper has been accepted as per our fast peer review process:

Authors Name: **Chintham Reddy Lalithaditya, Mrs.S.Sophana Jennifer**

Paper Title: **Road Lane-Line Detection using Python, OpenCV**

Paper Status: **Accepted**

Paper Id: **IJ-2403243649**

for possible publication in **International Journal of All Research Education & Scientific Methods, (IJARESM), ISSN No: 2455-6211", Impact Factor : 8.536**,

in the current Issue, **Volume 12, Issue 3, March- 2024**.

Best Regards,

Editor-in-Chief,
IJARESM Publication, India
An ISO & UGC Certified Journal
http://www.ijaresm.com