IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 26, Issue 4, Ser. 3 (July. – August. 2024), PP 32-42
www.iosrjournals.org

Robust Autonomous Navigation With Lane Tracking

Syed Rayan[1], Mohammed Abdul Wali Farooqui[2], Mrs. Ayesha Mariyam[3]
[1,2] Be CSE(AI&DS) Undergraduate Student, Muffakham Jah College Of Engineering & Technology, Hyderabad, Telangana State, India, 500 034
[3]Assistant Professor, CS&AI Department, Muffakham Jah College Of Engineering & Technology, Hyderabad, Telangana State, India, 500 034

Abstract:
Enhancing road safety and reducing accidents to save lives remains an essential goal for Advanced Driver Assistance Systems (ADAS). A foremost challenge self-driving vehicles face involves robustly detecting lane paths and road boundaries amidst dynamic driving conditions. At its core, this operation relies on lane detection protocols that pinpoint the roadway, discern the vehicle's position within it, and analyze the current direction. Leading techniques apply computer vision to reliably extract lanes from feed from on-board cameras. However, performance consistency eludes existing vision-based detectors when illuminated scenes transition to shadows or indirect lighting. This research puts forth an augmented vision-based method to enable precise, real-time lane detection resilient to light variation. Front-facing cameras supply road imagery for processing algorithms that filter and fit hyperbolic lane edge candidates via Hough transforms. Unlike predecessors confined to high-contrast straight lanes, this technique reliably handles low-contrast, curved, unpainted lanes across diverse lighting or weather. Extensive testing proves the method's swiftness and accuracy across street types validates its value for integration in next-generation ADAS. In summary, this paper presents an adaptable solution to a critical ADAS capability – reproducible lane detection despite ubiquitous challenges. The techniques' consistent roadway demarcation under diverse conditions positions this method to heighten autonomous navigation safety.

Key Word*: Hough Line Detection, Raspberry Pi, Lane Boundary Detection, Edge Detection.*

Date Of Submission: 22-07-2024 Date Of Acceptance: 02-08-2024

I. Introduction

Research into intelligent vehicle technologies has expanded lately, with Advanced Driver Assistance Systems (ADAS) integrated in commercial vehicles to improve autonomous safety. Lane detection critically enables ADAS by supplying drivers core lane intelligence like structure and vehicle positions. Precision outputs and minimal latency prove vital for reliable awareness. By bolstering prompt, accurate lane boundaries amidst diverse, dynamic driving settings, innovations to lane detection algorithms and sensors will prove pivotal in unlocking ADAS potential to lower accidents via enhanced assistance across complex environments. Further real-time, robust systems stand to heighten autonomous navigation safety [5].

Vision-based Lane detection involves locating boundaries without prior roadway data. Its accuracy directly enables reliable lane tracking across frames. While numerous executable-lane marking detection approaches exist, achieving real-time performance given hardware and software constraints remains vital for viable systems. Moreover, lane perception forms just one component within extensive visual analysis like segmentation and sign recognition required for semi-autonomy. Thus, lane detection methodology must balance processing simplicity and efficiency to meet tight response times and computational limits of embedded self-driving platforms. Further innovations targeting the precision-efficiency tradeoff that boost lane detection speed and accuracy will prove critical to advance dynamic scene understanding for autonomous navigation in diverse, complex settings.

Statistical techniques have been developed to classify lane and road lines depending on the intensity and color of painted lines with respect to the background color and intensity. HSI color model has been used to detect lane markings to overcome variability of lighting conditions. Edge detection-based models have been used to detect both marked and unmarked roads [7]. Hough Line transform-based – line detection has been adopted to overcome the occlusion problems. A fuzzy adaptive mechanism is used to enhance the contrast between the shadowed and unshaded pixel values to overcome the shadow problem. Our approach will be based on edge detection and Hough transform to detect road and lane lines.

Autonomous functionality pervades increasing mobility platforms beyond just passenger vehicles, with innovations transferring from automotive prototypes towards drones, robots, warehousing equipment, and future smart infrastructure [2]. Core self-driving capabilities like perception, planning, and redundancy to GPS aid embedded autonomy across form factors. Miniature rovers, industrial shipping, robotic warehousing and last-mile delivery all stand to benefit from vehicle intelligence breakthroughs pioneered in the automotive realm. Cross-disciplinary collaborations on next generation transportation promise increased safety, efficiency, and accessibility powered by modular software and sensors that bring autonomous advantages within reach for myriad applications on roads as well as in the air and sea. The main aim of this research work is to detect both sides of the lane and maintain the vehicle inside of those lanes, to detect the curved road ahead of the vehicle, and to prepare a working model by using a Raspberry Pi. The following sections of this paper consists of literature review, methodology and experimental results.

II. Literature Review

According to the World Health Organization, approximately 1.25 million people die in traffic accidents each year. It is difficult to introduce contemporary systems in our country because they are manufactured in other countries. The methodology used in various papers related to this research are discussed below:

Kitae Hwang et al. [2] have proposed an affordable autonomous driving system test platform using a remotely-controlled car. A Raspberry Pi device installed in the RC vehicle handles low-level control. All self-driving algorithms connect via a separate AI server that performs intensive perception, planning, and control computations. This modular approach facilitates testing and enhancing critical functionality like lane holding, obstacle avoidance, sign recognition, parking, etc. Leveraging inexpensive off-the-shelf hardware and segregated processing enables rapid design, debugging and refinement of building blocks essential for real-world autonomous navigation. Overall, the platform lowers barriers for innovation by offering flexible, low-cost experimentation to refine algorithms and evaluate integration strategies for safe, reliable vehicle autonomy.

Michael G. Bechtel et al. [3] have proposed a work that presents DeepPi, an inexpensive autonomous car research platform. Hardware comprises readily available components: a Raspberry Pi computer controlling a web camera-equipped remote-control car. DeepPi employs real-time end-to-end deep learning for self-driving. A deep convolutional neural network model accepts each video frame from the forward-facing camera and outputs a predicted steering direction. This approach continuously generates control commands by directly linking raw sensor inputs to driving maneuvers via a trained deep neural architecture. Overall, DeepPi offers a customizable testbed to develop and evaluate deep learning powered autonomous functionality using easily accessed hardware and computation resources.

Ketan Bodhe et al. [4] have developed a system with new technology that uses a pattern matching process in which cameras detect a unique pattern that will be printed on the roads. This pattern will be recorded by the camera and processed by a Raspberry Pi before ordering the automobile to drive in the desired direction. Special patterns will be deployed alongside the route to assess what type of road is present ahead. The concept of collision detection is also implemented in this.

Mr. Nihal A Shetty et al. [5] puts forth a pattern matching based autonomous navigation system. Cameras identify printed markers on roadways to locate the vehicle and detect nearby obstacles from imagery. A Raspberry Pi processes the captured patterns and wider scenes to derive appropriate driving maneuvers. Specifically, matches to navigation symbols instruct steering, while analysis of surrounding pixels identifies proximate objects to trigger braking. Additional patterns deployed along the route indicate upcoming road traits to prime sensor parameters or driving policy adjustments. Overall, the approach offers a customizable testbed to develop and assess affordable computer vision-driven self-driving techniques with onboard computation and interpretation of meaningful symbols encoding navigation directives.

Chanho Lee et al. [6] has suggested that the technique encompasses three core phases: system initialization, lane marking detection, and lane marking tracking over successive frames. The first two stages set up requisite regions of inspection for robust tracking - demarcating a broad initial search rectangle alongside a narrowed lane angle-defined area. This lateral tracking via predefined zones constitutes the predominant operation, enabled by one-time identification of lane markers and estimation of orientation to focus the continuing lane boundary verification.

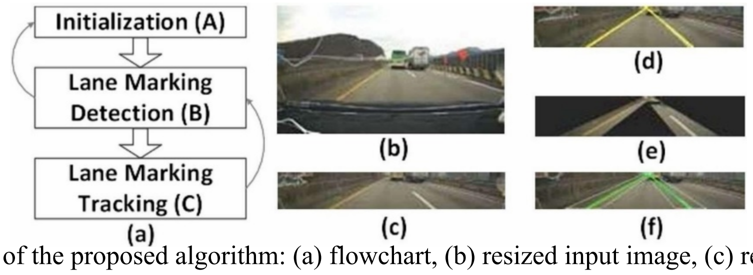


Figure 1: Procedure of the proposed algorithm: (a) flowchart, (b) resized input image, (c) rectangular ROI, (d) lane markings are detected in the rectangular ROI, (e) Λ-ROI, and (f) lane markings are tracked in the Λ-ROI.

Qiao Huang et al. [7] have proposed a principal benchmark for viable lane detection systems involving achieving real-time performance given current computational constraints. As lane markings generally appear straight in close-range imagery across common driving scenarios, this work adopts straight-line models instead of complex curves to represent roadways. Focusing on simple lane delineations sufficiently reveals limitations of prevailing detection algorithms - environmental influences like weather and lighting changes more significantly impact perception than road curvature alone. By probing failure points using simplified lane approximations, the approach emphasizes improving field reliability considering restricted hardware resources and stringent latency requirements inherent to autonomous navigation systems.

Gibrael Al Amin Abo Samra [8] has developed a system that has been implemented using MATLAB and tested on many road images in different places, at different day times and with different camera poses (positions and orientation). Considering comparisons with other systems, no-training and no-prior knowledge is needed for this system to work properly in most conditions. One criterion and five constraints have been applied on the selected maximum intensity Hough lines to detect the lanes lines. The output results showed that only targeted lines have been detected and distracting lines due to lighting effects, shadows, even real lines drawn due to vehicle friction have been rejected.

Most existing autonomous driving systems are designed for overseas driving styles, complicating adoption for India's road conditions. The challenges relating to the Indian driving style are:

1. Addressing navigation logic discrepancies between India's left-side driving system and foreign systems that assume right-lane operation poses a significant obstacle in the absence of proper localization, potentially leading to navigation errors and failures.
2. Coping with intense traffic chaos on Indian roads surpasses the capabilities of present algorithms, risking compromised safety.

The contributions of this paper involve:

1. Lane detection on both sides for precise vehicle positioning within lanes.
2. Advanced detection of upcoming curved roads.
3. Implementation of a functional model using Raspberry Pi for real-world application.

III. Existing Methodology

The provided figure illustrates the block diagram of a proposed lane detection system designed for Raspberry Pi.

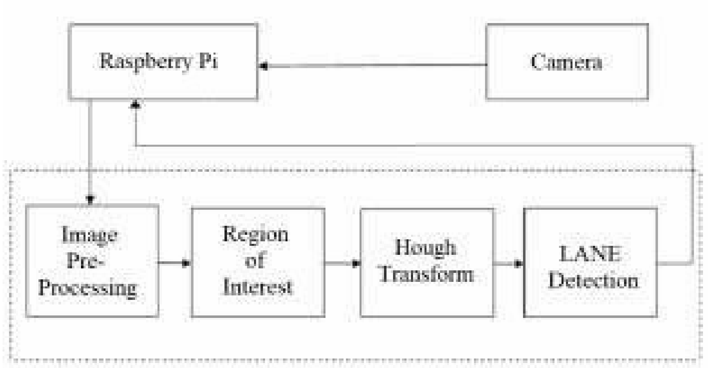


Figure 2: Block Diagram of Lane Detection System (of the Existing System)

Explanation of each block is given below:

1. Acquire input image: Hardware such as a camera is employed to capture the input image.
2. Image Preprocessing: Improving the image quality involves preprocessing. This includes applying processes such as noise reduction, edge detection, and managing contrast and color.
3. Region of Interest (ROI): To assess the computational complexity of lane identification and the LDI system, the ROI is crucial for detection. Only the designated areas are identified or considered for subsequent processing. The images within these selected ROIs are then utilized in lane detection through a proposed algorithm. Choosing specific ROIs contributes to a reduction in the processing time for frames.
4. Hough Transform: Following the detection of Canny edges in images, the Hough Transform is applied to identify the desired pixels in the image. In this system, the utilization of Hough Transform is employed to detect lane markings from the image data.
5. Lane Detection: In this case, the lane is distinguished by using a distinct color.

Two important algorithms Canny Edge Detection and Hough Transform are used to implement Lane Detection System.

IV. Proposed Methodology

In this project, we have adopted a modular design strategy to enhance code organization and flexibility. Each distinct task is encapsulated within a separate module, allowing easy addition or removal of functionalities without extensive code modifications. This modular structure also facilitates the reuse of modules in different projects. At the core of our system is the main module, which serves as the orchestrator. It seamlessly connects with various modules responsible for specific tasks such as webcam interaction, motor control, and lane detection.

Webcam Module –

A process involving the capture of videos using a webcam, subsequent conversion of these videos into individual frames or images, and transmission of these frames to a main module for further processing. Initially, the system records videos in real-time through the webcam. These videos are then broken down into individual frames, essentially still images, which can be analyzed separately. The transformation of the video into frames allows for more granular analysis and manipulation of the visual information. Subsequently, these frames are transmitted to a central processing module, often referred to as the main module, where various computational tasks such as image recognition, computer vision, or other forms of analysis take place. This modular approach enables efficient and distributed processing of visual data, facilitating diverse applications in fields such as computer vision, surveillance, or any domain requiring real-time video analysis.

Figure 3: ESP32 Camera Module

Lane Detection and Tracking Module –

Utilizes a pixel summation method to analyze image columns. By summing up pixel values (0 for black, 255 for white), we determine the direction and degree of turning required. The calculated curve values are sent back to the main module for further action.

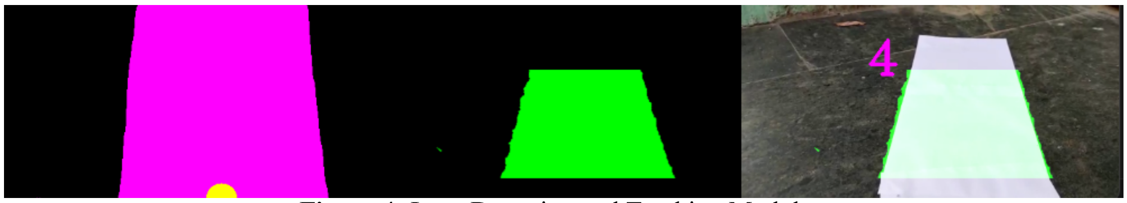


Figure 4: Lane Detection and Tracking Module

Utility File –

MainRobotLane.py

This is the main script that runs the lane detection pipeline and controls the robot's motors. It imports the necessary modules (Motor, getLaneCurve, WebcamModule) and defines the main loop. In the loop, it captures an image from the camera, runs the lane detection algorithm (getLaneCurve), and then controls the motor based on the detected lane curve.

LaneDetection.py

This file contains the core lane detection algorithm. It processes the input image through various steps, including thresholding, perspective transformation, histogram analysis, and curve fitting. The getLaneCurve function returns the estimated lane curve, which is used to control the robot's motors.

Utlis.py:

This file contains utility functions used by the lane detection algorithm. It includes functions for image thresholding, perspective warping, trackbar initialization, drawing points on images, getting the histogram, and stacking images for display purposes.

WebcamModule.py

This module handles the camera interface. It provides a function getImg to capture an image from the camera and resize it to a specified size.

MotorModule.py

The file provides a convenient way to control two motors connected to a Raspberry Pi using GPIO pins. The Motor class encapsulates the logic for setting motor speeds and directions, allowing users to control the robot's movements by calling the move and stop methods with appropriate arguments.

Motor Module –

Receives curve values from the main module. Controls the robot's motors based on specified speeds and turn angles. This structured approach not only ensures clean and strategic coding but also allows seamless integration and removal of modules, providing adaptability to different projects.

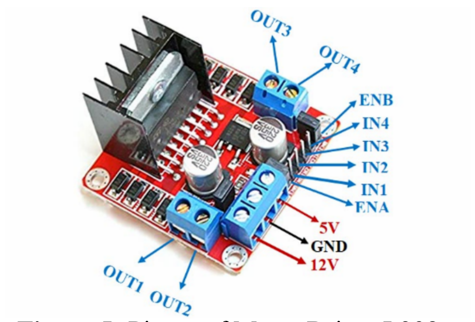


Figure 5: Pinout of Motor Driver L298n

Despite deploying the code on a Raspberry Pi, we choose to develop and debug on a PC for efficiency. This two-step process streamlines coding, accelerates debugging, and enables easy integration into the Raspberry Pi environment once each module is prepared. Our lane detection method, employing the pixel summation technique, is straightforward yet effective. This step-by-step development and testing approach ensures clarity at each stage, contributing to the project's overall success.

Working Structure of Lane Detection –

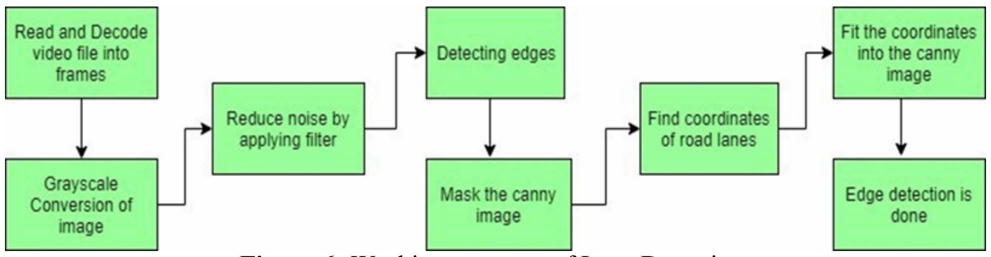

Read and Decode video file into frames
Grayscale Conversion of image
Reduce noise by applying filter
Detecting edges
Mask the canny image
Find coordinates of road lanes
Fit the coordinates into the canny image
Edge detection is done

Figure 6: Working structure of Lane Detection

Detecting the region of interest (ROI - road boundaries) is an important task in adaptive traffic monitoring and adaptive signal control systems. The region of interest can assist in lane finding and enable the system to adapt to different environmental conditions and camera viewing positions to find an optimum view. To detect the region of interest, we select the edge-based approach using Hough line transform as the main technique for detecting straight lines, introducing some practical enhancements, and fine-tuning methods to get better results. The detection is performed in three steps: edge detection, Hough transform, and line selection.

Working Process –

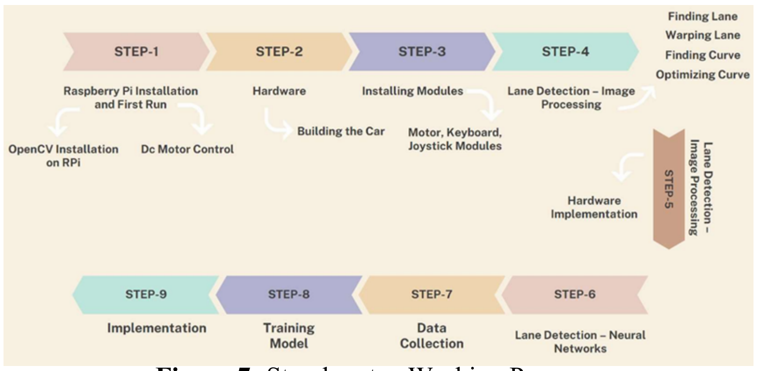


Figure 7: Step-by-step Working Process

Step – 1: Raspberry Pi Installation and First Run
Installation of OpenCV and DC Motor control is to be done first within the Raspberry Pi that is being used.

Step – 2: Hardware
The assembling of hardware components together for building the car needs to be done carefully and with at most care.

Step – 3: Installing Modules
The models such as the joystick, mouse and keyboard are to be installed for using them.

Step – 4: Lane Detection – Image Processing
This is the most important step, to detect the lanes and the curves. The dataset of different images is being trained. The image-processing is done to detect the lane boundaries. Finding Lane, Wrapping Lane, Finding Curve, and Optimizing Curve is done in this step.

Step – 5: Hardware Implementation
This step also does the image processing, but this is now implemented in the hardware model.

Step – 6: Lane Detection – Neural Networks
In this step, A dual model utilizing instance segmentation is constructed using a Convolutional Neural Network (CNN).

Step – 7: Data Collection
The data is collected in this step, the training dataset and testing dataset, which are then ready for processing.

Step – 8: Training Model
In this step, the training model is trained by the help of training dataset and the needed algorithms are used to make the Machine Learning Model efficient and increase the accuracy.

Step – 9: Implementation
The implementation is the final step where we test our model with the help of a testing dataset. We can take real-life examples and try to test our model on those as well. The accuracy plays a vital role here as the training and test datasets are not completely same, so the model should be able to accurately detect the lane boundaries and the road curves.

V. Product Image

This prototype has the rigid body of the chassis that is attached with 4 tires including the DC Motors, that relate to a Motor Driver L298n. Front-side has the Pie Cam setup that is used to scan the area ahead. A Raspberry Pi 3 Model B/B+ is residing on the chassis that has the Pie cam connected to it. A Battery of 3.7V is constantly powering this component for its functioning. Jumper wires have been used for connections of the various said components to their intended ports.

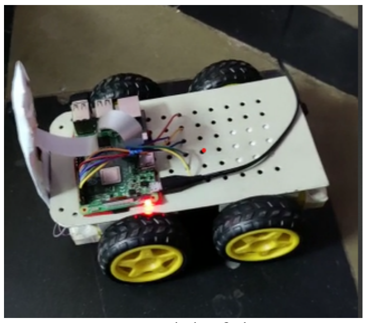


Figure 8: Prototype Model of the Proposed System

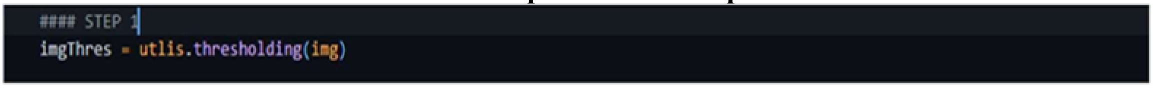
VI. Tech-Stack Used

A Software Requirements Specification (SRS) fully describes the intended behavior of a software system. It contains use cases outlining user interactions, as well as non-functional requirements that impose design and implementation constraints like performance, quality standards, and limitations. The SRS provides a complete set of functional user interactions and non-functional technical requirements for the system under development.

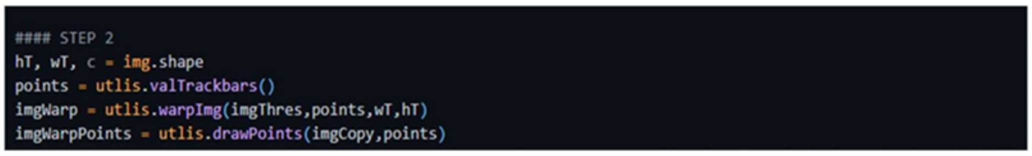
Table 1: Software Requirements and Hardware Requirements

SOFTWARE REQUIREMENTS	HARDWARE REQUIREMENTS
Python - Version 3.9	Laptop/PC/Raspberry Pi Model B/B+
OpenCV - Version 4.7	Processor: i5 and Above
Windows - Operating System	Speed: 1.1Ghz and Above
Raspberry Pi - Operating System (64-bit)	RAM: 4GB and Above
	Hard disk: 1TB
	Mouse: Optical mouse
	DC Motors
	Motor Driver L298n
	Raspberry Pi 3 Model B/B+
	Battery Case
	Batteries 3.7V
	Jumper Wires
	Tires
	Chassis
	Pie Cam of 5MP

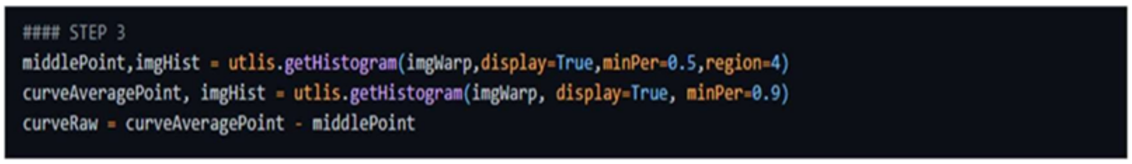
VII. Experimental Setup


```
#### STEP 1
imgThres = utlis.thresholding(img)
```
Figure 9: Code for Thresholding Function

We can detect the white A4 paper path using either color or edge detection, but we will implement color detection through a thresholding function in Utilities since that simplifies finding the path based on its color.
Here we are simply converting our image to HSV color space and then applying a range of colors.


```
#### STEP 2
hT, wT, c = img.shape
points = utlis.valTrackbars()
imgWarp = utlis.warpImg(imgThres,points,wT,hT)
imgWarpPoints = utlis.drawPoints(imgCopy,points)
```
Figure 10: Code for Converting Image to HSV Color Space

We only need to process the current curve of the path, not ahead, so cropping the image provides the immediate area. However, a bird's eye view is required to easily find the curve, so we'll warp the cropped image to an overhead perspective by defining initial points. This gives an angled view of just the nearby path to detect the current curve.



```
#### STEP 3
middlePoint,imgHist = utlis.getHistogram(imgWarp,display=True,minPer=0.5,region=4)
curveAveragePoint, imgHist = utlis.getHistogram(imgWarp, display=True, minPer=0.9)
curveRaw = curveAveragePoint - middlePoint
```

Figure 11: Code for Cropping and Wrapping the Image

Since our warped binary image has only black or white pixels, we can sum the pixel values vertically to find the curve in the path. This summation of pixels in the y-direction allows us to identify the curve by analyzing the distribution of black and white pixel values.

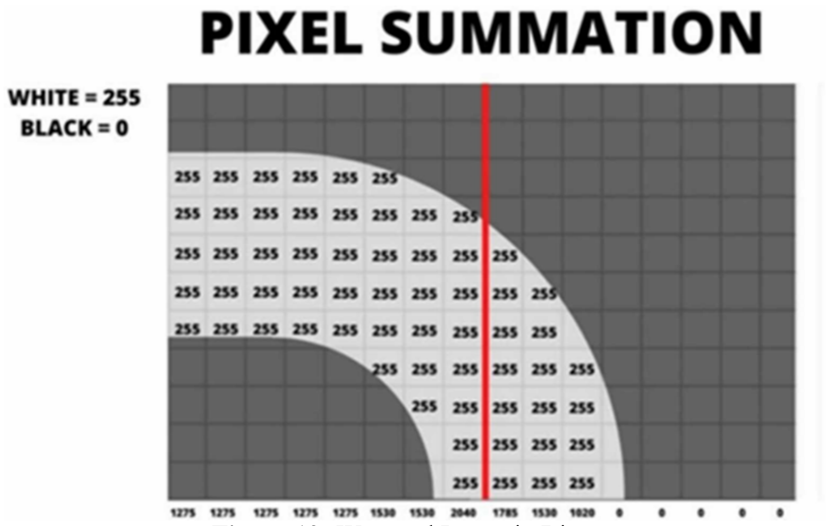


Figure 12: Wrapped Image in Binary

The picture above shows all the white pixels with 255 value and all the black with 0. Now if we sum the pixels in the first column, it yields 255+255+255+255+255 = 1275.

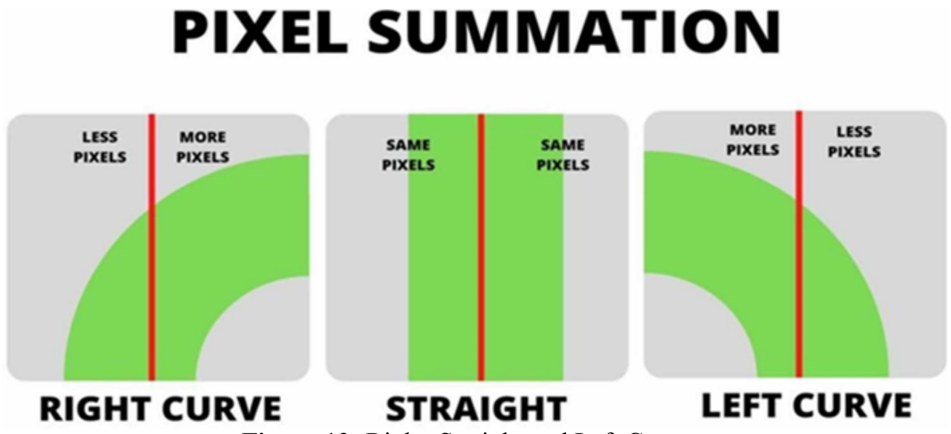


Figure 13: Right, Straight and Left Curves

When the path curves right, there are more pixels on the right side, and vice versa when it curves left. A straight path has roughly equal pixel amounts on both sides since the distribution is even. Analyzing the pixel sums on each side reveals information about the direction and severity of curves.

About the Dataset –

Our custom dataset comprises more than 3500 annotated images of trained scenes in the jpg format, meticulously curated to cover a wide range of real-world scenarios. Each image is accompanied by pixel-level annotations of lane markings, providing invaluable ground truth for training and evaluating lane detection algorithms. The dataset encompasses diverse lighting conditions, weather conditions, road geometries, and traffic scenarios, ensuring the robustness and generalization of our lane detection model and by confirming the quality and consistency of the manual annotations will be important for optimal model performance and it is generated using a script in which it converts the video into images with respect to frames of the video.

VIII. Implementation

STEP – 1: Data Collection

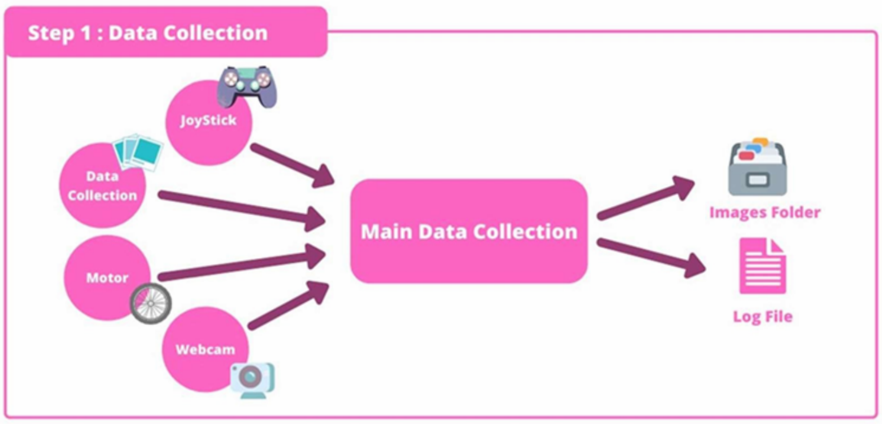


Figure 14: Step – 1 Showing the Data Collection step

In this step we create the data set using the pi with the help of a joystick in which the data set may contain about 2000-2400 pictures of data (It may vary) which is saved in the images folder with the log file. It contains the timestamp of every photo which is taken in the comma separated value format.

STEP - 2: Training

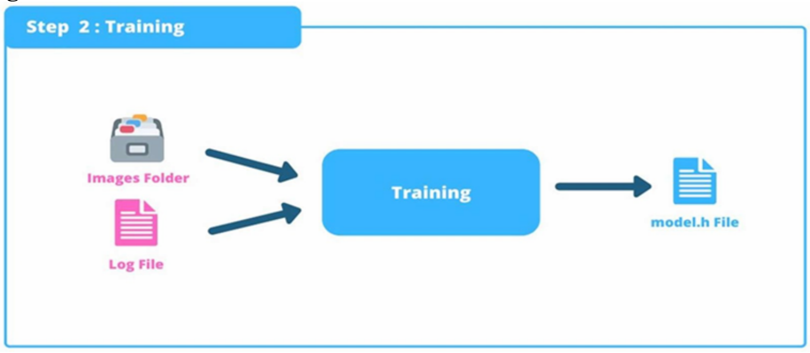


Figure 15: Step – 2 Showing the Model Training step

Now in this step we take the file/data which was created in step 1, now we select the image folder and Log file and train the model with the help of TensorFlow for building the model, Pandas for Data Frame. Here is the detailed explanation.

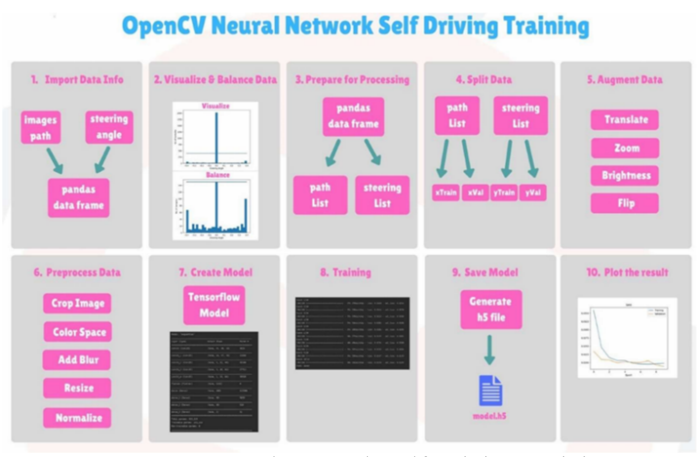


Figure 16: Neural Network Self Driving Training

STEP - 3: Implementation

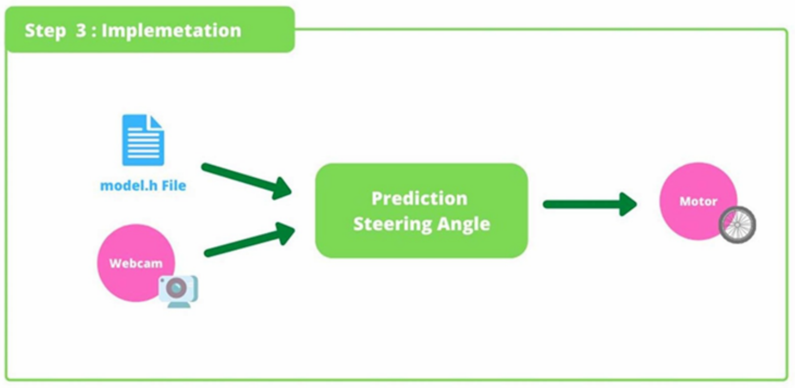


Figure 17: Step – 3 Showing the Implementation step

Now we collect the model .h file which was created in step – 2, now implemented on the main.py file which is used to predict the angle of steering in the lane detection.

IX. Result & Discussion

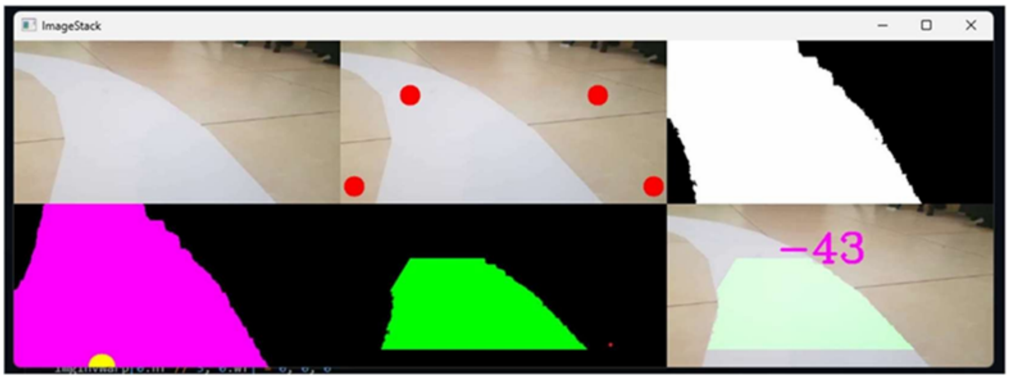


Figure 18: Output of the Prototype

We can see that the lane boundaries are being correctly detected using Edge detection and Hough Transform. We get our ROI and display the type of curve on the road that is ahead of us.

Our model car was successful in keeping itself inside of the lane boundaries, and was able to tackle the sharp turns of the road ahead of it in an efficient way.

To minimize storage space and reduce computing complexity, the original images are scaled down into pixels. In the suggested methodology, the RGB segmentation methodology is used to do picture pre-processing following the picture acquisition phase. In the suggested method, a filter is applied to each channel threshold area to pick only those portions of the image where pixel values are within the range of the target item. To smooth down the image and fill up the smaller areas, the median filter is utilized.

X. Conclusion

The goal of our proposed system for autonomous vehicles is to make vehicle operation more efficient by reducing human effort. After careful testing, we found that our system outperforms the performance of the typical human driver. In addition, our system has a higher degree of consistency, which leads us to believe it has the capability to reduce basic human error in driving. Such an improvement in performance is especially important in situations where human drivers can exhibit inconsistencies or lapse in attention. We have also addressed the challenges of lane detection for self-driving vehicles. We understand the complexities of changing road conditions and diverse markings, as well as occlusion from other vehicles, as well as variations in lighting, shadows, etc. Our proposed solution for self-driving lane detection demonstrates robustness against these real-world challenges. All in all, our proposed solution for autonomous vehicles not only reduces human effort for vehicle operation, but also provides superior, consistent performance, making it a very promising solution for improving road safety and overall efficiency.

XI. Future Scope

The lane detection model has a lot of room for improvement with more effective mathematical modeling techniques. Beyond its current applications, this technology is relevant in a variety of areas such as traffic monitoring and flow analysis, control and security systems. An interesting avenue for further development is to integrate the lane detection model with computer vision algorithms specially designed for obstacle detection. This would enable the development of a powerful collision warning system. By combining the technology with other relevant parameters, the model would be able to support the development of a sophisticated lane departure warning system that would improve overall road safety. With its versatility and adaptability, the lane detection model is well-positioned for future improvements and new applications, making it an invaluable asset in the continuous development of intelligent transportation systems.

References

[1]. Mr. Mustafa Surti, Dr. Bharati Chourasia, "Real Time Lane Detection System Using Python And Opencv On Raspberry Pi", Ijream Vol-05, Issue-06, Sep 2019.

[2]. Kitae Hwang, In Hwan Jung, Jae Moon Lee, "Implementation Of Autonomous Driving On Rc-Car With Raspberry Pi And Ai Server", Webology, Volume 19, Number 1, January, 2022.

[3]. Michael G. Bechtel, Elise Mcellhiney, Minje Kim, Heechul Yun, "Deeppicar: A Low-Cost Deep Neural Network-Based Autonomous Car", University Of Kansas, Usa, Indiana University, Usa 30 Jul 2018.

[4]. Ketan Bodhe, Himanshu Taiwade, Janhvi Ingle, Juhi Patre, Kajal Singh, Sakshi Polkamwar, Shreeya Bajirao, "Image Processing & M.L Based Driverless Car With Image Detection System", Priyadarshini Institute Of Engineering And Technology, Nagpur, Maharashtra, India. Vol-7 Issue-3 2021.

[5]. Mr. Nihal A Shetty, Mr. Mohan K, Mr. Kaushik K, "Autonomous Self-Driving Car Using Raspberry Pi Model", International Journal Of Engineering Research & Technology (Ijert), Special Issue – 2019.

[6]. Chanho Lee, Senior Member, Ieee, And Ji-Hyun Moon, "Robust Lane Detection And Tracking For Real-Time Applications", Ieee Transactions On Intelligent Transportation Systems, Vol. 19, No. 12, December 2018.

[7]. Qiao Huang And Jinlong Liu, "Practical Limitations Of Lane Detection Algorithm Based On Hough Transform In Challenging Scenarios", International Journal Of Advanced Robotic Systems March-April 2021: 1–13.

[8]. Gibrael Al Amin Abo Samra, "Automatic Detection Of Roads/Lanes Boundaries Using Geometric Constraints Imposed On The Hough Transform Detected Lines", Ain Shams Journal Of Electrical Engineering (Asjee) Vol. 2. December, .2009, Pp.131-142.