2020

Lane Detection of Autonomous Vehicles Using Clustering and Augmented Sliding Windows Techniques

Keerti Chand Bhupathi
z1841800@students.niu.edu

ABSTRACT

LANE DETECTION OF AUTONOMOUS VEHICLES USING CLUSTERING AND AUGMENTED SLIDING WINDOWS TECHNIQUES

Keerti Chand Bhupathi, MS
Department of Electrical Engineering
Northern Illinois University, 2020
Dr. Hasan Ferdowsi, Director

In this thesis, detection of solid lines and dashed lines of lanes using augmented sliding window technique, clustering technique and combination of both augmented sliding window and clustering is discussed. The lane points are extracted using image processing techniques. The performance of lane detection using these techniques is analyzed. Lanes are simulated in a laboratory for the input data set. The input data set consists of curved lines of dashed and solid lines which split and merge. Further, partially obscured lanes are also tested with the algorithm. The center of the lanes from vehicle to horizon is found based on the lane width. The percentage of successful detection of lanes using various techniques is calculated and their performance is compared. The average detection accuracy of clustering and augmented sliding window is nearly 98% for the considered input set of video frames. Missing lane markings of obscured lanes are also estimated using the relative position of adjacent lanes.

Keywords*:* lane detection; autonomous vehicles; clustering; augmented sliding windows

NORTHERN ILLINOIS UNIVERSITY
DEKALB, ILLINOIS

AUGUST 2020

LANE DETECTION OF AUTONOMOUS VEHICLES USING CLUSTERING AND AUGMENTED SLIDING WINDOW TECHNIQUES

BY

KEERTI CHAND BHUPATHI

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

Thesis Director:
Dr. Hasan Ferdowsi

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Hasan Ferdowsi, for his valuable suggestions and support in completing this thesis. He has extended great advocacy and encouragement in the completion of this thesis. Furthermore, I would also like to extend my sincere gratitude towards my advisory committee, Dr. Veysel Demir and Dr. Benedito Fonseca, for their support throughout my thesis work.

I would like to thank my parents and my brother for their prayers and continuous support. I would like to thank my husband, who has always motivated and supported me in every aspect of my life.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Background

With growing traffic congestion and the number of road accidents, there is a need for the development of advanced driving assistance systems. Some of the advanced driving assistance systems which are available in most of the vehicles are anti-braking system, cruise control system, parking assist, blind spot assist and many more. Some of the primary objectives behind the development of autonomous vehicles are to reduce traffic congestion, reduce fuel consumption, increase safety and decrease the number of road accidents due to human error. Autonomous vehicles are self-driving vehicles. Fully autonomous vehicles are expected to roam on the roads in full potential in the near future. Several researchers have been working on developing autonomous vehicles. According to NHTSA [1] and SAE [2], there are six levels of autonomy. Level 0 is no driving automation and the human driver handles everything manually. Level 1 provides driver assistance by taking control of either steering or acceleration/deceleration of the driving task. For example, Level 1 functions can be stated as electronic stability control, automatic braking, etc. Level 2 provides partial driving automation by taking control of both steering and acceleration/deceleration like adaptive cruise control with lane keeping.

Level 3 is a conditional driving automation, which handles all driving tasks under certain conditions, but the human driver should always be ready to take back control whenever needed. Level 4 is a high driving automation level that performs all driving tasks, in certain circumstances, without having any expectation that the human driver would intervene. Level 5 is a fully driving automation, where the vehicle should perform all the driving tasks in all conditions, without any expectation of human intervention. However, the human driver should always be perceptive to the driving environment.

Some of the necessary functions of autonomous vehicles include sensor system, planning system, communication system, user interface system and control system. The vehicle receives information about the surrounding environment in the form of inputs through a sensor system. Briefly put, in an autonomous vehicle, these inputs are processed, planned and provide outputs that control the vehicle in a safe and optimum way with an expectation to be better than a human driver. Figure 1 shows different types of sensors which can be used in autonomous vehicles.

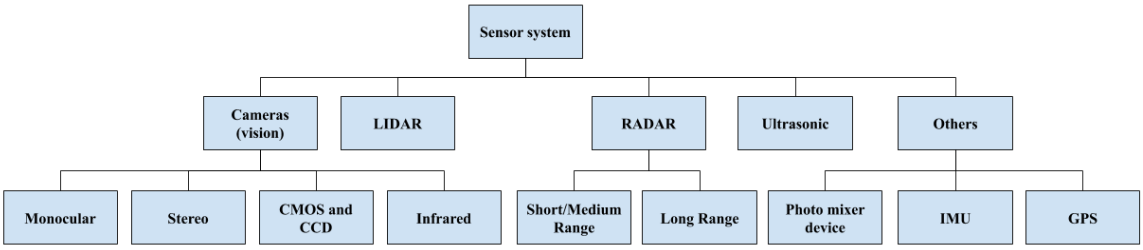


Figure 1. Different Types of Sensors Used in Autonomous Vehicles.

Sensor System

Cameras

Cameras are the vision sensors where one or more cameras are used to detect, analyze, track the color and contrast of the images. Different types of cameras that can be used are explained below.

Monocular camera. It consists of one lens and produces one image output at any point of time. It can be used for detection of obstacles, pedestrians, lanes, traffic sign monitoring, etc. It might not be reliable for distance estimation as it lacks depth information.

Stereo camera. It consists of two or more lenses. It extracts information from two or more images. It is useful in extracting 3D information. It can be used for traffic sign recognition, lane, pedestrian and obstacle detection. It is more accurate than a monocular camera.

CMOS and CCD. In each pixel of complementary metal oxide semiconductor (CMOS) cameras, photons are converted to voltage separately. Transistors measure and amplify the signal from each pixel. In charge coupled device (CCD) cameras, the total number of photons per pixel is measured [3].

Infrared cameras. These are of two types. Active cameras and standard digital cameras or passive cameras. Active cameras illuminate the scene. Passive cameras use an infrared sensor where every pixel is considered as a temperature sensor that can capture thermal radiation emitted by any material. Their range is about 750 to 1400 nm.

Lidar

It is a light-detection and ranging sensor generally mounted on the top of the car, that scans the surrounding environment and provides a 3D representation of the target. It measures the distance of the vehicle or any object from the target by calculating the time taken for the pulsed laser light to reflect back from the target to the sensor position [4]. It has a range up to 100m. They can be used for emergency braking systems, obstacle detection, collision avoidance, etc.

Radar

Radars are operated in short range (0.3-30m), medium range (30-80m) and long range (80-200m). They emit microwaves and estimate the speed and distance of the target by measuring change in frequency. They are used for cross traffic alerts, blind spots, adaptive cruise control, etc.

Ultrasonic Sensors

They operate within a range of 25-400cm. They generate sound waves to determine the presence of an object within the range of the sensor. It can be used to assist in parking. This sensor can only be used at low speeds.

Other Sensors

Photonic mixer devices (PMD) are smart sensors enabling fast optical sensing, faster imaging and high resolution. Inertial measurement unit (IMU) and Global positioning system (GPS) are useful for localization of the vehicle. IMU and GPS improve the distance measurements with lidar and radar. The functions of different sensors in an autonomous vehicle are shown in Figure 2. No single sensor provides all the necessary data. Thus, information from different sensors is linked to produce a common object representation, i.e., sensor fusion [5], to enhance the performance of the system.

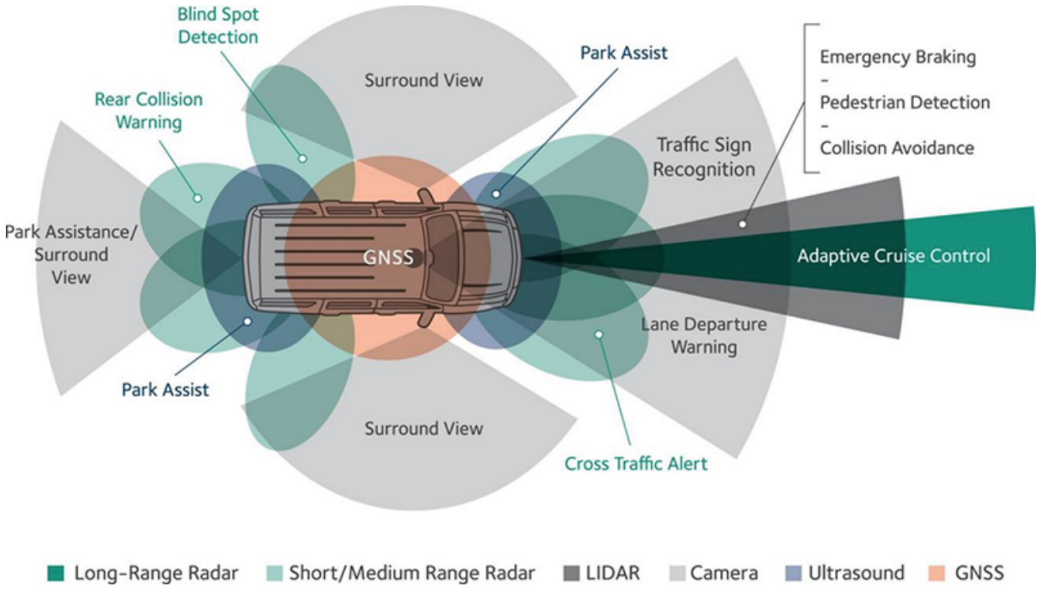


Figure 2. Representation of Functions of Sensors [6].

Planning System

The sensor data is given to the planning module, which processes the input and plans a path and sends this information to the control system. According to [7], the planning system is divided into three levels. Task level plans a proper mission schedule and travelling routes. Action level analyzes and converts the mission into a set of actions and responses for the problems with the surrounding environment. Motion planner searches the optimum trajectories to fulfill the assigned actions like taking a turn, collision avoidance, etc. The control system calculates the proper throttle and steering angle.

Communication System

Communication system is responsible for the interaction between the autonomous vehicle and external system. It can be of three types: Vehicle to Infrastructure (for example, receiving the information about traffic signal timing), Vehicle to Pedestrian (for example, interaction with pedestrians [8] when they are crossing the road, at intersections, etc.) and Vehicle to Vehicle (for example, speed information of the ego-vehicle to maintain a safe distance). User interface system generally consists of USB ports, touch screen, microphone, speakers and other smart features.

Control System

It mainly consists of steering unit, throttle unit and braking unit, along with other hardware units like engine, controllers, wheels, etc. Steering unit adjusts the direction or heading of the vehicle. Throttle or acceleration unit controls the speed of the motor/engine of the vehicle. Braking unit decelerates the vehicle by providing friction to slow the vehicle. There is an automatic control system of the vehicle that controls objects. In [9], the control system computer receives the information about the vehicle's state through CAN bus. The control commands are used to control the vehicle's steering, gear, brakes, acceleration, lights and horn, etc.

Some of the key functionalities in autonomous vehicles are lane detection, localization, obstacle avoidance, pedestrian detection, adaptive cruise control, traffic sign detection, traffic light detection, pose estimation, path planning and many more. The vehicle uses various vision cameras

(sensors as stated above) to receive information (for example, images) of the surrounding environment. This information might contain lanes, road signs, road signals, vehicles around it, etc. In this paper, lane detection is analyzed. Lane detection consists of localization of road, calculation of the relative position between vehicle and road, and the heading direction of the vehicle [10].

Lane detection is one of the important functions in the perception of an autonomous vehicle to steer around. The problems that are encountered in lane detection are due to the shadows of large trees and buildings, improper/erased lane markings, light (glare caused by bright sun) and extreme weather conditions like heavy rain, snow, fog, etc.

In ideal conditions, using only lanes and road information, lanes can be easily identified. But this is not the situation all the time. While a vehicle is moving, there are some unidentifiable obstacles and objects on the road that come across the view of the vehicle, that have to be distinguished from the lanes. Hence, this causes difficulty to identify lane line points. There are also many constraints to consider for implementing lane detection. Depending upon the brightness of the image, the sensitivity of color thresholds, and the edge detection thresholds, kernel size for image blur filter should be adjusted. Also based on camera position, the region of interest selection and coordinates for a top-view image are selected.

Python programming language is used to code along with other necessary libraries like OpenCV (a library of programming functions mainly aimed at real-time computer vision), NumPy (a library for Python programming language that operates mathematical functions on arrays) and Sklearn (scikit-learn is a machine-learning library for Python with various classification, regression, clustering algorithms, etc.).

In this paper, lane detection of straight lines using Hough transform, lane detection of curved lines using augmented sliding window technique, lane detection of curved lines using clustering technique, and lane detection of multiple curved lines using augmented sliding window and clustering technique are analyzed and the performance of the algorithms is compared. The lane detection algorithms are tested on images extracted from a video obtained from the camera sensor fitted on the vehicle. The camera sensor used is an Intel RealSense depth series camera.

The images from the camera are processed through a series of steps in a particular algorithm. Various techniques and algorithms that are used for image processing are explained in Chapter 2. Chapter 3 explains all the lane detection algorithms that are analyzed in this paper. Chapter 4 describes the comparison of different lane detection techniques and their performance. Chapter 5 deals with conclusion and future scope.

Problem Statement

The lanes can be of various types and there can be single lanes or multiple lanes formed by dashed, solid, straight or curved lines with different curvatures. Hough transform, sliding window technique and clustering are some of the techniques which can be used to detect lane lines.

Hough transform, sliding window technique and clustering algorithms are analyzed, and detecting straight lines and slightly curved lines work well using these techniques. But when the lanes curve sharply or turn, these techniques do not efficiently track the lane lines. In this paper, another approach, called augmented sliding window technique, which efficiently tracks sharply

curved lanes and lanes with sharp turns, is explored. Augmented sliding window technique is also combined with clustering technique to efficiently detect dynamically changing lanes which split and merge. The performance of these algorithms on input video sequences that include solid lines, dashed lines and possibly obscured lines is studied.

Literature Review

Lane detection is very important for advanced driving assistance systems. In the early 1980's, due to the scarcity of computational power, researchers used to take a picture, analyze it and blindly drive for some distance before capturing the next picture [11]. In 1986, few 8086 processors and Kalman filters were used for lane detection. According to [10], several techniques that have been used for lane detection are GOLD algorithm, LOIS algorithm, AURORA and LANA algorithm, etc. GOLD algorithm is a generic obstacle and lane detection system which uses stereo images from a camera and performs perspective transformation [12]. LOIS algorithm is a deformable template approach. It locates the best lane edges by using a shape model, likelihood function and an optimization algorithm [13]. According to [14], AURORA is the acronym of automotive run-off-road avoidance system. It tracks the lane marks by using a bi-normalized adjustable template correlation method. LANA stands for lane finding in another domain algorithm, which uses frequency domain features and provides information about gradient values of edges [15].

Simply put, in order to implement lane detection, one should know where the lines are and those lines on the road are lanes. To know the information about where lines are, edge detection

techniques are used. To detect that those are lines, Hough transform is used by many researchers [10, 16, 17]. The input image considered for lane detection undergoes a series of image preprocessing steps which include color image to grayscale conversion and noise reduction by filtering, and road boundaries are detected by fitting hyperbola pairs to the edges of the lane after applying edge detection and Hough transform in [10]. The algorithm was able to detect the straight lines and slightly curved lines but could not detect the lanes during sharp curves and shadows.

A hyperbola pair road model is used for the lane detection in [18]. CCD cameras are used for image acquisition and the input image is processed for enhancement and edge detection. First a triangular lane region is considered, and shape and location information are acquired. A pair of boundary points is extracted based on the lane width distance. The input data set consists of images with straight and curve lanes and solid and dashed markings under different weather conditions. Average error rate is given as 2.99 %, which explains that detection rate is approximately 97%.

In [17], a road boundary and line detection algorithm combine Hough transform, canny edge detector and least square method for fitting and Kalman filter to minimize the adaptive region of interest. It also predicts future road boundaries location. The algorithm is simulated in a Pro-SiVIC simulator and exhibits good performance for discontinuous and dashed road lines. Only for sharp curves, the speed is limited to a certain extent. Polynomial equation model is used for fitting measured data with the help of least square error method. Such fitting would be complex in the case of sharp curves and it might not be a correct approximation of the original curved line.

Hough transformation can detect lines and circles but detecting curves is a little complex. The lanes on the road are either straight or curved lines (dashed and solid lines). So, other

techniques like inverse perspective transform, sliding windows, and clustering techniques are some of the techniques which can be used for advanced lane detection.

Reliable lane markings are used to detect the lane in [19]. Region of interest of grayscale image is selected. AND operation is performed on two different images, one is grayscale image and the other is Sobel edge-detected image. The output of AND yields a strengthened image based on a particular threshold value. Then, a BEV (bird's-eye view) image is generated and histogram is calculated. Sliding window (rectangles of fixed width and height) is scanned based on reliable lane marks using histogram. Least square method is used to find the parameters of second-order polynomial equation of all the points obtained from the sliding windows. The algorithm presented good performance for straight lanes, worn-out lanes, nonparallel lanes, and slightly curved lanes. But this algorithm might face difficulty for sharp curves as it needs higher model fitting than a second-degree polynomial.

A three-feature-based lane detection algorithm is presented in [20]. A lane vector is calculated using image processing techniques that generate lane boundary information. The starting position, direction/orientation and gray-level intensity values are the three features of a boundary. The algorithm continuously detects the lanes in the road and if the road width goes beyond a particular threshold value and has a big variation, then the current lane found will be discarded and the previous one is considered as the current one using a weighted distance metric. The constants of the metric are optimized using the evolutionary algorithm.

The performance comparison of the algorithm using manual selection of constants and using the optimization algorithm is presented. This three-feature-based automatic lane detection algorithm displayed better performance in inclined, dotted, shady and wet roads. The input data

set consists of 200 images and the paper didn't explain the performance of lane detection in sharp turns.

In [21], lane detection using line segment detection is introduced. To begin with, the non-road region is extracted, then the BEV image is obtained. Line segment detection is performed on the BEV image. It calculates a vector for all the pixels in the image frame. The vector gives information about magnitude and orientation of the pixel points of lines. The line segments obtained are filtered out based on distance and angle. Left and right lanes are categorized using angle and then the points obtained are fitted to a polynomial equation of second order. The results shown in the paper exhibit good performance with better execution time. Dashed lines and solid lines are differentiated, and lane keep assist is implemented. Using second-order polynomial curve fitting might not be suitable in all cases because of different properties of lines in the bottom half (x-coordinate value for a couple of y-coordinate points might be almost the same in image space) and top half of the image (especially in case of sharp curve, y-coordinate value would be almost the same for a couple of values of x-coordinate in image space). It would be really difficult to fit a sharp curve using a single equation.

The authors in [22] introduced a three-level (low-level, mid-level, high-level) processing model for lane detection. Low-level processing performs grayscale conversion, noise reduction, contrast enhancement, and ROI (region of interest) selection. Mid-level processing performs binary conversion, canny edge detection, thresholding based on image gradient orientation, stretching image and connecting edges (dilation), providing labels to connected pixels, selection of labels using histogram, and canny filter for edge thinning. High-level processing consists of dividing the cropped image into sections and applying Hough transform, and several thresholds

like angle, distance, etc., are checked to find the best lane points. The algorithm provided 87-98% accuracy and it faced few limitations due to glare, shadows and poor road infrastructure.

In [23], a bi-directional sliding window search method is introduced which uses line-by-line scanning for each lane boundary. At first, a fusion of edge distribution function and Hough transform is used to find lane boundaries. Then the starting point of the sliding window, lane width and center of the lane are initialized, followed by the application of bi-directional sliding window for the detection of lane points. A detection rate of 94 % is achieved.

In [24], input image is converted to bird's-eye view image (BEV) and a custom edge detection is applied on BEV image to reduce noise. Then lines are detected using a Hough transform and sliding window approach is applied in which the window is moved from bottom to top of the image to find meaningful or valid pixels in the window. Then obtained points are subjected to lane fitting. The algorithm yielded good results in multiple lighting conditions, lane changing and curved roads but failed for images consisting of lanes with irregular curves.

Sometimes, the clutter on the road or unnecessary objects are also considered as required points. Also, when detecting lanes during sharp turns, there is a possibility that the sliding windows of one lane might clash with another lane. In such cases, clustering techniques will be useful to detect the feature points.

In [25], an adaptive approach for segmenting the region of interest is employed. Line segments are detected, non-lane points are removed using the K-means clustering approach. Out of the remaining points, lane points are extracted. This approach provided an average correct detection rate of 93.15 %. However, there are a few limitations in detection of lanes due to curves and glare in the images.

Color-based clustering is done in [26]. The lane marks are extracted by segmenting the image using K-means clustering in CIE lab feature space. It was explained that this algorithm is more suitable for initial lane marking detection.

In [27], the line segments are clustered based on their slope and Y-intercept for the detection of left and right lanes. Initially, image preprocessing techniques (Sobel gradient operation and canny edge detection) are performed. Line segments are detected using probabilistic Hough transform. Then these line segments are clustered by using agglomerative clustering and weighted regression. The algorithm showed good performance for straight lines and curve-like segments when lane markings are missing or occluded.

In [28], lane detection is achieved using DBSCAN (density-based spatial clustering of applications with noise) clustering technique and RANSAC (random sample consensus) fitting. At first, the image is subjected to inverse perspective transform. The transformed image is converted to grayscale. Then the image is binary segmented based on adaptive thresholding technique. Next, morphological corrosion is applied on images to reduce any noise due to small objects and to smooth the boundaries. Feature extraction points are the inputs to DBSCAN clustering technique that result in different clusters. Parabolic fitting using RANSAC is done to fit the lanes.

The results shown in [28] mentioned an accuracy greater than 95% for different scenarios where parts of the lane marks are damaged, have light irradiation, interference and other factors. In [29], lane detection is achieved based on histogram calculation. The input image is converted to a BEV image, and the histogram of the BEV image is calculated. The peaks are separated using the mean-shift clustering algorithm. Then the sliding window is moved from the bottom to the top of the image to find the valid pixels. Lane fitting is done with a quadratic equation using the least

squares method. Using a single sliding window in case of sharp turns will not capture the orientation of the lane in all cases. This might lead to completely ignoring the sharp turn.

In [30], the authors put forth two lane detection models. The first model consists of a Gaussian steerable filter for edge detection and Hough transform. The second model uses Gaussian mixture model (an unsupervised machine learning data-clustering algorithm) for segmentation and hyperbola pair model using RANSAC (random sample and consensus). Initially, image preprocessing steps like region of interest and increasing contrast of images are applied and then the first model is employed to detect the lanes. Lane detection confidence value is given and if it is low, then a second model is used for detection. By the end, the Kalman filter is used to track the lanes. For performance analysis, sampling and voting scheme with prior geometric information are presented. The algorithm worked well for the problems like shadows, poor visualization, curvy roads, intersections and other road markings. It exhibited few limitations with lanes consisting of sudden curves (hyperbola-pair model could not approximate a smooth curve), missing or no lane markings and curbs (curbs are detected as lane boundaries).

CHAPTER 2

TECHNIQUES USED IN LANE DETECTION

The algorithms for lane detection mainly consist of extracting useful information from images and then trying to detect lanes. Image processing techniques like edge detection, color-based thresholding, perspective transform, etc., are used to extract lane information. To detect lanes, algorithms like sliding window, Hough transform, clustering, etc., can be used. Images are represented as matrices where rows and columns of the matrix correspond to image resolution. Storing images as matrices is useful to perform various operations on them. In this chapter, various techniques and algorithms used in this paper are explained.

Experimental Staging

Simulated Lanes in the Laboratory

Various types of lanes exist in the real world. Commonly observed lanes are formed with solid and dashed lines, where the outermost lines are solid lines and the lines that form intermediate lanes are dashed lines. The lanes can be straight or curved, and in the case of curved lanes, the radius of curvature varies in the lanes. Some lanes also have very sharp curvature, at which point

they can be categorized as lanes that either turn left or right. In some scenarios, the number of lanes increase and decrease depending on road conditions. Such lanes can be referred to as splitting and merging lanes. Generally, a lane that splits is always the outermost lane, and when a lane splits, a new dashed line starts besides it. When lanes merge, the dashed lane corresponding to the lane that merges inwards will no longer be visible. These lane properties are used to simulate various types of lanes in the laboratory. A black background and white tape are used to retain the color conditions of real-world roads and lanes. Table 1 show the lanes simulated in the laboratory. Tables 2 and 3 show some of the lane images as seen from the vehicles' camera and complex lane scenarios.

Table 1

Simulated Lanes

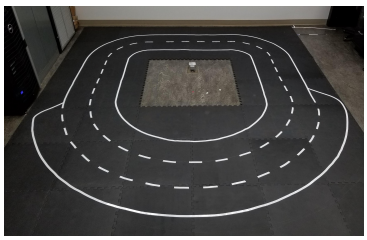
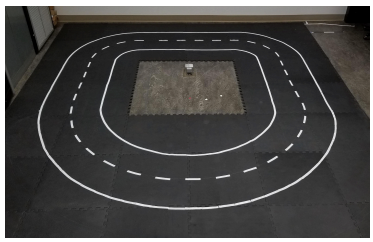
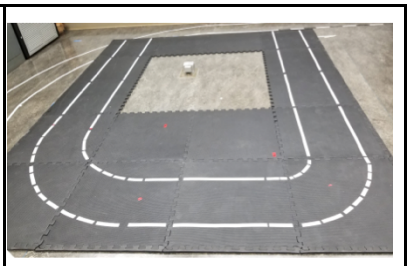
		

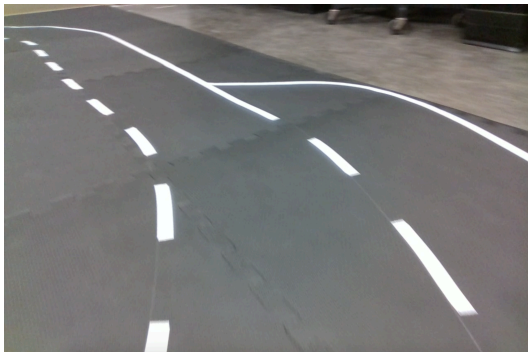
Table 2

Lane Images as Seen from Vehicle Camera

		
Straight Line with Solid Lines	Turning Lane with Solid and Dashed Lines	Sharp Curved Lane with Solid and Dashed Lines

Table 3

Complex Lanes

 Splitting Lane	 Merging Lanes

Intel RealSense Camera

The sensor for capturing input is an Intel RealSense D435 camera. Intel RealSense D400 series can be used for powerful image processing and depth perception. It is generally sold as shown in the Figure 3. It mainly consists of three parts. For example, in the case of D415, it consists of a depth module (top), interposer (middle) and D4 vision processor (bottom) as shown in Figure 4. The stereo depth module is connected to a 50-pin connector on vision processor D4 board using an interposer. The interposer can be a rigid interposer or flex interposer as shown in Figure 4.

Figure 3. Intel RealSense D400 Series.

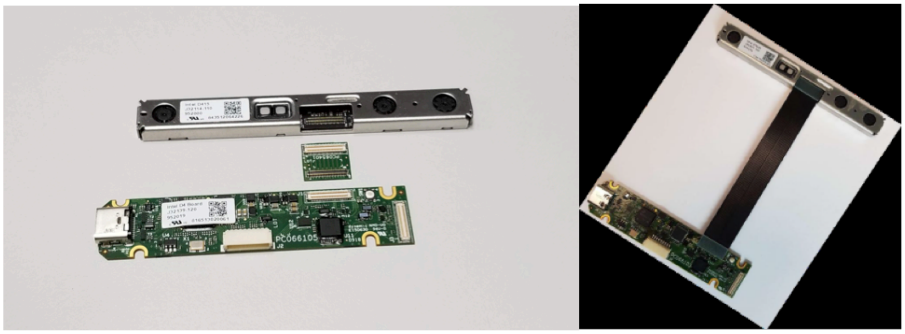


Figure 4. Intel RealSense D415 (Left), Flex Interposer (Right).

The host processor is connected to the vision processor D4 using a USB peripheral type-C cable. The USB port on the host processor should be USB 3.1 so that RGB camera and stereo depth module can be accessed properly. The vision processor D4 communicates to the host processor through USB 2.0/ USB 3.1 Gen 1 and receives the sensor data from the stereo depth module. For a D415 camera, the fps (frames per second) and resolution of depth module and RGB camera is 90 fps, 1280 x 720 pixels and 30 fps, 1920 x 1080 pixels respectively. The range of the camera varies with different lighting conditions. To work with the camera on the host processor, Intel RealSense SDK 2.0 should be installed depending on the type of operating system. Pyrealsense2 library is used for accessing the camera when programming using Python.

Reading Images from Camera

Video is a sequence of frames (images) that are received from a camera at a specific rate. This is called frame rate. The frame rate can be faster or slower depending on camera configuration. Figure 5 shows how frames are received from the camera using SDK.

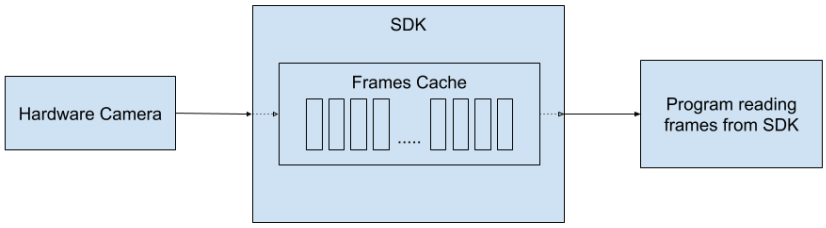


Figure 5. Reading Frames from Intel RealSense Camera.

SDK contains a cache which stores all the frames it receives from the camera. When frames are read from SDK, it provides the earliest frame from the cache. When the rate at which we read frames from SDK is slower than the rate at which frames are stored in this cache, the frame received by the program will not be the latest. If the algorithm receives such older frames, the vehicle's position provided by the algorithm relative to the lane is different from the vehicle's current position; i.e., the program may run lane detection on an older frame where the vehicle was present a couple of milliseconds ago. These seconds of difference can cause accidents as lane detection will not provide the correct position of the vehicle at each instant of time. Hence, when running an algorithm in real time, the program should not wait to read the next frame until the current frame is completely processed. Instead, it should keep reading the frames asynchronously from SDK, and when previous frame processing is completed, the most recently received frame

from SDK is used. Such a program which reads the latest frame from SDK is created. Another program to save all the frames received from the camera is also written, which can be used to save videos for later processing.

Image Formats

Image processing techniques are used to perform various operations in lane detection. This section explains how images are stored and operated on, which is useful to understand how these techniques work. All images have resolution which represents the size of the image. For example, a 1280 x 720 (width x height) resolution image contains a table of 720 rows and 1280 columns, where every cell in the table represents a color or value. These cells are also called pixels. The data in these cells vary depending upon the format of the image. Hence, these images can be stored as a matrix that enables us to run various operations on them. Some of the images that are dealt with are color image, grayscale image and binary image.

Color Image

The camera used to record lane information creates different types of images (RGB, BGR, infrared, depth, etc.). The lane detection algorithms implemented in this paper deal with BGR color images. In a BGR color image, every pixel value contains another 1 x 3 matrix, where each column represents blue, green and red intensity values. Each value varies between 0 and 255. This format

of matrix is used to create output images in our algorithm to easily visualize detected lines. Color images are also useful to analyze performance (Chapter 4, Procedure section) of the detection. Figure 6 is an example of an input raw image.

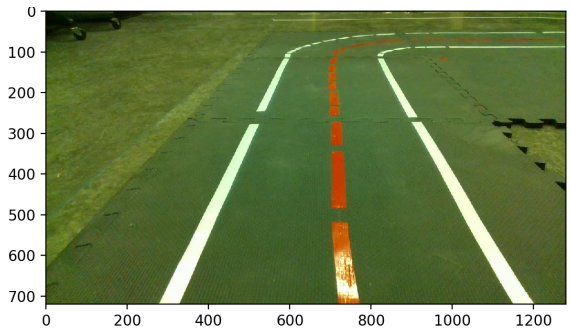


Figure 6. Color Image.

Color Components Extraction

Color-based thresholding is performed to extract the required colors based on the color information of the lanes. This can be used to filter white color components in an image, with which we can extract white lines for the lanes, thereby filtering unnecessary objects before lane detection is performed.

Lane lines with different colors are also used for performance analysis (Chapter 4, Procedure section). Using color filters, specific lines can be identified. In Figure 6, three lane lines are seen, where the left solid line and right solid line are in white color. Middle dashed line is in red color. Figure 6 is subjected to red color extraction and the output is shown in Figure 7. Such types of images are used to analyze the performance of algorithms. Using color filters, identification of expected line pixels can be done, and these pixels can be compared with pixel data obtained from the outputs.

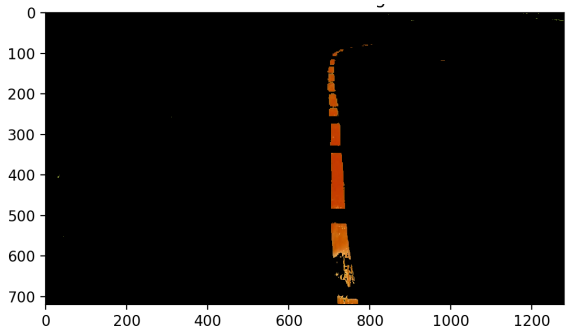


Figure 7. Red Color-Extracted Image.

Grayscale Image

Grayscale image is similar to a binary image, but instead of ones and zeros, the pixel values contain any value between 0 and 255. Conversion of color image to grayscale image reduces computational time on the image as color images consist of three components and grayscale images consist of only two components. This image is usually used to represent intensity-related data such as heat maps, etc. Grayscale images are used to represent clustered images in lane detection algorithms. Grayscale conversion is shown in the Figure 8.

Binary Image

Binary image is the simplest type of image where every pixel has a value of 1 or 0. Figure 9 shows an example of a binary image. It can be seen that areas of the image that have data are

seen as white and other areas are seen as black. Black areas will have pixel values of 0, whereas white areas will have pixel value of 1. As binary images contain only ones and zeros, such an image is helpful in identifying which part of the image has the information needed. Such binary images are used for various checks in the algorithms in this paper. For example, after getting line boundaries, the image can be converted to binary, which is represented by a matrix of ones and zeros. Now for a given section of image (selected by row and column numbers), it can be identified whether lane points exist or not. This technique is used in the sliding window and augmented sliding window approaches (Chapter 3) to identify if lane points exist in specific windows. Binary images are also used to calculate the histogram of the image (Chapter 2, Histogram of an Image section) and in the performance analysis (Chapter 4, Procedure section).

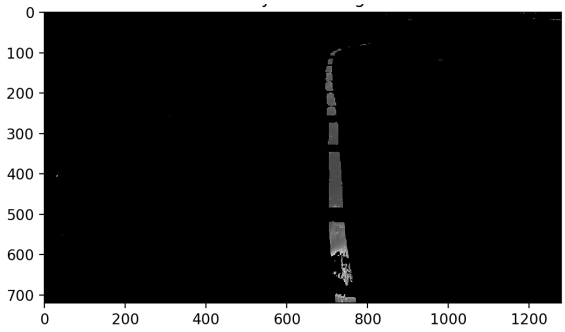


Figure 8. Grayscale Image.

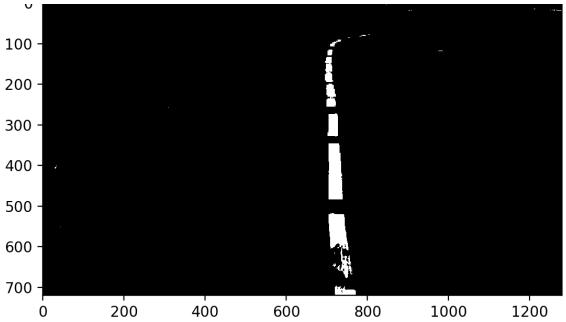


Figure 9. Binary Image.

Canny Edge Detection

Canny edge detection is an image processing technique that generates borders of images using gradient thresholding. Figure 10 shows the respective canny-edge-detected image of Figure 9. As seen from the figures, the canny-edge-detected image contains much less data compared to color-filtered image. Using canny-edge-detected images, the number of pixels the algorithm works with will be vastly reduced, thereby greatly improving the computational time. Canny edge detection has two input thresholds: low-level threshold and high-level threshold. The gradient values of pixels should be greater than the high-level threshold in order to be accepted as an edge. All the pixels whose gradient intensity values are lower than the low-level threshold are not considered as edge points. But if there are any points between these two thresholds and if that point has connectivity to an edge point, then it is also considered as an edge. Optimum selection of these thresholds produces smoother edges.

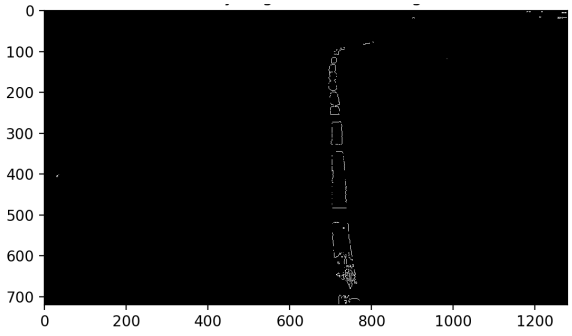


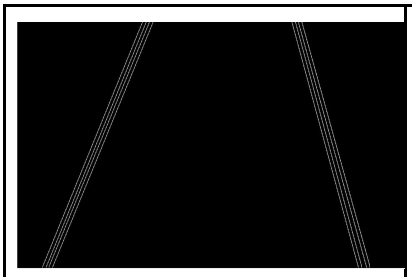
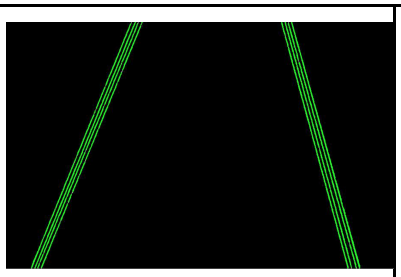
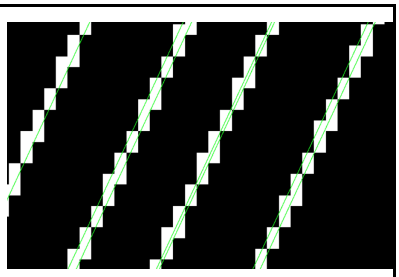
Figure 10. Canny Edge Detection.

Hough Transform

Hough transform is an algorithm which provides a line equation that satisfies a given set of points in a 2D space. This algorithm can be used to detect straight lines or circles in an image. Hence, this algorithm can be used in lane detection for straight lines. Table 4 shows a couple of examples which show lines detected using a Hough transform for a given set of points.

Table 4

Hough Transform Outputs

		
(a) Edge-Detected Image	(b) Hough Lines Detected	(c) Closer Look at Hough Lines

Perspective Transform

Perspective transform is a geometrical transformation technique that transforms the viewing angle perspective of the image. The original image coordinates are changed to new coordinates where, for example, the original image seen from a car can be altered to a bird's-eye view image or top-view image. In [31], this transformation technique is clearly explained. OpenCV library has an inbuilt function to perform this transform. The input arguments should be selected

carefully in order to obtain a proper top-view image. Perspective transform is very useful in case of lane detection. Table 5 and Table 6 show perspective transform applied to images containing straight and curved lanes.

Table 5

Perspective Transform Applied to Straight Lanes

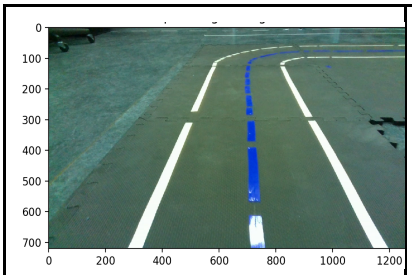
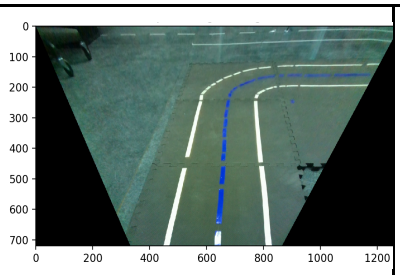
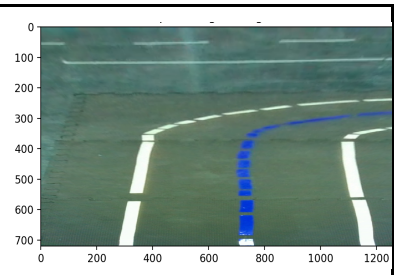
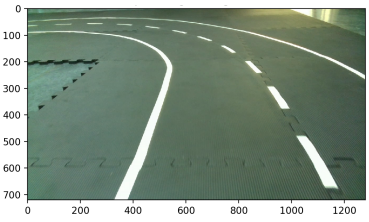
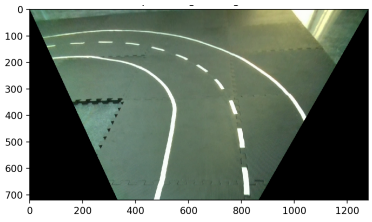
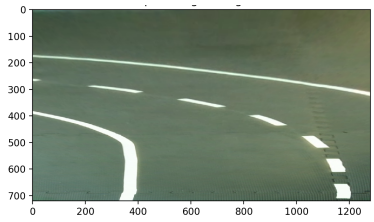
		
(a) Original Image	(b) Perspective Transform That Does Not Crop Image	(c) Perspective Transform That Crops Image

Table 6

Perspective Transform Applied to Curved Lanes

		
(a) Original Image	(b) Perspective Transform That Does Not Crop Image	(c) Perspective Transform That Crops Image

From the above images, bird's-eye view of the input images can be observed. Usually the far ends of the lane lines are closer to each other, which can cause difficulty to categorize them into specific lines. Applying perspective transform on an image increases the distance between the

lanes as shown in the above figures. Hence, this technique is used in lane detection. Perspective transform takes input values that represent the shape of current image and output image. The dimensions to apply perspective transform on the image should be provided such that the image is stretched and not cropped. Incorrect dimensions provided for perspective transform can crop the image, and the resulting image loses lane information from the image. Figure 11 provides an example of how images can be cropped with incorrect parameters of perspective transform. It can be seen that parts of the lane are lost due to the crop.

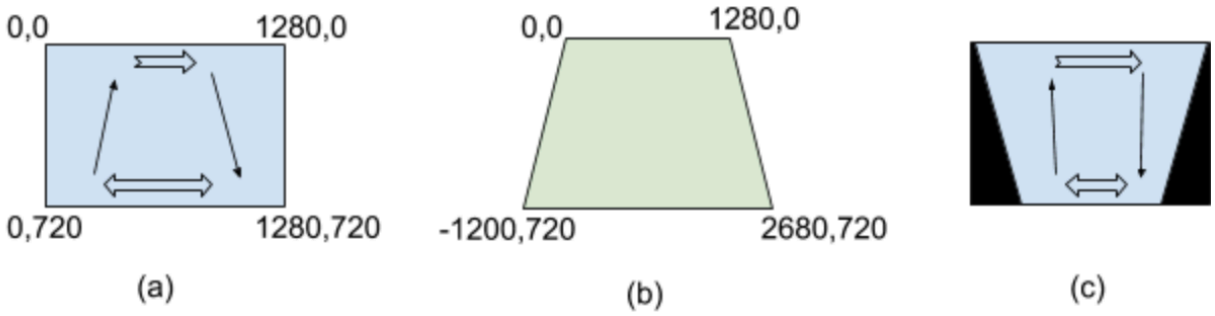


Figure 11. (a) Original Image (b) Perspective Transform Dimensions (c)Transformed Image.

In Figure 11, parts (a) and (b), shows input image of 1280 x 720 resolution and output image dimensions respectively. The result would be an image represented as (c), which shows the top part of the input image is stretched (note that bottom part of the image is not cropped), thereby giving a bird's-eye view of the image. The parameters for perspective transform vary depending upon resolution, position and angle of the camera mounted on the vehicle. Hence, whenever the camera position is changed, the perspective transform parameters should be calibrated such that straight lanes converging on the far end from the vehicle should be visible as straight vertical lines parallel to each other.

Sliding Windows

Sliding windows are used in analyzing lane detection techniques in this paper. Sliding windows are small boxes (can be any shape depending upon the data) of variable dimensions which are used to search part of the image for required information instead of searching the whole image. The term “sliding” is used because the window slides as needed to search points in an image. Figure 12 shows an example of sliding windows where the window at the bottom is the first window that has been placed, and they are moved to the top of the image. Criteria to determine where to place the first window and how to slide it depend on the kind of lane detection algorithm used and are explained in Chapter 3, Lane Detection section.

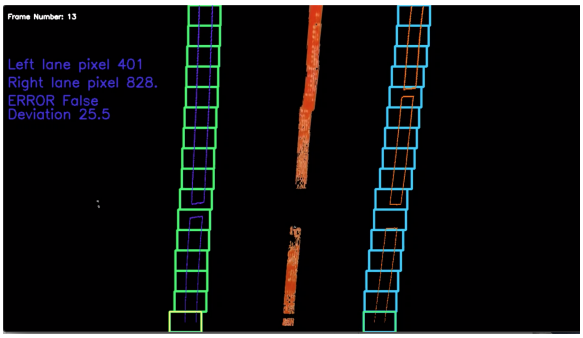


Figure 12. Sliding Windows Formed on Image.

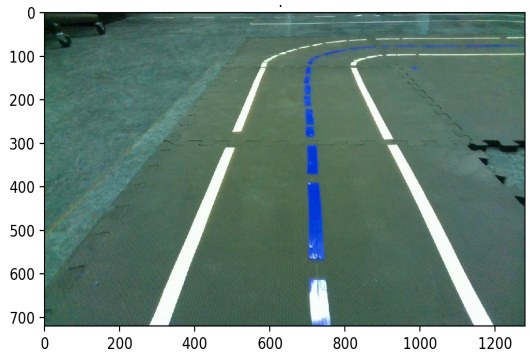
Histogram of an Image

Histogram calculation is used to retrieve meaningful data for our lane detection techniques. On the input image, white color filter and binary image conversion are applied and then the image is converted to a warped image. This binary BEV (bird's-eye view) image now contains vertical lane lines as perspective transform was applied on it. Histogram is calculated by combining ones per each column (vertically), and hence, the peak values in histogram indicate where the most lane points were present (since lanes are straight vertical lines). By setting appropriate threshold values, which vertical sections of the image have lane lines and how many lanes are present can be identified. Note that any white objects that are present in the image apart from lanes will affect the reliability of this approach. The initial sliding window boxes are created using this approach in sliding-window-based lane detection approaches.

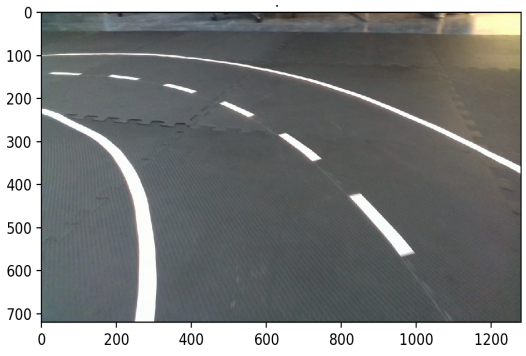
It can be seen from Table 7 that lane lines are efficiently detected in an image. But this approach does not work when the vehicle moves along sharply curved lanes and turns. In these scenarios, lane lines will not be vertical even after applying perspective transform. The resulting histogram does not provide reliable information to tell where the lane line points exist in the image. Hence, tracking the movement of lanes is needed to reliably know where the initial sliding window should be placed.

Table 7

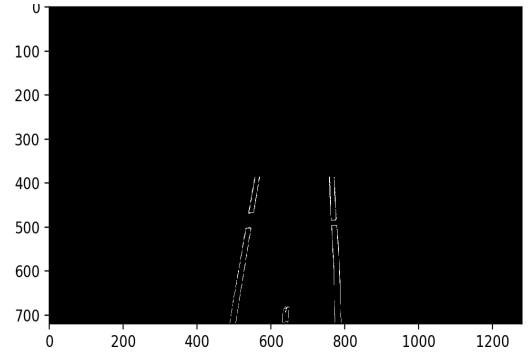
Histogram Representation



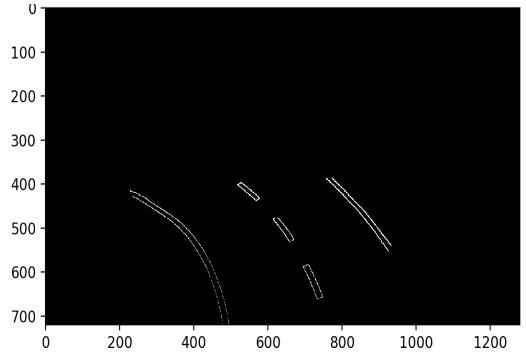
(a) Input Image with Straight Lines



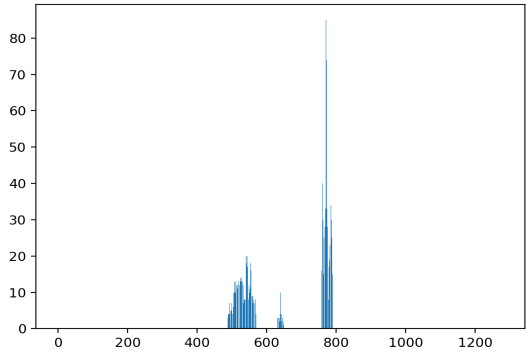
(b) Input Image with Curved Lines



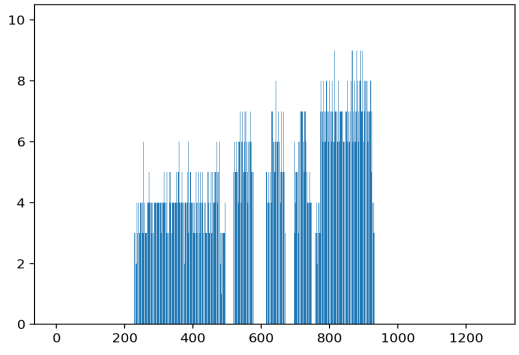
(c) Perspective Transform on Binary Image



(a) Perspective Transform on Binary Image



(b) Histogram on Transformed Image



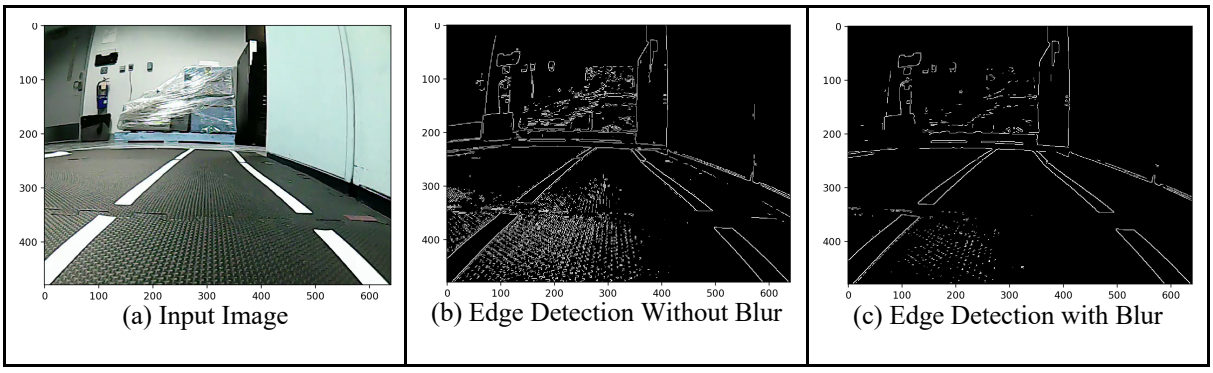
(c) Histogram of Transformed Image

Image Blurring

Image blurring is a technique that greatly reduces the small disturbances in an image. Table 8 shows how unwanted data can be omitted from input images. Such unwanted data can be usually seen in a white-component-extracted image, when an input image contains light shade, grass, soil or other debris, etc. Using blur technique, the image pixels blend together with surrounding pixels, where any white pixels which are referred to earlier are blended to respective environments. Hence, this helps to reduce unwanted data in the input image. In this paper, Gaussian blur technique is used.

Table 8

Significance of Blur



Clustering Technique

Clustering is an image processing technique that forms clusters in an image. Clusters are blobs on the image which can be identified. Figure 13 shows an input image and clusters that are detected in the image.

DBSCAN (density-based spatial clustering of applications with noise) is one of the clustering techniques that can be used to form clusters in an image. This technique is clearly explained in [32]. DBSCAN takes Epsilon and MinPts as two input parameters. DBSCAN forms clusters in an image using the following criteria:

1. Any pixel in an image that has at least MinPts number of points around it within a distance of Epsilon are considered as core points.
2. Any pixel in an image that has core points within distance of Epsilon are considered as reachable points from core points.
3. Any pixel in an image that has reachable points within distance of Epsilon (and not necessarily core point) are also considered as reachable points.
4. All remaining points which are not reachable are considered as noise points.
5. A core point forms clusters with all reachable points around it (where reachable points can also be core points).

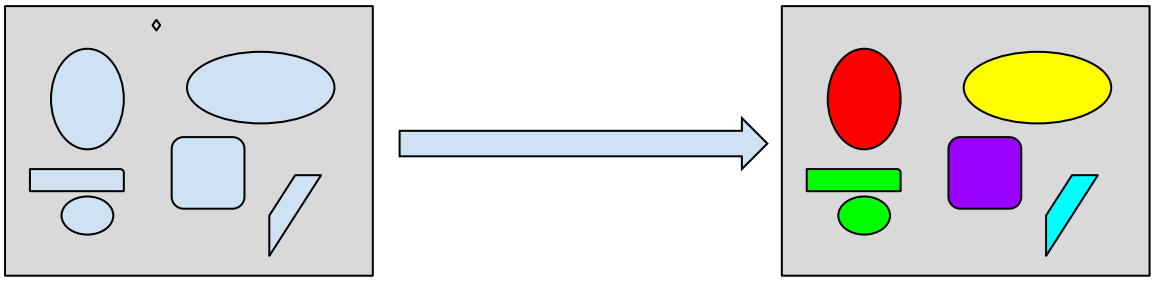


Figure 13. Representation of DBSCAN Clustering.

For example, Figure 13 is considered for explanation on which clustering is being applied. Each color in the output image represents a cluster. This is done for visualization purposes. Actual

clustered image will be a grayscale image where each pixel has a value of 0-255, where pixel value represents cluster label. Based on the MinPts and Epsilon parameters, values are provided in the above figure:

- No points that form rhombus are categorized as core points. Hence, no points in the rhombus are visible in a clustered image.
- All the other shapes have points that are either core or reachable points. Hence, they are visible in clustered images.
- Distance between some points in the circle and rectangle is less than Epsilon and all core points in the circle and rectangle are reachable. Hence, points in these shapes are categorized as the same cluster, and all other shapes are categorized as separate clusters.

DBSCAN clustering provides the total number of clusters formed, labels to identify the clusters, and information about which cluster each pixel belongs to. These details are used in lane detection to distinguish various lanes. This is explained further in Chapter 3, Lane Detection Using Clustering section.

Down Sampling and Up Sampling Image

Images can be downsized using a pyramid approach. When downsizing an image, all pixels of the image are grouped by 4 pixels (2 row pixels and 2 column pixels), and each of these 4 pixels in the group are combined together to form a single pixel. The resulting image will be smaller by a factor of 2. If an image of resolution 1280 x 720 is down sampled once, the resulting image would be of resolution 640 x 360, which when down sampled again results in an image of 320 x

180 resolution. Figure 14 represents how the image is down sampled and the bottom level of the pyramid represents the actual image. And every level above from the bottom in the pyramid represents a down-sampled image.

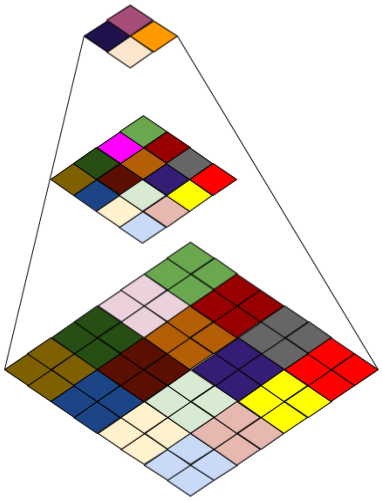
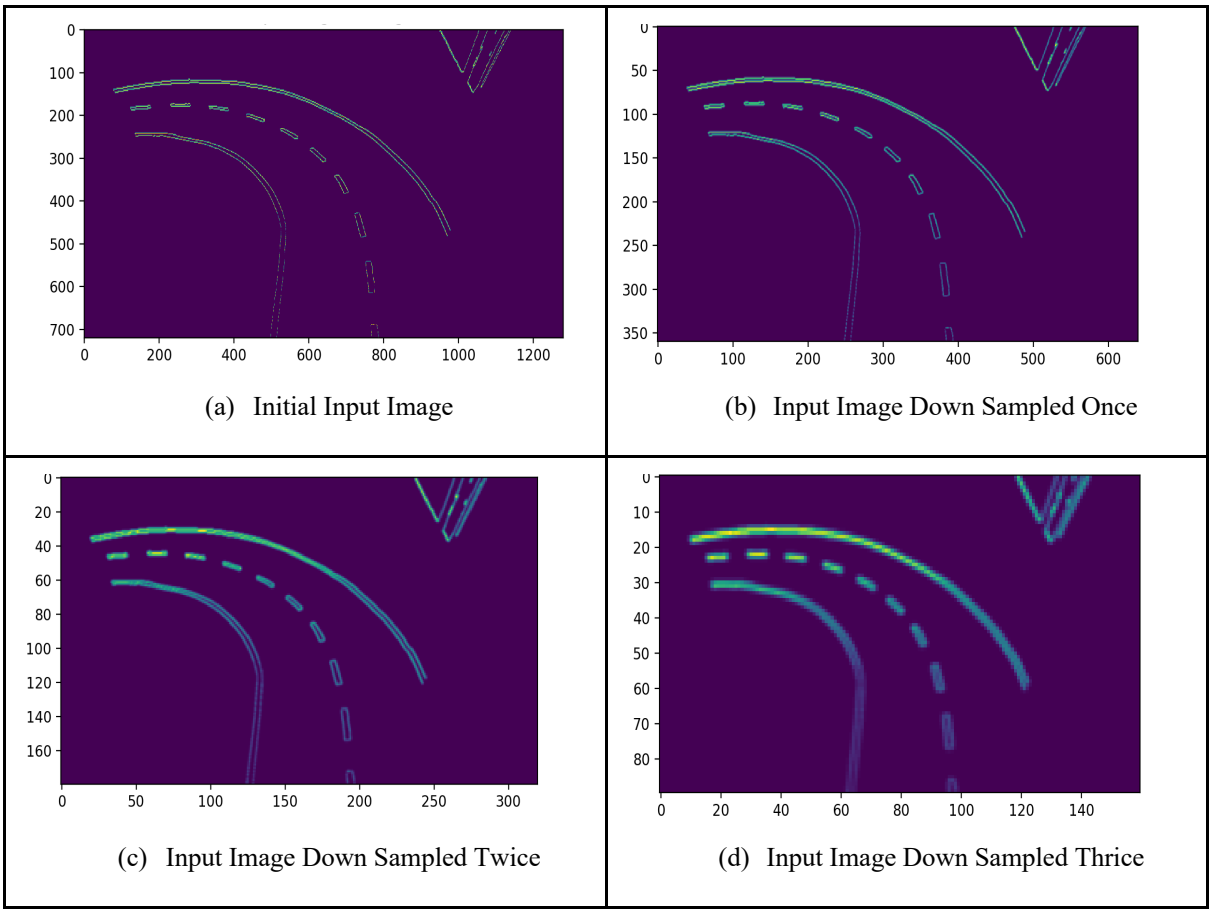


Figure 14. Image Pyramids.

OpenCV library in Python provides an implementation of this downscaling which is called PyrDown. When combining values of each set of 4 pixels, it uses Gaussian blur technique to down sample. Table 9 shows multiple images every time it is downscaled.

Table 9

Image at Different Down Sampling Levels



(a) Initial Input Image

(b) Input Image Down Sampled Once

(c) Input Image Down Sampled Twice

(d) Input Image Down Sampled Thrice

In the lane detection algorithm used in this paper, clustered image is upscaled to original input image size without losing cluster data (Chapter 3). This is basically moving from pyramid top down. Every pixel in a down-sampled image is then divided into 4 more pixels. OpenCV library provides a PyrUp method that does up sampling of the image, but the Laplacian pyramid used for up sampling produces an image that blends together adjacent pixels. While this may be ideal for color image up sampling, it will not work to scale up a clustered image, as pixel values have to be retained for the lane detection algorithm to work. Equations 1 and 2 show the comparison between scaled-up clustered images using PyrUp and Kronecker product.

A custom up-sampling algorithm is implemented that uses Kronecker product to retain the pixel data while up sampling the image. Kronecker product is used to multiply matrices as shown in Equation 1.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1\times\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} & 2\times\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \\ 3\times\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} & 4\times\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{pmatrix} \quad \text{(Eq. 1)}$$

As the clustered image is a grayscale image, which is basically a matrix of numbers, we can apply the Kronecker product on it. Equation 2 shows an exemplar application of Kronecker product between an array (which can be considered as our clustered image) and a unit matrix without changing the input values. It can be observed from the above example that the nonunit matrix size is increased by a factor of 2 while replicating the values of the matrix instead of modifying them. Hence, we use this approach to up sample clustered images to input image size.

$$\begin{pmatrix} 6 & 9 \\ 2 & 5 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 6\times\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} & 9\times\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \\ 2\times\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} & 5\times\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 6 & 6 & 9 & 9 \\ 6 & 6 & 9 & 9 \\ 2 & 2 & 5 & 5 \\ 2 & 2 & 5 & 5 \end{pmatrix} \quad \text{(Eq. 2)}$$

CHAPTER 3

LANE DETECTION

Lane detection is one of the crucial functions in an autonomous vehicle. Lane detection involves capturing surrounding information from cameras and sensors, then analyzing and determining the vehicle's position with respect to the current lanes on the road. In this paper, cameras are used to collect surrounding lane information. Data retrieved from cameras are images. To extract information from images, various image processing techniques are used. Chapter 2 explained how cameras are used to retrieve images, various lanes that are simulated in the laboratory and the techniques used for lane detection in this paper. Lane detection has two main categories, extracting useful information from images and detecting lanes from the extracted information. Extracting useful information involves modifying the image so that unnecessary data can be filtered out while retaining the lane information. It also modifies the image to reduce its size, which improves computational time. Lane detection involves using algorithms like sliding window, clustering, histogram and augmented sliding window on extracted feature points; identifying lines that form the lane; and calculating the vehicle’s position relative to lane. This chapter explains in detail how information extraction and lane detection are performed for these algorithms. Figure 15 shows basic steps of all lane detection algorithms that are implemented in this chapter.

The yellow sections show steps which are optionally performed for a few algorithms. Image acquisition is done using a camera or prerecorded video sequence. Once frames are available, image preprocessing steps are performed to extract useful information, and all points that correspond to respective lanes' lines are identified. Vehicle's current position with respect to lane lines is calculated.

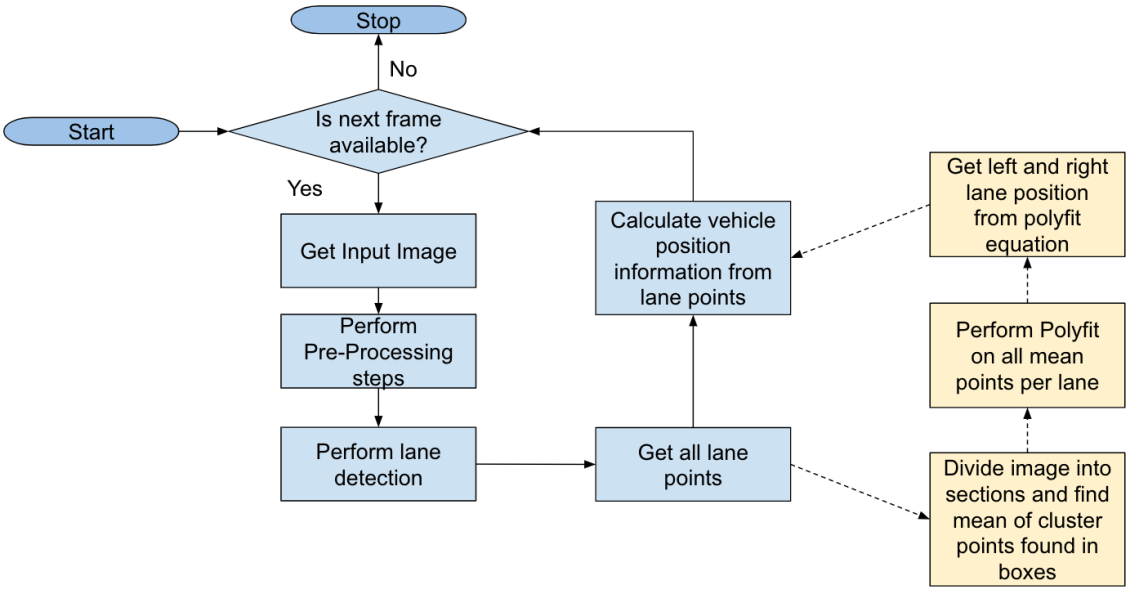


Figure 15. High-Level Flowchart of All Algorithms Implemented in This Chapter.

Lane detection using Hough transform applies polynomial fitting of degree 4 on all detected points per lane, and curve equation is retrieved. This equation represents the lane line. Averaging this equation with respective line equations in the past few frames provides a confidence value for the detected lane lines. This averaging also helps us to ignore any error-prone frames that contain incorrect lane information. For example, in a couple of frames the lane lines are not showing up correctly, which causes wrong lane lines to be detected.

Hence, a few past frames equations can help the algorithm to keep going for the next couple of frames. It is very difficult to approximate sharply curved lines and turning lines with a single equation. Hence, polyfit is not calculated for such lanes. In such scenarios, total points detected in the current lane are used for providing the detected line's confidence.

Once lane line points are detected, the position of lane lines is retrieved. The closest left and right lane lines from vehicles are determined and deviation of vehicle from lane center is calculated. Figure 16 shows an example of various lane lines detected on a 1280x720 image; 10, 350, 850 and 1200 are respective lane line positions on x-axis. Note that the origin in image space is located in the top left. When a polynomial fitting equation is available, lane positions can be retrieved by substituting 719 for y. When working with sharp curves, polynomial fitting cannot be used. In this case, the bottommost points of each line are found, and their mean x-axis value is used as a lane position. Hence, lane center is calculated as 500. As the camera mounted on the vehicle will be at a constant place and is in the middle of the vehicle, the vehicle position is always taken as 640. The deviation of vehicle from lane center is found.

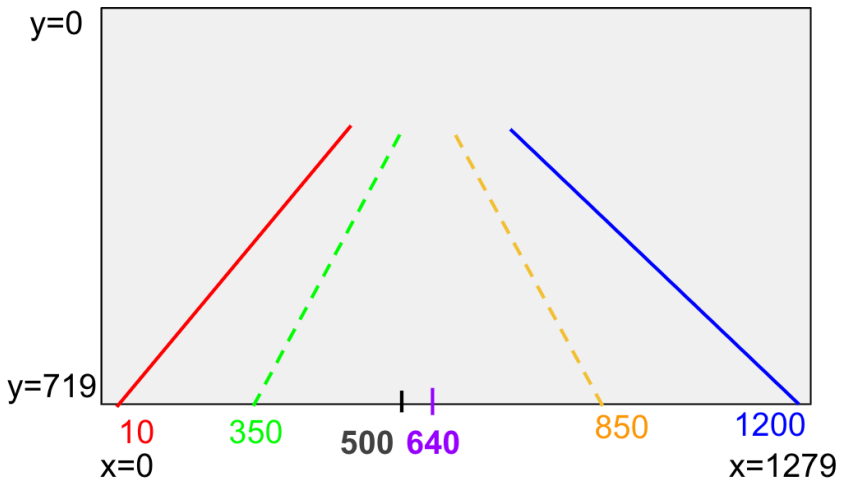


Figure 16. Representation of Lane Lines Detected.

For all the lane detection algorithms in this chapter, common image preprocessing steps are used. Figure 17 provides a series of steps that illustrate these. Input image is a BGR (blue, green, red) image. This BGR image is subjected to white color thresholding (as the simulated lanes are white in color). Once, the white components are extracted, the image is converted to grayscale. To reduce low-intensity pixel points, Gaussian blur is performed. Then canny edge detection is done to find the edge pixel values of lane boundaries that reduces the total number of points in the search space, thereby increasing the processing speed of the algorithm. ROI is selected to remove non-lane points from the image. For curved lanes and for instances where the far view of lanes looks like they are intersecting, perspective transform is applied on such a data set. Finally, the transformed image is further processed for lane detection.

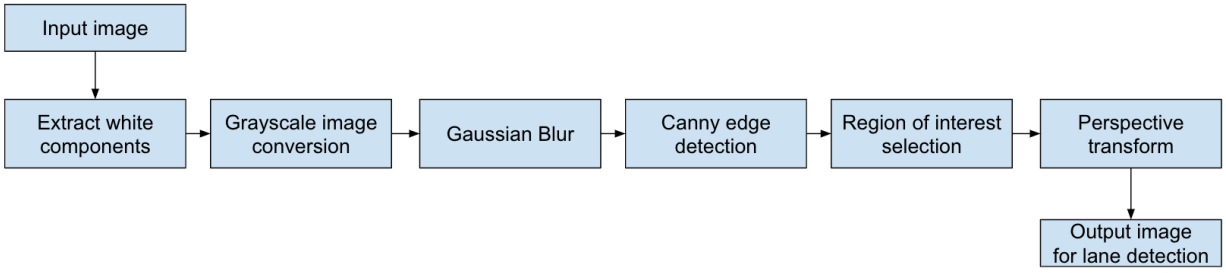


Figure 17. Image Preprocessing Steps.

Sliding-window-based lane detection algorithms need to know lane starting position to start detecting lanes. When straight lines or slightly curved lines are present, a histogram calculated in the bottom half of the image can provide lane positions. If sharply curved lanes exist, peaks calculated using histogram will not be efficient to detect lane positions as explained in Chapter 2. In such cases, tracking boxes can be helpful to track the lane positions. Histogram calculation per

frame is a time-consuming operation compared to tracking boxes. Hence, histogram can be used on the first few frames when straight lanes are found (generally, when a vehicle starts from rest, it is expected to capture straight lines), and tracking boxes can be used in subsequent frames. Figure 18 shows steps of determining tracking windows initially. Tracking boxes are initially placed based on histogram of image. Every time lanes are detected, a cache is used to store lane positions. The mean of the past few positions from cache is used to determine the next set of tracking windows. This approach helps move tracking boxes as lane position changes steadily. If in any set of frames lines are not detected, histogram can be used again to reset lane starting positions.

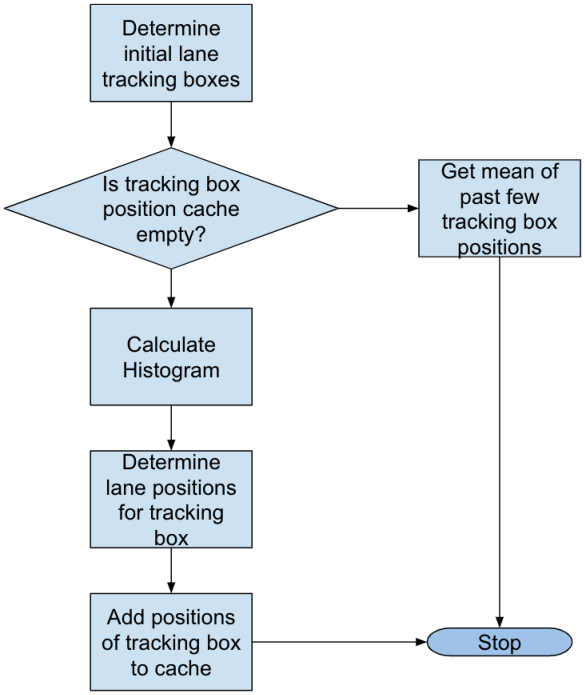


Figure 18. Selection of Tracking Windows.

Lane Detection of Straight Lines Using Hough Transform

At start upon preprocessing the input image, Hough transform is applied on the canny-edge-detected image (Figure 19). Slopes of Hough lines are determined, and they are categorized into left and right lines. The slope of the vertical line is undefined, and the slope of the horizontal line is zero.

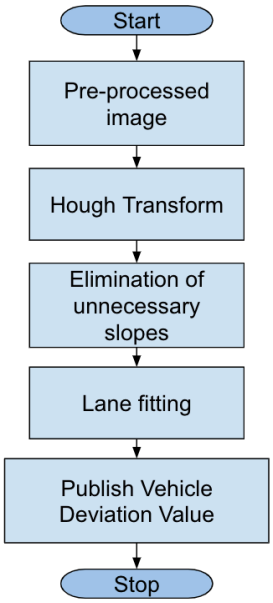


Figure 19. Lane Detection Using Hough Transform.

Similarly, the lines that are moving from left to right that are upwards have a positive slope and the lines that are moving from top to bottom that are downwards will have negative slope. This is the understanding for a normal coordinate system. Here as origin lies at the top left corner of the image, these conditions are reversed. Next, the corresponding points are fitted to a second-order polynomial equation using least mean square error method. As we have the line equation,

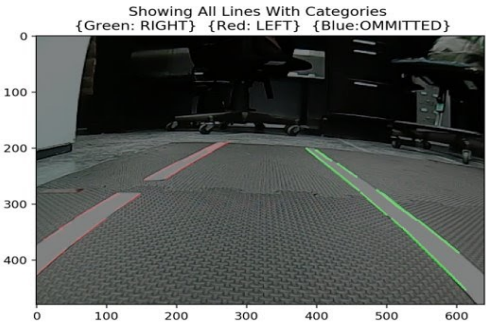
the corresponding points of the left line and right line of a lane are found. Using these, the center of the lane is calculated. The distance from the vehicle to the center of the lane is determined. This value enables lane keep assist. Using this value, the controller can control the vehicle to stay in the respective lane.

Results and Discussion

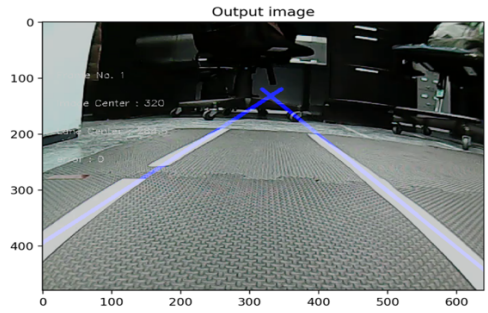
Table 10 shows how Hough transform was used to detect lanes. Hough transform works very efficiently on straight lanes. However, if the lanes start to curve, the line equation produced by Hough transform will not align with all the actual lane points. Consider Table 10 (c), which shows an image where the red and green lines represent multiple left and right Hough lines detected on top of lanes. When a single line equation is fitted on all lane points, the lines are generated as shown in (d). As it can be seen, these line equations do not provide information about the curved lane. This can be crucial information for vehicles, as at high speeds vehicles can skid on curved lanes. Also, these line equations can provide a wrong heading direction to the vehicle. In the next sections, algorithms which try to overcome this are analyzed.

Table 10

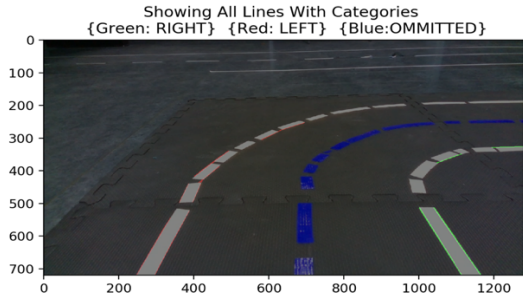
Outputs of Lane Detection Using Hough Transform



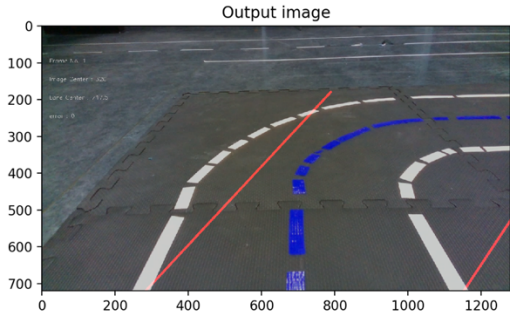
(a) Detection of Left (Red) And Right Lines (Green)



(b) Output Image



(c) Detection of Left and Right Lines of Curved Lane



(d) Output Image

Lane Detection of Slightly Curved Lines Using Sliding Window

After the image preprocessing steps, the output is a bird's-eye view image (BEV). The dimensions of the sliding window and the number of windows is defined. Histogram of the BEV image is obtained. Based on the peak values of histogram, left and right starting sliding windows are selected. If there are new pixel points in the window, those points are saved. After searching a window, the next window's horizontal position is determined by finding the mean of the points

found in the current window. If there are no points in the current sliding window, the current sliding window's horizontal position is taken as the next window's horizontal position. Next sliding window is created on top of the current sliding window. New window is searched for new points. This process continues until it reaches the predefined number of windows. The results are explained in the next section. Figure 20 explains the flowchart of the lane detection algorithm using a single sliding window.

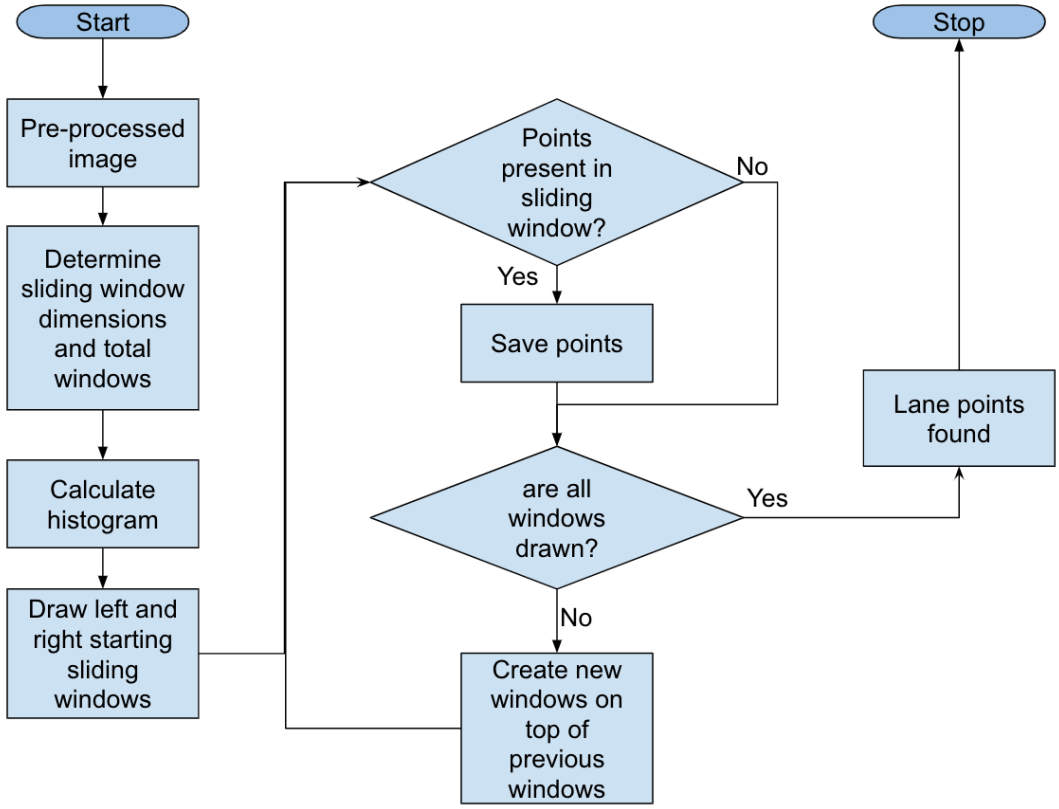


Figure 20. Process Flow of Lane Detection Using Sliding Window.

Results and Discussion

Figure 21 shows the results of the lane detection algorithm using a sliding window. As shown in the figure for an input image, white components are extracted, converted to grayscale, canny edge detection is performed, and the image is cropped to a particular region of interest. ROI image is converted to a BEV image; using the histogram, the starting points of sliding windows are determined and nine sliding windows are moved from bottom to top part of the image. Next, polyfit is implemented to find the lane points and the output is overlaid onto the input image for good visualization.

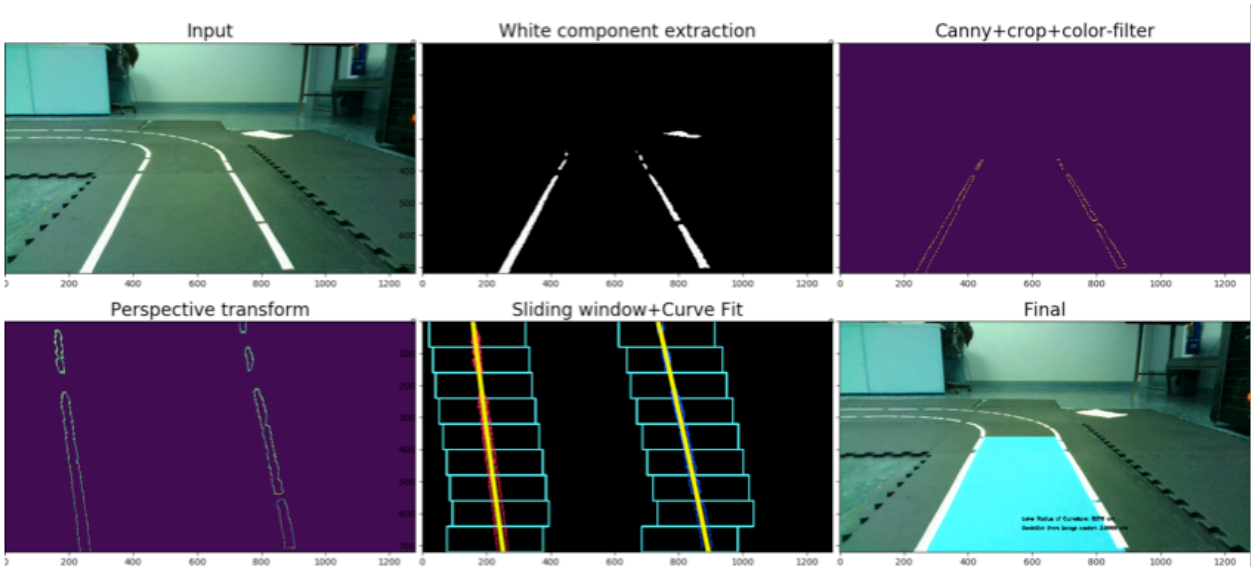


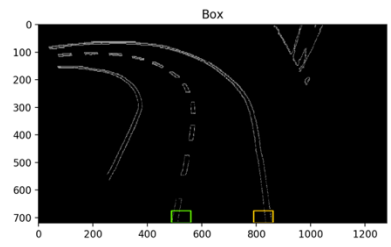
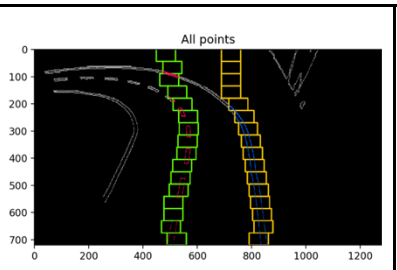
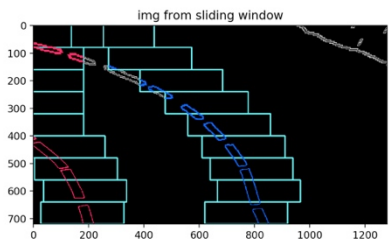
Figure 21. Lane Detection of an Input Frame Using Sliding Window Approach.

When sliding windows are continuously built on top of the starting windows to find the pixel points, such continuity, irrespective of the absence of points in a window, might lead to clashing of the sliding windows into other lane points. Such a scenario is shown in Table 11. Table 11 shows three images; the leftmost image is the BEV image where starting boxes of sliding

windows are selected. Once a window is searched it keeps on searching other windows as shown in the middle image and the rightmost image. It can be seen that the left and right lines in the bottom part of the image are farther away from each other than the lines in the top part. Hence, an optimum size of the sliding window should be selected. Also, the movement of the sliding window should be altered so that it doesn't clash with the other lanes. This led to the idea of augmented sliding window technique. This technique is explained in a later section after clustering. The first image in Table 11 also has a few outliers (non-lane points at top right of the image), and there is a good chance that such kind of non-lane points might be considered as the points inside the sliding windows in the coming future frames. This led to the idea of using a clustering technique so that non-lane points can be removed, or the individual lane points can be stored and cross-checked. This technique is presented in the next section.

Table 11

Examples of Sliding Window Failure

 (a) Starting Windows	 (b) All Windows	 (c) Curved Lane Scenario

The flowchart of the working of lane detection using clustering technique is shown in Figure 22. Initially, the input image is subjected to preprocessing techniques. The preprocessed image is a BEV image. DBSCAN clustering is used to segment the lanes in the image. The two input parameters Minpts and Eps are carefully selected so that the left lane is one cluster and the right lane is another cluster. The initial tracking windows are selected using histogram of the BEV image as explained in Figure 18. Now that the initial windows are obtained, they are checked for any points inside the initial windows. If points are present, then the corresponding cluster label of the points is found. All the points in the image with that particular label are detected as left lanes, if the points are in the left starting window and vice versa. If there are no points inside the initial window, then that frame is skipped, and the next frame is considered as the input image.

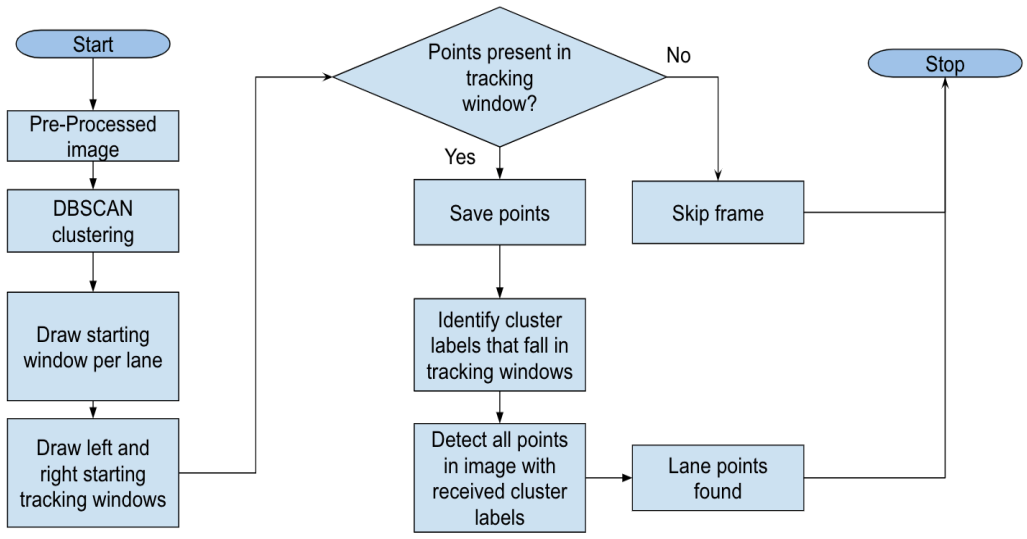


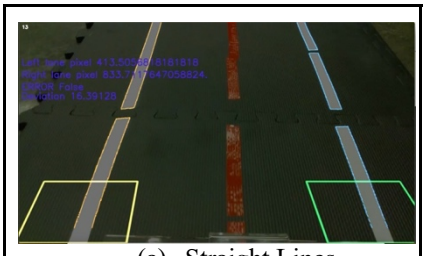
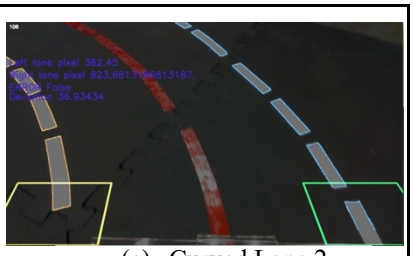
Figure 22. Process Flow of Lane Detection Using Clustering.

Results and Discussion

Table 12 shows the outputs of lane detection using clustering. Three different scenarios of lane outputs are presented. The image 12(a) is a straight-line input, where the left and right clusters are detected. The output image displays the information of the left lane and right lane pixel positions. It also shows the calculated deviation of the vehicle with respect to the detected center of the lane. The second image is a scenario of sharp curves and the third image shows slightly curved lanes. In all the following images, clustering technique detects the left and right lanes successfully. For images where the dashed lines' gap is more, in such a case, dashed lines are considered as different multiple clusters. Clustering technique cannot detect the dashed lines as a single line, especially when dashed lane gaps are bigger or equal to lane width. This flaw led to the idea of combining sliding windows and clustering technique for detecting dashed lines. This is explained in Figure 25.

Table 12

Outputs of Lane Detection Using Clustering

 (a) Straight Lines	 (b) Curved Lane 1	 (c) Curved Lane 2

Augmented sliding window approach is built on top of sliding window approach. This is a newly proposed method in this paper to efficiently detect sharply curved lanes and lanes which are turning. This approach uses a set of five sliding windows instead of a single sliding window. The steps for augmented sliding window technique are shown in Figure 23.

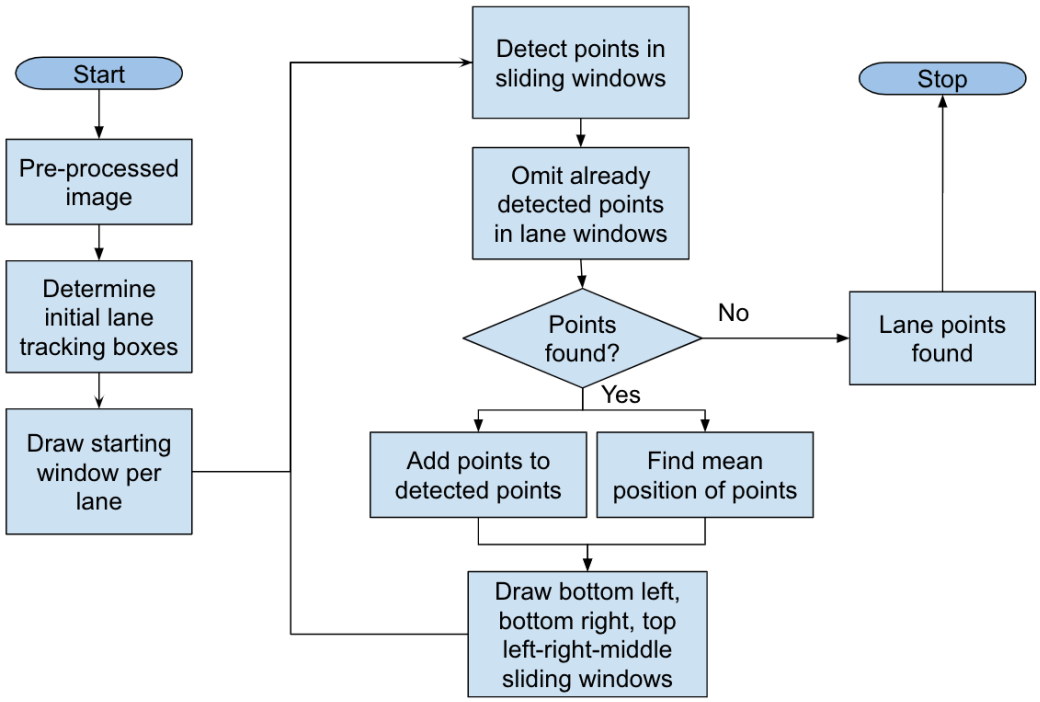


Figure 23. Process Flow of Lane Detection Using Augmented Sliding Window.

After the input image is preprocessed, tracking boxes are determined as shown in Figure 18. Initial sets of sliding windows are placed based on tracking boxes. Lane points that fall under the current set of sliding windows are found. Any points that are already found in the previous set of sliding windows are omitted and new points are stored. Unlike the sliding window approach,

the next set of windows can be adjusted horizontally and vertically. The mean of new points' position (horizontal and vertical) is used to place the next set of sliding windows. This enables the windows to track more efficiently. This process continues on the current frame until no new points are found. All lane points are then returned for calculating vehicle deviation from lane center.

The shape of these sliding windows is as shown in Figure 24. Section (a) represents the shape of windows that is used. Initially, bigger windows are used, and after detecting 75% of the lane, windows are adjusted to the size as seen in (b) or (c). These shapes are observed to efficiently track the line points even when they have sharp turns and curves. When the window's vertical length is bigger than the gap between dashed line segments, these windows can efficiently track dashed line points. During sharp turns, it's observed that lanes move horizontally and parallel to each other. In these scenarios, due to these lanes being farther away from the vehicle, the distance between lanes is visually observed as very small. They are smaller than the dashed lane line segment gaps. Sections (b) and (c) present the shape of these windows when lanes are being tracked in such positions. If the lane is turning left, then the top right box's width is minimized (c), and if the lane is turning right, the top left box's width is minimized (b).

As this approach stops sliding windows when no new points are observed, this causes the sliding window approach to stop tracking all points when lanes have large gaps, but this also prevents tracking non-lane points that do not belong to a lane.

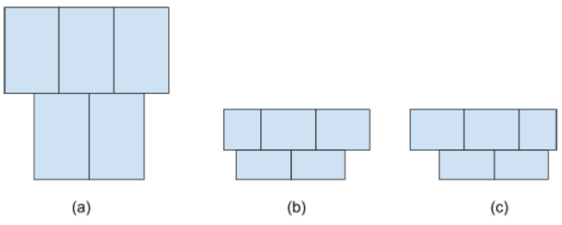


Figure 24. Augmented Sliding Windows.

Results and Discussion

Table 13 shows the movement of augmented sliding windows to track sharply curved lanes. Starting windows are placed at the bottom of each lane line, and sliding windows move along the lane while detecting new points with each movement.

Table 13

Movement of Sliding Windows

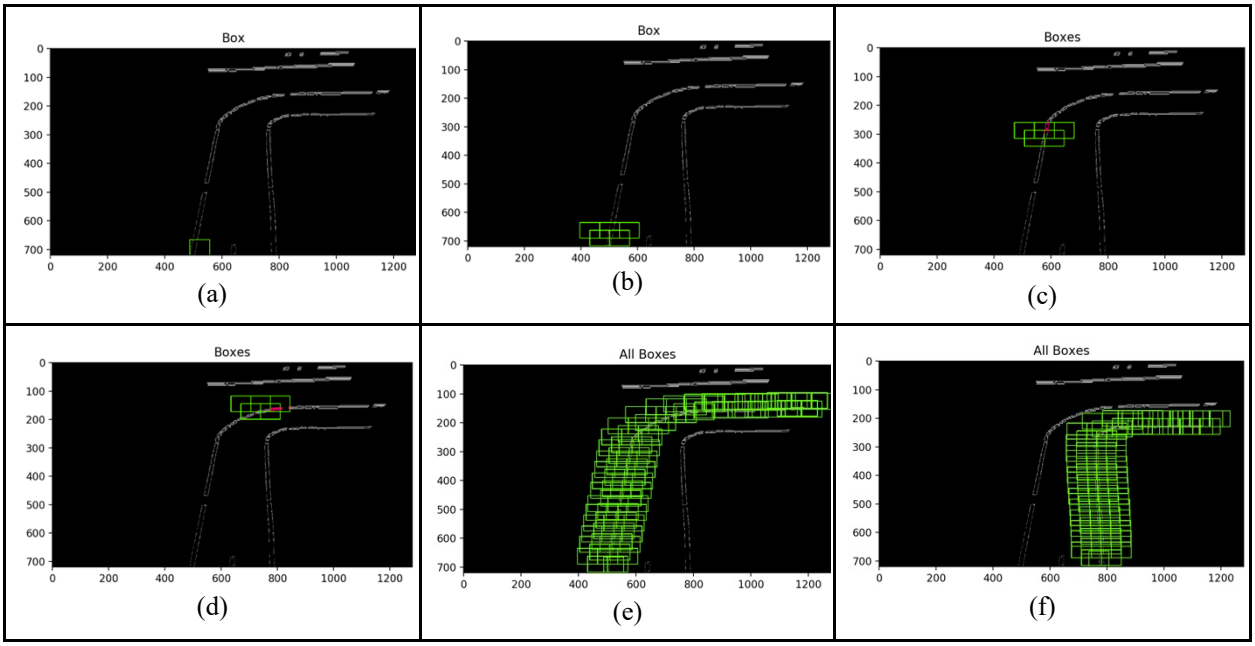
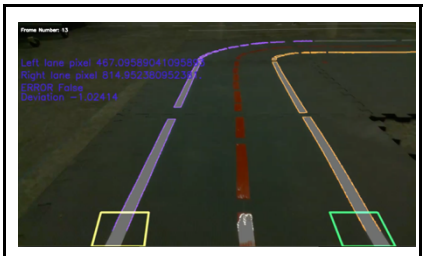
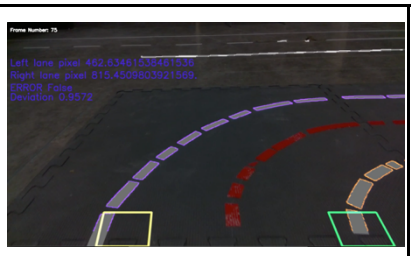
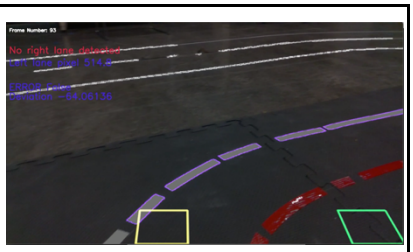


Table 14 shows visualized outputs of lane detection using augmented sliding window algorithm. It can be seen that straight lines with turns and sharply curved and partially visible lanes are all detected. Yellow and green boxes are the tracking boxes that track lane movement. The vehicle's deviation is efficiently calculated in all scenarios.

Table 14

Outputs of Lane Detection Using Augmented Sliding Window

		
(a)	(b)	(c)

Lane Detection of Sharply Curved Lines and Turns Using Augmented Sliding Window and Clustering

The working of the lane detection using augmented sliding window and clustering techniques is shown in Figure 25. After the image preprocessing steps, DBSCAN clustering is applied onto the BEV image. Using the initial lane tracking windows, a cluster label for points inside the respective tracking windows is found. Total points in the image with each cluster label are retrieved. All points in the solid line fall under the same cluster due to lack of gaps. Hence, solid lane points can be distinguished from dashed lane points based on the number of points in the cluster. Tracking boxes that contain solid lane points are found and all respective points are categorized as solid lanes for each of these boxes. The rest of the tracking boxes correspond to dashed lanes. Augmented sliding window is applied per tracking box and dashed lane points per tracking box are retrieved. The lane points are used to calculate vehicle deviation.

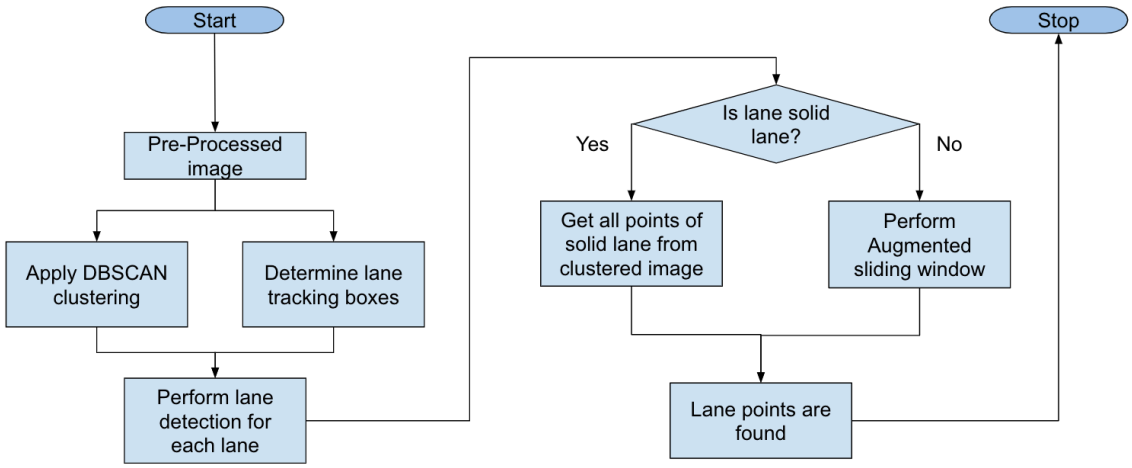


Figure 25. Process Flow of Lane Detection Using Augmented Sliding Window and Clustering.

Results and Discussion

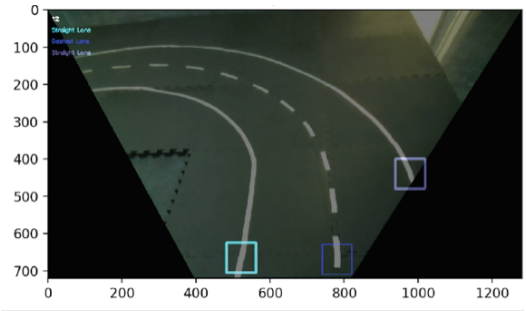
Table 15 shows the results of the lane detection using augmented sliding window and clustering techniques. The first image is the bird's-eye view image with information of initial tracking windows (a). Next, the image is scaled down and then subjected to DBSCAN clustering (b). The second image is a PyrDown clustered image. This image provides the information of cluster labels and the number of points and their pixel positions of a particular cluster. It is assumed to have prior information of whether the lane is turning left or right (either from maps or past frames). Based on that, if the vehicle is taking a left turn at a particular time, then the algorithm starts detecting rightmost lane points first and vice versa. This in combination with augmented sliding window dimensions avoids detecting lane points from one lane while detecting another lane. Generally, the rightmost and leftmost lines are solid lines. The output image with all detected solid line points is shown in (c). Now, the detected rightmost solid line points are saved and

removed before further processing; (d) shows the remaining points to be detected. Next, the rightmost line is detected and found to be a dashed line (based on the number of points pertaining to the cluster label found for tracking windows). Hence, an augmented sliding window is applied, and the new valid points are searched in the windows. This is shown in (e). The output is a dashed line and is added to output (f). This dashed lane is removed before further processing, and the only remaining clustered image is then detected. All detected lane points are now visualized in (h).

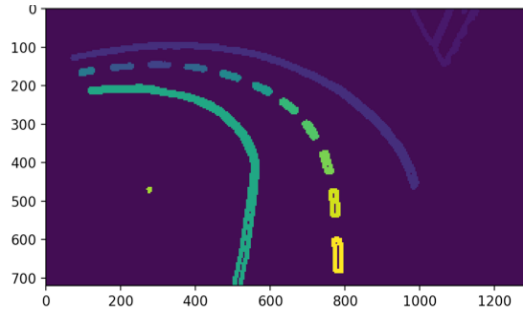
After the output image, the image is scaled up using a custom PyrUp function, which is mentioned in the last section of Chapter 2. Then the position of the vehicle with respect to the center of the lane is calculated. The final output of the lane detection using augmented sliding window and clustering technique for an input frame is shown in Figure 26.

Table 15

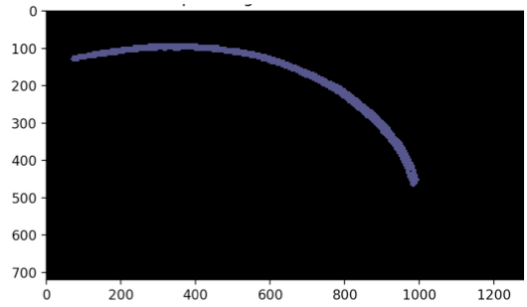
Intermediate Results of Lane Detection Using Augmented Sliding Window and Clustering



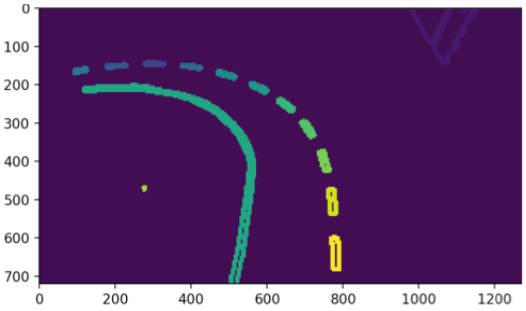
(a) Tracking Windows



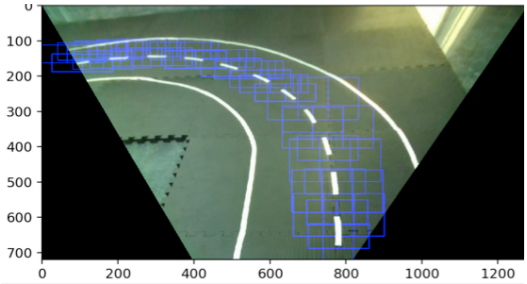
(b) PyrDown Clustered Image



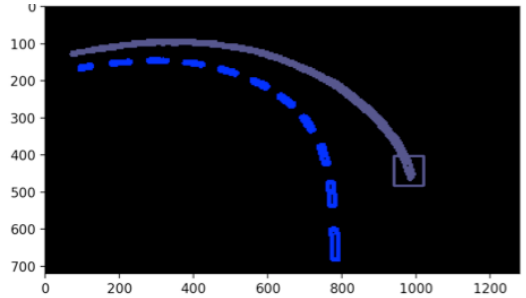
(c) Output Image with New Solid Line



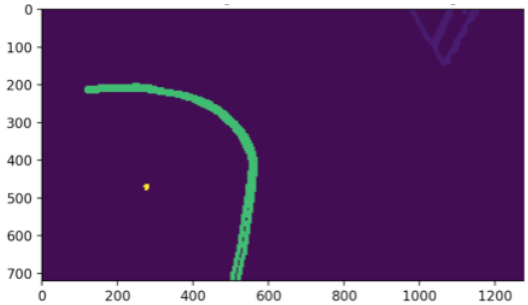
(d) Clustered Image After Filtering Solid Line



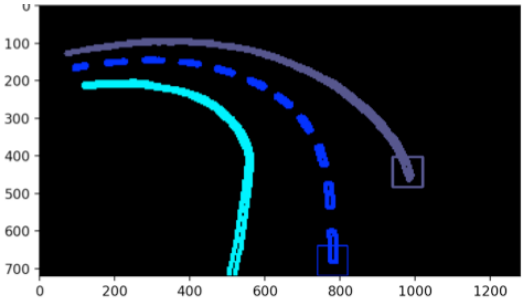
(e) Augmented Sliding Window Boxes



(f) Output Image with New Dashed Line



(g) Clustered Image After Dashed Line Filtering



(h) Output Image with New Solid Line

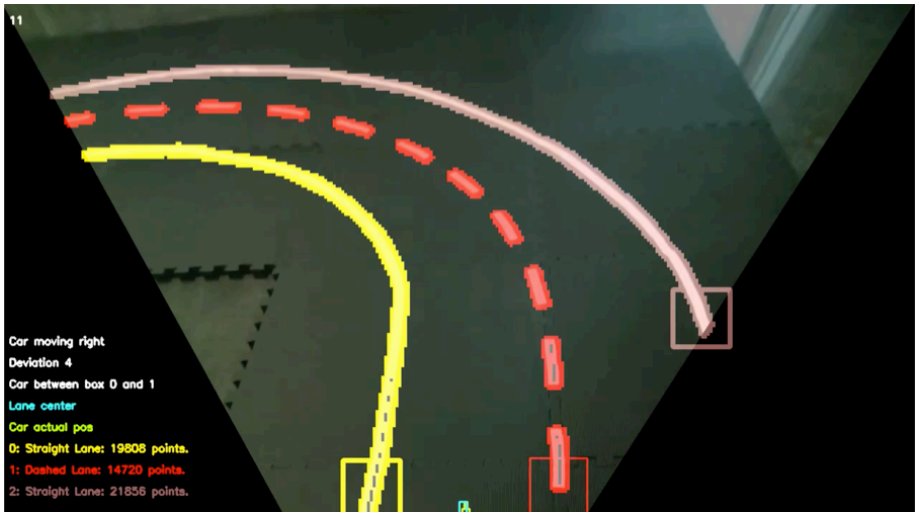


Figure 26. Final Output of the Input Frame as Shown in Table 15 (a).

In Figure 26, the following information is displayed:

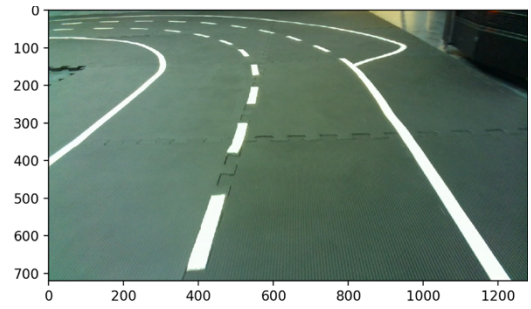
1. Car moving right – it means that the vehicle is on the right side of the center of the lane
2. Deviation 4 – indicates the distance between vehicle and lane center in terms of pixels
3. Car between 0 and 1 – indicates that the car is in between tracking windows with labels 0 and 1
4. Lane center – constant text used to show the color of lane center box drawn in the output image
5. Car actual pose – represents color of vehicle position box
6. 0: Straight lane/ 1: Dashed lane/ 2: Straight lane – displays the tracking box label and number of points in that particular lane tracked by this box

Lane Split Scenario

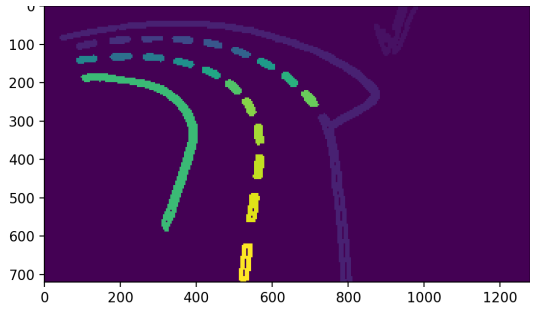
Table 16 demonstrates the lane splitting. The information about when the vehicle will come across the split lane is needed. This can be obtained from Maps. As the algorithm runs, it is being assumed that the information about when the car is approaching the split lane is provided by Maps. If this information of the lane split in a particular frame is not provided, every line is processed twice in every frame in order to detect the split. The split lane is usually a solid line that divides into a solid and dashed line. The image (a) in Table 16 is the input image where lanes are split. Clustering technique is used to detect all the points of the solid lane as shown in (c). Augmented sliding window is then run on the same line to check new points that are not there in the points of the detected solid lane using clustering, which is shown in (e). Since clustering detects the line points that are continuous, an augmented sliding window will detect the dashed line that occurs when the lane is split as seen in (g). These new points that are detected in the augmented sliding window are considered the start of lane splitting (h). As the starting point of lane split is found, a new tracking box at this point is drawn. Then perform augmented sliding window technique like for a dashed lane on this box as shown in (i). With this, all the dashed lane points that occur from the solid lane after the split will be detected as shown in (j). As the vehicle moves toward the split lane, this combination of clustering and augmented sliding window is continued until the vehicle crosses the location on the road where the split happens. The final output of the input image (a) in Table 16 is shown in Figure 27.

Table 16

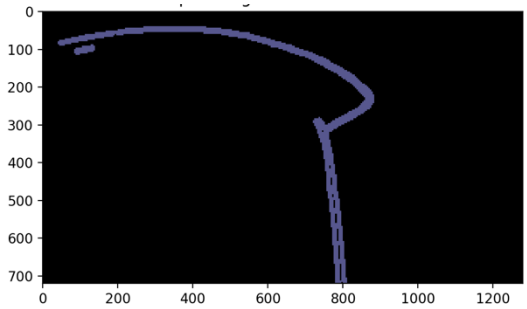
Demonstration of Lane Splitting



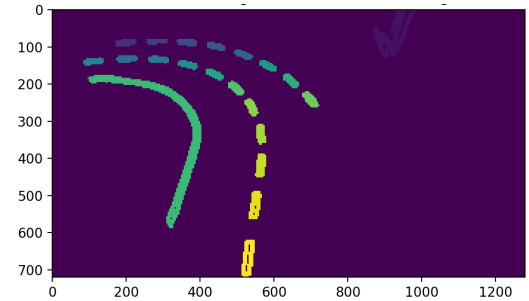
(a) Input Image with Lane Split



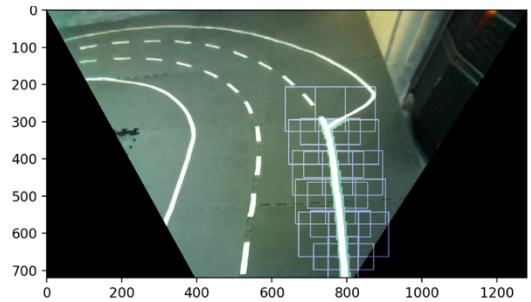
(b) Clustered Image



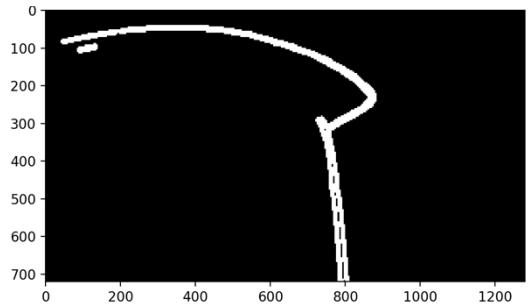
(c) Solid Lane Detected Via Clustering



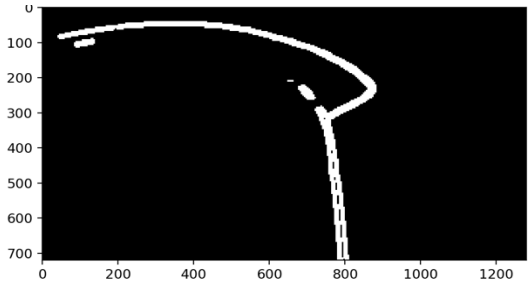
(d) Remaining Points to Be Detected Shows New Line Points to Be Detected



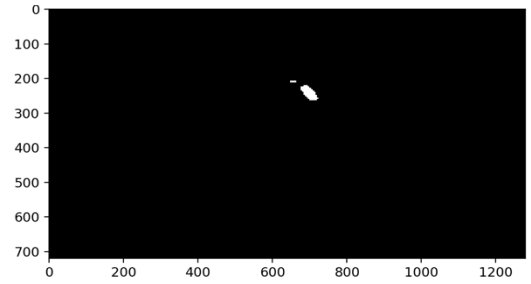
(e) Augmented Sliding Window Performed on Same Solid Line



(f) Points Detected with Clustering



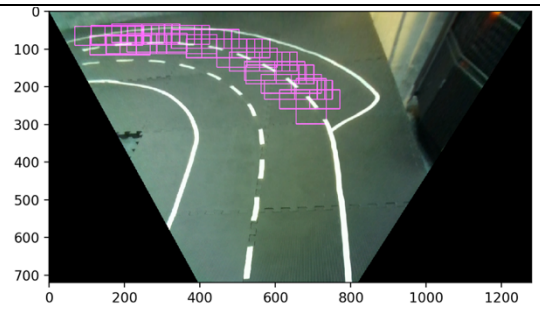
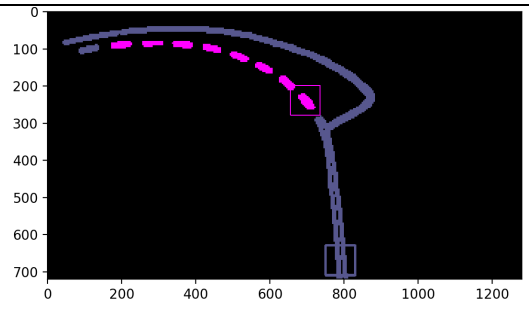
(g) Points Detected Via Augmented Sliding Window



(h) Points Detected by Augmented Sliding Window but Not in Clustering

(continued on following page)

Table 16 continued

 (i) Augmented Sliding Window Technique Performed Where New Points Detected	 (j) New Split Lane Points Detected

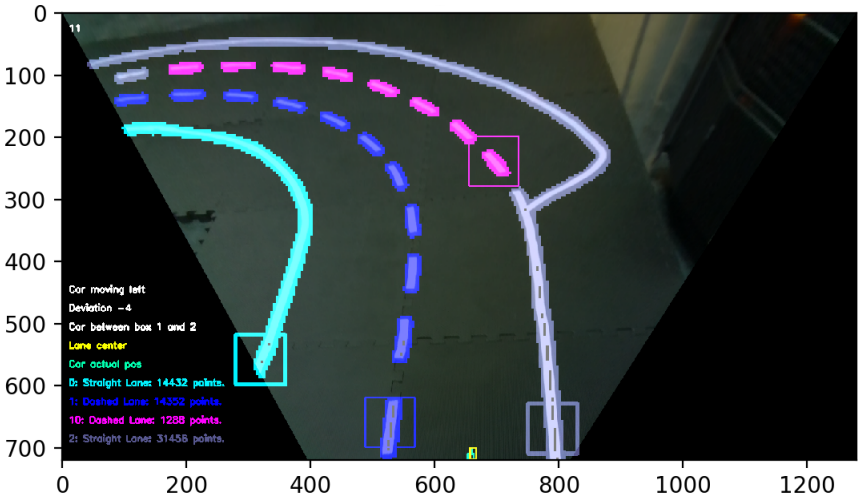


Figure 27. Output Image of Splitting Lane.

Merging of Lanes

Merging of the lane is simpler than splitting lane detection. When the lines merge, a solid lane and a dashed line converge to form a single solid line (Figure 28). Clustering and augmented sliding window techniques are run on both respective lines before the merge. Clustering technique

detects all the points in solid line and augmented sliding window technique detects all points in dashed line that occur till the point where the merge happens.

Vehicle location is again required to know, which tells us, when a vehicle has crossed the location, at which point the lanes merge. Once the vehicle crosses this location, the tracking box that was detecting the dashed lane is removed, as this lane no longer exists. If this information is not retrieved from the map, merging of lanes can still be detected when both tracking windows start to track the same lane. In the real world when a lane splits or merges it usually happens on either the leftmost line or rightmost line and the leftmost line and rightmost lines are solid lines. This information is used in the algorithm.

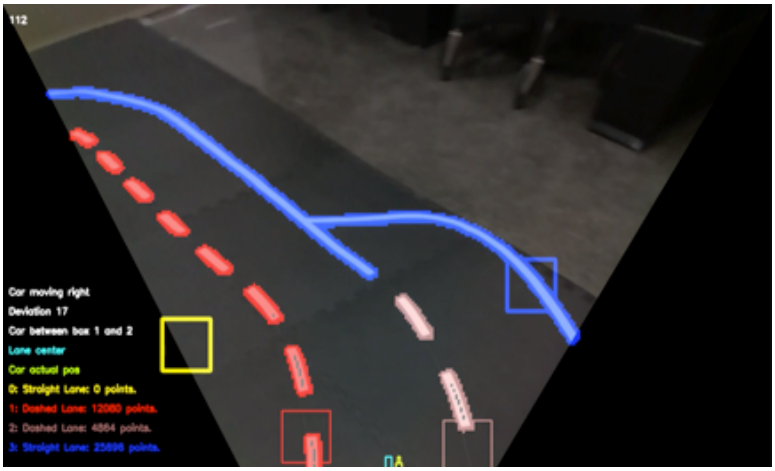


Figure 28. Output Image of Merging Lanes.

Obscured Lanes

Obscured lanes are the lines which are partially occluded due to worn out or faded lines or, in few cases, tire marks might occlude lane lines. Table 17 shows a sample obscured lane where a

solid line is not continuously visible. This section provides the details about how the lane position is estimated depending on adjacent lanes.

Table 17

Obscured Lane Input at Various Stages in Lane Detection

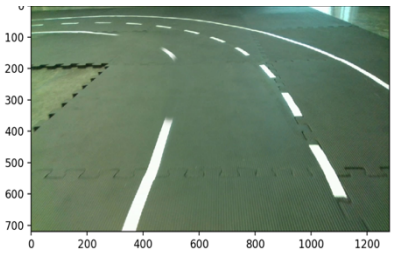
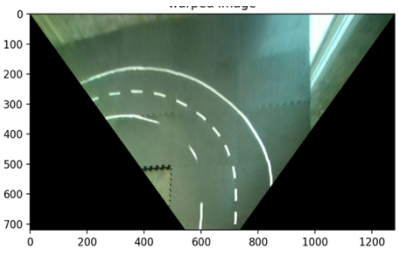
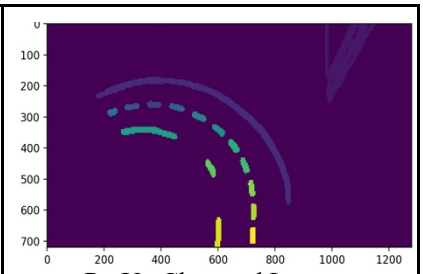
 Input Image	 Warped Image	 PyrUp Clustered Image

Figure 29 shows the final output from lane detection using augmented sliding window and clustering technique without lane estimation. Whether clustering technique or sliding window technique is applied, the left solid lane is not completely tracked due to lack of continuity in lane line points. Hence, lane position is estimated based on adjacent lane points.

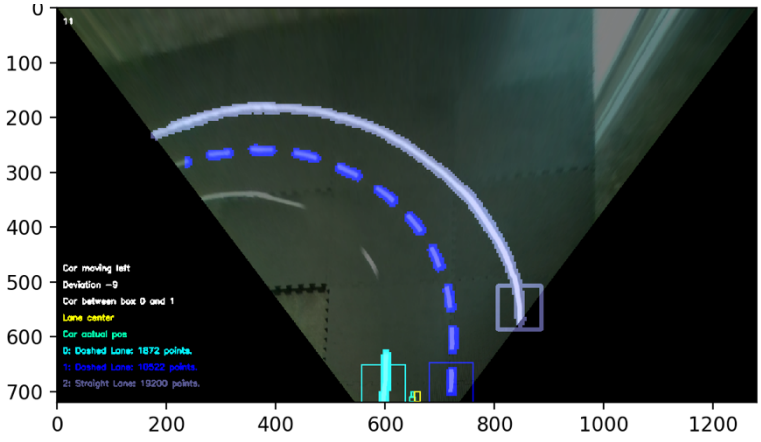
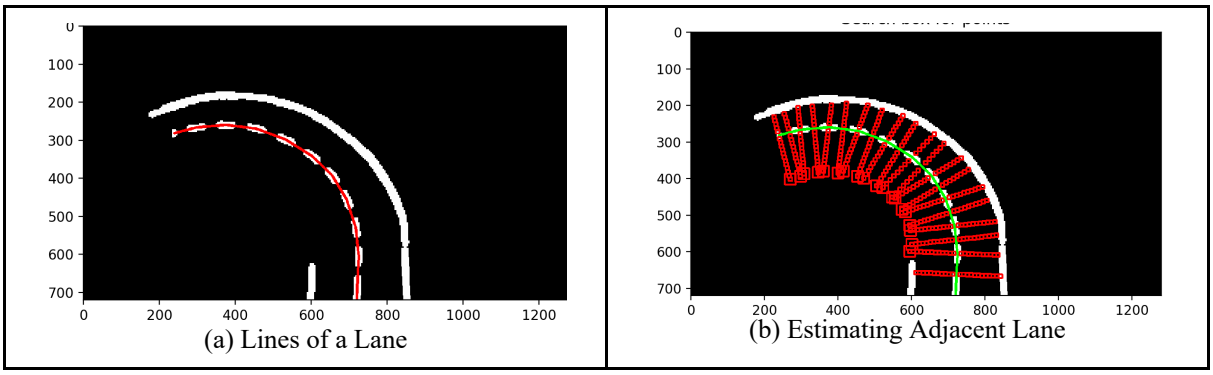


Figure 29. Lane Detection Output Without Estimation.

The output from the lane detection algorithm provides all the points present in individual lines, including partially detected solid line points. Dashed line adjacent to the partially detected solid lane has most of the points detected. Hence, we take all lane points on this dashed lane, and based on the lane width determined in the past few frames, the lane points on the left of the dashed lane are estimated. Table 18 (a) shows the lane points detected by lane detection algorithm and center lane points of the dashed lane and (b) shows how lane points in solid line are estimated. The detected points are then bound together as lane and midpoints are calculated as seen in Figure 30.

Table 18

Estimation of Obscured Lines



(a) Lines of a Lane

(b) Estimating Adjacent Lane

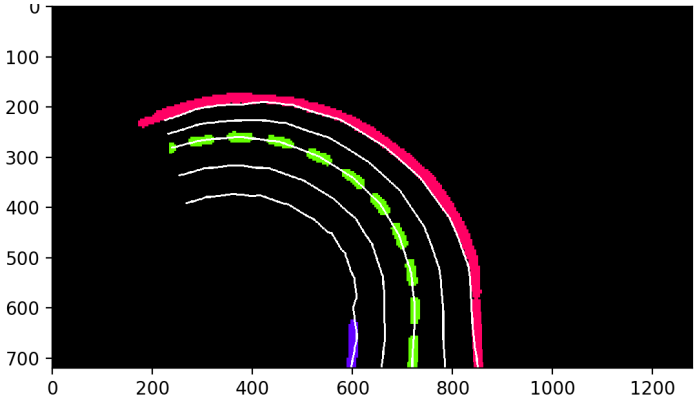


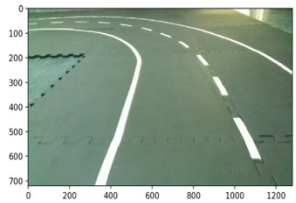
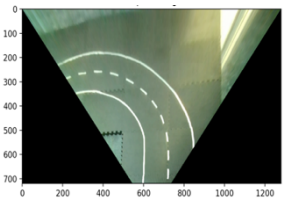
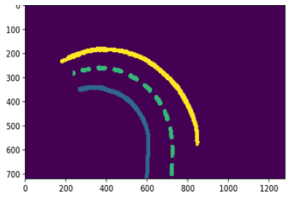
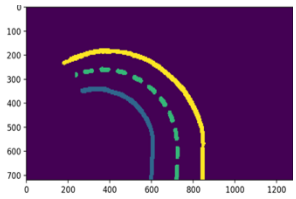
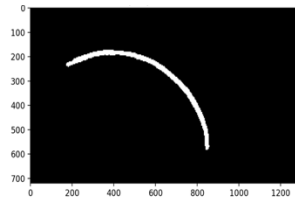
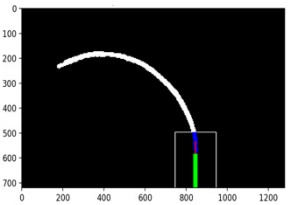
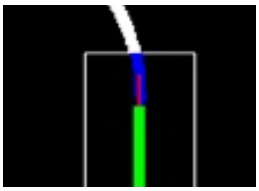
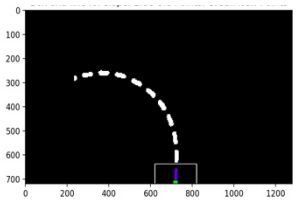
Figure 30. Final Output After Estimation.

Finding Center of the Lanes

To reduce errors during the calculation of the center of the lane and to find the center of the lane of adjacent lanes, the missing points at the bottom of the lane lines are constructed based on the slope of the existing line. This process is shown in Table 19: (a) is the input image, (b) is the warped image. From (b), it can be seen that the right lane points at the bottom of the image are missing; (c) is the output of the lane detection algorithm; and (d) is the image obtained after reconstructing the missed part. This reconstruction is shown in the images from (e) to (f): (e) is the separated line, (f) describes the small part of the line to find slope and extends the missing part of the line. The lanes' bottom points are considered, and slope is calculated. Based on it, the obtained slope of a line is constructed until the bottom of the image, that is until 'y' value reaches its peak value at 720. From (g) and (h), it can be clearly seen that the blue part is the selected part to find slope and the green part shows the extended line.

Table 19

Approximating the Missed Part of the Lane

 (a) Input Image	 (b) Warped Image	 (c) Image Before Extension	 (d) Image After Extension
 (e) Input Points of Lane	 (f) Blue-Old Points, Green- New Points	 (g) Solid Line Extension	 (h) Dashed Line Extension

The output of the lane detection algorithm are the respective points of each lane. As the lanes are curved, in order to calculate the midpoints, the steps shown in Figure 31 are followed. A particular lane is selected, and a set of consecutive points is selected on this lane line. The slope of lines formed by these consecutive points is calculated. For each of these lines, perpendicular lines are calculated, and the adjacent lane line points are found. The midpoints of respective points are calculated.

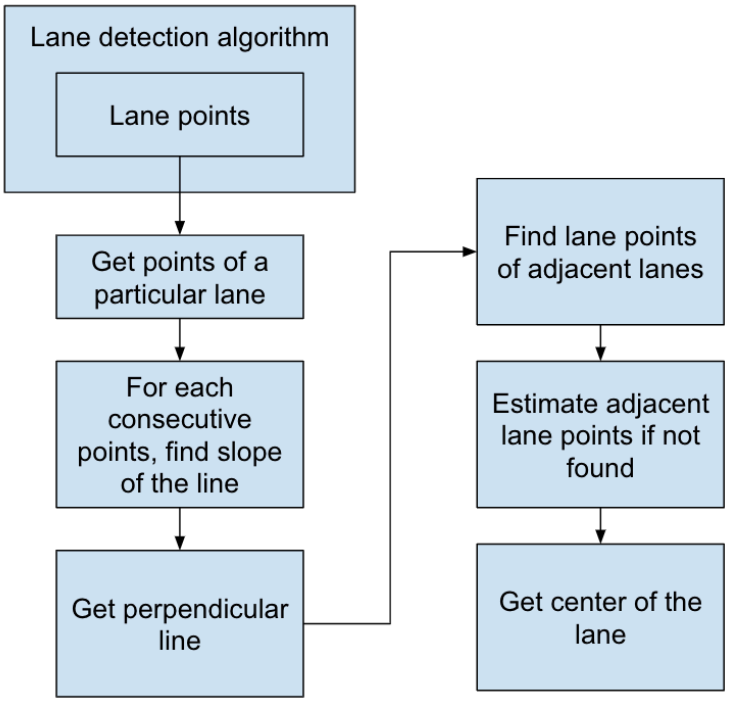
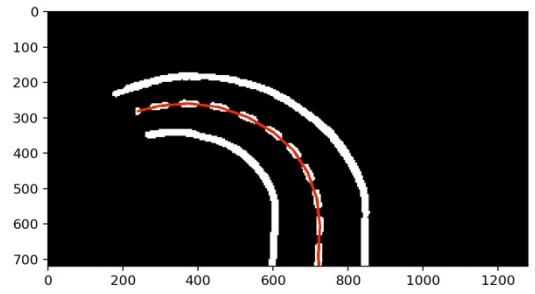


Figure 31. Flowchart for Finding Center of the Lane.

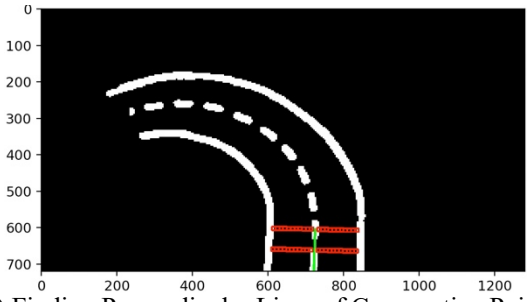
Table 20 shows the results of finding center of the lane. In Table 20, (a) shows the image where the dashed line points are considered for detecting the adjacent line points, and (b) represents the formation of perpendicular lines using point and slope. The point on the adjacent line where the perpendicular line intersects is computed. Using the points on the adjacent line and the current line, the center of the lane is found as shown in the image (d) in Table 20.

Table 20

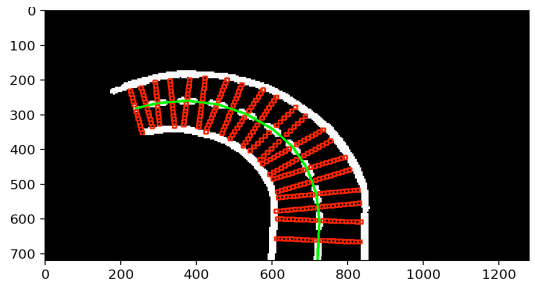
Results of Finding Center of the Lane



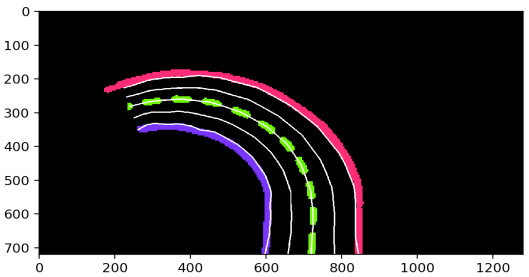
(a) Finding Consecutive Points for A Dashed Line



(b) Finding Perpendicular Lines of Consecutive Points



(c) Finding Adjacent Points Throughout the Lane



(d) Final Output Showing Midpoint Line for Two Lines

CHAPTER 4

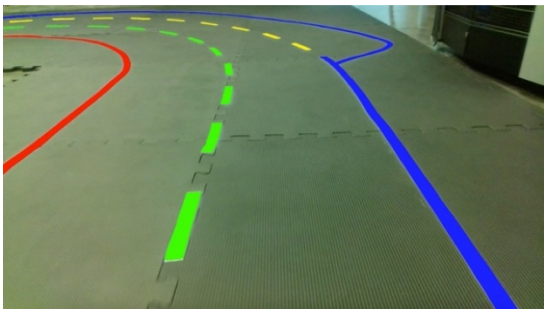
PERFORMANCE EVALUATION

Procedure

To evaluate efficiency of the algorithm, lanes with colored lines are created. This can be done by using different colors when drawing lanes as seen in Table 21. The lines can be identified by the colors; for example, blue color corresponds to solid lane that splits, green corresponds to dashed lane and red corresponds to left solid lane, etc.

Table 21

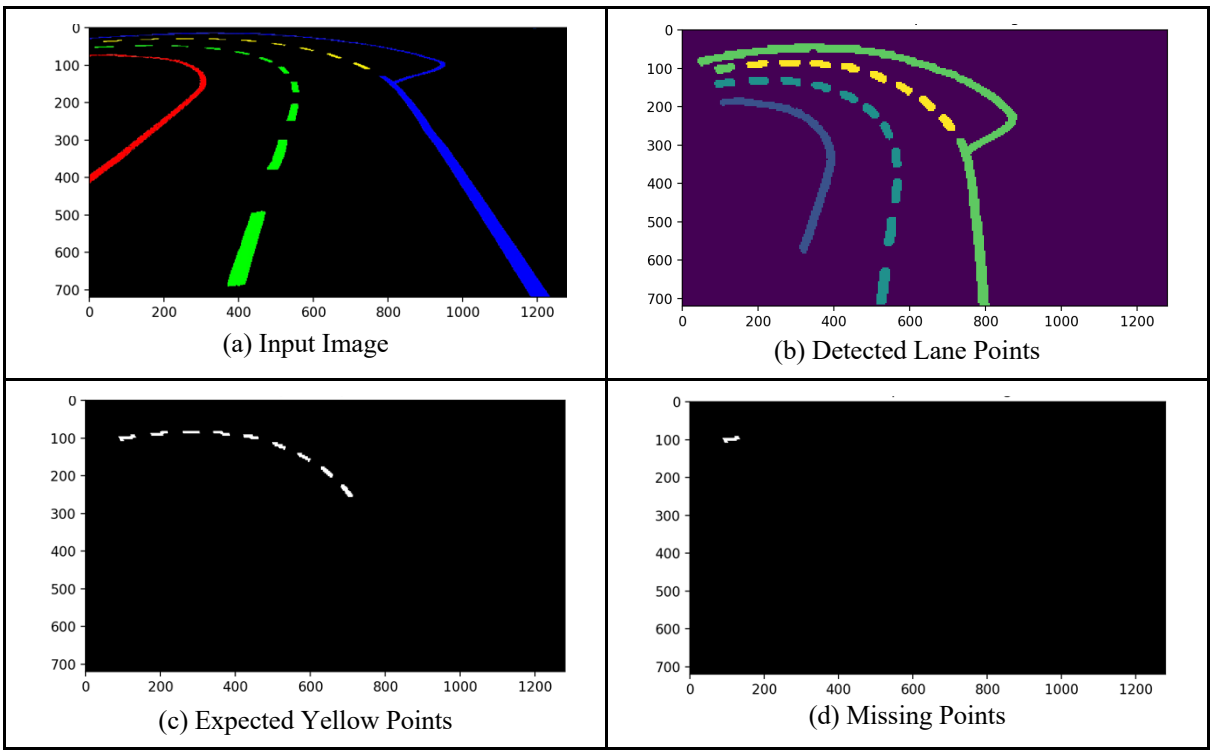
Input Image for Performance Evaluation

 (a) Normal Input Image	 (b) Colored Lines

By performing color extraction, lane points that belong to specific lanes are identified, and separate binary images are stored per lane that represent ones where line points exist, and zero otherwise. These images represent expected points per line; (c) in Table 22 shows such an image.

Table 22

Steps of Performance Evaluation



All lane points are extracted as seen in Table 22 (a) and are provided as input to lane detection algorithms. Algorithms produce output in the form of grayscale images, where every pixel that is detected as part of the line is associated with a predefined line number. If a pixel does not belong to any line, it gets a value of 0. Hence, points detected for a line can be extracted by filtering this grayscale image with a predefined lane number. Such an image is shown in Table 22 (b), which shows expected yellow line pixels along with other lane points.

For each lane, this output image is compared with an image that contains expected pixels for that lane, and the percentage of total detected points is calculated. Table 22 (d) shows expected yellow line points that were not detected. The percentage of lane detection is recorded per frame for reference if needed. Figure 32 displays output of performance detection performed on a single frame. Percentage of detection per lane is displayed in the output. To compare different lane detection algorithms, only outputs of lane detection are compared, without adding any estimation or lane extension technique specified in Chapter 3, Obscured Lanes section.

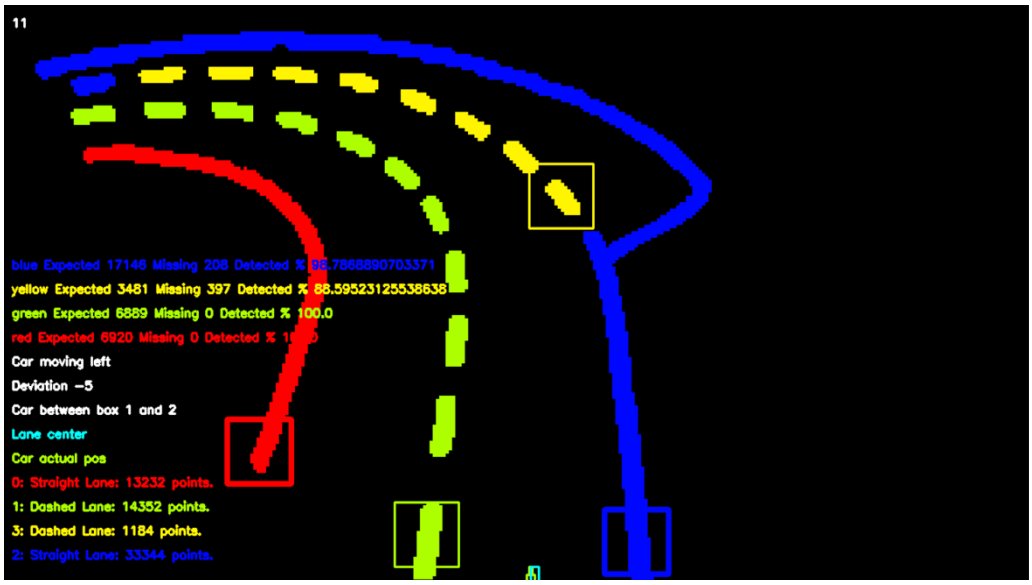


Figure 32. Visual Output of Performance Analysis.

Six input videos are used to test this algorithm. The first and second input data sets consist of 196 and 223 frames respectively. The input frames in these two videos consist of two lines (left green line, right blue line). These lines were made at the very beginning of this project (when the objective was just to make some lane lines); hence, solid and dashed lines are not considered in these frames (the lines are not uniformly spaced as seen in Figure 33). Red line is not a detectable

line; it is the center of the lane which is used to maintain uniform lane width and for checking the center of a detected lane manually.

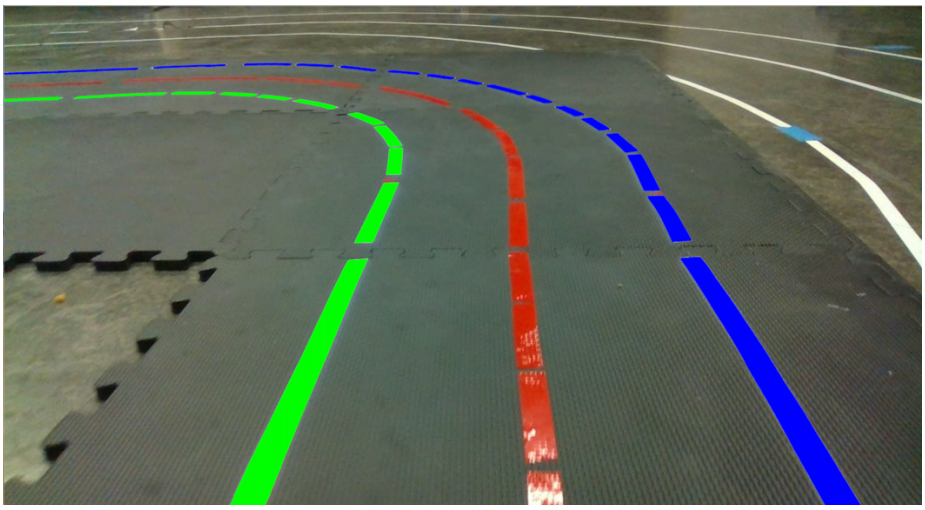


Figure 33. Input Lanes Scenario in First and Second Video Sequences.

The next four sets of videos have different curvature, lane width, number of lanes and well-defined dashed and solid lines when compared to the first two videos. Input videos 3 and 4 contain three lane lines, two solid and one dashed line as seen in Figure 34. The lanes do not split or merge. Input videos 5 and 6 have complex scenarios where the lane splits and merges. There are three lane lines, and the third lane splits and merges in these videos. One of the input frames from this video is shown in Table 21(a). Hence, this lane contains two solid lines and two dashed lines (one of these exists only when the lane splits). In this data set, the leftmost line (red) and rightmost line (blue) are solid lines as shown in Table 21(b). The second leftmost line (green) and second rightmost line (yellow) when there are four lines are dashed lines.

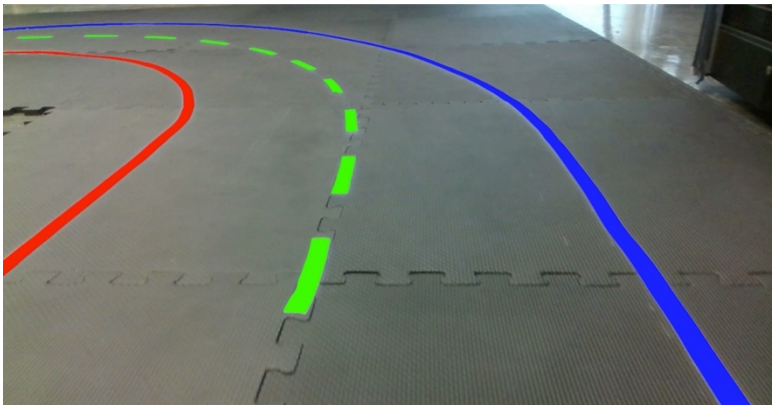


Figure 34. Expected Image.

Input Video 1

Table 23 shows the percentage of successful lane detection for input video 1 using respective lane detection algorithm. As the input video consists of irregularly spaced lines and also it doesn't consist of split and merge lanes, using augmented sliding window technique and clustering combined technique is not necessary.

Table 23

Percentage of Lane Detection for the Input Video Sequence 1

Technique	Left Line (Green) (%)	Right Line (Blue) (%)	Overall Accuracy (%)
Sliding Window	69.72	49.43	59.57
Augmented Sliding Windows	99.99	99.02	99.50
Clustering	99.99	98.35	99.17

From the Table 23, it can be inferred that the input video 1 consists of 196 frames and the average accuracy of lane detection using sliding window, augmented sliding windows and clustering are 59.57 %, 99.50 % and 99.17 % respectively.

Input Video 2

The second input video consists of 223 frames and consists of frames similar to Figure 33. Table 24 shows the determined percentage of lane detection. The average overall accuracies for input video 2 of lane detection using sliding window, augmented sliding windows and clustering technique are 87.13 %, 99.67 % and 99.67 % respectively.

Table 24

Percentage of Lane Detection for the Input Video Sequence 2

Technique	Left Line (Green) (%)	Right Line (Blue) (%)	Overall Accuracy (%)
Sliding Window	82.68	91.59	87.13
Augmented Sliding Windows	99.99	99.35	99.67
Clustering	99.99	99.35	99.67

Input Video 3

Third input video consists of 247 frames. It consists of input images similar to Figure 34. Table 25 shows the performance output for input video 3 of lane detection algorithm using single sliding window, augmented sliding windows, clustering and the combined technique of augmented sliding windows and clustering.

Table 25

Percentage of Lane Detection for the Input Video Sequence 3

Technique	Left Solid Line (Red) (%)	Middle Dashed Line (Blue) (%)	Right Solid Line (Blue) (%)	Overall Accuracy (%)
Sliding Window	59.46	34.60	16.26	36.77
Augmented Sliding Windows	99.39	97.30	99.10	98.59
Clustering	99.99	86.72	99.98	95.56
Clustering and Augmented Sliding Windows	99.99	97.35	99.99	99.11

Input Video 4

This input data set consists of a total of 295 frames similar to Figure 34. Table 26 shows the percentage of detection of lanes using various techniques for an input video 4. The overall average accuracies using sliding window, augmented sliding windows, clustering and combination of clustering and augmented sliding windows for the input video 4 are 53.70 %, 95.56 %, 90.60 % and 96.30 % respectively.

Table 26

Percentage of Lane Detection for the Input Video Sequence 4

Technique	Left Solid Line (Red) (%)	Middle Dashed Line (Blue) (%)	Right Solid Line (Blue) (%)	Overall Accuracy (%)
Sliding Window	62.31	53.76	45.03	53.70
Augmented Sliding Windows	88.08	98.93	99.67	95.56
Clustering	87.59	84.74	99.49	90.60
Clustering and Augmented Sliding Windows	90.46	98.71	99.75	96.30

Input Video 5

The input video 5 consists of 222 frames in which the lines that merge and split are observed as shown in Table 21. Table 27 shows the percentage values for successfully detected lanes of input video 5 using the combined augmented sliding windows and clustering technique. As this input video consists of merge and split lines, the sliding window technique, augmented sliding windows and clustering cannot be applied. Hence, the performance of the combined technique is determined.

Table 27

Percentage of Lane Detection for the Input Video Sequence 5

Left Solid Line (Red)	**Middle Dashed Line (Green)**	**Middle Right Dashed Line (Exists During Merge-Yellow)**	**Right Solid Line (Involved in Split and Merge- Blue)**	**Overall Accuracy (%)**
91.74	97.70	86.76	99.31	93.87

Input Video 6

The input video 6 consists of 205 frames similar to the fifth video. Table 28 shows the percentage values for successfully detected lanes of input video 6 using the combined augmented sliding windows and clustering technique. The average overall accuracy obtained is 99.02 %.

Table 28

Percentage of Lane Detection for the Input Video Sequence 6

Left Solid Line (Red)	Middle Dashed Line (Green)	Middle Right Dashed Line (Exists During Merge-Yellow)	Right Solid Line (Involved in Split and Merge- Blue)	Overall Accuracy (%)
99.99	97.84	99.81	98.44	99.02

Single Sliding Window Technique

Single sliding window technique is performed on the input video with three lanes used for performance evaluation in Chapter 3, Augmented Sliding Window and Clustering section. A snapshot of the frame on which single sliding window is performed is shown in Figure 35. As seen from this figure, the single sliding window technique does not efficiently track the points of curved lanes.

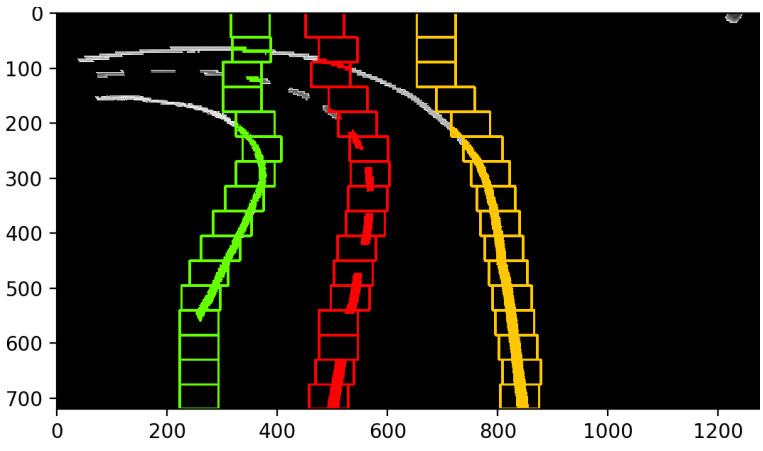


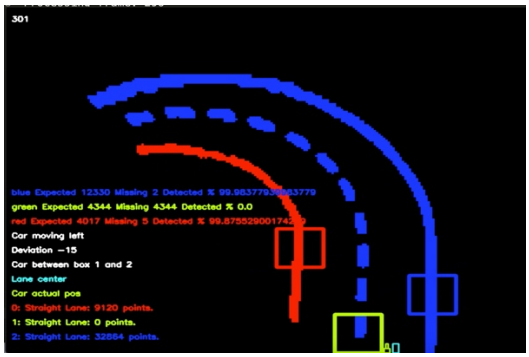
Figure 35. Single Sliding Window Technique Applied on Curved Lanes.

Clustering Technique

The Epsilon value in DBSCAN clustering technique is adjusted as per the lane properties such that all lanes are distinctly categorized as separate clusters as seen in Table 29 (a). The Epsilon is adjusted such that (distance between dashed line gaps) < epsilon < (distance between lane lines), and a good perspective warp enables us to do this. As lane line gaps are not always consistent, occasionally the lane lines may be closer than dashed lane gaps, as seen in Table 29 (b). In such a case, all points of different lane lines are categorized as lane line points of one cluster. This affects the detection percentage in our analysis.

Table 29

Clustering Technique Outputs

	
(a) Good Clustering	(b) Inefficient Clustering

Performance Comparison

The performance of the lane detection algorithms analyzed in this paper is compared in Table 30. Same data sets are used to compare the performance of lane detection using input videos 3 and 4. It can be seen that augmented sliding window technique with clustering performs better than other lane detection algorithms for the considered input data scenario.

Table 30

Performance Comparison of Analyzed Lane Detection Algorithms

Technique	**Total Accuracy (%)**
Single Sliding Window	45.23
Augmented Sliding Window	97.07
Clustering Technique	93.08
Augmented Sliding Window and Clustering Technique	97.70

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

In this paper, lane detection of autonomous vehicles using augmented sliding window technique is implemented. It is also combined with clustering techniques to efficiently detect the lanes that are splitting. Various types of lanes are simulated in the laboratory and lane detection algorithms are applied on this data set. Obscure lanes which have missing lane markings are also detected by calculating their expected positions relative to adjacent lane markings. Vehicle's position relative to lanes is found, and midpoint of all lanes from vehicle to horizon are also calculated. Performance of various lane detection techniques is compared. It was observed that augmented sliding window technique and clustering technique provides efficient results, and clustering technique is more dependent on lane properties (lane width, distance between dashed lines) than augmented sliding window technique. Performance analysis technique was created to automate the percentage of accurate detection of lane points instead of manually checking for efficiency of detected lanes. Image upscaling technique was created to increase speed of the performance of clustering. The augmented sliding windows dimensions and perspective warp constants are dependent on lane properties such as lane width and dashed lane gaps.

Good perspective warp constants provide well-separated lane points, which can avoid accidentally detecting lane points across lanes. All lane detection algorithms that are implemented in this paper use input threshold parameters for various image processing techniques like blur (kernel size of filter), canny edge detection (low-level and high-level threshold values), etc. These constants are dependent on lighting conditions and are fine tuned for lighting conditions in the laboratory. These constants and properties need to be adjusted when lighting conditions change. A dynamic algorithm can be used to automatically tune these constants depending on lane conditions. The lane estimation technique currently used relies on calculating estimated lane positions based on adjacent lanes. This is a time-consuming process. An algorithm that tracks lane positions on image instead of using adjacent lanes can help avoid these intensive calculations. The proposed algorithm also should be tested in various test situations on larger sets of frames in future.

BIBLIOGRAPHY

[1] National Highway Traffic Safety Administration (NHTSA), U.S, Department of Transportation. Accessed on April 13, 2020. [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety#issue-road-self-driving.

[2] SAE, Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. (J3016-2018). Accessed on April 13, 2020. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/.

[3] M. Taraba, J. Adamec, M. Danko and P. Drgona, "Utilization of modern sensors in autonomous vehicles," *2018 ELEKTRO*, Mikulov, 2018, pp. 1-5.

[4] S. Gargoum and K. El-Basyouny, "Automated extraction of road features using LiDAR data: A review of LiDAR applications in transportation," *2017 4th International Conference on Transportation Information and Safety (ICTIS)*, Banff, AB, 2017, pp. 563-574.

[5] Q. Li, L. Chen, M. Li, S. Shaw and A. Nüchter, "A Sensor-Fusion Drivable-Region and Lane-Detection System for Autonomous Vehicle Navigation in Challenging Road Scenarios," in *IEEE Transactions on Vehicular Technology*, vol. 63, no. 2, pp. 540-555, Feb. 2014.

[6] Shahian-Jahromi Babak, Syed A Hussain, Burak Karakas and Sabri Cetin, "Control of Autonomous Ground Vehicles: A Brief Technical Review", *Fourth International Conference on Mechanics and Mechatronics Research, IOP Publishing*, vol. 224, 2017.

[7] W. Liu et al., "Autonomous vehicle planning system design under perception limitation in pedestrian environment," *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM),* Siem Reap, 2015, pp. 159-166.

[8] A. Rasouli and J. K. Tsotsos, "Autonomous Vehicles That Interact with Pedestrians: A Survey of Theory and Practice," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, March 2020, pp. 900-918.

[9] Zhao Pan, Jiajia Chen, Yan Song, Xiang Tao, Tiejuan Xu, and Tao Mei. "Design of a Control System for an Autonomous Vehicle Based on Adaptive-PID," *International Journal of Advanced Robotic Systems,* vol 9, Issue 2, August 2012.

[10] A. Assidiq, O. O. Khalifa, M. R. Islam and S. Khan, "Real-time lane detection for autonomous vehicles," *2008 International Conference on Computer and Communication Engineering*, Kuala Lumpur, 2008, pp. 82-88.

[11] U. Franke, D. Gavrila, S. Gorzig, F. Lindner, F. Puetzold and C. Wohler, "Autonomous driving goes downtown," *in IEEE Intelligent Systems and their Applications,* vol. 13, no. 6, Nov.-Dec. 1998, pp. 40-48.

[12] M. Bertozzi and A. Broggi, "GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection," *in IEEE Transactions on Image Processing*, vol. 7, no. 1, Jan. 1998, pp. 62-81.

[13] K. Kluge and S. Lakshmanan, "A deformable-template approach to lane detection," *Proceedings of the Intelligent Vehicles '95*. Symposium, Detroit, MI, USA, 1995, pp. 54-59.

[14] Mei Chen, T. Jochem and D. Pomerleau, "AURORA: a vision-based roadway departure warning system," *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, Pittsburgh, PA, USA, 1995, vol.1, pp. 243-248.

[15] C. Kreucher and S. Lakshmanan, "LANA: a lane extraction algorithm that uses frequency domain features," *in IEEE Transactions on Robotics and Automation,* vol. 15, no. 2, April 1999, pp. 343-350.

[16] D. Liu, Y. Wang, T. Chen and E. T. Matson, "Application of Color Filter Adjustment and K-Means Clustering Method in Lane Detection for Self-Driving Cars," *2019 Third IEEE International Conference on Robotic Computing (IRC)*, Naples, Italy, 2019, pp. 153-158.

[17] F. Bounini, D. Gingras, V. Lapointe and H. Pollart, "Autonomous Vehicle and Real-Time Road Lanes Detection and Tracking," *2015 IEEE Vehicle Power and Propulsion Conference (VPPC)*, Montreal, QC, 2015, pp. 1-6.

[18] Qiang Chen and Hong Wang, "A Real-time Lane Detection Algorithm Based on a Hyperbola-Pair Model," *2006 IEEE Intelligent Vehicles Symposium*, Tokyo, 2006, pp. 510-515.

[19] Y. Yenİaydin and K. W. Schmidt, "A lane detection algorithm based on reliable lane markings," *2018 26th Signal Processing and Communications Applications Conference (SIU)*, Izmir, 2018, pp. 1-4.

[20] Young Uk Yim and Se-Young Oh, "Three-feature-based automatic lane detection algorithm (TFALDA) for autonomous driving," *in IEEE Transactions on Intelligent Transportation Systems*, vol. 4, no. 4, Dec. 2003, pp. 219-225.

[21] A. Mahmoud et al., "Real-Time Lane Detection-Based Line Segment Detection," *2018 New Generation of CAS (NGCAS)*, Valletta, 2018, pp. 57-61.

[22] D. C. Andrade et al., "A Novel Strategy for Road Lane Detection and Tracking Based on a Vehicle's Forward Monocular Camera," *in IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 4, April 2019, pp. 1497-1507.

[23] Y. Fan, W. Zhang, X. Li, L. Zhang and Z. Cheng, "A robust lane boundaries detection algorithm based on gradient distribution features," *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD),* Shanghai, 2011, pp. 1714-1718.

[24] T. T. Duong, C. C. Pham, T. H. Tran, T. P. Nguyen and J. W. Jeon, "Near real-time ego-lane detection in highway and urban streets," *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, Seoul, 2016, pp. 1-4.

[25] W. Liu and S. Li, "An Effective Lane Detection Algorithm for Structured Road in Urban", In: Yang J., Fang F., Sun C. (eds) *Intelligent Science and Intelligent Data Engineering. IScIDE 2012. Lecture Notes in Computer Science*, vol 7751. Springer, Berlin, Heidelberg, 2013, pp.759-767.

[26] C. Ma and M. Xie, "A Method for Lane Detection Based on Color Clustering," *2010 Third International Conference on Knowledge Discovery and Data Mining, Phuket*, 2010, pp. 200-203.

[27] Rudra N. Hota, Shahanaz Syed, Subhadip Bandyopadhyay and P. Radha Krishna. "A Simple and Efficient Lane Detection using Clustering and Weighted Regression," *15th International Conference on Management of Data (COMAD)*, Mysore, India, December 9-12, 2009.

[28] J. Wang, W. Hong and L. Gong, "Lane detection algorithm based on density clustering and RANSAC," *2018 Chinese Control and Decision Conference (CCDC)*, Shenyang, 2018, pp. 919-924.

[29] J. He, S. Sun, D. Zhang, G. Wang and C. Zhang, "Lane Detection for Track-following Based on Histogram Statistics," *2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, Xi'an, China, 2019, pp. 1-2.

[30] Y. Xing, C. Lv, H. Wang, D. Cao and E. Velenis, "Dynamic integration and online evaluation of vision-based lane detection algorithms," in *IET Intelligent Transport Systems*, vol. 13, no. 1, 1 2019, pp. 55-62.

[31] M. Venkatesh and P. Vijayakumar, "Transformation Technique," *International Journal of Scientific and Engineering Research*, Volume 3, Issue 5, May-2012, pp. 735-738.

[32] Martin Ester, Hans-Peter Kriegel, Jorg Sander and Xiaowei Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, AAAI Press, pp. 226-231.