# Chapter 1: INTRODUCTION

Lane detection is an important enabling or enhancing technology in a number of intelligent vehicle applications, including lane excursion detection and warning, intelligent cruise control and autonomous driving.

Various lane detection methods have been proposed. They are classified into infrastructure-based and vision-based approaches. While the infrastructure-based approaches achieve highly robustness, construction cost to lay leaky coaxial cables or to embed magnetic markers on the road surface is high. Vision based approaches with camera on a vehicle have advantages to use existing lane markings in the road environment and to sense a road curvature in front view.

Vision-based location of lane boundaries can be divided into two tasks: lane detection and lane tracking. Lane detection is the problem of locating lane boundaries without prior knowledge of the road geometry. Lane tracking is the problem of tracking the lane edges from frame to frame given an existing model of road geometry. Lane tracking is an easier problem than lane detection, as prior knowledge of the road geometry permits lane tracking algorithms to put fairly strong constraints on the likely location and orientation of the lane edges in a new image. Lane detection algorithms, on the other hand, have to locate the lane edges without a strong model of the road geometry, and do so in situations where there may be a great deal of clutter in the image. This clutter can be due to shadows, puddles, oil stains, tire skid marks, etc. This poses a challenge for edge-based lane detection schemes, as it is often impossible to select a gradient magnitude threshold which doesn't either remove edges of interest corresponding to road markings and edges or include edges corresponding to irrelevant clutter.

Detection of long thick lines, such as highway lane markings from input images, is usually performed by local edge extraction followed by straight line approximation. In this conventional method many edge elements other than lane makings are detected when the threshold of edge magnitude is low, or, in the opposite case, edge elements expected to be detected are fragmented when it is high. This makes it difficult to trace the edge elements and to fit the approximation lines on them.

## 1.1 Project Aim

In any driving scenario, lane lines are an essential component of indicating traffic flow and where a vehicle should drive. It's also a good starting point when developing a self-driving car! In this project, we will be showing how to build your own lane detection system in OpenCV using Python. Here's the structure of our lane detection pipeline:

- Reading Images

- Color Filtering in HLS
- Region of Interest
- Canny Edge Detection
- Hough Line Detection
- Line Filtering & Averaging
- Overlay detected lane
- Applying to Video

With the rapid development of society, automobiles have become one of the transportation tools for people to travel. In the narrow road, there are more and more vehicles of all kinds. As more and more vehicles are driving on the road, the number of victims of car accidents is increasing every year. How to drive safely under the condition of numerous vehicles and narrow roads has become the focus of attention. Advanced driver assistance systems which include lane departure warning (LDW), Lane Keeping Assist, and Adaptive Cruise Control (ACC) can help people analyze the current driving environment and provide appropriate feedback for safe driving or alert the driver in dangerous circumstances. This kind of auxiliary driving system is expected to become more and more perfect.

Lane detection is a hot topic in the field of machine learning and computer vision and has been applied in intelligent vehicle systems. The lane detection system comes from lane markers in a complex environment and is used to estimate the vehicle's position and trajectory relative to the lane reliably. At the same time, lane detection plays an important role in the lane departure warning system. The lane detection task is mainly divided into two steps: edge detection and line detection.

Qing in his research paper proposed the extended edge linking algorithm with directional edge gap closing. The new edge could be obtained with the proposed method. Mu and Ma proposed Sobel edge operator which can be applied to adaptive area of interest (ROI). However, there are still some false edges after edge detection. These errors will affect the subsequent lane detection. Wang et al. proposed a Canny edge detection algorithm for feature extraction. The algorithm provides an accurate fit to lane lines and could be adaptive to complicated road environment. In 2014, Srivastava proposed that the improvements to the Canny edge detection can effectively deal with various noises in the road environment. Sobel and Canny edge operator are the most commonly used and effective methods for edge detection.

Line detection is as important as edge detection in lane detection. With regard to line detection, we usually have two methods which include feather-based method and model-based methods. Niu used a modified Hough transform to extract segments of the lane profile and used DBSCAN (density based spatial application noise clustering) clustering algorithm for clustering. In 2016, Mammeri used progressive probabilistic Hough transform combined with maximum stable extreme area (MSER) technology to identify and detect lane lines and utilized Kalman filter to achieve continuous tracking. However, the algorithm does not work well at night.

In this project, we propose a lane detection method that is suitable for all kinds of vehicles. First, we preprocessed each frame image and then selected the area of interest (ROI) of the processed images. Finally, we only needed edge detection vehicle and line detection for the ROI area. In this study, we introduced a new preprocessing method and ROI selection method. First, in the preprocessing stage, we converted the RGB color model to the HSV color space model and extracted white features on the HSV model. At the same time, the preliminary edge feature detection is added in the preprocessing stage, and then the part below the image is selected as the ROI area based on the proposed preprocessing.

## 1.2 Objective

The objective of this project is to use traditional Computer Vision techniques to develop an advanced and robust algorithm that can detect and track lane boundaries in a video. The pipeline highlighted below was designed to operate under the following scenarios:

- It can detect *exactly* two lane lines, i.e. the left and right lane boundaries of the lane the vehicle is currently driving in.
- It cannot detect adjacent lane lines.
- The vehicle must be within a lane and must be aligned along the direction of the lane.
- If only one of two lane lines have been successfully detected, then the detection is considered invalid and will be discarded. In this case, the pipeline will instead output a lane line fit (for both left and right) based on the moving average of the previous detections. This is due to the lack of an implementation of the lane approximation function (which is considered as future work).

The objective of this project is to use traditional Computer Vision techniques to develop an advanced and robust algorithm that can detect and track lane boundaries in a video. The pipeline highlighted below was designed to operate under the following scenarios:

- To detect exactly two lane lines, i.e. the left and right lane boundaries of the lane the vehicle is currently driving in.
- To carry load and transfer load between two divisions of any Industry/Factory.
- To use radium tapes to make lane which will be detected by machine as it will reduce the electricity cost of the factory with manual labour as well.
- To detect both the lanes because if only one of two lane lines have been successfully detected, then the detection is considered invalid and will be discarded. In this case, the pipeline will instead output a lane line fit (for both left and right) based on the moving average of the previous detections. This is due to the lack of an implementation of the lane approximation function (which is considered as future work).

## 1.3 Project Description

### 1.3.1 Goal

The goal of this project is to build up a simple image pipeline (take a frame from video as an input, do something, return a modified version of the frame), which allows detecting lane lines in simple conditions: sunny weather, good visibility, no cars in sight, only straight lanes. One more thing*: our lane line detector should be linear.*

### 1.3.2 Dependencies

- Python 3.7
- NumPy
- Matplotlib (for charting and visualising images)
- OpenCV 4.1
- MoviePy (to process video files)

### 1.3.3 Project structure

- **lane_tracker.ipynb**: Jupyter notebook with a step-by-step walkthrough of the different components of the pipeline
- **test_images/**: Folder containing a set of images for test purposes
- **readme_images**: Directory to store images used within this README.md
- **challenge_video.mp4**: Video containing uneven road surfaces and non-uniform lighting conditions
- **challenge_video_output.mp4**: Resulting output on passing the challenge_video through the pipeline
- **project_video.mp4**: Video with dark road surfaces and non-uniform lighting conditions
- **project_video_output.mp4**: Resulting output on passing the project_video through the pipeline

### 1.3.4 Pipeline

The various steps involved in the pipeline are as follows, each of these has also been discussed in more detail in the sub sections below:

- Compute the image/ video calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Apply a perspective transform to rectify image ("birds-eye view").
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.

- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

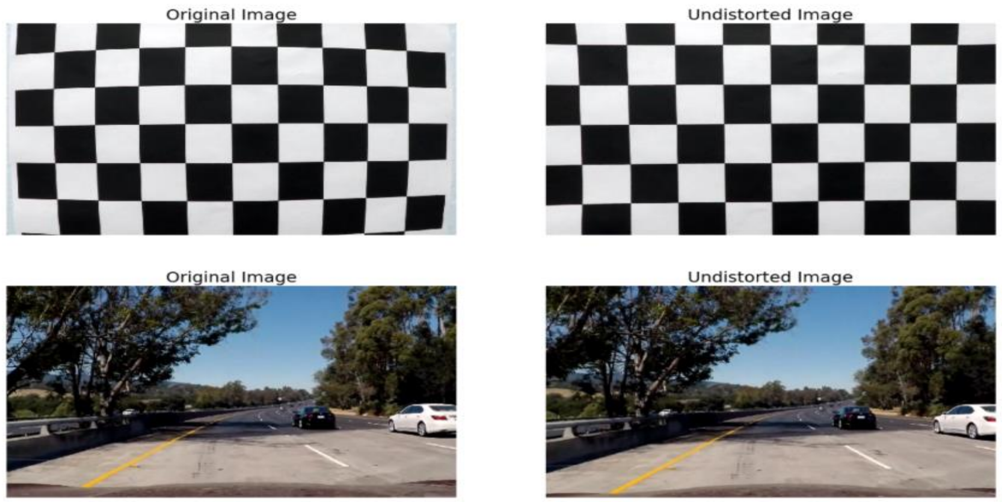


Fig.1.1

1.3.5 Generating a thresholded binary image

Many techniques such as gradient thresholding, thresholding over individual colour channels of different color spaces and a combination of are to be experimented with over a training set of images with the aim of best filtering the lane line pixels from other pixels. The experimentation yielded the following key insights:

1. The performance of individual color channels varied in detecting the two colors (white and yellow) with some transforms significantly outperforming the others in detecting one color but showcasing poor performance when employed for detecting the other. Out of all the channels of RGB, HLS, HSV and LAB color spaces that were experimented with the below mentioned provided the greatest signal-to-noise ratio and robustness against varying lighting conditions:
    - White pixel detection: R-channel (RGB) and L-channel (HLS)
    - Yellow pixel detection: B-channel (LAB) and S-channel (HLS)
2. Owing to the uneven road surfaces and non-uniform lighting conditions a strong need for Adaptive Thresholding is to be realized.

1.3.6 Lane Line detection: Sliding Window technique

A wrapped *thresholded binary image* where the pixels are either 0 or 1; 0 (black color) constitutes the unfiltered pixels and 1 (white color) represents the filtered pixels. The next step involves mapping out the lane lines and determining explicitly which pixels are part of the lines and which belong to the left line and which belong to the right line.

The first technique employed to do so is: **Peaks in Histogram & Sliding Windows**

1. We first take a histogram along all the columns in the lower half of the image. This involves adding up the pixel values along each column in the image. The two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. These are used as starting points for our search.

2. From these starting points, we use a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame.

### 1.3.7 Lane Line detection: Adaptive Search

After detecting the two lane lines, for subsequent frames in a video, we will search in a margin around the previous line position instead of performing a blind search.

Although the Peaks in Histogram and Sliding Windows technique does a reasonable job in detecting the lane line, it often fails when subject to non-uniform lighting conditions and discolouration. To combat this, a method that could perform adaptive thresholding over a **smaller** receptive field/window of the image was needed. The reasoning behind this approach was that performing adaptive thresholding over a smaller kernel would more effectively filter out our 'hot' pixels in varied conditions as opposed to trying to optimise a threshold value for the entire image.

### 1.3.8 Applying Canny Detector

The Canny Detector is a multi-stage algorithm optimized for fast real-time edge detection. The fundamental goal of the algorithm is to detect sharp changes in luminosity (large gradients), such as a shift from white to black, and defines them as edges, given a set of thresholds. The Canny algorithm has four main stages:

### 1.3.9 Noise reduction

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}$$

As with all edge detection algorithms, noise is a crucial issue that often leads to false detection. A 5x5 Gaussian filter is applied to convolve (smooth) the image to lower the detector's sensitivity to noise. This is done by using a kernel (in this case, a 5x5 kernel) of normally distributed numbers to run across the entire image, setting each pixel value equal to the weighted average of its neighboring pixels.

1.3.10 Intensity gradient

The smoothened image is then applied with a Sobel, Roberts, or Prewitt kernel (Sobel is used in OpenCV) along the x-axis and y-axis to detect whether the edges are horizontal, vertical, or diagonal.

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

1.3.11 Non-maximum suppression

Non-maximum suppression is applied to "thin" and effectively sharpen the edges. For each pixel, the value is checked if it is a local maximum in the direction of the gradient calculated previously.

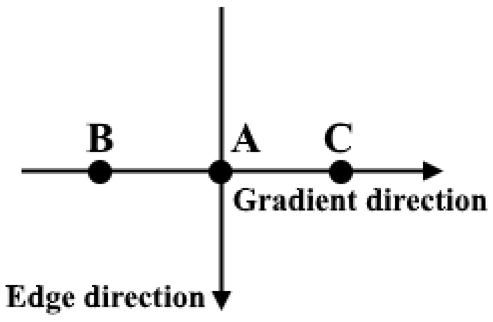


Fig. 1.2

1.3.12 Hysteresis thresholding

After non-maximum suppression, strong pixels are confirmed to be in the final map of edges. However, weak pixels should be further analyzed to determine whether it constitutes as edge or noise. Applying two pre-defined minVal and maxVal threshold values, we set that any pixel with intensity gradient higher than maxVal are edges and any pixel with intensity gradient lower than minVal are not edges and discarded. Pixels with intensity gradient in between minVal and maxVal are only considered edges if they are connected to a pixel with intensity gradient above maxVal.

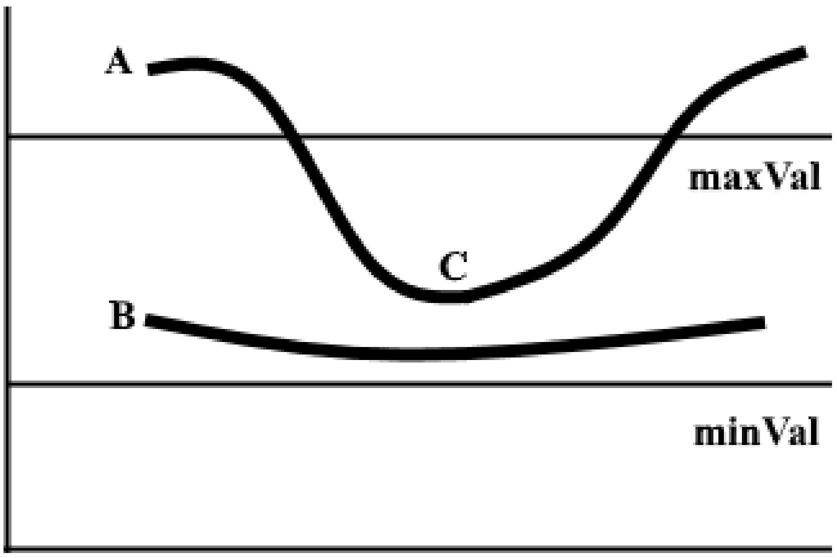


Fig. 1.3

1.3.13 Segmenting lane area

We will handcraft a triangular mask to segment the lane area and discard the irrelevant areas in the frame to increase the effectiveness of our later stages.

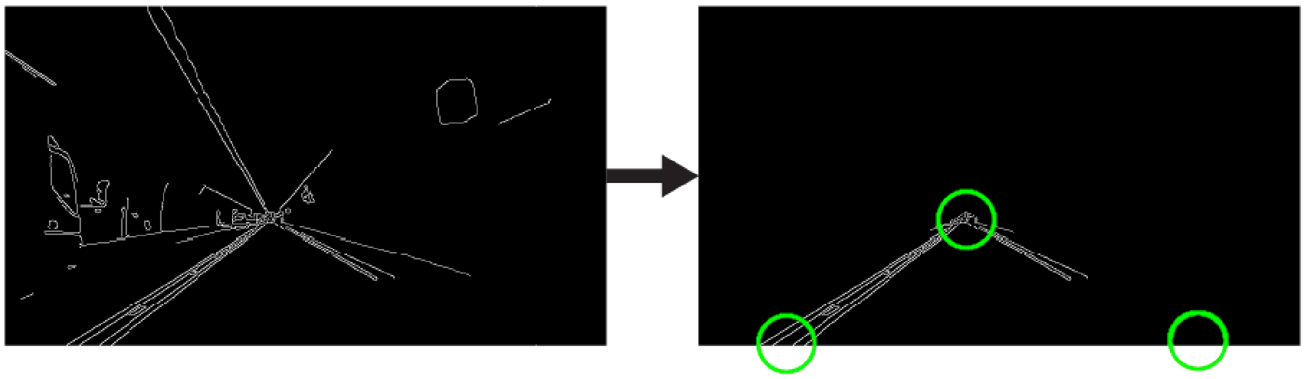


Fig. 1.4

1.2.14 Hough transforms

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing.[1] The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses. The Hough transform as it is universally used today was invented by Richard Duda and Peter Hart in 1972, who called it a "generalized Hough transform"[2] after the related 1962 patent of Paul Hough.[3][4] The transform was popularized in the computer vision community by Dana H. Ballard through a 1981 journal article titled "Generalizing the Hough transform to detect arbitrary shapes"

## Theory

In automated analysis of digital images, a sub problem often arises of detecting simple shapes, such as straight lines, circles or ellipses. In many cases an edge detector can be used as a pre-processing stage to obtain image points or image pixels that are on the desired curve in the image space. Due to imperfections in either the image data or the edge detector, however, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line/circle/ellipse and the noisy edge points as they are obtained from the edge detector. For these reasons, it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects (Shapiro and Stockman, 304).

The simplest case of Hough transform is detecting straight lines. In general, the straight line *y = mx + b* can be represented as a point (*b*, *m*) in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter *m*. Thus, for computational reasons, Duda and Hartproposed the use of the Hesse normal form

,

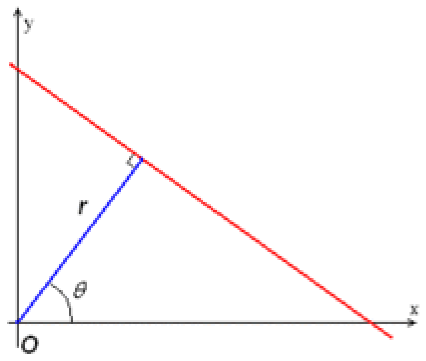


Fig 1.5

It is therefore possible to associate with each line of the image a pair . The plane is sometimes referred to as *Hough space* for the set of straight lines in two dimensions. This representation makes the Hough transform conceptually very close to the two-dimensional Radon transform. (They can be seen as different ways of looking at the same transform.

Given a *single point* in the plane, then the set of *all* straight lines going through that point corresponds to a sinusoidal curve in the ($r,\theta$) plane, which is unique to that point. A set of two or more points that form a straight line will produce sinusoids which cross at the ($r,\theta$) for that line. Thus, the problem of detecting collinear points can be converted to the problem of finding concurrent curves.

## Implementation

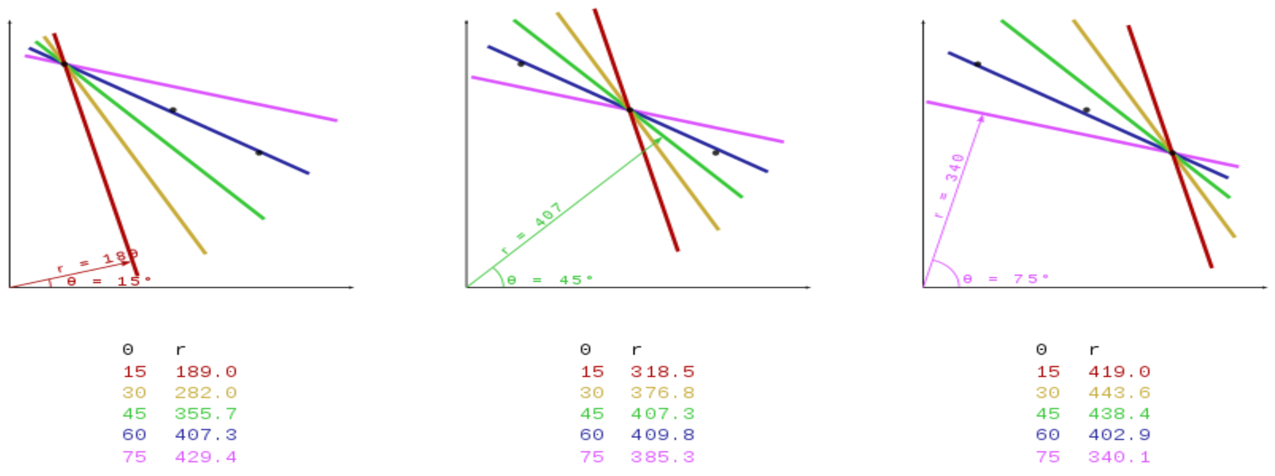
The linear Hough transform algorithm uses a two-dimensional array, called an accumulator, to detect the existence of a line described by The dimension of the accumulator equals the number of unknown parameters, i.e., two, considering quantized values of r and θ in the pair (r,θ). For each pixel at *(x,y)* and its neighborhood, the Hough transform algorithm determines if there is enough evidence of a straight line at that pixel. If so, it will calculate the parameters (r,θ) of that line, and then look for the accumulator's bin that the parameters fall into, and increment the value of that bin. By finding the bins with the highest values, typically by looking for local maxima in the accumulator space, the most likely lines can be extracted, and their (approximate) geometric definitions read off. (Shapiro and Stockman, 304) The simplest way of finding these *peaks* is by applying some form of threshold, but other techniques may yield better results in different circumstances – determining which lines are found as well as how many. Since the lines returned do not contain any length information, it is often necessary, in the next step, to find which parts of the image match up with which lines. Moreover, due to imperfection errors in the edge detection step, there will usually be errors in the accumulator space, which may make it non-trivial to find the appropriate peaks, and thus the appropriate lines.

The final result of the linear Hough transform is a two-dimensional array (matrix) similar to the accumulator—one dimension of this matrix is the quantized angle θ and the other dimension is the quantized distance r. Each element of the matrix has a value equal to the sum of the points or pixels that are positioned on the line represented by quantized parameters (r, θ). So the element with the highest value indicates the straight line that is most represented in the input image.

## Examples

### Example 1

Consider three data points, shown here as black dots.



| θ | r |
|---|---|
| 15 | 189.0 |
| 30 | 282.0 |
| 45 | 355.7 |
| 60 | 407.3 |
| 75 | 429.4 |

| θ | r |
|---|---|
| 15 | 318.5 |
| 30 | 376.8 |
| 45 | 407.3 |
| 60 | 409.8 |
| 75 | 385.3 |

| θ | r |
|---|---|
| 15 | 419.0 |
| 30 | 443.6 |
| 45 | 438.4 |
| 60 | 402.9 |
| 75 | 340.1 |

Fig-1.6

- For each data point, a number of lines are plotted going through it, all at different angles.
- These are shown here in different colours.

- To each line, a support line exists which is perpendicular to it and which intersects the origin. In each case, one of these is shown as an arrow.
- The length (i.e. perpendicular distance to the origin) and angle of each support line is calculated. Lengths and angles are tabulated below the diagrams.

From the calculations, it can be seen that in either case the support line at 60° has a similar length. Hence, it is understood that the corresponding lines (the blue ones in the above picture) are very similar. One can thus assume that all points lie close to the blue line.

Example 2

The following is a different example showing the results of a Hough transform on a raster image containing two thick lines.

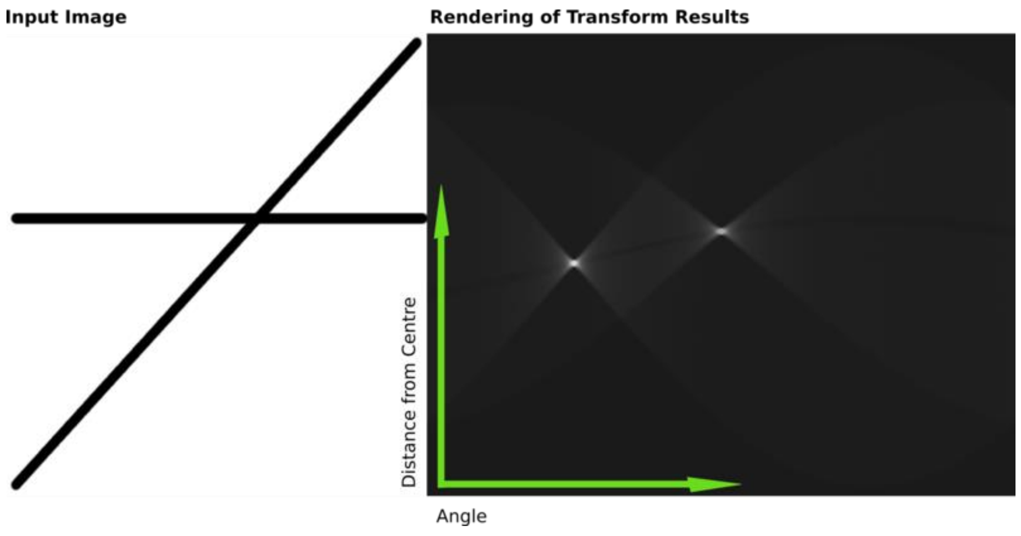


Fig- 1.7

The results of this transform were stored in a matrix. Cell value represents the number of curves through any point. Higher cell values are rendered brighter. The two distinctly bright spots are the Hough parameters of the two lines. From these spots' positions, angle and distance from image center of the two lines in the input image can be determined.

Various Extensions

An improvement suggested by O'Gorman and Clowes can be used to detect lines if one takes into account that the local gradient of the image intensity will necessarily be orthogonal to the edge. Since edge detection generally involves computing the intensity gradient magnitude, the gradient direction is often found as a side effect. If a given point of coordinates (*x,y*) happens to indeed be on a line, then the local direction of the gradient gives the $\theta$ parameter corresponding to said line, and the *r* parameter is then immediately obtained. (Shapiro and Stockman, 305) The gradient direction can be estimated to within 20°, which shortens the sinusoid trace from the full 180° to roughly 45°. This reduces the computation time and has the interesting effect of reducing the number of useless votes, thus enhancing the visibility of the spikes corresponding to real lines in the image.

## Kernel-based Hough transform (KHT)

Fernandes and Oliveira [9] suggested an improved voting scheme for the Hough transform that allows a software implementation to achieve real-time performance even on relatively

large images (e.g., 1280×960). The Kernel-based Hough transform uses the same parameterization proposed by Duda and Hart but operates on clusters of approximately collinear pixels. For each cluster, votes are cast using an oriented elliptical-Gaussian kernel that models the uncertainty associated with the best-fitting line with respect to the corresponding cluster. The approach not only significantly improves the performance of the voting scheme, but also produces a much cleaner accumulator and makes the transform more robust to the detection of spurious lines.

## 3-D Kernel-based Hough transform for plane detection (3DKHT)

Limberger and Oliveira suggested a deterministic technique for plane detection in unorganized point clouds whose cost is in the number of samples, achieving real-time performance for relatively large datasets (up to points on a 3.4 GHz CPU). It is based on a fast Hough-transform voting strategy for planar regions, inspired by the Kernel-based Hough transform (KHT). This 3D Kernel-based Hough transform (3DKHT) uses a fast and robust algorithm to segment clusters of approximately co-planar samples, and casts votes for individual clusters (instead of for individual samples) on a spherical accumulator using a trivariate Gaussian kernel. The approach is several orders of magnitude faster than existing (non-deterministic) techniques for plane detection in point clouds, such as RHT and RANSAC, and scales better with the size of the datasets. It can be used with any application that requires fast detection of planar features on large datasets.

## Hough transform of curves, and its generalization for analytical and non-analytical shapes[edit]

Although the version of the transform described above applies only to finding straight lines, a similar transform can be used for finding any shape which can be represented by a set of parameters. A circle, for instance, can be transformed into a set of three parameters, representing its center and radius, so that the Hough space becomes three dimensional. Arbitrary ellipses and curves can also be found this way, as can any shape easily expressed as a set of parameters.

The generalization of the Hough transform for detecting analytical shapes in spaces having any dimensionality was proposed by Fernandes and Oliveira.In contrast to other Hough transform-based approaches for analytical shapes, Fernandes' technique does not depend on the shape one wants to detect nor on the input data type. The detection can be driven to a type of analytical shape by changing the assumed model of geometry where data have been encoded

(e.g., euclidean space, projective space, conformal geometry, and so on), while the proposed formulation remains unchanged. Also, it guarantees that the intended shapes are represented with the smallest possible number of parameters, and it allows the concurrent detection of different kinds of shapes that best fit an input set of entries with different dimensionalities and different geometric definitions (e.g., the concurrent detection of planes and spheres that best fit a set of points, straight lines and circles).

For more complicated shapes in the plane (i.e., shapes that cannot be represented analytically in some 2D space), the Generalised Hough transform [12] is used, which allows a feature to vote for a particular position, orientation and/or scaling of the shape using a predefined look-up table.

Circle detection process

Altering the algorithm to detect circular shapes instead of lines is relatively straightforward.

- First, we create the accumulator space, which is made up of a cell for each pixel. Initially each cell is set to 0.
- For each edge point (i, j) in the image, increment all cells which according to the equation of a circle could be the center of a circle. These cells are represented by the letter in the equation.
- For each possible value of  found in the previous step, find all possible values of which satisfy the equation.
- Search for local maxima in the accumulator space. These cells represent circles that were detected by the algorithm.

If we do not know the radius of the circle we are trying to locate beforehand, we can use a three-dimensional accumulator space to search for circles with an arbitrary radius. Naturally, this is more computationally expensive.

This method can also detect circles that are partially outside of the accumulator space, as long as enough of the circle's area is still present within it.

Detection of 3D objects (Planes and cylinders)

Hough transform can also be used for the detection of 3D objects in range data or 3D point clouds. The extension of classical Hough transform for plane detection is quite straightforward. A plane is represented by its explicit equation  for which we can use a 3D Hough space corresponding to  and This extension suffers from the same problems as its 2D counterpart i.e., near horizontal planes can be reliably detected, while the performance deteriorates as planar direction becomes vertical (big values of  and amplify the noise in the data). This formulation of the plane has been used for the detection of planes in the point clouds acquired from airborne laser scanning [13] and works very well because in that domain all planes are nearly horizontal.

For generalized plane detection using Hough transform, the plane can be parameterized by its normal vector (using spherical coordinates) and its distance from the origin resulting in a three dimensional Hough space. This results in each point in the input data voting for a sinusoidal surface in the Hough space. The intersection of these sinusoidal surfaces indicates presence of a plane.[14] A more general approach for more than 3 dimensions requires search heuristics to remain feasible

Hough transform has also been used to find cylindrical objects in point clouds using a two step approach. The first step finds the orientation of the cylinder and the second step finds the position and radius.

Using weighted features

One common variation detail. That is, finding the bins with the highest count in one stage can be used to constrain the range of values searched in the next.

Carefully chosen parameter space

A high-dimensional parameter space for the Hough transform is not only slow, but if implemented without forethought can easily overrun the available memory. Even if the programming environment allows the allocation of an array larger than the available memory space through virtual memory, the number of page swaps required for this will be very demanding because the accumulator array is used in a randomly accessed fashion, rarely stopping in contiguous memory as it skips from index to index.

Consider the task of finding ellipses in an 800x600 image. Assuming that the radii of the ellipses are oriented along principal axes, the parameter space is four-dimensional. (x,y) defines the center of the ellipse, and a and b denote the two radii. Allowing the center to be anywhere in the image, adds the constraint 0<x<800 and 0<y<600. If the radii are given the same values as constraints, what is left is a sparsely filled accumulator array of more than 230 billion values.

A program thus conceived is unlikely to be allowed to allocate sufficient memory. This doesn't mean that the problem can't be solved, but only that new ways to constrain the size of the accumulator array are to be found, which makes it feasible. For instance:

1. If it is reasonable to assume that the ellipses are each contained entirely within the image, the range of the radii can be reduced. The largest the radii can be is if the center of the ellipse is in the center of the image, allowing the edges of the ellipse to stretch to the edges. In this extreme case, the radii can only each be half the magnitude of the image size oriented in the same direction. Reducing the range of a and b in this fashion reduces the accumulator array to 57 billion values.
2. Trade accuracy for space in the estimation of the center: If the center is predicted to be off by 3 on both the x and y axis this reduces the size of the accumulator array to about 6 billion values.
3. Trade accuracy for space in the estimation of the radii: If the radii are estimated to each be off by 5 further reduction of the size of the accumulator array occurs, by about 256 million values.
4. Crop the image to areas of interest. This is image dependent, and therefore unpredictable, but imagine a case where all of the edges of interest in an image are in the upper left quadrant of that image. The accumulator array can be reduced even further in this case by constraining all 4 parameters by a factor of 2, for a total reduction factor of 16.

By applying just the first three of these constraints to the example stated about, the size of the accumulator array is reduced by almost a factor of 1000, bringing it down to a size that is much more likely to fit within a modern computer's memory.

In the Cartesian coordinate system, we can represent a straight line as y = mx + b by plotting y against x. However, we can also represent this line as a single point in Hough space by plotting b against m. For example, a line with the equation y = 2x + 1 may be represented as (2, 1) in Hough space.

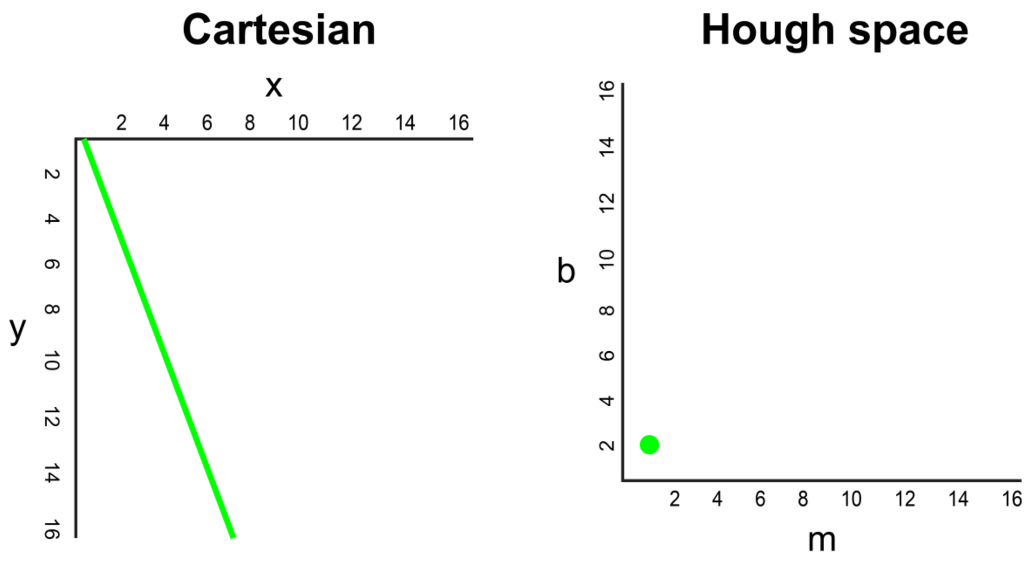


Fig-1.8

Now, what if instead of a line, we had to plot a point in the Cartesian coordinate system. There are many possible lines which can pass through this point, each line with different values for parameters m and b. For example, a point at (2, 12) can be passed by y = 2x + 8, y = 3x + 6, y = 4x + 4, y = 5x + 2, y = 6x, and so on. These possible lines can be plotted in Hough space as (2, 8), (3, 6), (4, 4), (5, 2), (6, 0). Notice that this produces a line of m against b coordinates in Hough space.

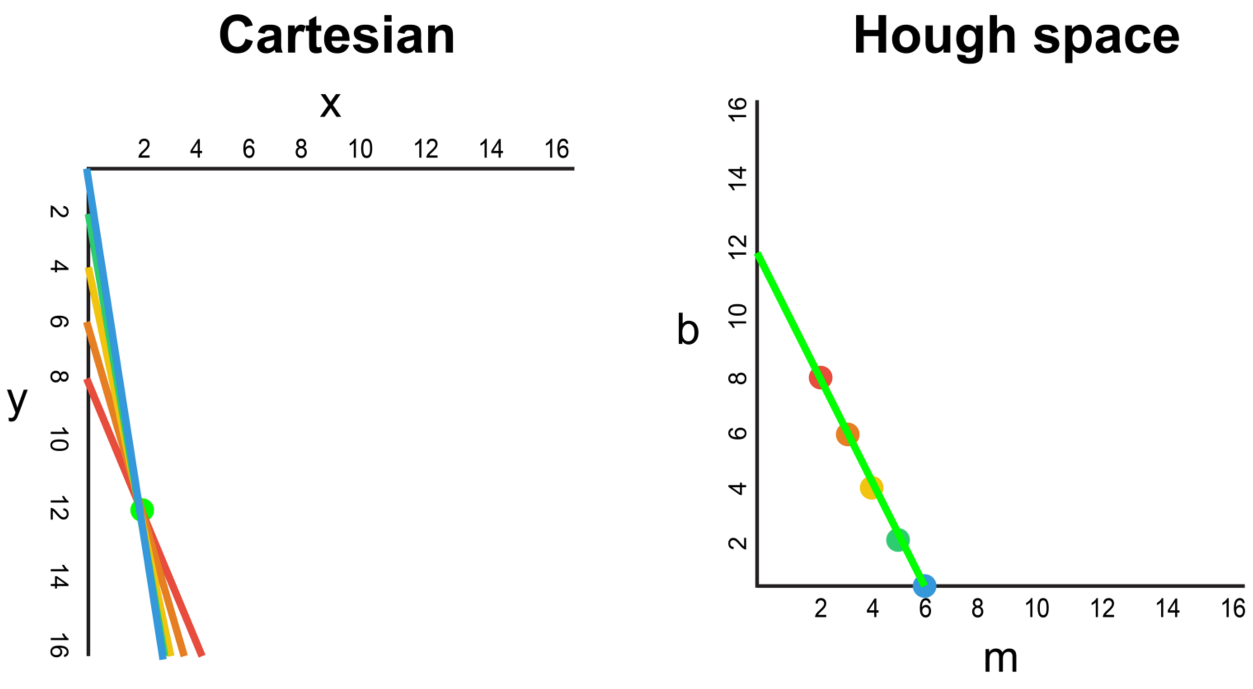


Fig.-1.9

Whenever we see a series of points in a Cartesian coordinate system and know that these points are connected by some line, we can find the equation of that line by first plotting each point in the Cartesian coordinate system to the corresponding line in Hough space, then finding the point of intersection in Hough space. The point of intersection in Hough space represents the m and b values that pass consistently through all of the points in the series.

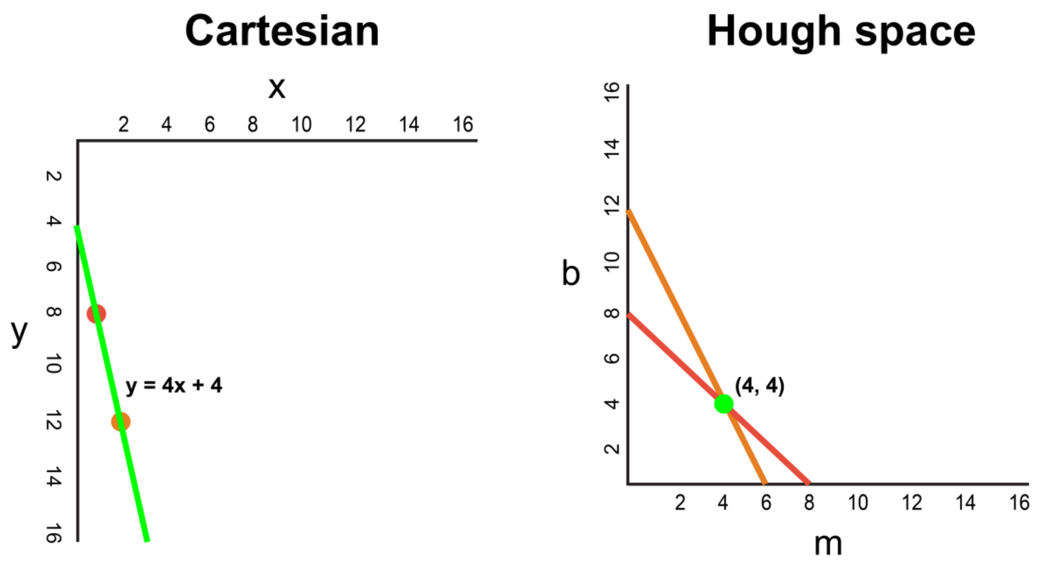


Fig-1.10

Since our frame passed through the Canny Detector may be interpreted simply as a series of white points representing the edges in our image space, we can apply the same technique to identify which of these points are connected to the same line, and if they are connected, what its equation is so that we can plot this line on our frame.

For the simplicity of explanation, we used Cartesian coordinates to correspond to Hough space. However, there is one mathematical flaw with this approach: When the line is vertical, the gradient is infinity and cannot be represented in Hough space. To solve this problem, we will use Polar coordinates instead. The process is still the same just that other than plotting m against b in Hough space, we will be plotting r against θ.

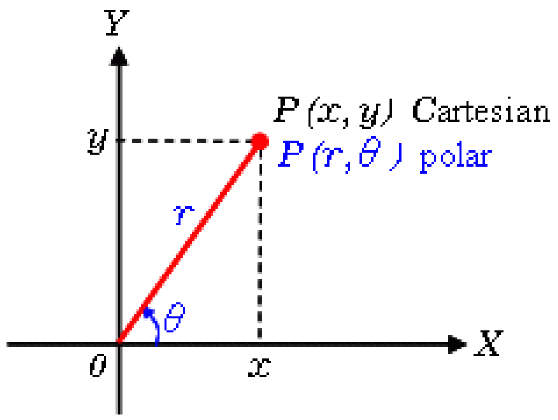


Fig- 1.11

1.2.15 Visualization

The lane is visualized as two light green, linearly fitted polynomials which will be overlayed on our input frame.

Lane detection is a complicated problem under different light/weather conditions. In this project we analysis the easy case first: the images are captured from the crossover above the road, assume the lanes to be detected are straight, at daytime and with good weather condition. The lane markings can be solid or dash lines. Other than detecting the lane markers, the mid-line of each lane is also calculated to identify the position of the vehicle with respect to lane makings, which is useful for autonomous driving.

## 1.3 Scope

The lane detection problem, at least in its basic setting, does not look like a hard one. In this basic setting, one has to detect only the host lane, and only for a short distance ahead. A relatively simple Hough transform-based algorithm, which does not employ any tracking or image-to-world reasoning, solves the problem in roughly 90% of the highway cases. In spite of that, the impression that the problem is easy is misleading, and building a useful system is a large-scale R&D effort. The main reasons for that are significant gaps in research, high reliability demands, and large diversity in case conditions.

The reliable intelligent driver assistance systems and safety warning systems is still a long way to go. However, as computing power, sensing capacity, and wireless connectivity for vehicles rapidly increase, the concept of assisted driving and proactive safety warning is speeding towards reality. As technology improves, a vehicle will become just a computer with tires. Driving on roads will be just like surfing the Web: there will be traffic congestion but no injuries or fatalities. Advanced driver assistant systems and new sensing technologies can be highly beneficial, along with large body of work on automated vehicles.

## 1.4 Background of Project

The goal of this project is to build up a simple image pipeline (take a frame from video as an input, do something, return a modified version of the frame), which allows detecting lane lines in simple conditions: sunny weather, good visibility, no cars in sight, only straight lanes.

## 1.5 Operation Environment

The Lane Detector is a project and shall operate in all systems, for a particular system we are taking a Windows 10 laptop with Python Version 3.7.4, OpenCV and its packages such as Matplotlib, Pillow, Numpy, PIP.  For text editor we are using Jetbrains by Pycharm community edition 2019.2.1.

## 1.5 Report Structure

The project Real-time Object detection and Recognition is primarily concerned with the Image processing in real-time and whole project report is categorized into five chapters.

Chapter 1: Introduction- introduces the background of the problem followed by rationale for the project undertaken. The chapter describes the objectives, scope and applications of the project. Further, thechapter gives the details of team members and their contribution in development of project which is then subsequently ended with reportoutline.

Chapter 2: Review of Literature- explores the work done in the area of Project undertaken and discusses the limitations of existing system and highlights the issues and challenges of project area. The chapter finally ends up with the requirement identification for present project work based on findings drawn from reviewed literature and end user interactions.

Chapter 3: Proposed System - starts with the project proposal based on requirement identified, followed by benefits of the project. The chapter also illustrate software engineering paradigm used along with different design representation. The chapter also includes block diagram and details of major modules of the project. Chapter also gives insights of different type of feasibility study carried out for the project undertaken. Later it gives details of the different deployment requirements for the developed project.

Chapter 4: Implementation - includes the details of different Technology/ Techniques/ Tools/ Programming Languages used in developing the Project. The chapter also includes the different user interface designed in project along with their functionality. Further it discusses the experiment results along with testing of the project. The chapter ends with evaluation of project on different parameters like accuracy and efficiency.

Chapter 5: Conclusion - Concludes with objective wise analysis of results and limitation of present work which is then followed by suggestions and recommendations for further improvement.

# Chapter 2: Review of Literature

## 2.1 Study of Existing Techniques

### 2.1.1 Results using Hough Transform

Mariut et al in his research paper proposed a simple algorithm that detected the lane marks, lane mark's characteristics and had the ability to determine the travelling direction. It used the well known Hough transform to detect the potential lines in the images. To ensure the right detection of the lane mark, they had developed a technique that extracts the inner margin of the lane. The margins are highlighted by generating the magnitude image.

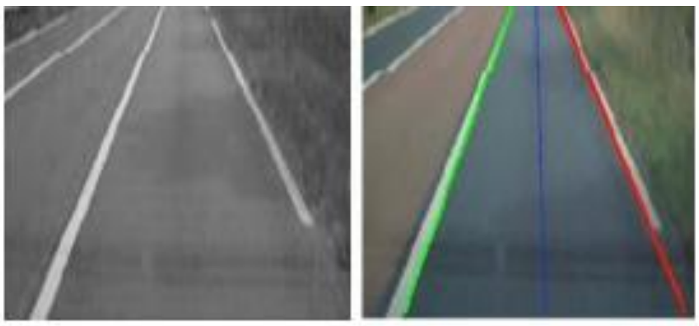


**Figure a) Input Image b) Detected Lanes[1]**

Fig-2.1

### 2.1.2 Results using Hough Transformation and Filters

T.T Tran et al in his research paper proposed an adaptive method based on HSI color model to detect lane marking. First, they converted RGB-based image to its HSI-based image. However, HSI color model was improved by the change in the way to calculate the intensity (I) component from RGB color images. From observing the color images of the road scene in HIS color space, they utilized the limited range of color. Hence, H, S and I component were used in this method. The proposed method can label the location of lane marking accurately.

**Figure : a)Input Image b) Detected Lanes [2]**

Fig-2.2

### 2.1.3 Results of Lane Detection based on HSI model

S. Srivastava et al in his research paper proposed an efficient ways of noise reduction in the images by using different filtering techniques in this paper. The main objective was to design, develop, implement and subsequently simulate an efficient lane detection algorithm which will provide high quality results in the case when noise is present in the signal. Various filters used for comparison were median, wiener, and hybrid median filters.

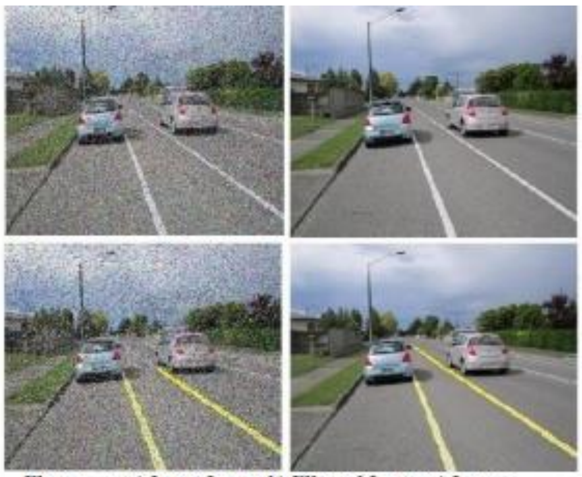


Figure : a) Input Image b) Filtered Image c) Image without Filter d) Output Image [20]

Fig-2.3

## 2.2Requirement Analysis

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users. It is an early stage in the more general activity of requirements engineering which encompasses all activities concerned with eliciting, analyzing, documenting, validating and managing software or system requirements.

Requirements analysis is critical to the success of a systems or software project. The Requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

## 2.3 Problem Definition

The problem of detecting straight lines in gray-scale images as an inverse issue. Our formulation is based on use of the ROI, which relates the parameters determining the location and orientation of the lines in the image to the noisy input image. The advantage of this formulation is that we can then approach the problem of line detection within a regularization framework and enhance the performance of the Hough-based line detector through the incorporation of prior information in the form of regularization. We discuss the type of regularizes that are useful for this problem and derive efficient computational schemes to solve the resulting optimization problems enabling their use in large applications.

Finally, we show how our new approach can be alternatively viewed as one of finding an optimal representation of the noisy image in terms of elements chosen from a dictionary of lines. This interpretation relates the problem of Hough-based line finding to the body of work on adaptive signal representation

## .2.4 Requirement Specification

- The lane lines are mapped from the roads which are a result of determination of pixels which would be a part of the right and left lines of the lane.
- A way which would help to reduce traffic problems as lesser changing of lanes would take place.
- A method which would combine for the concept of self-driving cars.

# Chapter 3: Proposed System

## 3.1 Proposal

Our main contribution in this project is to do a lot of work in the preprocessing stage. We proposed to perform color transform of HSV in the preprocessing stage, then extract white, and then perform conventional preprocessing operations in sequence. Moreover, we selected an improved method proposed in the area of interest (ROI). In this project, based on the proposed preprocessing method (after HSV color transform, white feature extraction, and basic preprocessing), one-half part of the processed image is selected as the area of interest (ROI). In addition, we performed twice edge detection. The first is in the preprocessing stage, and the second is in the lane detection stage after the ROI is selected. The purpose of performing twice edge detection is to enhance the lane recognition rate.

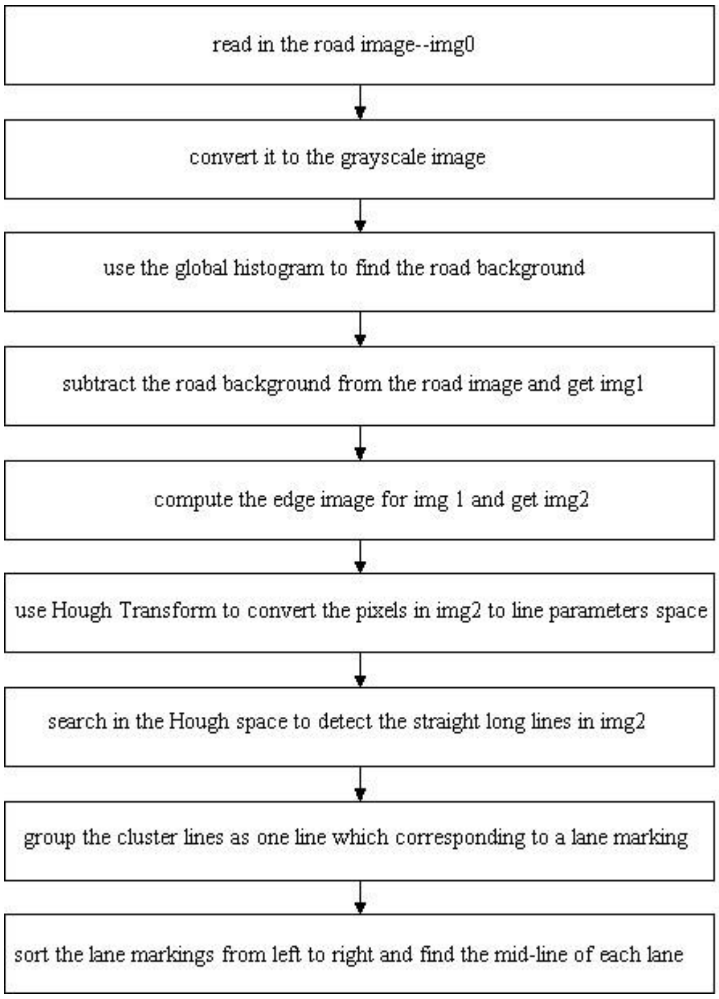

read in the road image--img0
convert it to the grayscale image
use the global histogram to find the road background
subtract the road background from the road image and get img1
compute the edge image for img 1 and get img2
use Hough Transform to convert the pixels in img2 to line parameters space
search in the Hough space to detect the straight long lines in img2
group the cluster lines as one line which corresponding to a lane marking
sort the lane markings from left to right and find the mid-line of each lane

Fig -3.1 The lane detection algorithm

## 3.2 Feasibility

Feasibility study is the process of determination of whether or not a project is worth doing. Feasibility studies are undertaken within tight time constraints and normally culminate in a written and oral feasibility report. The contents and recommendations of this feasibility study helped us as a sound basis for deciding how to precede the project. It helped in taking decisions such as which software to use, hardware combinations, etc.

### 3.2.1Technical feasibility

This assessment is based on an outline design of system requirements, to determine whether the company has the technical expertise to handle completion of the project.[6][7][8] When writing a feasibility report, the following should be taken to consideration:

- A brief description of the business to assess more possible factors which could affect the study
- The part of the business being examined
- The human and economic factor
- The possible solutions to the problem

At this level, the concern is whether the proposal is both *technically* and *legally* feasible (assuming moderate cost)

The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

Method of production

The selection among a number of methods to produce the same commodity should be undertaken first. Factors that make one method being preferred to other method in agricultural projects are the following:

- Availability of inputs or raw materials and their quality and prices.
- Availability of markets for outputs of each method and the expected prices for these outputs.
- Various efficiency factors such as the expected increase in one additional unit of fertilizer or productivity of a specified crop per one thing.

Production technique

After we determine the appropriate method of production of a commodity, it is necessary to look for the optimal technique to produce this commodity.

Project requirements

Once the method of production and its technique are determined, technical people have to determine the projects' requirements during the investment and operating periods. These include:

- Determination of tools and equipment needed for the project such as drinkers and feeders or pumps or pipes …etc.
- Determination of projects' requirements of constructions such as buildings, storage, and roads …etc. in addition to internal designs for these requirements.
- Determination of projects' requirements of skilled and unskilled labor and managerial and financial labor.
- Determination of construction period concerning the costs of designs and consultations and the costs of constructions and other tools.
- Determination of minimum storage of inputs, cash money to cope with operating and contingency costs.

Project location

The most important factors that determine the selection of project location are the following:

- Availability of land (proper acreage and reasonable costs).
- The impact of the project on the environment and the approval of the concerned institutions for license.
- The costs of transporting inputs and outputs to the project's location (i.e., the distance from the markets).
- Availability of various services related to the project such as availability of extension services or veterinary or water or electricity or good roads ...etc.

Legal feasibility

It determines whether the proposed system conflicts with legal requirements, e.g., a data processing system must comply with the local data protection regulations and if the proposed venture is acceptable in accordance to the laws of the land.

Operational feasibility study

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.[10]

The operational feasibility assessment focuses on the degree to which the proposed development project fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes.

To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, reducibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviors' are to be realized. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

Time feasibility

A time feasibility study will take into account the period in which the project is going to take up to its completion. A project will fail if it takes too long to be completed before it is useful. Typically this means estimating how long the system will take to develop, and if it can be completed in a given time period using some methods like payback period. Time feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable? Some projects are initiated with specific deadlines. It is necessary to determine whether the deadlines are mandatory or desirable.

.

### 3.2.2 Operational Feasibility

Operation feasibility is a measure of how people feel about the system.

Operational Feasibility criteria measure the urgency of the problem or the acceptability of a solution. Operational Feasibility is dependent upon determining human resources for the project. It refers to projecting whether the system will operate and be used once it is installed.

The essential questions that help in testing the operational feasibility of a system are following.

Does management support the project?

Are the users not happy with current business practices? Will it reduce the time (operation) considerably? If yes, then they will welcome the change and the new system.

Have the users been involved in the planning and development of the project? Early involvement reduces the probability of resistance towards the new system.

Our proposed project “Line Detector” is operationally feasible since there is no need for special training of staff member and whatever little instructing on this system is required can be done so quite easily and quickly as it is essentially This project is being developed keeping in mind the general people who one have very little knowledge of computer operation, but can easily access their required database and other related information. The redundancies can be decreased to a large extent as the system will be fully automated.

Technical Feasibility

Technical feasibility determines whether the work for the project can be done with the existing equipment, software technology and available personnel. Technical feasibility is concerned with specifying equipment and software that will satisfy the user requirement.

- In technical feasibility the following issues are taken into consideration.
- Whether the required technology is available ornot
- Whether the required resources are available
- Manpower- programmers, testers &debuggers
- Software and hardware

### 3.2.3 Economic Feasibility

Economic feasibility has great importance as it can outweigh other feasibilities because costs affect organization decisions. The concept of Economic Feasibility deals with the fact that a system that can be developed and will be used on installation must be profitable for the Organization. The cost to conduct a full system investigation, the cost of hardware and software, the benefits in the form of reduced expenditure are all discussed during the economic feasibility.

During the economic feasibility test we maintained the balance between the Operational and Economical feasibilities, as the two were the conflicting. For example, the solution that provides the best operational impact for the end-users may also be the most expensive and, therefore, the least economically feasible.

We classified the costs of our proposed project “Lane Detector” according to the phase in which they occur. As we know that the system development costs are usually one-time costs that will not recur after the project has been completed. For calculating the Development costs, we evaluated certain cost categories.

- Personnel costs
- Computer usage costs
- Training costs
- Supply and equipment’s costs
- Cost of any new computer equipment and software.

## 3.4 Data Flow Diagram

Flow Diagram

In the flow diagram, we describe our proposed method of detecting lanes, by extracting small line like structures of the lane using ROI gradient operator and canny edge detection, probabilistic Hough Transform for line segment detection, and then clustering the line segments.

We used clustering method to group the line segments with their slope and Y-intercept in two different stages:

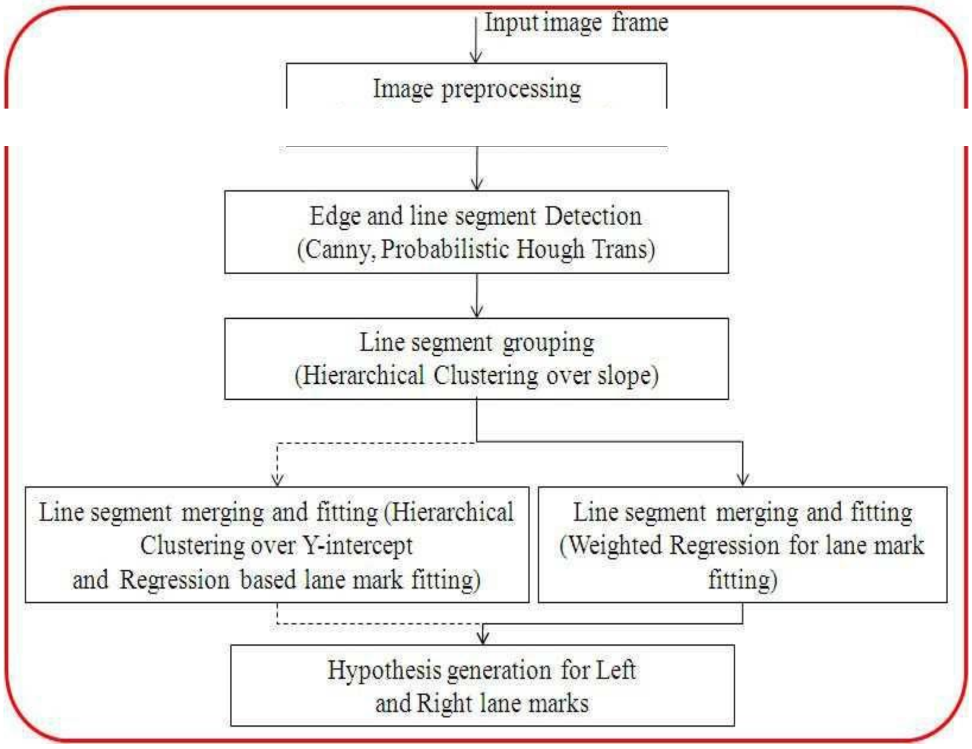
(i) clustering with slopes of line segments as input and

(ii) clustering again with Y-intercepts of the grouped lines from previous step. This is mainly to get distinct distance between line segments with respect to their angle and positions separately, as they are values of different scales (which may hamper the distance computation for proper clustering). Secondly, this two-stage separation helps us to compute weight according to Y-intercepts for each line in their group and apply heteroscedastic regression to fit lane marks (line or curve fitting).

Most of the works in the literature, merging of the line segments is done by simple threshold or clustering on similar slope and position followed by curve fit-ting. In this work, we combine the line grouping and curve fitting in a single step in order to improve computational efficiency.

This is carried out by the use of weighted least square regression. Figure 10 shows the flow of the proposed solution. The dotted line in this diagram shows the two-stage clustering process, while the solid line shows the proposed method of weighted regression based on Y-intercepts.

The stepwise output for an image frame is shown:



Input image frame
Image preprocessing
Edge and line segment Detection (Canny, Probabilistic Hough Trans)
Line segment grouping (Hierarchical Clustering over slope)
Line segment merging and fitting (Hierarchical Clustering over Y-intercept and Regression based lane mark fitting)
Line segment merging and fitting (Weighted Regression for lane mark fitting)
Hypothesis generation for Left and Right lane marks

Fig -3.2

## 3.5 Block Diagram

In the view of above challenges and industrial demand, we developed a feature driven methodology for real-time lane detection for automotive vehicles. The block diagram in figure 1 explains the system overview of the method for Lane detection. The basic input is a video file taken from camera mounted on a moving vehicle. The flow of the method is depicted in this diagram. The major components in this approach in-cludes video capture and frame extraction, line/curve like mark detection, spurious segment removal, line merging and fitting curves. A brief description of these components is given below:

- Input frame extraction:From the video out-put of color video camera, the sequence of frames has been extracted and fed to the system for fur-ther processing. Here we divide the image frame into two parts: near view and farther view. In near view the lane marks are normally line like structure and in farther view the marks are mostly curve like. So we divide the image into two parts and process them separately.
- Line detection: In general, lane marks are line and curve like structures. The disconnected lane marks are identified with line detection algorithm. Though, line detection method is able to detect small line like segments in the lane curve, they are not always continuous in space and slope. Hence we further process these line segments and merge them for continuous lane mark detection.
- Spurious line removal and line merging: The spurious lines are removed and merging of lines is done by grouping them according to their similarity in slope and spatial closeness. The spu-rious lines are removed, based on hypotheses as follows:
  (i) The lane markings should be some-what parallel to each other and they should merge at vanishing Point.
  (ii) Starting point of the left and right lane mark should be in 3rd and 4th quadrant respectively for a given image divided into four quadrants.
  (iii) The slope of the line in the near sight should be approximately vertical and continuous in the latter part.

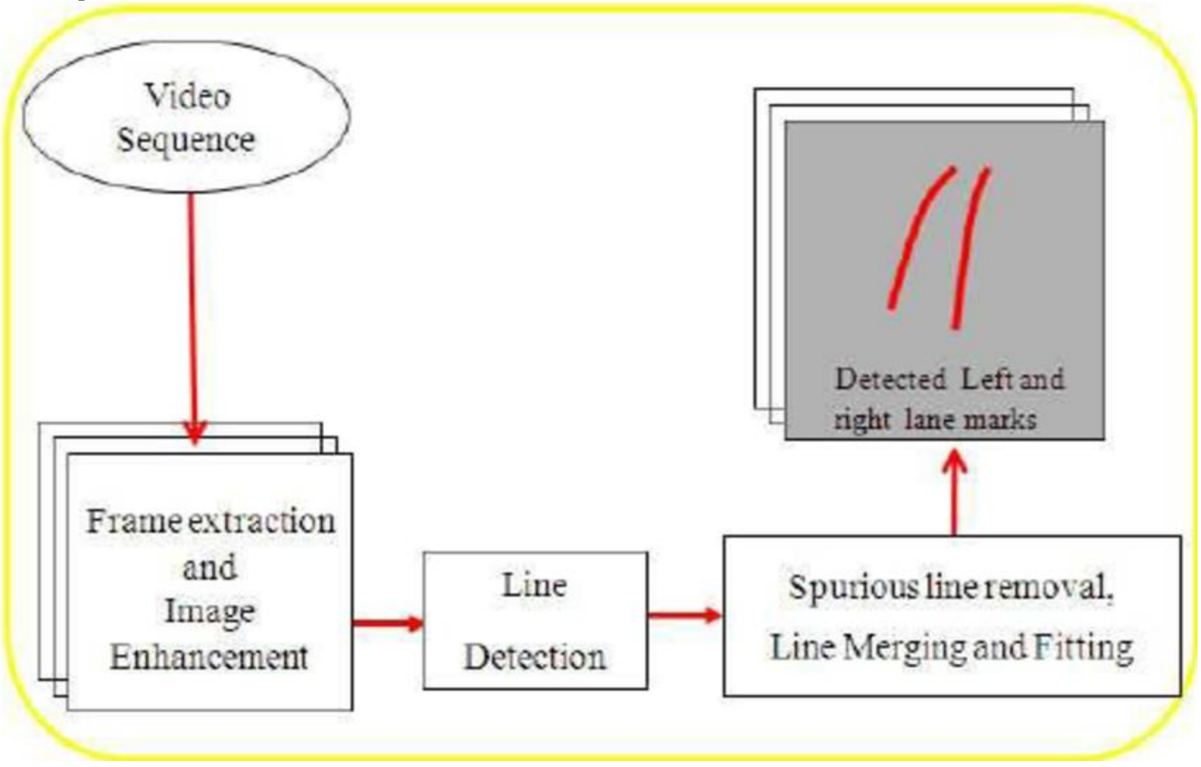

Video Sequence
Frame extraction and Image Enhancement
Line Detection
Spurious line removal, Line Merging and Fitting
Detected Left and right lane marks

Fig-3.3

## 3.7 Deployment Requirement

Functional Requirements

- The lane lines are mapped from the roads which are a result of determination of pixels which would be a part of the right and left lines of the lane.
- A way which would help to reduce traffic problems as lesser changing of lanes would take place.
- A method which would combine for the concept of self-driving cars.

Non-Functional Requirements

- Reliability:

The capability to maintain the specified level of performance is called reliability. This application is a web based application that runs on any device that has a browser.

- Availability:

The project will be available for 99% of the time.

- Security:

The business logic is hidden from the users and is much safer and thus avoids unauthorised or illegal access or database corruption. Security of the user's information is also safe as there is a login facility.

- Maintainability:

Maintenance is typically done after the software development has been completed. As the time evolves, so do the requirements and needs. It revolves around the understanding of the existing software and the effects of the change.

- Portability:

Portability is the ability of the system or application that can run in various environments. As the web application is based on the java language, the application is portable.

## 3.8 Software and Hardware Requirements

Software Requirements

- Jetbrains by Pycharm community edition 2019.2.1
- Python 3.7.4
- OpenCV
- Important packages such as Matplotlib
- Pillow
- Pip
- Numpy
- Opencv-python
- PyBundle
- Kiwisolver and many more library packages.

Hardware Requirements

- Intel Core i3 or above
- Windows 10
- RAM 4 GB or above
- Memory 500 GB or above

# Chapter 4: Implementation

## 4.1 Software Tools

### Python

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter`, `map`, and `reduce` functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers.

## OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

## Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

## Numpy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

## 4.2 Screenshots

## Screenshot 1

Detection of long thick lines, such as highway lane markings from input images, is usually performed by local edge extraction followed by straight line approximation. In this conventional method many edge elements other than lane makings are detected when the threshold of edge magnitude is low, or, in the opposite case, edge elements expected to be detected are fragmented when it is high. This makes it difficult to trace the edge elements and to fit the approximation lines on them.

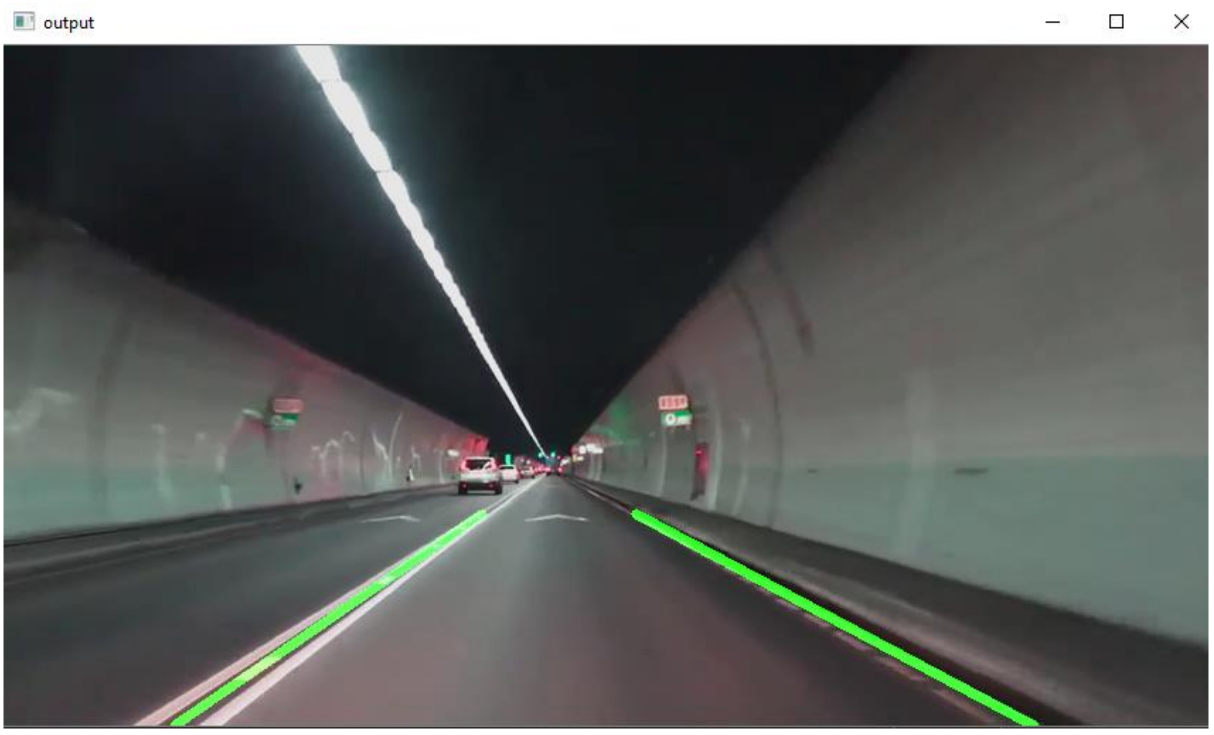


Fig-4.1

### Screenshot 2

The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection *explaining* why the technique works. (Wikipedia)

The Canny edge detection algorithm is composed of 5 steps:

1. Noise reduction;
2. Gradient calculation;

3. Non-maximum suppression;
4. Double threshold;
5. Edge Tracking by Hysteresis.

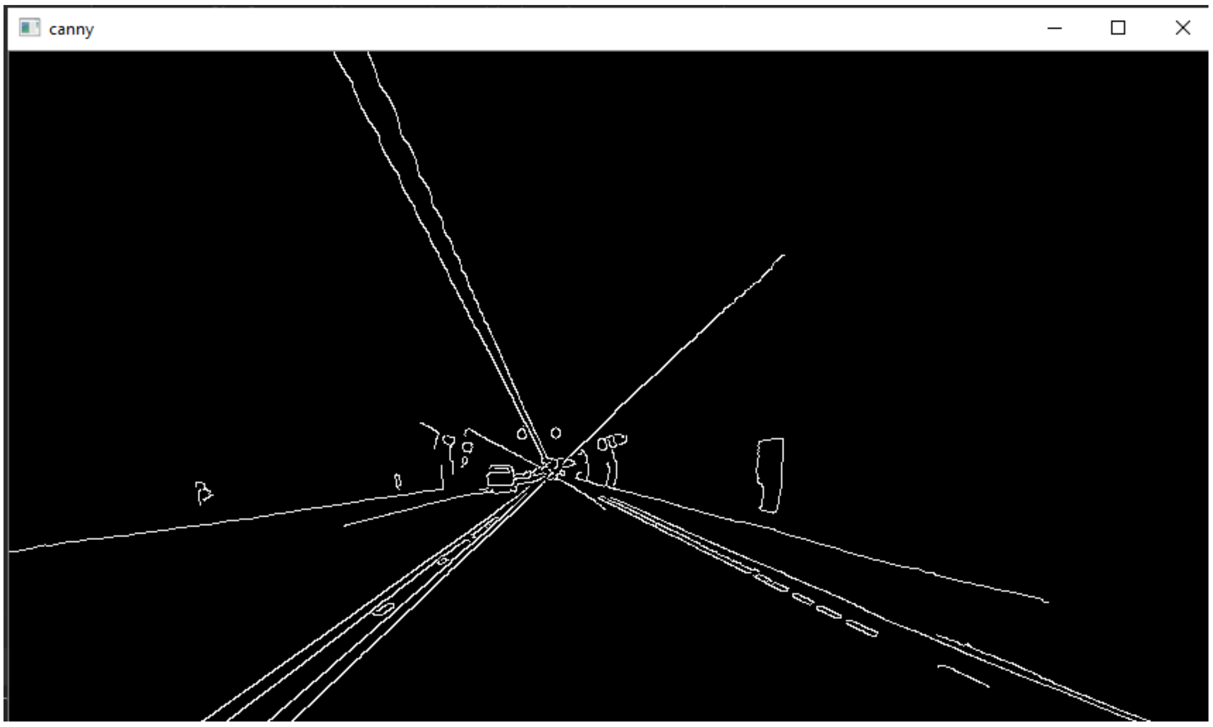


Fig-4.2

**Screenshot 3**

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the *classical* Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, *etc.* A *generalized* Hough transform can be employed in applications where a simple analytic description of a feature(s) is not possible. Due to the computational complexity of the generalized Hough algorithm, we restrict the main focus of this discussion to the classical Hough transform. Despite its domain restrictions, the classical Hough transform (hereafter referred to without the *classical* prefix) retains many applications, as most manufactured parts (and many anatomical parts investigated in medical imagery) contain feature boundaries which can be described by regular curves. The main advantage of the Hough transform technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise.

The Hough technique is particularly useful for computing a global description of a feature(s) (where the number of solution classes need not be known *a priori*), given (possibly noisy) local measurements. The motivating idea behind the Hough technique for line detection is that each input measurement (*e.g.* coordinate point) indicates its contribution to a globally consistent solution (*e.g.* the physical line which gave rise to that image point).

As a simple example, consider the common problem of fitting a set of line segments to a set of discrete image points (*e.g.* pixel locations output from an edge detector). Figure 1 shows some possible solutions to this problem. Here the lack of *a priori* knowledge about the number of desired line segments (and the ambiguity about what constitutes a line segment) render this problem under-constrained.

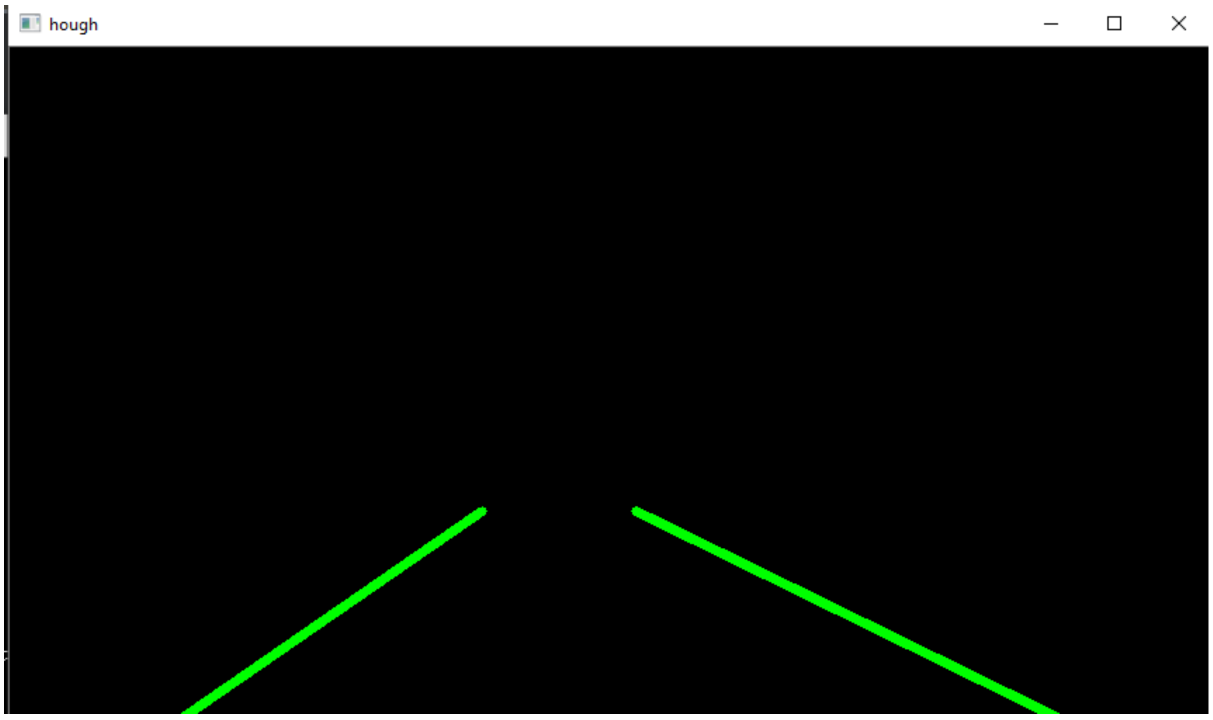


Fig -4.3

# Chapter 5: Conclusion

## 5.1 Conclusion

In this project, we proposed a new lane detection preprocessing and ROI selection methods to design a lane detection system. The main idea is to add white extraction before the conventional basic preprocessing. Edge extraction has also been added during the preprocessing stage to improve lane detection accuracy. We also placed the ROI selection after the proposed preprocessing. Compared with selecting the ROI in the original image, it reduced the nonlane parameters and improved the accuracy of lane detection. Currently, we only use the Hough transform to detect straight lane and track lane and do not develop advanced lane detection methods. In the future, we will exploit a more advanced lane detection approach to improve the performance.

## 5.2 Limitation

The Hough transform is only efficient if a high number of votes fall in the right bin, so that the bin can be easily detected amid the background noise. This means that the bin must not be too small, or else some votes will fall in the neighboring bins, thus reducing the visibility of the main bin.
Also, when the number of parameters is large (that is, when we are using the Hough transform with typically more than three parameters), the average number of votes cast in a single bin is very low, and those bins corresponding to a real figure in the image do not necessarily appear to have a much higher number of votes than their neighbors. The complexity increases at a rate of with each additional parameter, where the size of the image is space and is the number of parameters. (Shapiro and Stockman, 310) Thus, the Hough transform must be used with great care to detect anything other than lines or circles.

Finally, much of the efficiency of the Hough transform is dependent on the quality of the input data: the edges must be detected well for the Hough transform to be efficient. Use of the Hough transform on noisy images is a very delicate matter and generally, a denoising stage must be used before.

A recorder system to check the utility of the software on road and along with physical conditions is unavailable.

A running concept of research which is not accepted widely in the current world scenario.

## 5.3 Suggestions for Future

- Instead of line, it would be beneficial to use higher degree curve that will be useful on curved road.

- Even when we used information from previous frames, just averaging lanes might not be very good strategy. may be weight average or some form of priority value might work.

# Bibliography

[1]. D. Pomerleau, “RALPH: rapidly adapting lateral position handler,” in Proceedings of the Intelligent Vehicles '95. Symposium, pp. 506–511, Detroit, MI, USA, 2003.

[2]. J. W. Lee, C. D. Kee, and U. K. Yi, “A new approach for lane departure identification,” in Proceedings of the IEEE IV2003 Intelligent Vehicles Symposium, pp. 100–105, 2003.

[3]. J. W. Lee and U. K. Yi, “A lane-departure identification based on LBPE, Hough transform, and linear regression,” Computer Vision and Image Understanding, vol. 99, no. 3, pp. 359–383, 2005.

[4]. Zu Whan Kim “Robust Lane Detection and Tracking in Challenging Scenarios” March 2008.

[5]. C. J. Chen, B. Wu, W. H. Lin, C. C. Kao, and Y. H. Chen, “Mobile lane departure warning system in,” in Proceedings of the 2009 IEEE 13th International Symposium on Consumer Electronics, pp. 1–5, 2009.

[6]. H. Xu and H. Li, “Study on a robust approach of lane departure warning algorithm,” in Proceedings of the EEE International Conference on Signal Processing System (ICSPS), pp. 201–204, 2010.

[7]. J.-G. Wang, C.-J. Lin, and S.-M. Chen, “Applying fuzzy method to vision-based lane detection and departure warning system,” Expert Systems with Applications, vol. 37, no. 1, pp. 113–126, 2010.

[8]. Q. Lin, Y. Han, and H. Hahn, “Real-Time Lane Departure Detection Based on Extended Edge-Linking Algorithm,” in Proceedings of the 2010 Second International Conference on Computer Research and Development, pp. 725–730, Kuala Lumpur, Malaysia, May 2010.

[9]. H. Xu and H. Li, “Study on a robust approach of lane departure warning algorithm,” in Proceedings of the IEEE International Conference on Signal Processing System (ICSPS), pp. 201–204, 2010.

[10]. H. Aung and M. H. Zaw, “Video based lane departure warning system using hough transform,” in Proceedings of the International Conference on Advances in Engineering and Technology, pp. 85–88, Singapore, 2010.

[11]. Jiang Ryui “Lane detection and tracking Using a New Lane Model and Distance Transform”, January 2011.

[12]. Zaier Zaidi, “Evaluating Variable Speed Limits and Dynamic Lane Merging Systems in Work Zones: A Simulation Study” May 2012.

[13]. Ahron Bal Hilel, Ronnen Lener “Recent Progress in Road and Lane Detection: A Survey.”, December 2011.

[14]. Ni Wei and Ma Wanjing “Simulation Based Study on a Study Lane Assignment Approach for Freeway Weaving Section.” 2013.

[15]. S. Srivastava, M. Lumb, and R. Singal, “Improved lane detection using hybrid median filter and modified hough transform,” International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 1, pp. 30–37, 2014.

[16]. S.-C. Huang and B.-H. Chen, “Automatic moving object extraction through a real-world variable-bandwidth network for traffic monitoring systems,” IEEE Transactions on Industrial Electronics, vol. 61, no. 4, pp. 2099–2112, 2014.

[17]. F. Yuan, Z. Fang, S. Wu, Y. Yang, and Y. Fang, “Real-time image smoke detection using staircase searching-based dual threshold AdaBoost and dynamic analysis,” IET Image Processing, vol. 9, no. 10, pp. 849–856, 2015.

[18].P. N. Bhujbal and S. P. Narote, “Lane departure warning system based on Hough transform and Euclidean distance,” in Proceedings of the 3rd International Conference on Image Information Processing, ICIIP 2015, pp. 370–373, India, December 2015. V. Gaikwad and S. Lokhande, “Lane Departure Identification for Advanced Driver Assistance,” IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 2, pp. 910–918, 2015.

[19].Dinesh Kumar and Gurveen Kaur “Lane Detection Techniques: A Review.” Feb 2015

[20].J. Niu, J. Lu, M. Xu, P. Lv, and X. Zhao, “Robust Lane Detection using Two-stage Feature Extraction with Curve Fitting,” Pattern Recognition, vol. 59, pp. 225–233, 2015.

[21].M.-C. Chuang, J.-N. Hwang, and K. Williams, “A feature learning and object recognition framework for underwater fish images,” IEEE Transactions on Image Processing, vol. 25, no. 4, pp. 1862–1872, 2016.

[22].Y. Saito, M. Itoh, and T. Inagaki, “Driver Assistance System with a Dual Control Scheme: Effectiveness of Identifying Driver Drowsiness and Preventing Lane Departure Accidents,” IEEE Transactions on Human-Machine Systems, vol. 46, no. 5, pp. 660–671, 2016.

[23].A. Mammeri, A. Boukerche, and Z. Tang, “A real-time lane marking localization, tracking and communication system,” Computer Communications, vol. 73, pp. 132–143, 2016.

[24].J. Navarro, J. Deniel, E. Yousfi, C. Jallais, M. Bueno, and A. Fort, “Influence of lane departure warnings onset and reliability on car drivers' behaviors,” Applied Ergonomics, vol. 59, pp. 123–131, 2017.

[25].C.Y Kuo “On the Image Sesor Processing for Lane Detection and Control in Vehicle Lane Keeping Systems.” April 2019

[26].Aditya Singh Rathore “Lane Detection for Autonomous Vehicle using OPEN CV Library”, January 2019.

# Source Code

**Code of  Lane Simulation on Image**

```
import matplotlib.pylab as plt

import cv2

import numpy as np


def region_of_interest(img, vertices):

    mask = np.zeros_like(img)

   # channel_count = img.shape[2]

    match_mask_color = (255)

    cv2.fillPoly(mask, vertices, match_mask_color)

    masked_image = cv2.bitwise_and(img, mask)

    return masked_image


def draw_the_lines(img, lines):

    img = np.copy(img)

    blank_image = np.zeros((img.shape[0], img.shape[1], 3), dtype= np.uint8)


    for line in lines:

        for x1, y1, x2, y2 in line:

            cv2.line(blank_image, (x1, y1), (x2, y2), (0,255,0), thickness= 3)


    img = cv2.addWeighted(img, 0.8, blank_image, 1, 0.0)

    return img

image = cv2.imread('road.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


print(image.shape)
height = image.shape[0]
width = image.shape[1]


region_of_interest_vertices = [
    (0, height),
    (width/2, height/2),
    (width, height)
]
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
canny_image = cv2.Canny(gray_image, 572, 700)
cropped_image = region_of_interest(canny_image,
                np.array([region_of_interest_vertices], np.int32),)
lines = cv2.HoughLinesP(cropped_image,
                      rho= 6,
                      theta= np.pi/60,
                      threshold= 160,
                      lines= np.array([]),
                      minLineLength=40,
                      maxLineGap=25)
image_with_lines= draw_the_lines(image, lines)


plt.imshow(image_with_lines)
plt.show()
```

**Code of Video Simulation**

```
import cv2 as cv

import numpy as np

# import matplotlib.pyplot as plt



def do_canny(frame):

    # Converts frame to grayscale because we only need the luminance channel for detecting edges - less computationally expensive

    gray = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)

    # Applies a 5x5 gaussian blur with deviation of 0 to frame - not mandatory since Canny will do this for us

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

    # Applies Canny edge detector with minVal of 50 and maxVal of 150

    canny = cv.Canny(blur, 50, 150)

    return canny



def do_segment(frame):

    # Since an image is a multi-directional array containing the relative intensities of each pixel in the image, we can use frame.shape to return a tuple: [number of rows, number of columns, number of channels] of the dimensions of the frame

    # frame.shape[0] give us the number of rows of pixels the frame has. Since height begins from 0 at the top, the y-coordinate of the bottom of the frame is its height

    height = frame.shape[0]

    # Creates a triangular polygon for the mask defined by three (x, y) coordinates

    polygons = np.array([
                [(0, height), (800, height), (380, 290)]
            ])
    # Creates an image filled with zero intensities with the same dimensions as the frame
    mask = np.zeros_like(frame)
    # Allows the mask to be filled with values of 1 and the other areas to be filled with values of 0
    cv.fillPoly(mask, polygons, 255)
    # A bitwise and operation between the mask and frame keeps only the triangular area of the frame
    segment = cv.bitwise_and(frame, mask)
    return segment


def calculate_lines(frame, lines):
    # Empty arrays to store the coordinates of the left and right lines
    left = []
    right = []
    # Loops through every detected line
    for line in lines:
        # Reshapes line from 2D array to 1D array
        x1, y1, x2, y2 = line.reshape(4)
        # Fits a linear polynomial to the x and y coordinates and returns a vector of coefficients which describe the slope and y-intercept
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        y_intercept = parameters[1]
        # If slope is negative, the line is to the left of the lane, and otherwise, the line is to the right of the lane
        if slope < 0:
            left.append((slope, y_intercept))
        else:
            right.append((slope, y_intercept))
    # Averages out all the values for left and right into a single slope and y-intercept value for each line
    left_avg = np.average(left, axis = 0)
    right_avg = np.average(right, axis = 0)
    # Calculates the x1, y1, x2, y2 coordinates for the left and right lines
    left_line = calculate_coordinates(frame, left_avg)
    right_line = calculate_coordinates(frame, right_avg)
    return np.array([left_line, right_line])


def calculate_coordinates(frame, parameters):
    slope, intercept = parameters
    # Sets initial y-coordinate as height from top down (bottom of the frame)
    y1 = frame.shape[0]
    # Sets final y-coordinate as 150 above the bottom of the frame
    y2 = int(y1 - 150)
    # Sets initial x-coordinate as (y1 - b) / m since y1 = mx1 + b
    x1 = int((y1 - intercept) / slope)
    # Sets final x-coordinate as (y2 - b) / m since y2 = mx2 + b
    x2 = int((y2 - intercept) / slope)
    return np.array([x1, y1, x2, y2])
```

```
def visualize_lines(frame, lines):

    # Creates an image filled with zero intensities with the same dimensions as the frame

    lines_visualize = np.zeros_like(frame)

    # Checks if any lines are detected

    if lines is not None:

        for x1, y1, x2, y2 in lines:

            # Draws lines between two coordinates with green color and 5 thickness

            cv.line(lines_visualize, (x1, y1), (x2, y2), (0, 255, 0), 5)

    return lines_visualize


# The video feed is read in as a VideoCapture object

cap = cv.VideoCapture("input.mp4")

while (cap.isOpened()):

    # ret = a boolean return value from getting the frame, frame = the current frame being
projected in the video

    ret, frame = cap.read()

    canny = do_canny(frame)

    cv.imshow("canny", canny)

    # plt.imshow(frame)

    # plt.show()

    segment = do_segment(canny)

    hough = cv.HoughLinesP(segment, 2, np.pi / 180, 100, np.array([]), minLineLength = 100,
maxLineGap = 50)

    # Averages multiple detected lines from hough into one line for left border of lane and one
line for right border of lane

    lines = calculate_lines(frame, hough)
    # Visualizes the lines

    lines_visualize = visualize_lines(frame, lines)

    cv.imshow("hough", lines_visualize)

    # Overlays lines on frame by taking their weighted sums and adding an arbitrary scalar value of 1 as the gamma argument

    output = cv.addWeighted(frame, 0.9, lines_visualize, 1, 1)

    # Opens a new window and displays the output frame

    cv.imshow("output", output)

    # Frames are read by intervals of 10 milliseconds. The programs breaks out of the while loop when the user presses the 'q' key

    if cv.waitKey(10) & 0xFF == ord('q'):

def do_canny(frame):

    # Converts frame to grayscale because we only need the luminance channel for detecting edges - less computationally expensive

    gray = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)

    # Applies a 5x5 gaussian blur with deviation of 0 to frame - not mandatory since Canny will do this for us

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

    # Applies Canny edge detector with minVal of 50 and maxVal of 150

    canny = cv.Canny(blur, 50, 150)

    return canny



def do_segment(frame):

    # Since an image is a multi-directional array containing the relative intensities of each pixel in the image, we can use frame.shape to return a tuple: [number of rows, number of columns, number of channels] of the dimensions of the frame

    # frame.shape[0] give us the number of rows of pixels the frame has. Since height begins from 0 at the top, the y-coordinate of the bottom of the frame is its height
    height = frame.shape[0]

    # Creates a triangular polygon for the mask defined by three (x, y) coordinates

    polygons = np.array([

                        [(0, height), (800, height), (380, 290)]


        break

# The following frees up resources and closes all windows


def calculate_coordinates(frame, parameters):

    slope, intercept = parameters

    # Sets initial y-coordinate as height from top down (bottom of the frame)

    y1 = frame.shape[0]

    # Sets final y-coordinate as 150 above the bottom of the frame

    y2 = int(y1 - 150)

    # Sets initial x-coordinate as (y1 - b) / m since y1 = mx1 + b

    x1 = int((y1 - intercept) / slope)

    # Sets final x-coordinate as (y2 - b) / m since y2 = mx2 + b

    x2 = int((y2 - intercept) / slope)

    return np.array([x1, y1, x2, y2])


def visualize_lines(frame, lines):

    # Creates an image filled with zero intensities with the same dimensions as the frame

    lines_visualize = np.zeros_like(frame)

def do_segment(frame):
    # Since an image is a multi-directional array containing the relative intensities of each pixel
in the image, we can use frame.shape to return a tuple: [number of rows, number of columns,
number of channels] of the dimensions of the frame

    # frame.shape[0] give us the number of rows of pixels the frame has. Since height begins
from 0 at the top, the y-coordinate of the bottom of the frame is its height

    height = frame.shape[0]

    # Creates a triangular polygon for the mask defined by three (x, y) coordinates

    polygons = np.array([

                        [(0, height), (800, height), (380, 290)]

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


print(image.shape)

height = image.shape[0]

width = image.shape[1]


region_of_interest_vertices = [

    (0, height),

    (width/2, height/2),

    (width, height)

]

gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

canny_image = cv2.Canny(gray_image, 572, 700)

cropped_image = region_of_interest(canny_image,

            np.array([region_of_interest_vertices], np.int32),)

lines = cv2.HoughLinesP(cropped_image,

                rho= 6,
                 theta= np.pi/60,

                 threshold= 160,

                 lines= np.array([]),

                 minLineLength=40,

                 maxLineGap=25)

image_with_lines= draw_the_lines(image, lines)

def region_of_interest(img, vertices):

    mask = np.zeros_like(img)

   # channel_count = img.shape[2]

    match_mask_color = (255)

    cv2.fillPoly(mask, vertices, match_mask_color)

    masked_image = cv2.bitwise_and(img, mask)

    return masked_image


def draw_the_lines(img, lines):

    img = np.copy(img)

    blank_image = np.zeros((img.shape[0], img.shape[1], 3), dtype= np.uint8)


    for line in lines:

        for x1, y1, x2, y2 in line:

            cv2.line(blank_image, (x1, y1), (x2, y2), (0,255,0), thickness= 3)

    if cv.waitKey(10) & 0xFF == ord('q'):

def do_canny(frame):

    # Converts frame to grayscale because we only need the luminance channel for detecting
edges - less computationally expensive
```

```
    gray = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)

    # Applies a 5x5 gaussian blur with deviation of 0 to frame - not mandatory since Canny will
do this for us

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

    # Applies Canny edge detector with minVal of 50 and maxVal of 150

    canny = cv.Canny(blur, 50, 150)

    return canny




    img = cv2.addWeighted(img, 0.8, blank_image, 1, 0.0)

    return img

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



cap.release()

cv.destroyAllWindows()
```