# On-Road Object Detection using Computer Vision Techniques for Advanced Driver Assistance Systems (ADAS)

Project Report

Submitted by

**Sai Sravan Manne**
**EDM13B018**

in partial fulfillment for the award of the degree of

**Bachelor of Technology**

in

**Mechanical Engineering – Design and Manufacturing**


**Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, India**

April 2017

# BONAFIDE CERTIFICATE

This is to certify that the thesis titled "**Object Classification using Computer Vision Techniques for Advanced Driver Assistance Systems (ADAS)**" submitted by **Mr. Sai Sravan Manne(EDM13B018)** to the Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, for the award of **Bachelor of Technology in Electronics Engineering – Design and Manufacturing,** is a *bona fide* record of the project work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr.Binsu J Kailath**

Project Guide

Asst. Professor
Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127
India

Place: Chennai

Date:

# ACKNOWLEDGEMENTS

I would like to thank the Director of the Institute Prof. R. Gnanamoorthy for providing all infrastructural facilities required for my project as well as graduate study here. I wish to express my sincere gratitude to my Project Guide Dr. Binsu J. Kailath, Assistant Professor, IIITDM Kancheepuram, for the continuous support of my project work, for her patience and motivation.

I also wish to express my sincere gratitude to my Co-guide Dr. Sudha Natarajan, Autonomous Vehicle Program - R&D, Tata Elxsi Limited, Chennai, for her support and guidance for my project.

# ABSTRACT

The idea of a machine driving a complex system such as a car in real world scenario is no longer a myth. The manifestation of this idea is nothing but the self-driving vehicles and this is the current trend in the automobile industry. Several experts have forecasted that this would be the most disruptive technology by the year 2020. Though, this technology was completely developed by the year 2003, but the reason why it couldn't be commercialized is because of the immense cost of the system. So, getting a commercially viable product out of this technology is the goal of every automobile company currently in the market.

A typical self-driving system has several modules such as object detection, object recognition, cruise control and so on. The work discussed in this report is centred on object detection module. Object detection module has several sub systems such as radar based detection systems, ultrasound based detection system, vision (camera) based detection and GPS based navigation and detection system. The work presented in this report specifically focuses on vision based object detection techniques, in which the development of an efficient algorithm for this task, that can be implemented on a CPU, and still achieve the desired framed rate of 40 fps has been discussed.

In previous years several algorithm were published which use the approaches such as LIDAR based, convolutional neural network based and GPU based in order to achieve the desired frame rate. However, installing a LIDAR and GPU based solution into a system can increase the overall cost of the product (vehicle). This can lead to commercial failure of the product. In this regard, the algorithm discussed in this report is highly efficient as it is developed in OpenCV and can be implemented on any CPU with minimum required configuration and still achieve the desired frame rate.

# Contents

# List of Figures

\

# Nomenclature

ADAS - Advanced Driver Assist Systems

CNN - Convolutional Neural Network

RCNN - Region-based Convolutional Neural Network

GPU - Graphic Processing Unit

LIDAR - Light Detection and Ranging

CPU - Central Processing Unit

RADAR - Radio Detection And Ranging

FPS – Frames Per Second

# Chapter 1 Introduction

## 1.1 Motivation

Today, anywhere in the world, one could definitely find a person operating a smart phone, this is the extent by which the smart phones have penetrated into the market, and also, into our daily lives. Similarly, in the next 10 years this will be the exact scenario with the self-driving vehicles. The technology required to achieve this feat is currently in the testing phase and world's leading automobile manufacturers like Benz, BMW, Volvo etc have invested huge amounts in its development and are competing to release their autonomous driving vehicles by 2017.

Though this technology can completely change the phase of automobile industry, in the initial phase it is very costly. A lot of research is being done in the search for new algorithms and methods to reduce the system cost and to make it feasible. A simple example can illustrate the point here. Consider the case of Google Self Driving car and Tesla Model S, the first one is a complete self-driving car which doesn't even has a steering wheel, and has demonstrated successful operation in the 2012, whereas, the second one is semi auto pilot system, and has demonstrated successful operation in the year 2015. The current scenario is Tesla Motors have sold more than 76,000 vehicles in the year 2016 in USA and have even began their operation in UAE. However, the Google Self Driving Car was hardly even seen on roads. So, the fact is very simple, though Google had developed the working model a way before, it had the commercial element missing i.e. the cost of the car, so it is ultimately the commercial viability of the product that determines the sustenance in the market. Developing a commercially viable system is the main aim of this project.

## 1.2 Problem Statement

The chief objective is to develop an algorithm that can efficiently and accurately detect the on-road vehicles (both stationary and non-stationary). This entire project involves the following stages:

1. Understanding the various computer vision techniques currently being used for on road object detection.
2. Developing the initial design of the algorithm in MATLAB, and later implementation using Python with a major focus on optimizing its performance on GPU.

## 1.3 Overview of the Project

A typical self-driving system has several modules such as object detection, object recognition, cruise control and so on. The work discussed in this report is centred on object detection module. Object detection module has several sub systems such as radar based detection systems, ultrasound based detection system, vision(camera) based detection and GPS based navigation and detection system. The work presented in this report specifically focuses on vision based object detection techniques, in which the development of an efficient algorithm for this task, that can be implemented on a CPU, and still achieve the desired framed rate of 40 fps have been discussed. The major constraints involved in development of the algorithm are the input image size and the frame rate. The input image size and frames rate are interdependent quantities, the input image size can be associated with the number of computations involved. In general, for an accurate representation of a real time scenario an input image size of [640,140] is required. The algorithm developed in this project is tested on the datasets available in the KITTI Vision Benchmark Suite. The datasets are developed based on American road conditions such as streets and highways for different weather conditions such as rainy, cloudy, sunny etc. Currently, the algorithm is tested using datasets depicting highways, during cloudy and sunny weather conditions.

The basic difficulties that are encountered in developing this algorithm are the illumination variations in the backgrounds such as variation in brightness patterns due to shadows, reflections from sign boards and so on, these offer unnecessary noise that has to be filtered by the algorithm in order to achieve an accurate detection. Apart from these cluttering of similar colour objects beyond the road lanes, and four road junctions also cause detection errors. Even the broken white line lane markings are serious source of noise that often extend the detected object boundaries thereby leading to overlapping or cluttering of object detections.

The algorithm has two levels, in the first level it detects the road lane markings, and then separates the road from the rest of the image frame, and in the second level it processes the output from the first level using a custom designed filter, which basically supresses the illumination variations in the image. The lane detection is done using Hough transform, and since Hough Transform is widely used algorithm, there are pre-defined functions existing in OpenCV library that can perform the task in a highly resource efficient manner, i.e. efficient usage of memory and CPU. However, the design of the custom filter is the real challenge in this project, since there aren't any pre-defined functions in OpenCV that can optimize the algorithm efficiency. The implementation of this algorithm is discussed in detail in Chapter 3.

## 1.4 Contributions of the Thesis

In previous years several algorithm were published which use the approaches such as LIDAR based, convolutional neural network based and GPU based in order to achieve the desired frame rate. However, installing a LIDAR and GPU based solution into a system would increase the overall cost of the product (vehicle). This would lead to commercial failure of the product. In this regard, the algorithm discussed in this report is highly efficient.

The algorithm discussed in this report is developed in OpenCV-Python which can be implemented on any CPU and has demonstrated a frame rate of 38~40 fps when implemented on a Windows workstation with Intel i5 processor, 3.2 GHz and 8 MB RAM. Based on the literature survey done, which is discussed in the next section, the frame rate achieved by this algorithm is the maximum in comparison to any other on road vehicle detection algorithms published so far.

However, the algorithm discussed here is yet to be tested for night time and rainy road conditions.

## 1.5 Organization of the Report

The report is organized as follows: The various works that are published on on-road vehicle detection are discussed in Chapter 2, while the algorithm development and its implementation are discussed in Chapter 3. The stage wise output of the algorithm when tested on KITTI datasets are presented in Chapter 4 and the work is concluded in Chapter 5.

# Chapter 2 Literature Review

In the year 2016, many reports were published [1],[2],[3],[4],[7] with primary focus on on-road vehicle detection and lane detection techniques. A thorough study is made on the techniques employed in the abovementioned works and also the output characteristics for these works were noted down i.e. frame rate, input image size etc. This data is set as benchmark for the design of the current algorithm. Brief description of the various techniques employed in these papers are provided below.

## 2.1 Histogram of Oriented Gradients (HOG)

HOG is widely used to detect features in images. This technique was initially developed by Dala and Bill who are researchers at the French National Institute of Research in Computer Science. Basically, this algorithm functions by dividing the entire image into groups of rectangular cells, and then the histogram for individual cell gradients is constructed and normalized. This process of dividing the entire image into several small cells and treating them individually helps in avoiding the changes in colour intensities due to illumination pattern. Thus, this is the key factor that separates this descriptor in comparison to other descriptors. However, the orientation of object can cause significant change in the detected pattern. The following are the various stages that are involved in the implementation of the algorithm.

### 2.1.1 Gradient computation

This process is similar to that of an edge detection, which employs single dimension point discrete derivative mask. Even complex masks such as two dimensional Sobel mask or diagonal mask can also be used, however, the performance of these masks depends on their application involved. The most common filter kernels that are widely used for this purpose are [-1,0,1] and $[-1,0 1]^T$

### 2.1.2 Orientation binning

This step involves dividing the entire image into cells. The cells can be rectangular or circular depending on the feature pattern that needs to be detected. Now, the histogram of gradients needs to be calculated for individual cells. The direction of the gradient having maximum weight is treated as the direction of the gradient of the overall cell. The direction of the generated gradients can vary from 0 to 180 degree or from 0 to 360 degree whether it is signed or unsigned operation.

### 2.1.3 Block Normalization

The normalization factor computation technique used widely can be expressed as:

$$f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}} \tag{2.1}$$

where $f$ is the normalization factor,

$v$ is the vector containing the histograms in a given block,

$e$ is a constant.

The figure gives a brief outline of the operation performed by the HOG feature extraction process

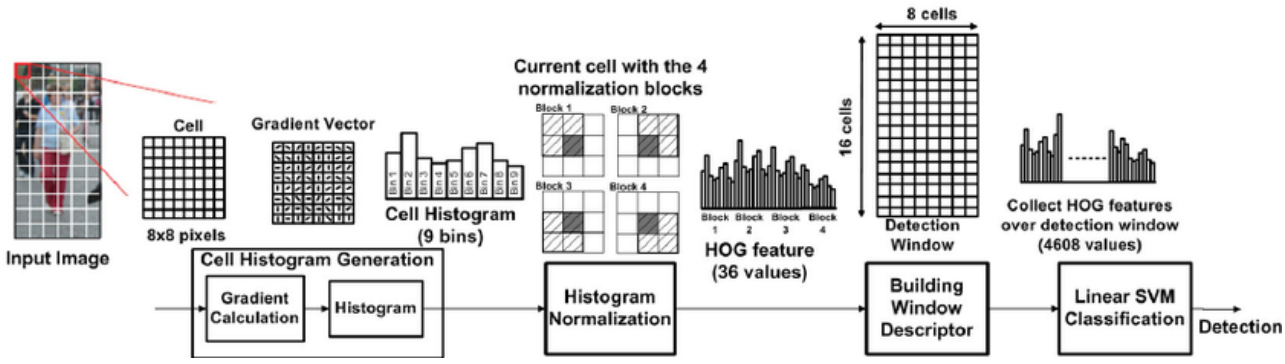


Figure 2.1: Flowchart of the HOG feature extraction

## 2.2 Support Vector Machine (SVM) Classifier

SVM classifier is widely used and its output can be illustrated by the figure below [2],[3],[5].

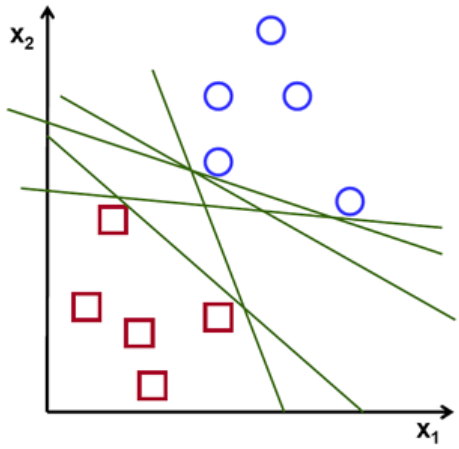


Figure 2.2: Graphic representation of SVM classifier operation

In the figure, the lines indicate the output of the classifier, the classifier tries to generate an optimum plane (in case of 3-dimentional data) to categorize the data available.

One common conclusion that can be drawn from the study of the various research works is that, all the systems that are using LIDAR based detection systems are highly efficient, whereas the output has a very low frame rate ranging from 8~13 fps. This can be attributed to the complexity involved in computing the three dimensional data that is generated, and so, in order to increase the frame rate of the system researchers have proposed the usage of GPU's. By parallelizing the algorithm implementation in GPU the overall frame rate can be increased but this will increase the overall cost of the system. This is the major setback with regard to the usage of LIDAR for implementation of the on-road vehicle detection systems.

A neural network based solution is reported wherein CNN's such as RCNN and FCNN are used to make region proposals for the detection, and these region proposals are later classified using SVM classifier [8]. This method is highly efficient similar to that of a LIDAR, but the accuracy of the detection greatly depends on the training dataset utilized. In reality one can find huge difference in the vehicles, streets and weather conditions prevailing in one country to another, which is actually a setback for this kind of approach, because a CNN can detect a vehicle or an obstacle only when it was trained previously with good number of similar objects.

Apart from the above discussion, the accuracy of a CNN also depends on the size of the weight matrix, and the number of layers involved. It has to be noted that both these quantities are

directly proportional to the execution time of the system. So, in order to achieve a high frame rate without compromising on the system performance, the authors have suggested the usage of GPU's. GPU can definitely upscale the frame rate of the system to a large extent, however, this comes at an increase in overall cost of the system.

However, since 2016, this trend has begun to change, as considerable amount of research is going on in the design of new algorithms for object detection, that can rely solely on the CPU itself. Non neural network based computer vision techniques began to surface up in research works, due to this new change. The current work described in this report is one such effort.

The following tabular column gives the input output characteristics of the literature survey done.

Table 2.1:Fram rate and image size data

| | **FPS** | **Image Size** |
|---|---|---|
| **[2],2016** | **24** | **[300,400]** |
| **[3],2016** | **10~13** | **LIDAR** |
| **[4],2016** | **36~38** | **[640,140]** |
| **[5],2015** | **4~6** | **[397,120]** |
| **[7],2016** | **30~32** | **[320,615]** |
| **[8],2016** | **5~6** | **[500,375]** |

# Chapter 3 Proposed Design for Optimizing the Processing Time

A detailed study is made on the common characteristics of the images in KITTI Dataset [6] and the same information is used to simplify the developed algorithm. The following are a few notable things that appear in over 80% of the images available in the KITTI Dataset:

1. Most of the roads are one way.
2. Roads have proper solid white lane line markings.
3. Roads have proper broken white lane line markings.
4. A four road junction appears for not more than 8 frames.
5. Only 50% of the image frame depicts the road (due to the mounting position of the camera).

Basing on the above-mentioned points, the following techniques or methods are incorporated into the algorithm.

1. Since only 50% of the image frame depicts the road, the entire image frame is first truncated into two halves (horizontally).
2. The solid white lane line markings can be used to separate the road from its adjacent surroundings, this in turn greatly reduces the noise that has to be filtered in order to detect the on road objects.
3. The picture frame can be divided into two halves (vertically), and the dominant line in both the halves can be found out using Hough transform which is illustrated in Fig. 4.6a and 4.6b.
4. The dominant lines found out in each frame are initially stored and compared with the previous values in the database, a new line is accepted or considered as lane only if it appears in consecutive frames (8 frames). This technique is especially useful to nullify bad and missing lane markings in certain image frames.

5. During every fifth iteration, the lines with least number of repetitions are discarded.
6. A filter is developed that can reduce the effect of illumination variations and broken white lane line markings which is discussed in detail in following sections.

Problems that were rectified due to the newly adopted techniques:

1. Reduction in the number of filters, thereby increasing the overall computation speed.
2. Reduction in the number of false detections that were mainly caused due to signboards and other bright objects in image frames.
3. Discontinuity in object detections due to variations in surrounding brightness.
4. Failing to detect objects that have zero relative velocity (moving with the same speed and in the direction similar to the vehicle on which the camera is mounted).

**Detailed description of the developed algorithm:**

The proposed system is implemented on windows workstation powered by Intel i5 with 8GB RAM at 3.2 GHz. The algorithm is initially developed in MATLAB because of the user-friendly interface and debugging techniques, later it is implemented in Python to make it completely open source, and also, to increase the frame rate.

The built-in functions utilized are as follows:

1. **Blob Analysis System Object:**

   The Blob Analysis object computes statistics for connected regions in a binary image. It computes the following parameters for the connected regions: area, centroid, bounding box, major-axis, minor-axis, orientation, eccentricity, equivalent diameter, perimeter.
2. **Dilation:**

   The binary dilation of A by B denoted $A \oplus B$, is defined as the set operation:

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \phi\} \tag{3.1}$$

   where $B$ is the reflection of the structuring element $B$. In other words, it is the set of pixel locations $z$, where the reflected structuring element overlaps with

foreground pixels in $A$ when translated to $z$. Dilation could also be defied wherein the structuring element is not reflected.

In the general form of *grayscale dilation*, the structuring element has a height. The grayscale dilation of *A(x,y)* by *B(x,y)* is defined as:

$$(A \oplus B)(x, y) = \max\{A(x - x', y - y') \mid (x', y') \in D_B\} \tag{3.2}$$

where $D_B$ is the domain of the structuring element $B$ and $A(x,y)$ is assumed to be $-\infty$ outside the domain of the image. In order to create a structuring element with nonzero height values, the syntax strel (nhood,height) could be used, where height gives the height values and nhood corresponds to the structuring element domain, $D_B$.

Most commonly, grayscale dilation is performed with a flat structuring element ($B(x,y) = 0$). Grayscale dilation using such a structuring element is equivalent to a local maximum operator:

$$(A \oplus B)(x, y) = \max\{A(x - x', y - y') \mid (x', y') \in D_B\} \tag{3.3}$$

3. **Erosion:**

The *binary erosion* of $A$ by $B$, denoted $A \ominus B$, is defined as the set operation $A \ominus B = \{z|(B_z \subseteq A\}$.

In other words, it is the set of pixel locations $z$, where the structuring element translated to location $z$ overlaps only with foreground pixels in $A$. In the general form of *gray-scale erosion*, the structuring element has a height. $D_B$ is the domain of the structuring element $B$ and $A(x,y)$ is assumed to be $+\infty$ outside the domain of the image.In order to create a structuring element with nonzero height values, the syntax strel (nhood,height) could be used, where height gives the height values and nhood corresponds to the structuring element domain, $D_B$. Most commonly, gray-scale erosion is performed with a flat structuring element.

4. **Hough Transform:**

The Hough Transform block implements the Standard Hough Transform (SHT). The SHT uses the parametric representation of a line:

$$rho = x * \cos(theta) + y * \sin(theta) \tag{3.4}$$

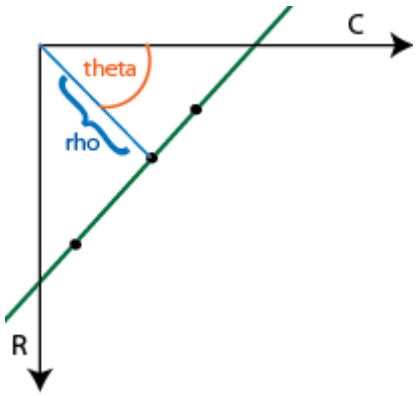


Figure 3.1: representation of rho and theta

The variable rho indicates the perpendicular distance from the origin to the line.
The variable theta indicates the angle of inclination of the normal line from the x-axis.
The range of theta is $-\frac{\pi}{2} \leq \theta < +\frac{\pi}{2}$ with a step-size determined by the Theta resolution (radians) parameter. The SHT measures the angle of the line clockwise with respect to the positive x-axis.
The Hough Transform block creates an accumulator matrix. The (rho, theta) pair represents the location of a cell in the accumulator matrix. Every valid (logical true) pixel of the input binary image represented by (R,C) produces a rho value for all theta values. The block quantizes the rho values to the nearest number in the rho vector. The rho vector depends on the size of the input image and the user-specified rho resolution. The block increments a counter which is initially set to zero, in the accumulator array cell pairs represented by (rho, theta) are found for each pixel. This process validates the point (R,C) to be on the line defined by (rho, theta). The block repeats this process for each logical true pixel in the image. The Hough block provides the resulting accumulator matrix as the output.

The figure below displays the flow chart of the designed algorithm:

.

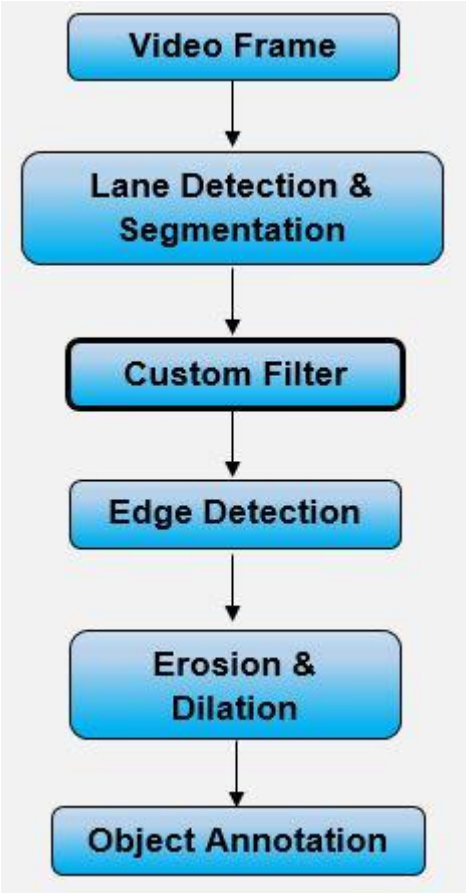


Figure 3.2: flowchart of the algorithm

The custom filter shown in the third stage of the algorithm does the following tasks

1. Limits the minimum and maximum colour intensities of pixels in between 0.25 to 0.3.
2. Smoothens the gradients in the pixel colour intensities.

Step by step implementation of the filter is discussed below.

1. Creating two individual masks, one mask for separating all the pixels with colour intensities greater than 0.3, while other for separating the pixels with colour intensities less than 0.3.
2. Invert (subtract the values from 1) all the colour intensity values greater than 0.3, and normalize the inverted values in between 0.2 and 0.53333
3. Make the first element in the other masked output i.e. values less than 0.3, and normalize the values in between 0.25 and 0.32142. Mask the output from this operation

with the mask initially developed to separate the pixel colour intensities greater than 0.3

4. Again the mask the output from step 3 with the mask that was developed in the lane detection phase
5. Add the outputs of the operations done in step 2 and 4.

# Chapter 4 Implementation and Performance Evaluation

## 4.1 System Configuration

The proposed system is implemented on windows workstation with AMD GPU. The GPU has 2GB graphics memory and the workstation is powered by Intel i5 with 8GB RAM, 3.2 GHz clock frequency.

## 4.2 Level wise description and output of the Algorithm:

**1.** Input image frame to the algorithm is shown in Fig. 4.1.

Figure 4.1: Input image frame to the algorithm

**2.** The truncated form of the input image is shown in Fig. 4.2.

Figure 4.2: the truncated form of Fig. 4.1

**3.** The truncated image is converted into a two dimensional quantity as shown in Fig 4.3

Figure 4.3: grayscale output

**4.** Edge detection filter is applied to find out the dominant edges in Fig. 4, the output of this operation is shown in Fig. 4.4

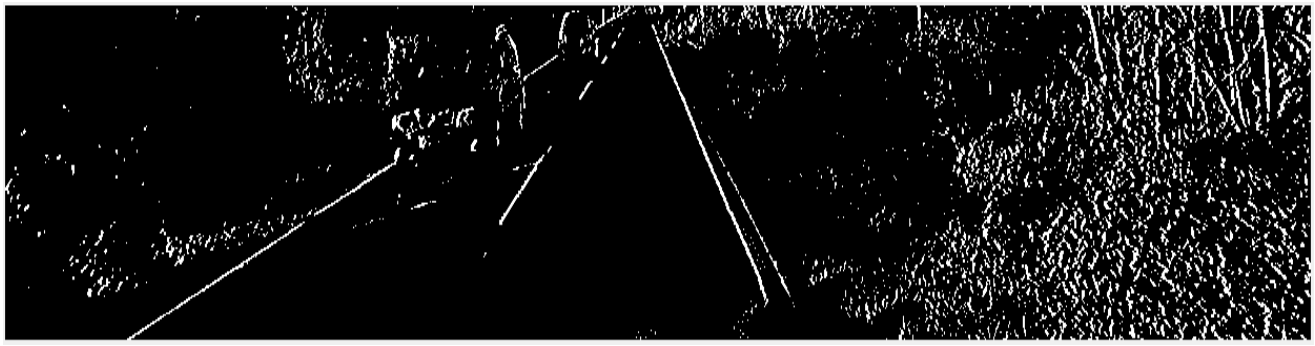

Figure 4.4: output from edge detection filter.

**5.** Now, Fig. 4.4 is subjected to a threshold operation where the values beyond the fixed threshold are treated as ones and the rest are made into zeros. The output of this operation is shown in Fig. 4.5

Figure 4.5: Output from threshold operation

**6.** Hough Transform is applied on the output in Fig. 4.5 and the output obtained is shown in Fig. 4.6a and 4.6b.

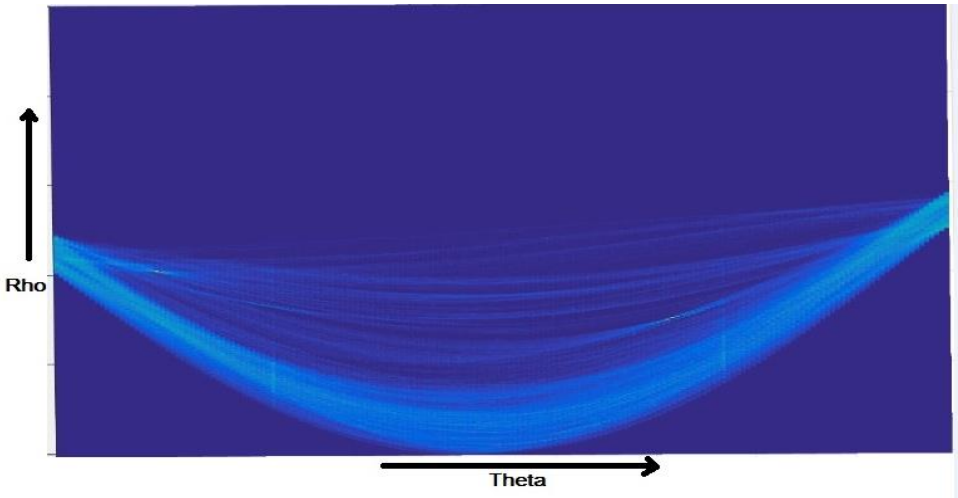


Figure 4.6a: plot of rho vs theta

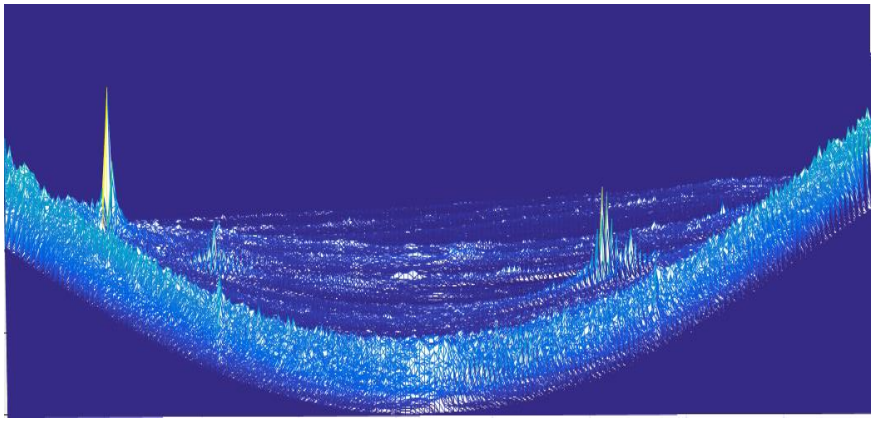

Figure 4.6b: frequency plot of rho vs theta

7. The rho and theta values having maximum frequency are found out from the peaks in Fig. 4.6b, these values correspond to the lane markings on the road. Then a polygon is drawn(pink colour) that fits in between the lines as shown in Fig. 4.7. The area occupied by the polygon corresponds to the area covered by the road.

Figure 4.7: solid white lane line detection output

8. The road detected in Fig. 4.7, is separated from the rest of the image, the output of this operation is shown in Fig. 4.8.

Figure 4.8: road separation from surrounding area

**9.** Fig. 4.9 displays the output of the built-in edge detection algorithm, it can be seen that the illumination variations on the road cause a lot of disturbances making it almost impossible to detect the vehicle.

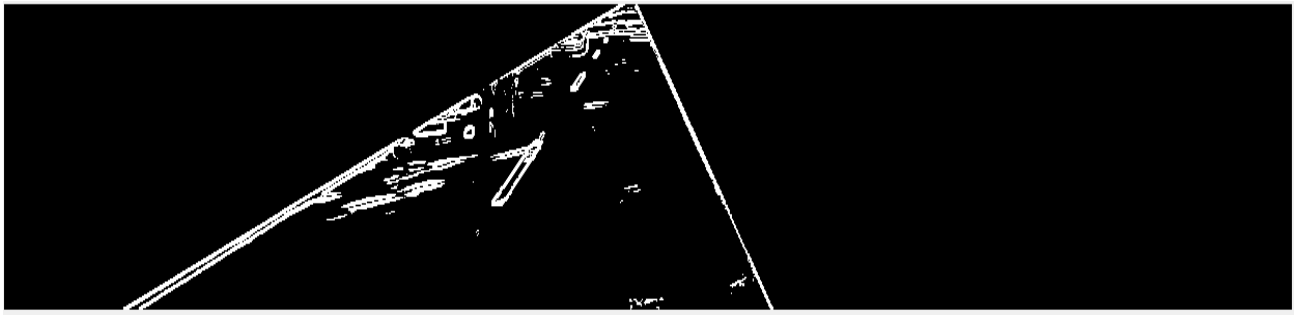

Figure 4.9: output from edge detection operation done on Fig. 4.8

**10.** In order to overcome the problem in step nine, a custom filter is developed to detect only the vehicles irrespective of the changes in illumination variations. The output of the filter can be seen in Fig. 4.10.

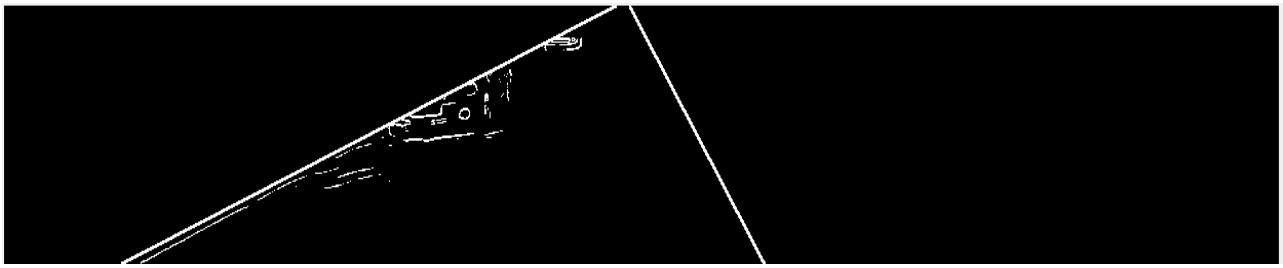

Figure 4.10: output from the custom designed filter

The designed filter smoothens the steep gradients in the colour patterns and at the same time compresses the values beyond a certain threshold (in this case it is 0.3). This process can be better understood by observing the following Fig. 4.11a, b, c and d.

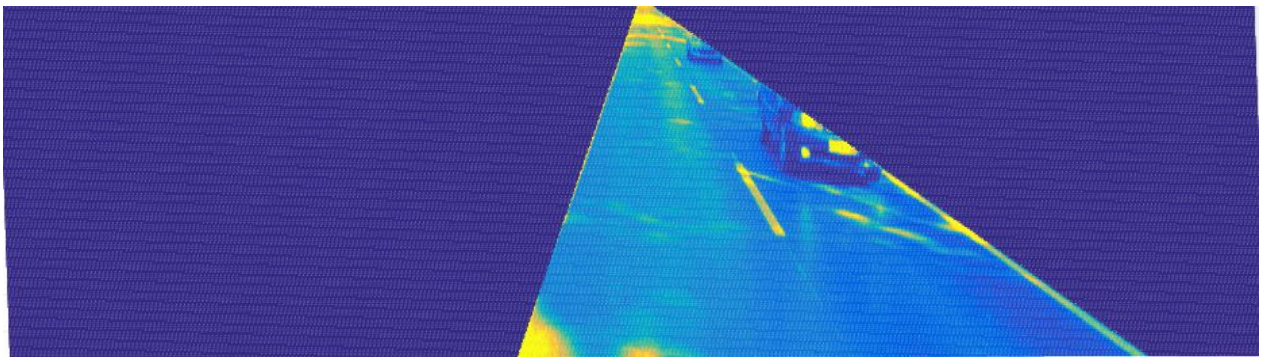

Figure 4.11a: brighter shades represented in yellow and darker shades are represented in blue.

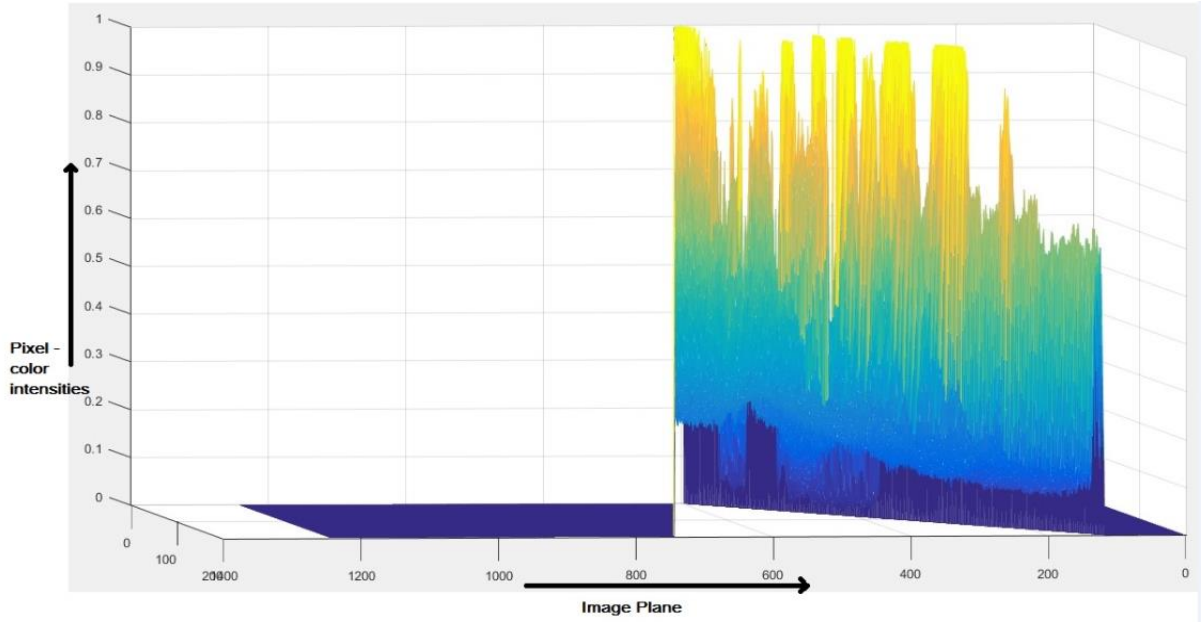


Figure 4.11b: variation in colour intensities

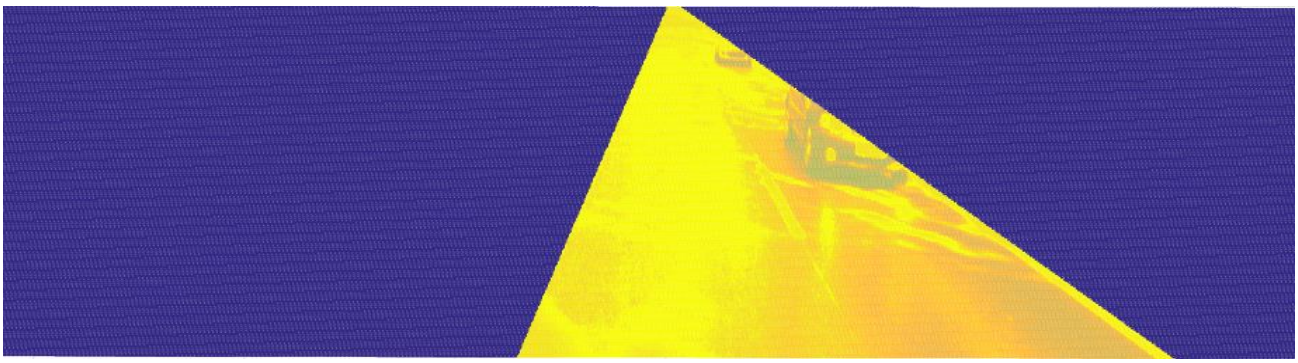

Figure 4.11c: output from the filter

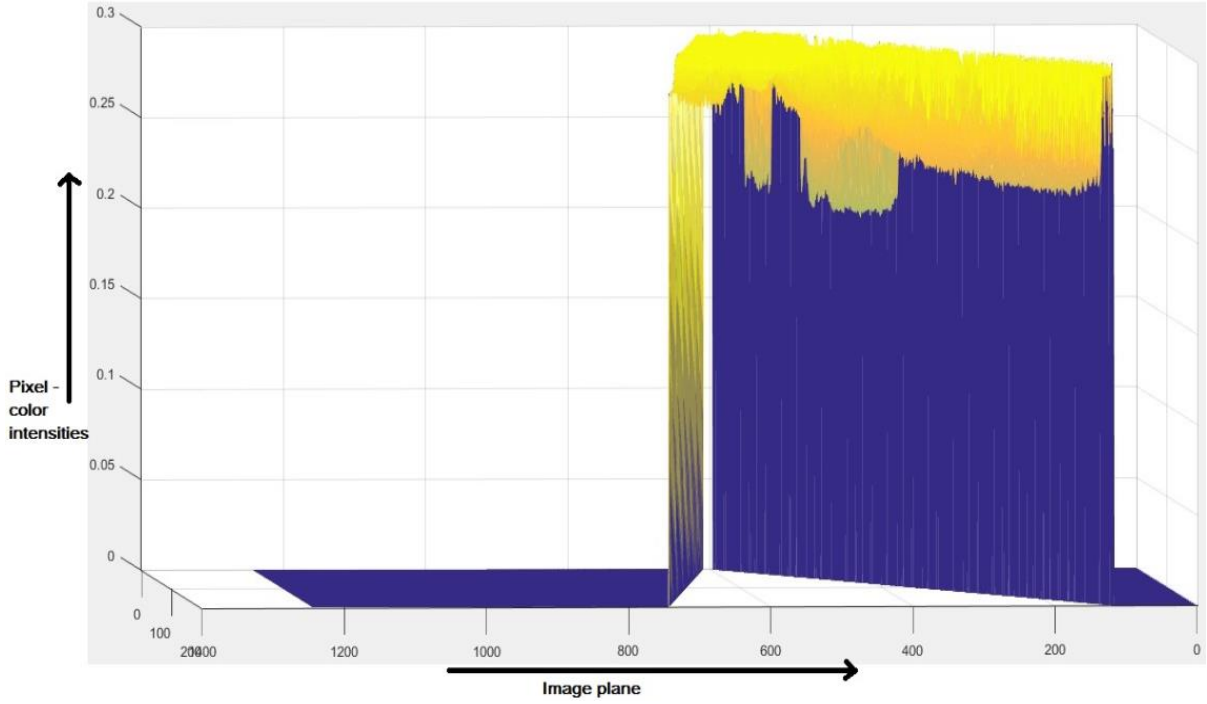


Figure 4.11d: reduced variation in colour intensities, max intensity is 0.3

**11.** The output shown in Fig. 4.10 is processed to remove the lane edges so as to facilitate the erosion and dilation operations. The output of this operation can be seen in Fig. 4.12.

Figure 4.12: output after removing the lane edges

**12.** Area filtration (white spaces with an area less than 10 units is removed) and erosion operations are performed on the output in Fig. 4.12 so as to enlarge the area covered by the vehicles. The output of this operation can be seen in Fig. 4.13.

Figure 4.13: output from area filtration and erosion operations

**13.** Built in MATLAB function called Blob Analysis is being used to find out the area and centre of the white spaces in Fig. 4.12 . The output of this function is a matrix consisting of centre coordinates and the height and width of the bounding boxes.

**14.** The output of the function in step 13 is further processed to track the detected white spaces (vehicles), which are further processed to remove false detections. However, the efficiency of the current method used isn't as expected. This has to be further improved to filter the unwanted detections. Bounding boxes are generated for the detected white spaces (vehicles), and the output is marked as shown in the Fig. 4.14.

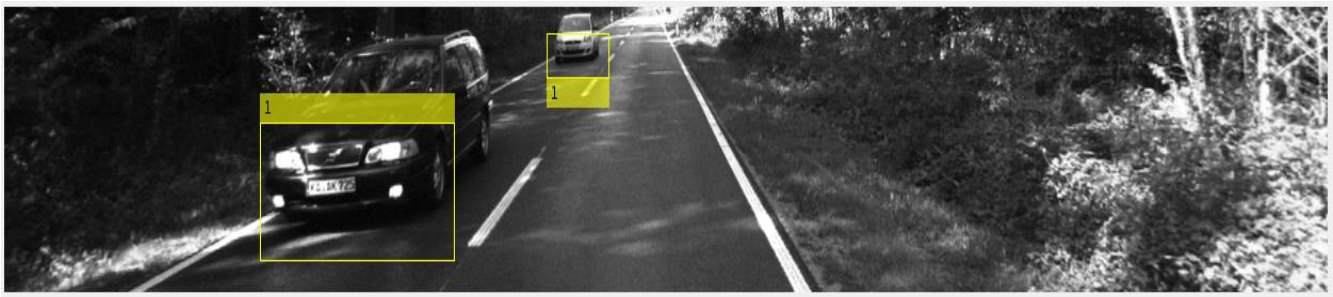


Figure 4.14: final output from the algorithm

The following is the level wise output of the algorithm for another dataset

1. Input image frame to the algorithm is shown in Fig. 4.15.

Figure 4.15: Input image frame to the algorithm

**2.** The truncated form of the input image is shown in Fig. 4.16.

Figure 4.16: the truncated form of Fig. 4.15

**3.** The truncated image is converted into a two dimensional quantity as shown in Fig 4.17.

Figure 4.17: the grayscale output

4. Edge detection filter is applied to find out the dominant edges in Fig. 4.17, the output of this operation is shown in Fig. 4.18.

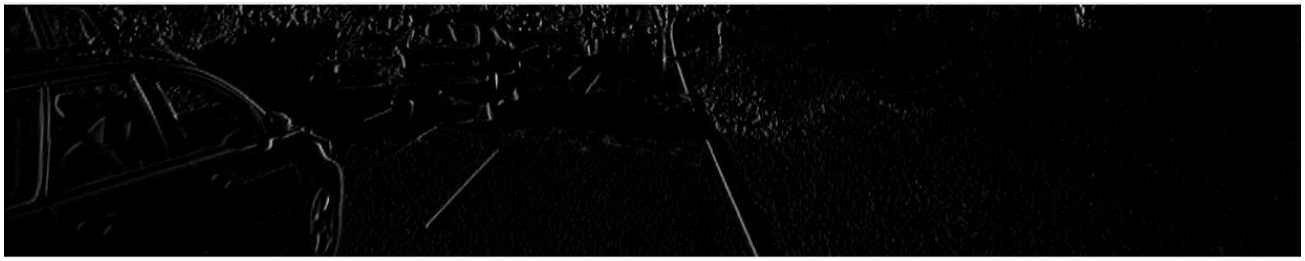

Figure 4.18: output from edge detection filter.

5. Now, Fig. 4.18 is subjected to a threshold operation where the values beyond the fixed threshold are treated as ones and the rest are made into zeros. The output of this operation is shown in Fig. 4.19

Figure 4.19: Output from threshold operation

6. Hough Transform is applied on the output in Fig. 6 and the output obtained is shown in Fig. 4.20a and 4.20b.

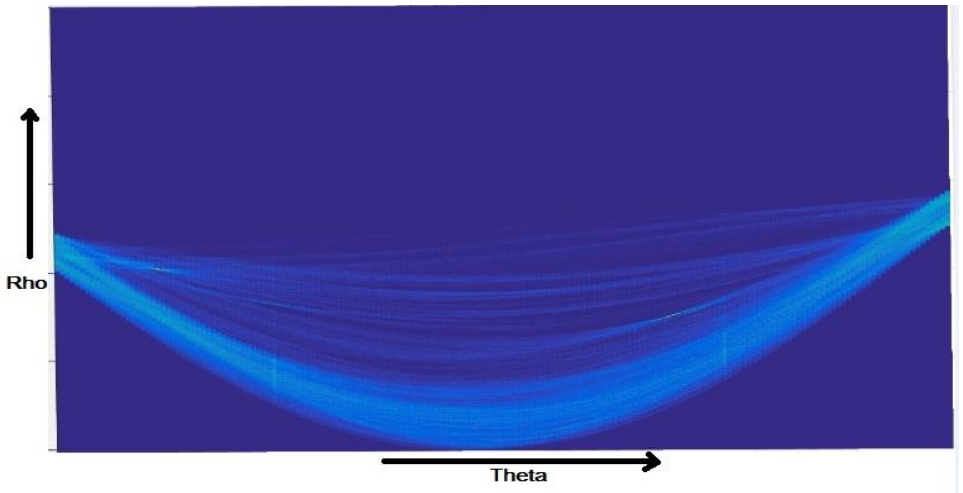


Figure 4.20a: plot of rho vs theta

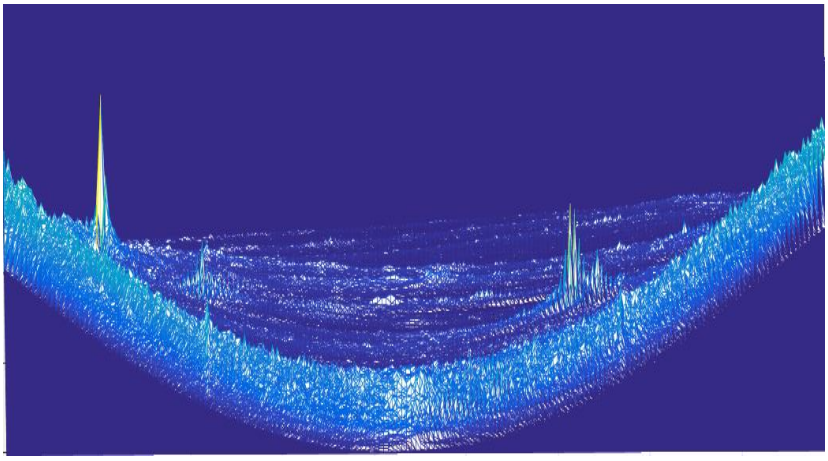

Figure 4.20b: frequency plot of rho vs theta

7. The rho and theta values having maximum frequency are found out from the peaks in Fig. 4.20b, which correspond to the lane markings on the road. Then a polygon is drawn (pink colour) that fits in between the lines as shown in Fig. 4.21. The area occupied by the polygon corresponds to the area covered by the road.

Figure 4.21: solid white lane line detection output

**8.** The road detected in Fig. 4.21, is separated from the rest of the image, and the output is shown in Fig. 4.22

Figure 4.22: road separation from surrounding area

**9.** Fig. 4.23 displays the output of the built-in edge detection algorithm, it can be seen that the illumination variations on the road cause a lot of disturbances making it almost impossible to detect the vehicle.

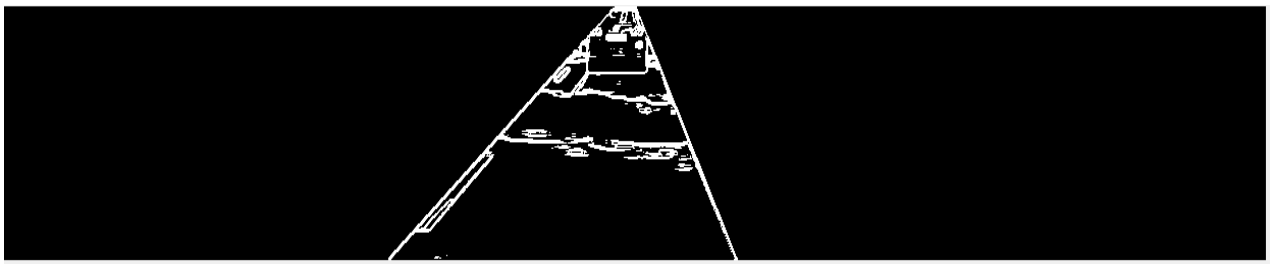

Figure 4.23: output from edge detection operation done on Fig. 4.22

10. In order to overcome the problem in step 9, a custom filter is developed to detect only the vehicles irrespective of the changes in illumination variations and broken white lane line markings. The output of the filter can be seen in Fig. 4.24 .

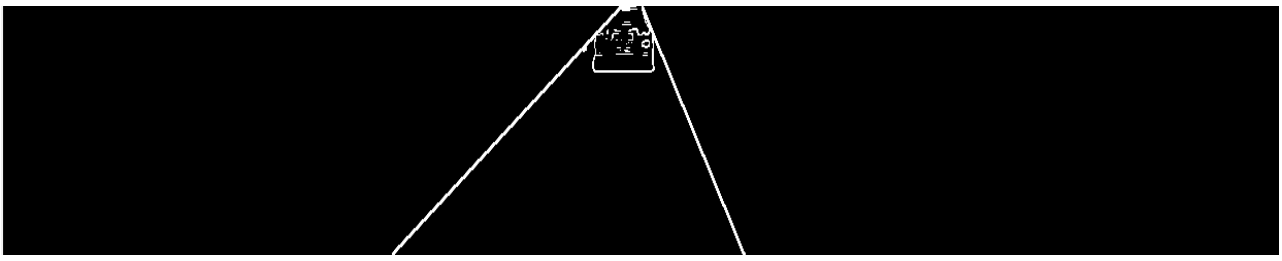

Figure 4.24: output from the custom designed filter

The designed filter smoothens the steep gradients in the colour patterns and at the same time compresses the values beyond a certain threshold (in this case it is 0.3). This process can be better understood by observing the following Fig. 4.25 a, b, c and d.

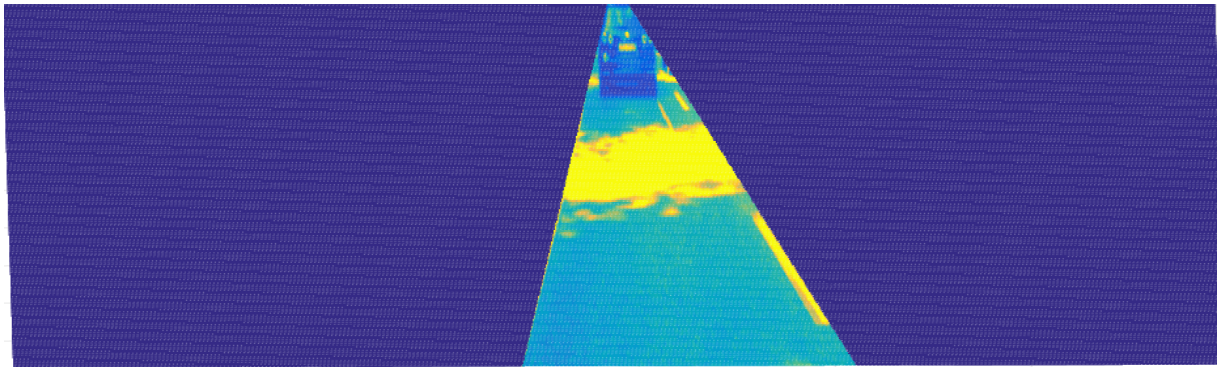

Figure 4.25a: brighter shades represented in yellow and darker shades are represented in blue.

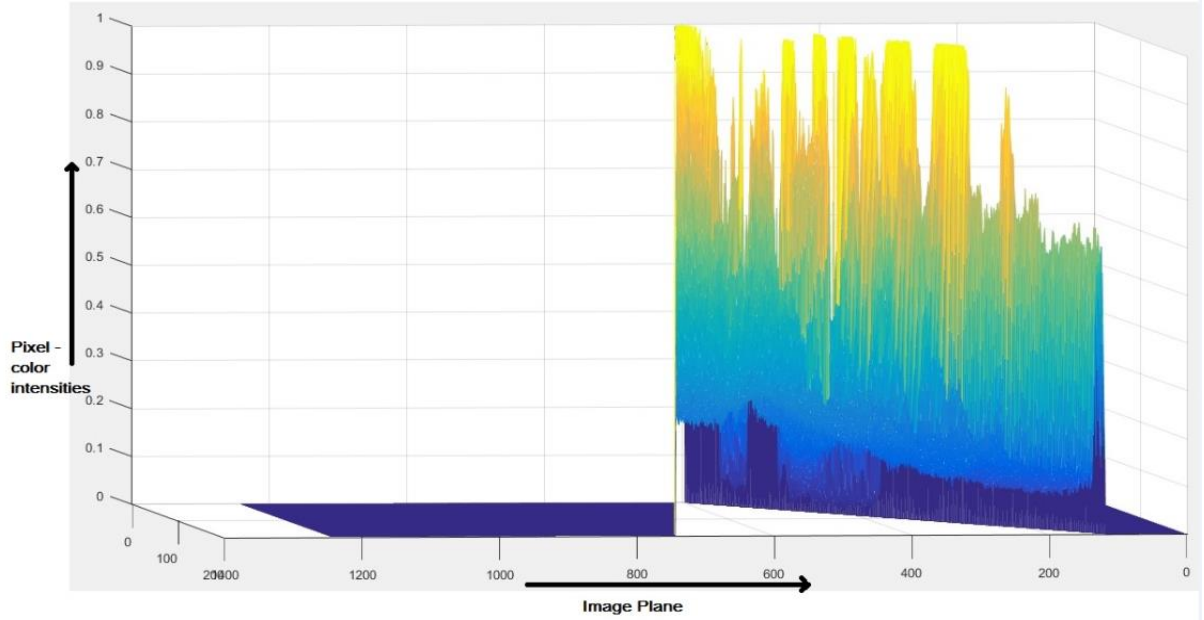


Figure 4.25b: variation in colour intensities

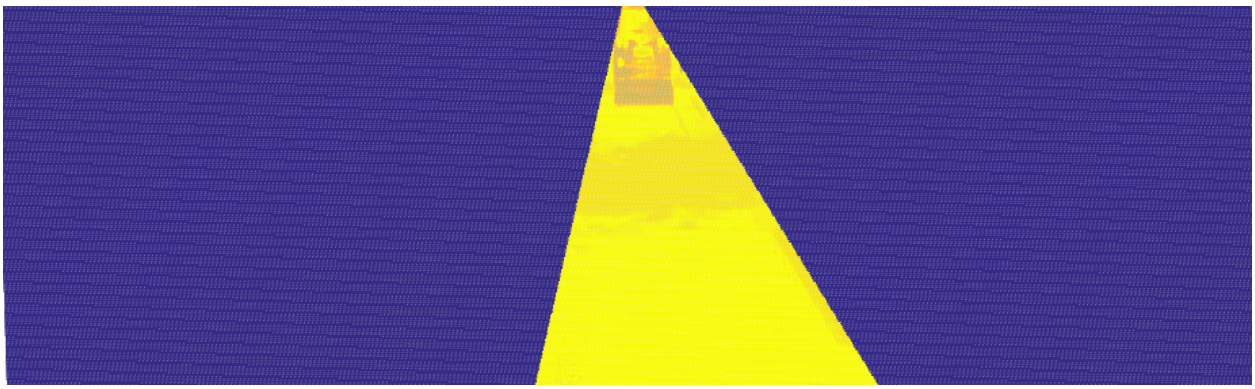

Figure 4.25c: output from the filter

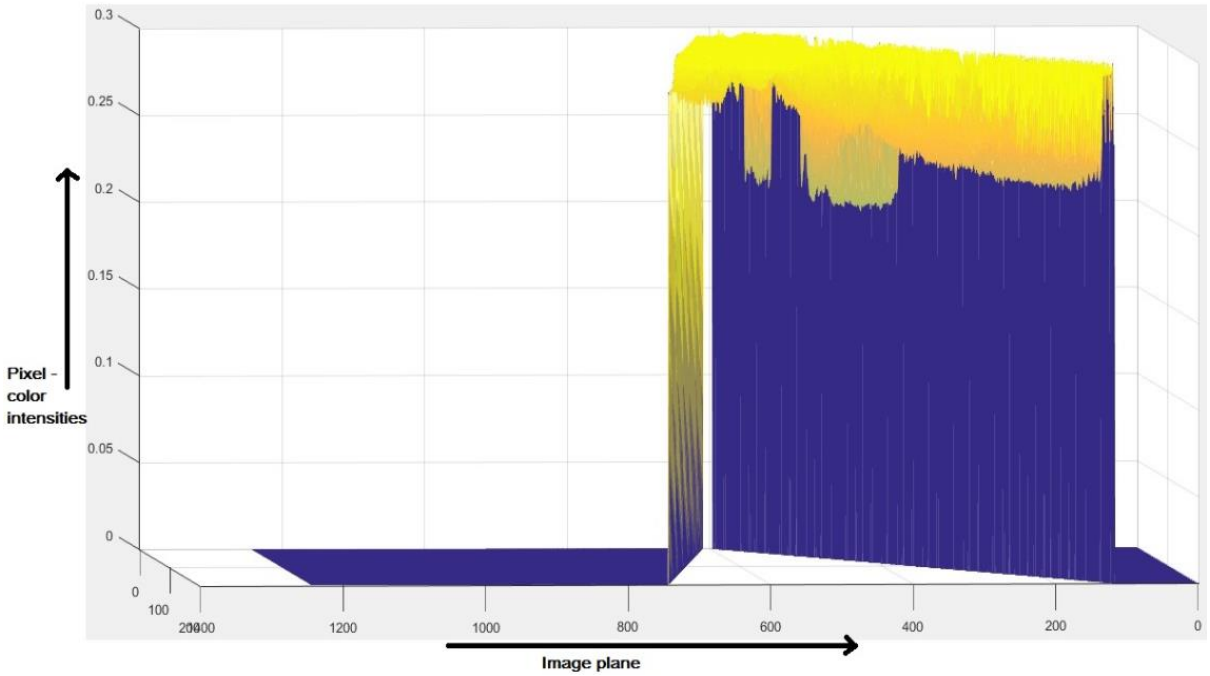


Figure 4.25d: reduced variation in colour intensities, max intensity is 0.3

**11.** The output shown in Fig. 4.24 is processed to remove the lane edges so as to facilitate the erosion and dilation operations. The output of this operation can be seen in Fig. 4.26.

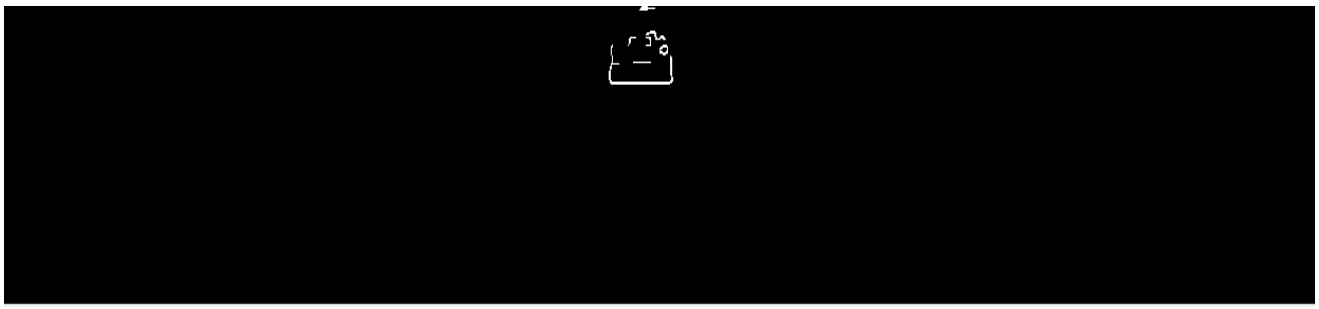

Figure 4.26: output after removing the lane edges

**12.** Area filtration (white spaces with an area less than 10 units is removed) and erosion operations are performed on the output in Fig. 4.26 this is being done in order to enlarge the area covered by the vehicles. The output of this operation can be seen in Fig. 4.27.

Figure 4.27: output from area filtration and erosion operations

**13.** Built in Matlab function called Blob Analysis is being used to find out the area and centre of the white spaces in Fig. 4.27. The output of this function is a matrix consisting of centre coordinates and the height and width of the bounding boxes.

**14.** The output of the function in step 13 is further processed to track the detected white spaces (vehicles), which are further processed to remove false detections. However, the efficiency of the current method used isn't as expected. This has to be further improved to filter the unwanted detections. Bounding boxes are generated for the detected white spaces (vehicles), and the output is marked as shown in the Fig. 4.28.


Figure 4.28: final output from the algorithm

The following table shows the frame rate achieved for two different input image sizes

Table 4.1: Input image size and FPS relation

| **Image Size** | **FPS** |
|---|---|
| **[375,1242]** | **17~19** |
| **[195,640]** | **38~40** |

# Chapter 5 Conclusion and Future Work

An efficient and accurate vehicle detection algorithm is discussed in this report that can be solely implemented on CPU. The frame rate achieved by the algorithm (40 fps) is by far the highest in comparison with the existing research works published in the field until now. Since the algorithm is coded in Python, there can be no copyright claims, so, it can be commercially used. Also, there is flexibility of importing the algorithm onto other operating systems such as Linux and also can be implemented on mobile computation platforms such as Raspberry Pi.

This algorithm is one of the sub modules of the self-driving vehicle system. So, this project can be further developed in order to integrate with RADAR based detection system, ultrasound based detection system or, GPS based navigation system. Also, additional features such as vehicle tracking, collision indication and lane change indication systems can be developed as an extension of this work.

# References


[1] Jisu Kim, Jeonghyun Baek and Euntai Kim, “On-road precise vehicle detection system using ROI estimation,” IEEE 17th International Conference on Intelligent Transport Systems 2016, Pages: 2251 – 2252.

[2] Zhipeng Di and Dongzhi He, “ Forward Collision Warning system based on vehicle detection and tracking,” IEEE International Conference on Optoelectronics and Image Processing (ICOIP) 2016, Pages: 10 – 14.

[3] Phanindra Amaradi, Nishanth Sriramoju, Li Dang, Girma Tewolde, and Jaerock Kwon, “Lane following and obstacle detection techniques in autonomous driving vehicles,” IEEE International Conference on Electro Information Technology (EIT) 2016, Pages: 0674 – 0679.

[4] Jang Woon Baek, Byung-Gil Han, Hyunwoo Kang, Yoonsu Chung, and Su-In Lee, “Fast and reliable tracking algorithm for on-road vehicle detection systems,” Eighth International Conference on Ubiquitous and Future Networks 2016, Pages: 70 – 72.

[5] Jisu Kim, Jeonghyun Baek, and Euntai Kim, “A Novel On-Road Vehicle Detection Method Using piHOG,” IEEE Transactions on Intelligent Transportation Systems 2015, Pages: 3414 – 3429.

[6] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun, “Vision meets Robotics: The KITTI Dataset,” International Journal of Robotics Research (IJRR) 2013.

[7] Huieun Kim, Youngwan Lee, Taekang Woo, and Hakil Kim, “Integration of vehicle and lane detection for forward collision warning system,” IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin) 2016, Pages: 5 – 8.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence 2016, Pages: 1 – 1.