TECHNISCHE UNIVERSITÄT
CHEMNITZ

Lane Detection
based on Contrast Analysis

Master Thesis

for the fulfillment of the academic degree
M.Sc. in Automotive Software Engineering

Faculty of Computer Science
Department of Computer Engineering
April 2016

Submitted by: **Surinder Kumar, 05.12.1977**

Supervisor: **Prof. Dr. Wolfram Hardt**
Dipl.-Inf. Daniel Reißner

Table of Content

List of Figures

List of Tables

Acknowledgment

This report is written during master thesis project at the Faculty of Computer Science, Department of Computer Engineering at Technical University of Chemnitz.
I would like to thank Prof. Dr. Wolfram Hardt for giving me the opportunity to work in his department on challenging research project which allow me to pursue different ideas, eventually leading to this thesis report. I would like to express my gratitude to several individuals who supported in completion of thesis project including, Stephan Blokzyl for providing help in choosing relevant image processing algorithm and methods and providing some ideas about how these methods are suitable for different hardware and software architecture. Dr. Ariane Heller, she always helps in solving organizational issues raised at each milestone. I would like to express my special thanks to Anantha R. Jayaram, a Linux device driver guru, who always ready to help me and supported for setting up hardware and software infrastructure.
Finally, I would like to thanks my supervisor Daniel Reißner, for providing valuable review and feedback comments, the time he spent for discussing problems and new idea for achieving goals of the project.
Last but not the least, I would like to express my heart-felt gratitude to my family members for their support and encouragement.

Abstract

Computer vision and image processing systems are ubiquitous in automotive domain and manufacturing industry. Lane detection warning systems has been an elementary part of the modern automotive industry. Due to the recent progress in the computer vision and image processing methods, economical and flexible use of computer vision is now pervasive and computing with images is not just for the realm of the science, but also for the arts and social science and even for hobbyists. Image processing is a key technology in automotive industry, even now there is hardly a single manufacturing process that is thinkable without imaging. The applications of image processing and computer vision methods in embedded systems platform, is an ongoing research area since many years.

OpenCV, an open-source computer vision library containing optimized algorithms and methods for designing and implementing applications based on video and image processing techniques. These method are organized in the form of modules for specific field including, user-graphic interface, machine learning, feature extraction etc [43].

Vision-based automotive application systems become an important mechanism for lane detection and warning systems to alert a driver about the road in localization of the vehicle [1]. In automotive electronic market, for lane detection problem, vision-based approaches has been designed and developed using different electronic hardware and software components including wireless sensor, camera module, Field-Programmable Gate Array (FPGA) based systems, GPU and digital signal processors (DSP) [13]. The software module consists on the top of real-time operating systems and hardware description programming language including Verilog, or VHDL.

One of the most time critical task of vision based systems is to test system applications in real physical environment with wide variety of driving scenarios and validating the whole systems as per the automotive industry standards. For validating and testing the advanced driver assistance systems, there are some commercial tools available including Assist ADTF from Elektrobit, EB company [43].

In addition to the design and strict real-time requirements for advanced driver assistance systems applications based on electronic components and embedded platform, the complexity and characteristics of the implemented algorithms are two parameters that need to be taken into consideration choosing hardware and software component [13]. The development of

vision-based automotive application, based on alone electronic and micro-controller is not a feasible solution approach [35] [13] and GPU based solution are attractive but has many other issues including power consumption.

In this thesis project, image and video processing module is used from OpenCV library for road lane detection problems. In proposed lane detection methods, low-level image processing algorithms and methods are used to extract relevant information for lane detection problem by applying contrast analysis at pixel level intensity values.

Furthermore, the work at hand presents different approaches for solving relevant partial problems in the domain of lane detection. The aim of the work is to apply contrast analysis based on low-level image processing methods to extract relevant lane model information from the grid of intensity values of pixel elements available in image frame. The approaches presented in this project work are based on contrast analysis of binary mask image frame extracted after applying range threshold. A set of points, available in an image frame, based lane feature models are used for detecting lanes on color image frame captured from video. For the performance measurement and evaluation, the proposed methods are tested on different systems setup, including Linux, Microsoft Windows, CodeBlocks, Visual Studio 2012 and Linux based Rasbian-Jessie operating systems running on Intel i3, AMD A8 APU, and embedded systems based (Raspberry Pi 2 Model B) ARM v7 processor respectively.

1 Introduction

> First they ignore you, then they laugh at you, then they fight you, then you win.
> --- Mahatma Gandhi

1.1 Overview

Humans recognize an object with little but this is not true for computer and machine vision-based systems. A variety of vision technologies are being employed for advance driver assistance systems (ADAS) applications and it is one of fastest growing area in automotive industry [62]. The advance driver assistance system applications such as lane departure warning systems (LDW), night vision, distance warning and vision-based self-parking systems will be among the major growth factors of the market for embedded vision systems. The embedded vision systems is an area of technology concerned with the entire real-time vision pipeline enabling machines to “see” and interpret data from computer vision software.

Lane detection and tracking is one of the most critical and challenging problem [3] for advanced driver assistance system applications in automotive industry. Many lane detection and lane tracking systems [1] [5] [6] [58] based on computer vision and image processing methods have been proposed for safety requirements and which provides relevant information related to road environment to the driver in intelligent transport system [45].

With the progress of high performance cost effective processing chips including CPUs, FPGAs [59] and GPUs [12], smart camera [14] and image sensors [26], embedded real-time operating systems [12] and computer vision-based methods [43] [50], it is becoming practical to incorporate image processing capabilities into a wide range of embedded vision systems [63]. The term embedded vision systems is emerged from image processing and computer vision technologies applied in the design and development process of embedded systems [70].

An image is a grid of pixels, in general 2D projection of a 3D scene captured by camera sensor. In image processing field, the process of converting continuous-space signal into discrete-space is called sampling and the samples are indexed by integer numbers. As in the case of 2D image, these integer numbers are rows and cols in image space. In the case of 3D camera space, there is one extra dimension added and each indexed coordinates called

pixel and has specific values called intensities value [39] in an image space. The intensity value is positive quantity depend on the data type and number of dimensions size of image. A gray-scale image contains only pixels of two intensity values, one for background pixels and one for object pixels. The contrast of the object is different from background values which help to recognize object from background in an image space. For example, higher the contrast filtering value shows more objects available with higher brightest area which separate the darkest and brightest areas in an image.

For computer vision (CV) based systems an image is a projection of a three dimensional (3D) structure into two-dimensions (2D). The projection process into two dimensional structure also include thrown-away objects which form noise and it carry many issues for the computer systems processing which is the main unit responsible extracting meaningful information from images [11]. One of the most interesting features of computer vision is that it lies on the boundary of several different academic disciplines, including, computer science, mathematics, industrial engineering and automotive software engineering. The goal of computer vision algorithms and methods is to make useful decision about the object and scenes based on capture data in an images in real physical environment [33]. Computer vision is a scientific and an inter-disciplinary research field of research that deal with methods for analyzing, capturing, manipulating images and vision-based data it has very tight links to Machine Learning (ML), image processing algorithms and optimization, Computer Graphics, Human Machine Interface (HMI) and Artificial Intelligence (AI) etc.

> *"Typically you will see some combination of lidar, radar, and high-resolution video cameras. But as image-processing algorithms improve, everything is in flux."*
> *In the panel discussion, Edward Ayrapetian, principal design engineer at sensor system vendor Nuvation.*
> *Article[1]: Advanced Driver Assistance Systems: Let the Driver Beware!*

Image processing have been an on-going research area in educational and commercial institutes. Computer Vision is part of image processing research field in which image manipulation, analysis, and retrieving information from images algorithms are premeditated on desktop based systems and processors where real-time processing requirement is not relevant. For example, image processing algorithms for medical image analysis. Most problems in image processing and pattern recognition have to cope with

1 Article: Advanced Driver Assistance Systems: Let the Driver Beware!, By Ron Wilson, Editor-in-Chief, Altera Corporation, July 21, 2014

high volumes data. In general it is very difficult to meet the hard real-time requirements of vision-based embedded systems. It is common practice to take image as reference to the resolution of digital imagery in terms of the sensor's packing density which is called pixels per unit area. The contrast of the image is simply the ratio of high intensity to low intensity, in either object or image space.

The function of determining the presence of a road line and locating it within the image frame is lane detection problem. This task requires mathematical based lane model [45] to discriminate lane from all other visual objects or patterns in the image after eliminating noise. In dynamic and cluttered visual scene, the location and appearance of the lane can change continually [58]. The task of the lane tracking algorithm is to follow the lane through a visual scene which involves establishing a temporal correspondence. The detected lane need to be constantly updated, maintaining an appropriate degree of correspondence over a time [60][24]. The lane detection functions requires a methods to discriminate lines between unwanted lines and other objects in the image and it may involves other type of task or sub-tasks, for example, classification of lines.

Computer vision based algorithms are developed on PC based computer systems available with direct power energy connection, large memory and processor with high computational power [53]. But these methods are not coping the embedded vision-based systems requirements [10] because embedded systems has different strict real-time processing requirements [12] [29]. Many automotive applications require computation to be performed under certain time constraints and even in hard real-time. In this case, computer vision-based method need to consider performance issues including computational complexity and accuracy while designing and developing real-time automotive applications [32] [35] [46].

Computer vision based systems development on embedded platform has been an ongoing research area in automotive domain and industrial applications, including GPS sensor based navigation system [82], lane departure warning systems (LDWS) [34] [27] [3] [6] and groups of application including in advance driver assistance systems (ADAS) [32], since many years now. There are many improved driver awareness systems available which helps in enhancing the wide range of advance driver assistance applications and road transport system [35] [45] and offer many benefits including safety, security and environment interaction for the driver.

A consumers survey report[2] by software supplier Cisco Systems suggests that more than half of global consumers (57%) would like to drive in a car controlled entirely by technology that does not require a human driver. The automotive industry is not meeting consumer demand in respect of technology [83]. The study in [83] acknowledges a ranges of current challenges including understanding the behavior of the transfer control between the vehicle and the driver, defining behavior of automated vehicles in wide range of environment conditions, and measurement of reliability of the communicated information between driver and the applied system.

1.2 Motivation

The advancement and continue evolution of hardware and software components [16] including, CPU, GPU, FPGA and DSP [80] and computer vision and image processing methods, including OpenCV library [43], has motivated the availability of increasing number of advanced driver assistance system application [7] in automotive domain.

This section provide an overview, the considerations of computer vision applications based on these hardware and software, and shows how existing computer vision-based methods are not suitable in designing and developing automotive embedded vision system applications [29] [12]. These consideration provide motivation behind this research project work.

It discussed several issues which are the main requirements need to consider while designing and implementing vision-based application in embedded systems platform. It provides an overview in context of an embedded vision systems containing camera or sensor module, several hardware storage and processing components including processor, memory, system-on-chip combination, FPGAs and GPUs etc., are explained as building blocks [53] [29] from which a vision system might be realized.

Automotive electronics is the second largest growth market for computer vision chips after mobile. A dramatic increase in safety requirements is a driving factor for the adoption of advance driving assistance systems (ADAS) [10]. In recent years, embedded vision-based methods has emerged as a powerful tools in designing and implementing robust vision-based systems for advanced driver assistance systems application including, lane detection and tracking systems [25]. The advance driver assistance market is one of the fastest

2 Survey Report-Consumers Desire More Automated Automobiles, Cisco Study, SAN JOSE, Calif. – May 14, 2013

growing application sector in current automotive electronics [7]. The market analysis firm *Strategy Analytics*[3], discussed in their report that the automotive companies and third party vendor automotive OEMs will be investing in about of $25B USD per year on advance driver assistance systems applications [7]. The market analysis *Strategy Analytics* company also predict, as shown in Figure 1.1, that the integration of image and vision processing functionalists is an essentional element of ADAS which offer additional and enhanced features beyond these are provided by the Radar, LiDAR, infrared, ultrasound and other sensing technologies[7].

In general, image processing and computer vision application development and implementation contain some pre-processing [5], analyzing image frame [18] [25] [34] and post-processing steps [13] including extracting information using mathematical [1] [6] or geometrical methods [3]. Embedded vision systems required to consider the entire real-time vision pipeline [12] [29] processing including starting from capturing image frame from camera or sensor module, operations that required filtering and segmentation of image data to the final output result which will be used by other systems to provide the services.

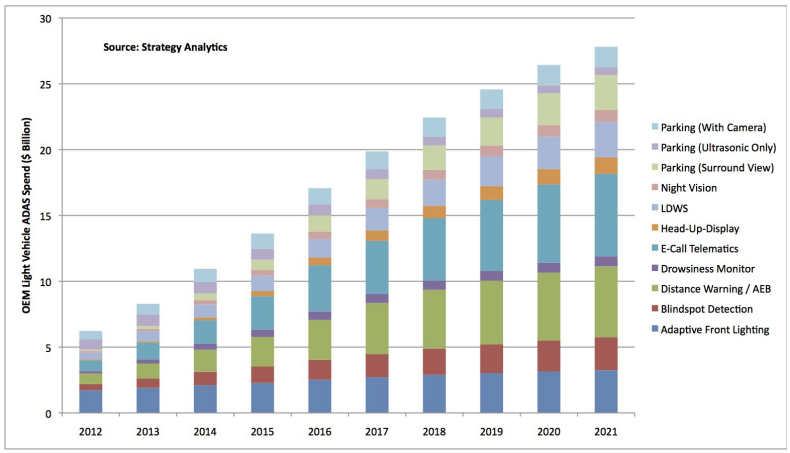

Figure 1.1: Diversity of advanced driver assistance and safety solution [7]

There is an immense option available of hardware and software based processing unit components including FPGAs, GPUs, DSPs for building real-time computer vision applications. These available choices create another issues in designing real-time systems applications, need to consider best suitable options after achieving best trade-off between several conflicting options [20]. In general, the following performance matrices are consider in designing and developing a typical embedded system i) Application performance ii) Power consumption matrices and iii) System size [20].

The object detection, an automotive safety application, has been tested on CPU and GPU

3 Strategy Analytics © 2015 www.strategyanalytics.com

processor based systems using CUDA module to abstract and classify vehicle data collected from vision sensors [46]. In CUDA module available in GPU processor, each block (core) perform same task on different data set available in the memory module. The major problems GPUs based systems are power consumption (minimum power consumption on 7800GTX was 50W), mapping the processing data to available processing unit and communication cost between host CPU and GPU.

For detecting and analyzing lines in an image Hough Transform (HT) [36] method has been used since many years now which extract straight the lines between the parameters space and Cartesian space. This method may lead to the misleading result if an image contain huge noise which lead to unwanted and high number of finite lines with defined end points to infinite lines [34]. In general Hough Transformation method is suitable for applying in applications where real-time applications performance is not required. For example applications, for designing and implementing software for construction architecture, interior and drawing, and in medical image scanning etc, are not required hard real-time performance constraints. The second drawback of Hough Transform method is its performance, which is not suitable for lane detection application for automotive domain where are resources for the computations are very limited and need to achieve result in hard real-time.

A report from Xilinx Company covered that many computer vision systems are implemented or prototyped based on many common computer vision-based functions, using optimized implementation of OpenCV library, targeting from PC based processors [79] [73], data intensive hardware accelerators GPU [], FPGA based SoCs [8] [59] to embedded platform [66] [80] including ARM Cortex A8. OpenCV library, which is an optimized implementation for computer vision-based applications, can also be further optimized using in-build parallel and multi-core processing capabilities [43] as per the application requirement to achieve real-time performance, for example using *parallel_for_* loop. Computer vision-based applications has been implemented in embedded system based platforms such as Zynq Base TRD [70] for capturing frame from camera using OpenCV function call and post-processing can be implement using in-build video processing unit available in Zynq Base TRD device which run on CortexTM-A9 processor cores. A computer vision based application for night vision has been implemented with low cost security camera on small size Raspberry Pi hardware module using OpenCV library

[76] [80].

Computer vision-based on-road systems are safety critical and require high real-time performance and these systems must perform reliably, in different road, weather and traffic conditions [45]. The design and development processes of computer vision-based advance driver assistance applications for intelligent vehicles also presents a number of unique challenges due to the availability of huge options of advanced technologies in hardware and software components integration.

A wide range support of open source library, OpenCV, and low cost embedded systems platforms, including Raspberry Pi, provide enough motivation to design, implement and test road lane detections methods on embedded system based processor ARM v7.

1.3 Thesis Project Objectives and Scope

Automotive industry has some strict and hard constraints in their advance driver assistance system applications in the terms of reliability, cost effectiveness and real-time performance [16]. Automotive third parties companies offering low-cost sensor capabilities [26] [30] [37] included reduced computational cost, help both research institute and business specific organizations with the goal to bring the potential future applications of their work into this domain in continue advancement in intelligent transport systems (ITS) [38]. Images or frames extracted from camera module mounted in vehicles provide significant information and data about the vehicle's current position and surrounding environment which need to be analyzed in order to extract relevant information or object to support real time advance driver assistance systems.

Lane detecting and tracking has been significant research area in computer vision society and automotive industry which play an important role for the safety and warning system to the driver. Most of the time low level image processing, including pre-processing is the first step in extracting object or relevant information from camera image [78]. These low level image processing methods includes converting color space image into gray-scale binary or into reduction of dimensional space 1D or 2D form. During the last decade research groups as well as a number of commercial organizations have started to deploy image processing algorithms based on computer vision in the automotive domain applications especially in advance driver assistance systems(ADAS), human-machine interface (HMI) [11], car navigation systems (CNS) [27].

Most of these applications have some common characters which are based on traditional computer vision and graphic research filed. For example, applying Canny Algorithms for edge detection, binary threshold for converting image or video frame into different format and extracting region of interest to reduce the number of computation on small size of image. For detecting road lane using computer vision library and applying image processing algorithms the following section provide a list of objects or goad defined in the research project.

Objectives

The aim of the presented thesis project work is to provide state of the art of lane detection methods applied in advanced driver assistance system applications based on image processing and computer vision methods. The representation of thesis project work deals with the lane detection problem in the field of advance driver assistance systems domain and proposed methods based on contrast analysis of image processing methods and functionalists available in OpenCV, an open source computer vision library.

The objective to develop lane detecting and tracking systems is to provide safety and warning feature to the driver while vehicles moving on the road. The central point of low level image processing algorithms is grid of pixels (an image frame) in direction of one, two or three dimensions which form channel and stored in specific data type in computer memory in form of digital images. Low-level vision algorithms take a digital image as an input, perform some pre-processing manipulation and measurement and extract useful information [31][66]. The goals or aims of this thesis project is given in the following list including:

- To review the current state of the art of lane detection and tracking methods based on image processing and computer vision methods.
- To present an up-to-date survey about the computer vision-based advance driver assistance applications implemented on different platforms
- To discuss the current advancements and challenges lies in embedded vision systems development.
- To apply low level image processing methods and algorithms for lane detection and to measure performance matrices.
- To implementation and test of these algorithms on different operating systems running on different processors and platforms including Microsoft Windows, Linux

and Rasbian-Jessie..

- Analyzing and measuring performance evaluation parameters on different platforms based on different frame rate.

The main contribution of this work is the implementation of low-level image processing and computer vision-based lane detection methods used to extract lane information from a camera image frame.

This thesis project proposes some methods for lane detection, including Light Flow Method, Direction Based Method and Bird Flow Method, based on contrast analysis of intensity value of pixel element by applying recursive methods on limited region of interest and updating the region of interest for next image frame.

Assumptions

The lane detection methods proposed int his project work has some assumptions which are listed in the following sections.

- The lane detection methods Light Flow Method and Direction Based Method can be used only for single road lane and Bird Flow Method can be used only for two left and right side road lanes.
- The *inRange* function available in OpenCV provide binary mask image, in which contain only 0 as black color and 255 as white color, based on the intensity value of each pixels available in image frame. In this case, the road lane color should be in bright white or yellow color. It is possible that proposed methods are not reliable in case of different color than these two white and yellow color value.
- The size of image frame used in this work includes 360x640 pixels for Light Flow Method and Direction Based Method and 480x640 pixels for Bird Flow Method. A small modification is required to use these methods for different size of image frame.
- The proposed methods are suitable for CPU and embedded systems based platform and are not implemented with the consideration of GPU and FPGA based systems platform. On other hand a small change in the implementation can be applied to run these methods on GPU based systems.

1.4 Outline of Thesis Report

The outline of this thesis project work is intended to provide a short introduction of the chapters included in the report.

Chapter 1 provide introduction about lane detection and tracking problems in automotive industry which help to understand current state of the art of lane detection problems and discuss the idea which lead the motivation behind this research project. It also shows how computer vision-based algorithm are not suitable for embedded vision-based systems. It also give the reason about this unsuitability and motivate us to start new research and implement different methods and techniques to achieve the relevant lane detection performance result. Special focus is given how lane detection problems is solved on the based of different hardware and software platforms using open source OpenCV library. At the end of the chapter, it provides a list of goals which were planned in this project work.

Chapter 2, provide some basic knowledge of OpenCV programming implementation and short example code which have been used in this thesis project. This section give short implementation and overview detail of the relevant methods used in lane detection methods. This section provide information only those functions and procedure from OpenCV library which were extensively used in proposed lane detection methods. A complete list of procedures and methods can be obtained from OpenCV web-page.

Chapter 3, discussed the current state of the art of lane detection and tracking systems available in current advanced driver assistance systems module in automotive industry. This section also provide some overview about current advancement and challenges available in applying vision-based methods for designing and developing advanced driver assistance systems applications. The advancement of vision-based embedded systems are accelerating at high clock speed including involvement of use of mobile devices and internet of things which attract the consumer market. But on other hand, integration of extensive use of hardware and software components within limited resource and power consumption matrices, hard real-time applications raises some critical issues and challenges.

Chapter 4, provide concepts architecture of the proposed method which shows how these are performed on captured image frame. This section also provide method flow chart for each proposed method with output result.

Chapter 5, discussed about implementation architecture of proposed lane detection methods

and explain how these methods perform to solve the lane detection problem.

Chapter 6, Performance evaluation, discuss the results obtained after performing proposed methods on different operating systems and software environment.

At the end, Chapter 7-Conclusion, summarizes and concludes the thesis project including some idea about future work which can be done on these proposed method for extending the algorithms include real-time camera module and updating the methods for detecting more the two lane on the road.

2 Basic OpenCV Operations

> We may have all come on different ships, but we're in the same boat now.
> --- Martin Luther

The following chapter introduces some basic methods and operations available in OpenCV library which are used in this thesis project work. This chapter focus on only some methods from OpenCV library which were intensively used and provide detailed information with examples. These examples are taken from implementation of lane detections methods and are the basic fundamentals for dealing lane detection methods proposed in this work. In case reader, wish to gain in-depth knowledge and a complete list of methods and operations available in OpenCV library, there are lot of tutorials available in the OpenCV[4] web page.

2.1 Reading and Writing Image or Video Files

The OpenCV offer some very easy to use in-build methods to capture frame from video or camera module or reading video file from memory.

The following *Example 1* shows how to read an image file. An image file is read into Mat image and displayed in Window "Win1". After this operation Canny threshold operation is applied to extract edges in the image using Canny function available in OpenCV. The result of edge detection operation is stored in `cannyImg` Mat variable which is used to save an image as canny image.

```
// Example 1: Reading image file from memory

void read_Image( )
{
 Mat image, cannyImg;

 namedWindow("Win1", CV_WINDOW_AUTOSIZE );
image = imread( "/Dataset/images/image1.jpg", 1 );
if (!image.data) {cout<<"No image data..."<<endl; return -1; }

 imshow( "Win1", image);              // Show image in Window

 Canny(image,cannyImg,100,200,3);  // Apply canny algorithm

 imwrite( "CannyImage.jpg",cannyImg);  // Save cannyimg
```

4 www.opencv.org

```
  waitKey(0);
  image.release();
  destroyWindow("Win1");
  }
```

The following *Example 2* shows how to read an image frame from video file save at specific memory location.

```
/Example 2: Reading Video Frame from memory

void read_VideoFrame( )
{
Mat frame;
 namedWindow("Win1", CV_WINDOW_AUTOSIZE );

 // Read image from memory
 VideoCapture video ("/Dataset/video/drive5.avi");

 video.read(frame) ;
 if (!frame.data) { cout<<"No image data..."<<endl; return -1; }

 while(video.read(frame))
 {
  imshow( "Win1", frame);
  char ch = waitKey(1); if (ch == 27){ break; }
 }
frame.release();
destroyWindow("Win1");

  }
```

The following example, *Example 3* shows how to read an image frame from camera module and save at specific memory location.

```
//Example 3: Reading Video Frame from Camera

void read_CameraFrame( )
{
Mat frame;
 namedWindow("Win1", CV_WINDOW_AUTOSIZE );

 VideoCapture video(0);  // Capture frame from camera

if (!video.isOpened()) { cout<<"No frame data..."<<endl; return -1; }

 if (!frame.data) { cout<<"No image data..."<<endl; return -1; }

 while(true)
 {
 video>>frame;
  imshow( "Win1", frame);
  char ch = waitKey(1); if (ch == 27){ break; }
 }
```

```
frame.release();
destroyWindow("Win1");

}
```

2.2 InRange

OpenCV offer an interesting inbuilt functionality called `inRange()` function. This function checks if input array elements lie between the specified lower and upper range. The following is the format of the `inRange()` function

InRange (InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)

src: The input image which can be 3D channel color image.

lowerb: The lower boundary array or a Scalar color value

upperb: The upper boundary array or a Scalar color value

dist: The output array of the same size as src array and CV_8U type.

The function perform the range operation as below:

```
If ( (src (i) <= lower (i) OR src (i) >= upperb (i) )
     dst (i)   =      0;       // Black color value
else
     dst (i)   =    255;       // White color value
```

The following Figure 2.1 shows the result of `inRange` operation with different color intensity values. From the diagram, it can be shown that more good result can be achieved with more increase in the color filter values.

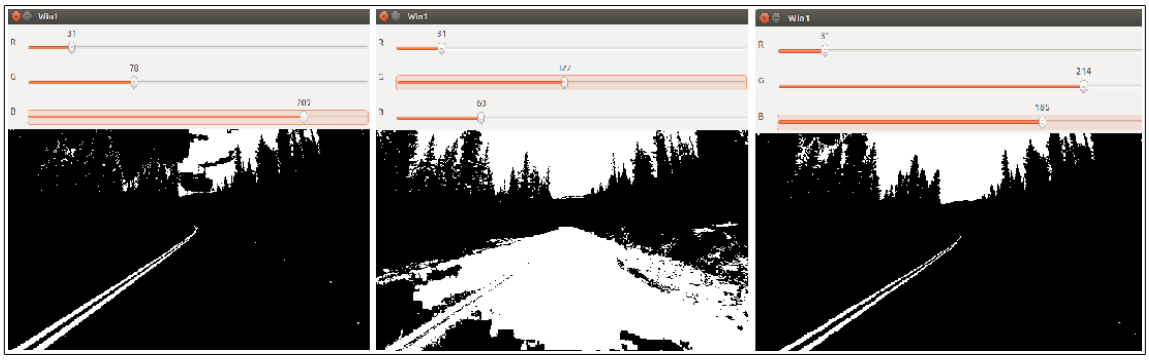

Figure 2.1: Applying intensity range threshold value (inRange)

The result of inRange operation is a binary image this has 0 or 255 values representing 0 as black and 255 as white value. From the Figure 2.1, can be recognize that road lane can be

parametrize use inRange filter operation and the location of the white pixel elements can be used to detect the road lane.

2.3 Drawing direction using Poly-lines

OpenCV provide some drawing and shapes methods which which easy to use in software application development. Existing shapes point can be converted into user-defined shapes as per the software application requirements. In this project work, a drawing shapes methods called *polylines( )* has been used to make direction shapes for guiding the driver to drive left, right or straight.

In the following *Example 4*, a straight line is designed from 6 points and which is drawn on the image frame as per the direction of the line extracted in an image frame.

```
//Example 4: Drawing straight direction

void gostraight (Mat &img)
{

int col =  img.cols/2;     // Starting point of drawing
int row =  img.rows/2+70;  // Starting row point

reg=Mat(img,Rect(col,row,100,100));
vector<Point> contour;

      contour.push_back(Point(50,80));  //1 Point
      contour.push_back(Point(50,20 )); //2 Point
      contour.push_back(Point(40,20 )); //3 Point
      contour.push_back(Point(55, 0 )); //4 Point
      contour.push_back(Point(70,20 )); //5 Point
      contour.push_back(Point(60,20 )); //6 Point
            contour.push_back(Point(60,80 )); //7 Point

const cv::Point *pts = (const cv::Point*) Mat(contour).data;
     int npts = Mat(contour).rows;
cout << "Number of polygon vertices: " << npts << std::endl;

polylines(reg, &pts,&npts, 1,true,
                Scalar(0,255,0), // colour RGB color value)
                  3,                 // line thickness
                     CV_AA, 0);
   }
```

In the following *Example 5*, a left side line direction is designed from 9 points and which indicate to drive left side in case there is left side road curve lane.

```
//Example 5: Drawing goleft direction
```

```
void goleft(Mat &img)
{

int col =  (img.cols/2)-20;      // Starting point of drawing
int row =  img.rows/2+70;    // Starting row point

reg=Mat(img,Rect(col,row,100,100));

vector<Point> contour;

      contour.push_back(Point(90,80));    //1 Point
      contour.push_back(Point(90,10 ));   //2 Point
      contour.push_back(Point(60,10 ));   //3 Point
      contour.push_back(Point(60, 0 ));   //4 Point
      contour.push_back(Point(40,15 ));   //5 Point
      contour.push_back(Point(60,30 ));   //6 Point
            contour.push_back(Point(60,20 ));   //7 Point
            contour.push_back(Point(80,20 ));   //8 Point
            contour.push_back(Point(80,80));    //9 Point

const cv::Point *pts = (const cv::Point*) Mat(contour).data;
      int npts = Mat(contour).rows;
cout << "Number of polygon vertices: " << npts << std::endl;

polylines( reg, &pts, &npts, 1,
          true,                // draw closed contour end to start)
          Scalar(0,255,0),           // colour RGB color value
          3,                       // line thickness
          CV_AA, 0);
   }
```

Example 6 is designed make right side direction from 9 points and which indicate to drive right side in case there is right side road curve lane.

```
//Example 6: Drawing goright direction

void goright(Mat &img)
{

int col =  (img.cols/2)-20;   // Starting point of drawing
int row =  img.rows/2+70;     // Starting row point

reg=Mat(img,Rect(col,row,100,100));

vector<Point> contour;

      contour.push_back(Point(10,80));     //1 Point
      contour.push_back(Point(10,10 ));    //2 Point
      contour.push_back(Point(40,10 ));    //3 Point
      contour.push_back(Point(40, 0 ));    //4 Point
      contour.push_back(Point(60,15 ));    //5 Point
```

```
    contour.push_back(Point(40,30 ));    //6 Point
          contour.push_back(Point(40,20 ));    //7 Point
         contour.push_back(Point(20,20 ));    //8 Point
          contour.push_back(Point(20,80));     //9 Point

const cv::Point *pts = (const cv::Point*) Mat(contour).data;
    int npts = Mat(contour).rows;
cout << "Number of polygon vertices: " << npts << std::endl;

polylines(reg, &pts,&npts, 1,
                true,                // draw closed contour
              Scalar(0,255,0),// colour RGB ordering
                3,                // line thickness
                  CV_AA, 0);
  }
```

2.4 Hough Lines Transformation

Hough Transform (HT) is a method for detecting horizontal and vertical lines in gray-scale or 2D image. The Hough Transform methods is extensive used in different type of application for extracting straight lines and it is computational very expensive operation. It simply detect lines aligned across the image which create some false detection to to an incidental pixel alignment [19] [27] [24].

```
//Example 7: Detecting lines using Hought Transform algorithm

void houghlines ( Mat &frame)
{
Mat graym,c_image;
vector<Vec2f> lines;
cvtColor(frame, gray, CV_BGR2GRAY);

Canny(gray, c_image, 50, 255, 3);

HoughLines(c_image, lines, 1, CV_PI/180, 150, 0, 0 );
for( size_t i = 0; i < lines.size(); i++ )
  {
    float rho = lines[i][0], theta = lines[i][1];

        Point pt1, pt2;
    double a = cos(theta), b = sin(theta);
    double x0 = a*rho, y0 = b*rho;

    pt1.x = cvRound(x0 + 1000*(-b));
    pt1.y = cvRound(y0 + 1000*(a));
    pt2.x = cvRound(x0 - 1000*(-b));
    pt2.y = cvRound(y0 - 1000*(a));
    line( R1, pt1, pt2, Scalar(0,0,255), 3, CV_AA);
```

```
    }
imshow("Win1", image);
    }
```

The method is expressed in x and y coordinates space (for example in rows and columns space) in an image. This allow to retrieve two end points of a single curve which may be point to the different line in an image and these points are stored in `Point` data structure available in OpenCV. For detecting relevant left or right line for lane detection method, all the points data structure need to check to extract the important end points which required more computations. In that case, in literature, the extended version method is used for the lane detection algorithms [18]. In the following *Example 7*, Hough Transform method is used to extract single line in an image. As shown in the Figure 2.2 two lines are extracted out of 431 lines detected by using Hough Transformation methods. The Hough Transform method was applied on single image frame. From this point, it is recommended that Hough Transform method is suitable for those application, where real-time performance parameter is not relevant.

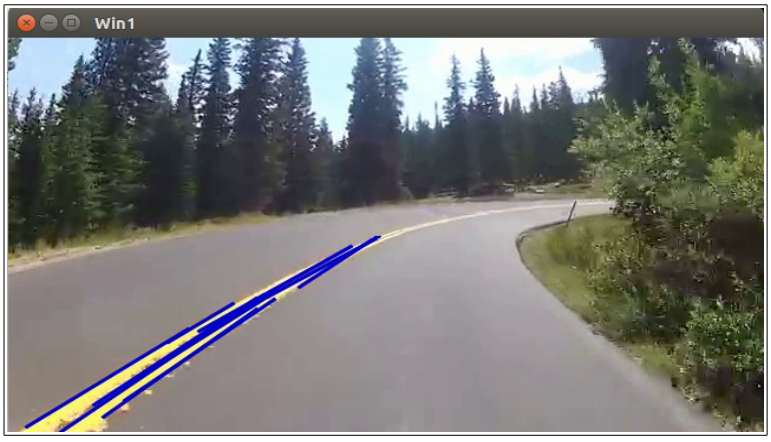

Figure 2.2: Lines extracted using Hough Transform

Detecting edges or straight lines in an image is the most common features of the modern computer vision applications. Historically, Hough Transform (HT) method has been used as the main means of detecting horizontal or vertical lines which is based on point-line duality. A point can be defined as a pair of *x* and *y* coordinates in image space.

The Hough Transform and contour detection methods mathematically elegant and are capable of detecting lines after removing considerable interfering noise but these methods are subject to severe computational problems because each point in image space gives

many numbers of votes in *x* and *y* coordinates. Lane model can be proposed on the based of more then two points extracted in image space. In general, a horizontal or vertical line can be drawn between any two points on any *x* and y coordinates in an image.

3 State of the Art

> "Learn from the mistakes of others... you can't live long enough to make them all yourselves!!"
> --- Chanakya, 370−283 BC

Computer vision-based applications implemented using image processing algorithms in combination with computer vision-based methods are commonly developed and tested on desktop based platforms, for example, image segmentation [54] a watershed transformation algorithms and contour lines recognition from commercially available printed topographic maps [55]. These platforms are provided with high performance processors, large scale memory and attached with direct power supply. The embedded systems based on image processing algorithms [66] and computer vision methods are finding their way into new embedded vision-based applications including automated inspection, advanced driver assistance systems, robots in manufacturing process intelligent vehicles and transport systems. The design and development methodology for embedded systems applications [25] are characterized with different requirements which are opposite to desktop based platforms [29].

This chapter provide state of the art general information of image processing research field, how image processing algorithms are used in computer vision-based application domain. It also discuss computer vision-based implementation are adopted in embedded vision-based systems in advanced driver assistance applications. In general, this chapter provide the overview of evolution of embedded vision systems from image processing and computer vision field. This chapter starts discussion with short introduction of image processing field, than computer vision-based systems, current state of the art of embedded vision systems (EVS), current challenges and advancements available in advance driver assistance applications in automotive industry.

The traditional problem of lane detection is to find the road lanes from the image frame captured by camera mounted in car and segments the lines after applying pre-processing steps and removing background and relevant noise. The state of the art methods in computer vision often formed such pre-processing steps as converting images in gray-scale, finding relevant region of interest (ROI) from the image, and applying some kind image processing algorithms to retrieve edges or line information and parametric data using

traditional Hough Transformation (HT) [19][27][24] or contour detections algorithms. These pro-processing tasks also creates many issues, including, first, the image frame reflects as highly congested environment with many different types of objects, sky, tree or other environment information which form as a kind of noise. Second, to find what is the notion of lane or line on the road and its properties from relevant region of interest portion of image frame. For example, in different road condition, lanes are in different color white or yellow, some time road marking are not clearly pained or missing road marks. Such conditions poses limitation on the set of available methods suitable for lane detection algorithms available in image processing and computer vision-based methods.

3.1 Embedded Vision Systems

The design and development methodology for embedded vision systems need to consider a real-time vision pipeline [20] in which involve hardware and software components, associated interconnection between input camera module to processing unit and memory size. The following section short overview of embedded systems real-time vision pipeline consideration and inter-relation between different components of embedded vision systems..

Image Processing: Image processing is branch of computer graphic research field, which provides a set of methods and algorithms used to enhance the quality of an image for further processing as input to computer vision based systems.

In general image processing algorithms can be distinguishes two levels, a low level image processing and high-level image processing [49]. The low-level image processing algorithms also called pre-processing algorithms usually used to perform basic image processing operations including converting image into different data structure, applying threshold methods, extracting relevant information like lines, circles, or different type of shapes and measuring image related parameters, for example, contour detection, Hough transformation, and image histogram calculations, etc. The high level image processing deals with the methods and algorithms in which the output of low-level algorithms is used as input parameters for further processing for designing and building computer vision-based applications, examples given in next section. Such algorithms are categories as image recognition or computer vision based algorithms and also partially belong to the area of artificial intelligence (AI). In general, image files are composed of multiple rows and

columns, in the form of one, two or three dimensional size as number of channels, required large amount of memory size and high computational processing power. For example, a single channel image with *1024x1014* pixels (1024 rows, and 1024 columns) each elements with color resolution of 8 bits per color requires 3MB of memory [49].

Computer Vision: Computer vision is a strongly multidisciplinary research area, and computer vision-based systems are becoming ubiquitous [50], integrating into embedded systems world including, games devices, smartphones and cameras, advanced driver assistance systems, robots used in manufacturing processes. Computer vision is the automatic analysis of images and videos by computers and has many applications in industry, for example, automatic reading of license plate [51], biometric security checks, inspecting product or industrial process [42] and guiding robots for complex product manufacturing [50].

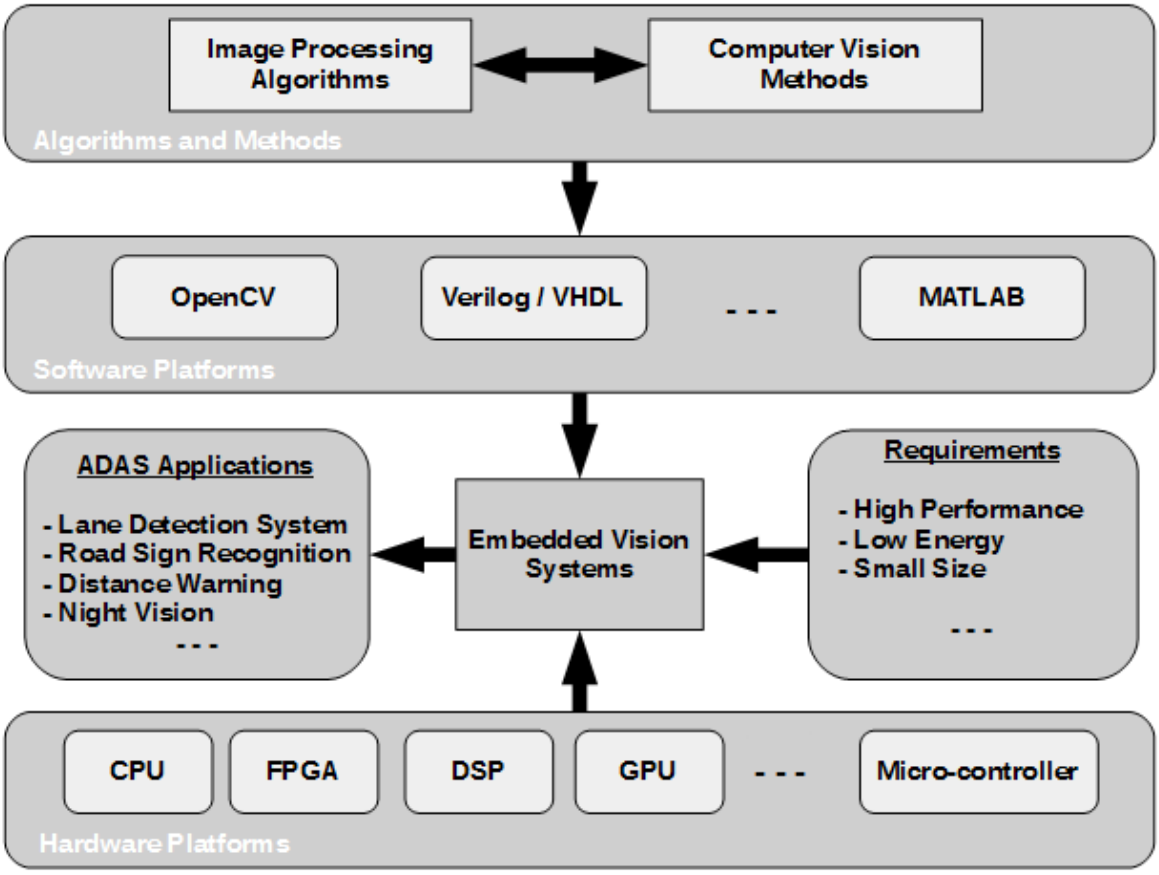

Figure 3.1: Embedded Vision System architecture

Preference concept allows integration of parameters to deal with disturbances by evaluation of situation by weighting of parameters [41]. Preference is used in image detection for

circle detection in chain of industrial carrier belt [42]. In research literature, both image processing and computer vision terms are overlapping each other and difficult to define clear boundary between them. In general, image processing methods and algorithms are used in computer vision methods. For example, intensively used canny threshold and smoothing operation techniques available in image processing field are used in computer vision-based systems to achieve application specific goals including lane detection systems and automatic number plate recognition system in automotive industry.

Embedded Vision Systems: The advancement of powerful hardware [72] in the form of fast and real-time processors [12], sufficiently large memory [22] and human machine interfaces, there is a strong and growing automotive market [53] [12] for advanced driver assistance systems (ADAS). Embedded computer vision systems a new discipline has been emerged within the image processing and computer vision community to deal with the new challenges called Embedded Vision Systems (EVS) [20].

A general architecture of embedded vision-based advanced driver assistance applications can be described as shown in Figure 3.1. The components involved in general architecture of embedded vision systems, shown in Figure 3.1, is designed on the based of current literature research and review in the area of image processing, computer vision and embedded systems implemented on vision-based architecture and platform. In general image processing algorithms are used in combination of computer vision methods using different hardware and software modules including OpenCV [57] [3] [58] [12] [17], hardware description language [52] Verilog or VHDL and MATLAB [17] [77]. These vision based applications has been designed and developed on different platforms including desktop CPU processor (ARM CPU processor [53]) [24] [60], FPGA [53] [59] and GPU based systems [13] [21], DSP (Blackfin 561 DSP Processor [5]) [16] [66] and micro-controller based processor[13] [66] [80]. The applications designed and developed under embedded vision systems methodologies has different requirement show in the right side of the Figure 3.1.

The applications in image processing and computer vision domain are characterized by high level of computational complexity, intensive floating point operations along with high volumes of memory requirements [25]. However, ADAS applications do not process only image data but also communicates with other hardware devices and interfaces [12]. These embedded platforms based applications has different requirements including, stringent size,

low power consumption, limited memory resources poses new research challenges in design and implementation methodology for embedded vision systems. Embedded vision systems need to consider the entire real-time vision pipeline [20] including capturing image frame from camera or sensor module, interconnection between camera module and processing unit, optimizing image analysis operations and realization of hardware components.

OpenCV Library: OpenCV, an open-source computer vision [43], library is written in C/C++ is a powerful tool for image processing and computer vision-based methods available for different systems architecture including Windows, Linux, Android operating systems and Real-Time Embedded Systems operating systems like Tizen IVI (Tizen In-Vehicle Infotainment) [12] is designed for computational efficiency with a strong focus on real-time applications. The OpenCV library is divided into several built in application specific modules as mentioned below [43].

- **opencv_core:** The core module contains the core functionalists the basic and dynamic data structures and arithmetic functions. For example, *Mat* data structure and operations etc.
- **opencv_imgproc:** Image processing functions and methods are available in this module including, image filtering, geometric transformation, image histogram calculations, object and feature detection.
- **opencv_highgui:** This module provide high level graphical user interface and media input/output interface, For example, reading and writing image or video files.
- **opencv_calib3d:** This module provide camera calibration, projecting 3D reconstruction view, two-view geometric estimation and stereo functions.
- **opencv_features2d:** This module provide common interfaces for feature matches description and detections, object categorization and drawing functions etc.
- **opencv_video:** The video module contains methods for motion analysis and estimation, feature tracking and foreground extraction etc.
- **opencv_ml:** OpenCV machine learning module contains a set of classes and functions for statistical classification, regression, and clustering of data.

Other important modules available in OpenCV are OCL module contain a set of classes and functions that implement and accelerate OpenCV functionality on OpenCL compatible devices and GPU module provide parallel implementation functionalists to utilize GPU

computational compatible devices including NVIDIA CUDA Run-time API. OpenCV can be also be used in combination with OpenCL for hardware acceleration of underlying heterogeneous computer platform. The OpenCV OCL and GPU module can be used for increasing the performance based on parallel implementation.

Computer vision has been the most popular domains of the embedded systems and automotive industry. In general, computer vision applications in their own domain are characterized by complex, intensive computations and requires more power consumption and memory size [25]. Automotive electronics and safety systems has some strict requirements to design and develop their products. For example, real-time performance, small size, low cost and long time reliability are some main requirements in embedded and automotive systems. To cope with such tight constraints, low-level programming language such as C or assembly language for software design, hardware description languages such as Verilog or VHDL has been used.

3.2 Embedded Vision-based ADAS Applications

The automated vehicle or self-driving vehicles will change like revolution the operations and experiences of driver which is long term goal of automotive industries. The ADAS (Advance Driver Assistance Systems) already include many safety and warning features which help driver follow the safe driver intrusions and to interact. Many computer vision-based advanced lane detection and tracking techniques [1] [3] have been designed and developed in a variety of environmental conditions. It can broadly be divided into three major categories [1] as described in the following section:

Region-based Methods: The region-based image processing methods [9] [3] [18] and [60], deal with limited size of region of interest (ROI) available in an image to extract relevant information about road lanes or objects in different road conditions. The region of interest based implementation of lane detection algorithms helps to reduce the computation time and increase the average frame rate. On other hand these methods creates some issues because there are chances that relevant information can be missed in decided optimal region of interest size.

Feature-driven Methods: Feature-driven method [3] [5] [6] has been applied in many FPGA and DSP based systems [66] and provide better qualitative results. In these methods, different type of lane detecting and tracking has been used including, vanishing point on

horizon [3], vehicle and road modeling [6] and Kalman filter based lane tracking method [1]. The extraction of the feature are based on pixel intensity values to identify the road lane and edge boundary of the objects. The main problem of these methods is removing relevant noises from input image frame including detecting connected components like sky, tree, or other un-defined shapes on the road, for example internal and external contour data extracted from an image frame.

Model-driven Methods: On the other hand, model-driven methods are based on geometric mathematical models such as linear-parabolic [24], vectorization of contour lines [55], Hough Transformation based line and circle detection [34] and selected region bands based lane analysis model [60]. These methods are suitable for modeling road and lane in combination of physical parameters of camera and sensors module.

Embedded vision based advanced driver assistance applications are growing rapidly in current market. The following section discuses a small number of embedded vision based advance driver assistance system applications available in current automotive industry.

Lane Detection and Tracking Systems: Lane detection is a crucial task to estimate the left-right edges of driving path on a traffic scene automatically. A single system composed with lane detecting and lane tracking is presented in [1] based on four-stages including (i) horizon localization (ii) lane region analysis (iii) adaptive line segmentation and (iv) river flow model. In first stage, pre-processing method used to retrieve and divide the image information in some real world defined scenes like sky, road, tree, and other objects. In second step, image is further divided into small region of interest (ROI) to classify the road and non-road on the base of pixels intensity range values. The third step involves applying some edge detection algorithms based on magnitude and orientation for extracting horizontal and vertical edges from the image. Using edge distribution functions adaptive line segment is used to split the near and far field regions.

In color image, pixels intensity values changes at very high ranges value which help to extract the road boundary and nearby field noise in the image. In general most of the time, pixels intensity values of the road share same range values in different road conditions. In this work, OpenCV in build function *`inRange(source image,lowerRange(), upperRange(), output mask)`* has been used to filter the image pixels intensity values and it help to extract white lane on road. The output of *`inRange( )`* is mask image file which is a 2D binary image each element of the image has 0 as black color and 255 as

white color. The element value of binary image is generated internally by apply some threshold operations.

The European project FP6-STREP-SPARC (Secure Propulsion using Advanced Redundant Control) has taken into account all the vehicle accidents and fixed objectives to drastically increase road safety. The SPARC project also developing the next generation trucks aiming to increase the vehicle safety in automotive industry [9]. In order to improve lane detection algorithms, a definition of the lane is proposed and several criteria are explained in detail. These definitions includes the aims of the camera based vision algorithm is to detect the position of the vehicle with respect to the road lane and perform the detection and tracking of obstacles [9]. To decrease the computational run-time of the algorithm, Lane Detection Algorithm (LDA) set to limited bounded area in camera space. The camera space is dived into 12 region of interest (ROI) region to keep the algorithm boundary and extract the required information from camera for further processing.

In project work, one method is implemented on the based of region of interest (ROI) to find middle point in the image space and extracting white pixels intensity values in that limited regions. In proposed thesis work only four region of interest as mask images has been used with maximum size *100x100* pixels. Region of interest (ROI) allow to set the bounded area for the algorithm. In OpenCV, in build function ROI has one more interesting feature that after processing and extracting the required information, computer ROI can be pointed or set easily to the original image.

Information and Warning Systems: Vision-based information and warning systems have appeared several years now and evolving a the basic assistance capabilities that are now becoming common as well. Ultrasound and cameras can extend the driver's perceptions car may break automatically in case of suddenly appear something in front of the car. Similarly, many drivers can now rely on autonomous acceleration and steering through a combination of front cameras and radar to keep the car in the center of its lane and cruise with traffic, even when speeds are continually changing. With the infotainment system to provide owners with a new degree of interpretive capabilities for, example, reading street signs or detecting objects around the car. Today cameras and interior displays could replace external rear-view mirrors, not only enhancing safety by eliminating the treacherous blind spot, but also streamlining the car's profile for improved gas mileage and reduced carbon emissions.

Object Recognition Systems: A complete object recognition systems consists of the following modules [11] [28].

- **Data acquisition:** In involves capturing data from camera module or sensors based module and transform received data correspondence to the training data sets which help in decision boundary as a functional mapping between the feature vectors and the correct class labels.
- **Pre-processing:** Pre-processing steps includes creating relevant region of interest, applying some binary operation to convert image in gray-scale and some relevant measurement related to the input image.
- **Feature extraction and feature selection:** This process applied to extract meaningful object and performing comparison operation with pre-defined object models.

Vehicle Localization: Vehicle's accurate position information obtained from affordable off-the-shelf GPS receivers is limited due to physical constraints and such information data render an essential part of upcoming automotive technologies such as vehicle- to-vehicle (V2V) communication [2]. Such new technologies offer new possibilities for vehicle-related applications in the domain of comfort and safety feature. A novel measurement model based on Bayesian filtering for lane accurate localization system is designed using sensor module and including camera sensor, GPS receiver and LDM (Local Dynamic Map) which are already available in the series vehicles to increase the technical and practical feasibility. The position information which is used by navigation systems generated from GPS signal based devices and correlation with information obtained from current motion state of the vehicle depend to a great extent on navigation systems.

In proposed thesis work, position of the vehicle is obtained using manipulation of pixel values in the image and region of interest. To extract the middle point and starting for lane detection, the lower portion of the image is divided in four region of interest and the mask operation is applied to get total number of white pixels.

Collision Warning Systems: The collision warning systems in automotive industry is an important active safety systems based on active sensors, however, active sensors have several disadvantages including low data acquisition speed, low resolution and high costs. On other hand, in past decade the rapid increase in the number and availability of smart and low cost camera modules[29], embedded vision systems has emerged as an area of

intensive and concentrated research in both the academic and commercial settings. This has resulted diverse and ever-expanding range of applications, including lane detection and tracking systems [57], [61] object detection [67], collision warning systems and pedestrian detection systems [68].

In this project work, the goal is to find the lane points from the 2D (two dimensional) mask image which is a binary image file containing 0 or 255 (0 as black color and 255 as white color value) values based on pixels color intensity values available in source image frame. Given a three channel color image captured from camera or video file, OpenCV provide some functionalists to create mask image after applying some filter and range methods including lower and upper color values. Hough Transform and Sobel operator method has been used in [27] [24] for lane detection using feature matching algorithms based on geometric model, extracting a set of control points of lane model, and detecting vanishing point using extended Hough transformation technique.

Pedestrian Detection Systems:

Pedestrian detection and tracking techniques widely applied is sub problem of object detection and currently widely applied in large number of intelligent transport systems, automotive advanced driver assistance application and other safety and security area including military, warehouses and high security area. A low-dimensional soft-output SVM pedestrian classifier, a pedestrian detection and tracking systems, for counting number of person and providing risk warning has been implemented using OpenCV library on Intel i3 processor under Microsoft Windows Visual Studio 2010 [78]. On other hand feature based pedestrian detection and tracking system, a vehicle prototype which offer high performance in term of several matrices, has been implemented using Haar cascade classifier trained with the Daimler Detection Benchmark data set and histogram-of-oriented-gradient (HOG) classifier based on OpenCV library [68].

3.3 Advancement and Challenges in EVS

The following section provide, a state of the art advancement and challenges currently available in embedded vision based advance driver assistance system applications in automotive industry. The statistics [15] indicates the majority of the accidents are caused by human error and it also shows that adding vision based applications and intelligent systems into vehicles can reduce traffic accidents and save the lives [13]. The advance

driver assistance systems that monitor the driver's intention, alert driver in many uncertain conditions of the vehicle by warning message and assist in vehicle guidance, are all being active systems need to work in different environment conditions. These systems also need to fulfill strict requirements [2] including providing high performance, high average frame processing rate, low power consumption and small systems size. On the top these requirements need to achieve in different type of road lane condition and environment to analyze and extract correct information for intelligent decisions [35].

Vision-based automotive safety and warning system applications are growing in current automotive industry [62] [16]. Computer vision-based lane detection and tracking applications designed and implemented in OpenCV based library has been already proposed by many educational research communities [12] [17] [18] [58]. The automotive commercial organizations now trying to port vision based applications from vision-sensor based applications including, as radar, LiDAR and GPS signal, into OpenCV based implementation [16] [53]. Image processing algorithms has been designed, implemented and tested on Android based operating systems on embedded vision platform using OpenCV library [56]. The main advantages of software based libraries application over sensor vision based application are the availability of low cost and high-resolution cameras and abundant information contained in video images. At the same time, computer vision based automotive applications also facing many challenges in porting their algorithms in software-based vision systems. One of the most astronomic issue lies in performing real-time algorithms on low cost and small size embedded systems within hard real-time constraints [29]. In the following section, new trends or advancements which are going in embedded vision-based advanced driver system applications are discussed.

Trend 1: Integration of Embedded vision systems

Due to the emergence of advanced technologies including low cost powerful processor, for example ARM Cortex A7, smart camera modules and heterogeneous computing environment, it has become possible to incorporate computer vision capabilities in to embedded systems [10]. For developing advance driver assistance system and safety applications, the most essential requirements for the embedded vision software is to deal with the real-time image processing capabilities which need to be performed in different environment and conditions. For that camera module and frame data needs to process and distribute to high-level vehicle control systems those responsible for activating the safety

and protection applications, for example, steering and braking and warning signals in form of audio or visual effects. Existing automotive industry standards such as CAN (Controller Area Network) and Ethernet (Audio/Video Bridging), MOST (Media Oriented Systems Transport) and FlexRay already playing as keystone interfaces used to distribute sensor and video data, and control operations across different vehicle subsystems [32]. Commercial organizations are turning to FPGA acceleration, for example Xilinx Kintex Ultrascale, are able to provide performance greater than 14 frame per second results, making them a great choice for to achieve the highest performance in the data center [10]. In [12], both lane departure warning (LDW) and driver fatigue detection DFD) systems implemented on open source based Tizen IVI platform using OpenCV libraries and described the issues in developing advanced driver assistance application based on FPGA systems. The run-time of both applications is measured as round 8.201 ms for 320x240 pixel image. A technique for human computer integration using OpenCV library has been proposed and implemented based on image processing methods in python and Matlab [77]. A robot control based on embedded vision system platform using OpenCV library has been designed and implemented on different hardware components including TS7800 embedded board, LCD displays, USB and RS232 ports and parallel implementation achieved using OpenCV library [80].

Trend 2: Involvement of High Performance Graphics Modules

The availability of low cost and high performance hardware, smart camera are enabling a wide range of innovative applications in the area of embedded vision system based platform [72]. Human face detection applications has been designed and implemented based on efficient texture analysis and image processing algorithms based on heterogeneous CPU+GPU platform for accelerating parallel computations [21]. OpenCL [40] is an industry standard for writing parallel programs targeting heterogeneous platforms, including OpenCL compatible desktop based CPU, FPGA, GPUs, and APUs. In [46] study shows that certain vision-based automotive applications can be ported to GPUs based systems which perform four times better in comparison of CPU based processors. At the same time study also posses some issues regarding high power consumption rate and mapping the multi-thread parallel implementation to execution core elements called processing units or processing elements. The solution, to reduce the power consumption resources, can be achieved if more than one application can be implemented on single

GPU. FPGA based heterogeneous SoCs, including Zynq-based Zedboard, Vivado HLS, SDAccel from Xilinx and Altera (OpenCL) are providing supports for OpenCV based applications [72].

Trend 3: Smart Cameras and Sensors

The camera based sensor technology, called CMOS (Complementary Metal-Oxide Semiconductor), is increasingly being used in camera based device like smart phones and digital cameras providing high resolution image, consume less power, delivering high performance and advanced features [26]. The small size and less expensive CMOS active-pixel sensor (APS) has been appeared an alternative CCD (charge-coupled device) cameras. The relative simplicity of CMOS-based cameras consume significantly less power, while offering higher levels of integration and a lower overall system bill-of-materials as compared to CCD-based systems. The integration of all camera functions on a single chip, significantly reducing component count, cost and board space allows for quick and easy application designs, enabling faster time-to-market for systems manufacturers [36]. The AIA (Automated Imaging Association) classify the embedded vision products in the following three domains.

Smart Camera: ADAS systems use a range of sensors and camera module including vision sensor-based (RGB sensor,), to capture information about the surroundings for implementation purpose for ADAS functionalists [47]. The automated imagining association refer smart cameras as a complete vision system in which basic image processing and software programs will be included in the camera body. For example, The National Instruments Corporation offer *NI 177x High-Performance Smart Cameras* are industrial, high-quality image sensors combined with powerful processors, all-in-one solutions for embedded and machine vision-based applications. These devices are designed

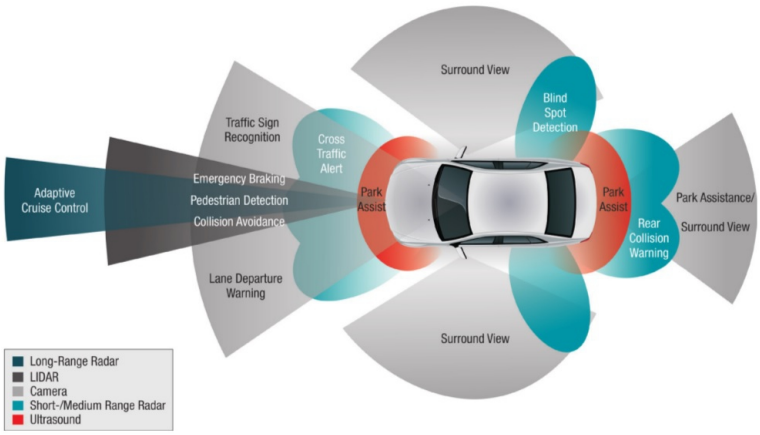

Figure 3.2: ADAS systems use camera and sensor based applications [16]

for easy to integrate with other programmable automation controllers and human machine interfaces as well as work with the entire suite of vision algorithms in the vision software platform [37].

Vision Sensor: Vision Sensor are a lower end smart camera which will give same performance as a high-end smart cameras. These self-contained vision sensor are robust and easy-to-use perform automated basic pre-processing operations (like filtering noise and meaningful data and signals stream) on input data and the results are forwarded to the next systems for further processing. For example, Vision Sensor product *PresencePLUS P4 Sealed OMNI*, available from Banner Engineering Corp Limited [30], a 640 x 480 CCD imaging, low power and small size device, performs inspections up to 2,000 parts per minute (PPM), available with direct connectivity to industrial standard network protocols included standard vision tools for locating geometric based information, edge and object detection, provides results in measurements.

Trend 4: Embedded Vision Processor: The growth markets of embedded vision based systems clearly show the need for a dedicated vision processor to carry out the image processing and vision tasks providing high performance result in a power-efficient manner. Moreover, image processing algorithms applied on embedded systems consume too much power which are mostly attached to the external battery. A new intelligent vision processor, for example CEVA-XM 4 systems-on-chip (SOC) [69], has been designed for on board range of computer vision processing application in smartphones, tablets, automotive safety, robotics, and infotainment. The XM4 intelligent vision processor shows huge gap in energy efficiency of 20x on average in comparison of mobile GPU [69]. Embedded vision processor is a small computing a system-on-chip system in which a camera can be attached to reduce the data communication operation cost and to increase the computational processing for high data intensive applications [22][36].

Trend 5: Heterogeneous Computing

The modern parallel computing and multi-core processing has changed the computational computing platform called heterogeneous computing paradigm in which multiple devices on a single systems can be used for concurrency and parallelism to improve performance and power issues [40]. Open Computing Language (OpenCL) is heterogeneous programming framework developed specifically to support multiple devices in single systems with the goals to bring together in a single device the best of both central

processing units (CPUs) and graphics processing units (GPUs) [65]. Support for OpenCL is rapidly growing as wide range of hardware platform venders including NVIDA, AMD, Intel, IBM, and Apple. The concept of *concurrency* in heterogeneous environment is called *kernels* is specially a functional code that is intended to be executed by one or more OpenCL compliant devices which form *host application* and *device application* relationship [65]. A host application interact with devices, manage their connected devices, create kernel functions and maps the kernel argument data to the associated devices.

Trend 6: Vehicle System of A Systems of Internet of Things (IoT)

The platform for the development of Cooperative Intelligent Transport Systems (C-ITS) in European Union was launched by the Commission in July, 2014. The objective of the C-ITS is to identify and determine most suitable deployment methods, including frequencies, hybrid communication, wireless network security, access to in-vehicle data V2V and V2I services to improve potential road safety and efficiency of road transport [38].

Trend 7: HMI in Automotive Domain

Advanced Human Machine Interface (HMI), a graphics-based visualization and monitoring system offer rendering capability found in today's graphical user interfaces. These technologies have changed the control panel layout from a rather motionless to a more flexible, dynamic and adaptable design. Smart applications have become available, providing new channels to give information to the driver. Voice-based HMI lower driver's distraction when commanding the vehicle or its options, for example in navigation devices and radios or mobile phones, provide direct access to the control, help driver in looking at the road and keep hands on steering, increase the driving performance.

> "*Cars are the biggest and oldest mobile devices... the face of mobility. We've [Ford] been around for over a century. But we welcome the competition from newcomers like Apple and Samsung.*"
>
> John Ellis (Head of Ford's developer program) at CES 2013

The automotive industry adopting HMI based technologies and benefits from the fast pace of technological advancement in the mobile and software space. In comparison with old model of cars, the modern cars are occupied with HMI -based user interaction and safety feature [77]. There are many more controllable elements within cars, numbering in the hundreds navigation systems, telematics, ride control and infotainment systems [23].

The following section provide short discussion on the challenges currently based by

automotive industry for applying computer vision-based methods in their safety and driver assisting application.

Challenge 1: Power and Energy Consumption

In educational research institutes and commercial organizations, energy consumption and internet of things (IoT) is always a hot field of research topic. Embedded systems technology is promoting the fast rise of the so-called Internet of Things billions of smart, connected communicating with devices machines, environments and vehicles. Even the applications which deliver service to the consumer required to process intensive data, need to perform communication operations and more to meet marketplace demands like real-time performance. The greater functionality also demands more complex, energy-consuming processing from battery powered and "plugged" devices alike. There are two most important considerations need to keep at the top preference list before designing embedded vision-based systems are reduction of energy consumption to maximize the battery life [25] and reduction of the area requirements to minimize cost and size.

Challenge 2: Measurements and Standardization

The embedded vision systems industry lacks the taxonomy evaluation models, benchmarks [71] and standards for embedded vision-based processing methods. Their are some existing standards available to measure and quantize the vision-based application in the area of image processing application but automotive industry is lack of any such standards.

During a U.S. Senate hearing, Duke University robotics professor Mary Cummings testified that

"There is a lack of "principled, evidenced-based tests and evaluations" for autonomous cars, ….. little information about automakers' test plans and data is available today for experts to measure the performance of self-driving cars...."

A hearing titled "Hands Off: The Future of Self-Driving Cars" on Tuesday, March 15 2016

There are some pre-defined benchmark data-sets available from some commercial organizations[5] and research institutes[6] for testing benchmark data set of vision-based applications [50]. The industry standard OpenCV based implementation of computer vision methods are leading to standardization the mobile imaging architecture [63] from SIMA (Standard Mobile Imaging Architecture). The international G3 started an initiative on standardization process, including its automotive and mechanical engineering industries

5 Daimler Pedestration Detection Benchmark Dataset
6 CamVid, Cambridge-driving Labeled Video Database

members and Machine Vision industry, launched a study group on embedded system standards to evaluate requirements, to identify technological candidates, and to describe possible implementation concepts for machine vision standards for embedded systems [64]. In [71] a flow taxonomy of embedded vision-based systems is proposed to characterize the vision functions and methods including from image capture stage, pre-processing methods, interconnection and final classification of the result after performing post-processing steps. A pixel-based measurement evaluation, a novel open-access data-set and benchmark for road area and ego-lane detection, has been introduced [74] for road area and ego-lane detection.

Challenge 3: Integration of Multiple Assistance Systems

The driver distraction issue can not be solved only by apply advanced HMI technologies, also need to manage HMI resources with other sub-systems components and application running jointly in advance driver assistance, integration of driving assistance, infotainment and informatics data and resources. On the top of all these requirements, advance driver assistance application imposing high level issues like scalability, flexibility and adaptability [6][7][13]. To resolve tight constraints on performance, cost and typical hard real-time requirement of embedded systems, most of the designers use low-level programming such as C for software design and Verilog or VHDL for hardware design [25]. The idea to implement a whole advanced driver assistance systems in FGPA or GPU is not a feasible solution because these architecture are not suited for high-level decision and serial processing tasks [13].

Challenge 4: Development Time to the Market

The design process for embedded vision systems need to have depth understanding of unique characteristics of the application and associated implementation constraints [20] [27]. Embedded vision-based implementation and design methodology is a critical and challenging task due to increasing complexity in integration of different hardware and software components and targeted operating systems platforms [25].

> "*Ask five people to develop a lane departure warning system and you'll get five different solutions. Each will likely start with a Matlab implementation that is ported to run on the selected hardware. If the developer is fortunate, the silicon will support image processing primitives to accelerate development.*"
>
> Tina Jeffrey, Article: The challenges of developing advanced driver assistance systems, New Electronics, Findlay Media Ltd, 07 November 2013

There are many commercial and open source tools available for creating and debugging software application design, from higher level of abstraction, are creating other issues like increase in the development time of systems product [8].

4 Concept Design and Methodology

> Don't be satisfied with stories, how things have gone with others. Unfold your own myth.
> --- Rumi

In this thesis project proposes lane detection methods based low-level image processing algorithms applied in the area of computer vision-based systems by analyzing the intensity pixel elements value and applying recursive contrast analysis on limited set of points for road lane model. The lane detection methods presented in this works use low level image processing methods including *inRange* binary threshold, based on pixels intensity values to extract the white pixels elements. The proposed design flow relies on the following main concepts. This section provide overview of these proposed lane detection methods and algorithms components.

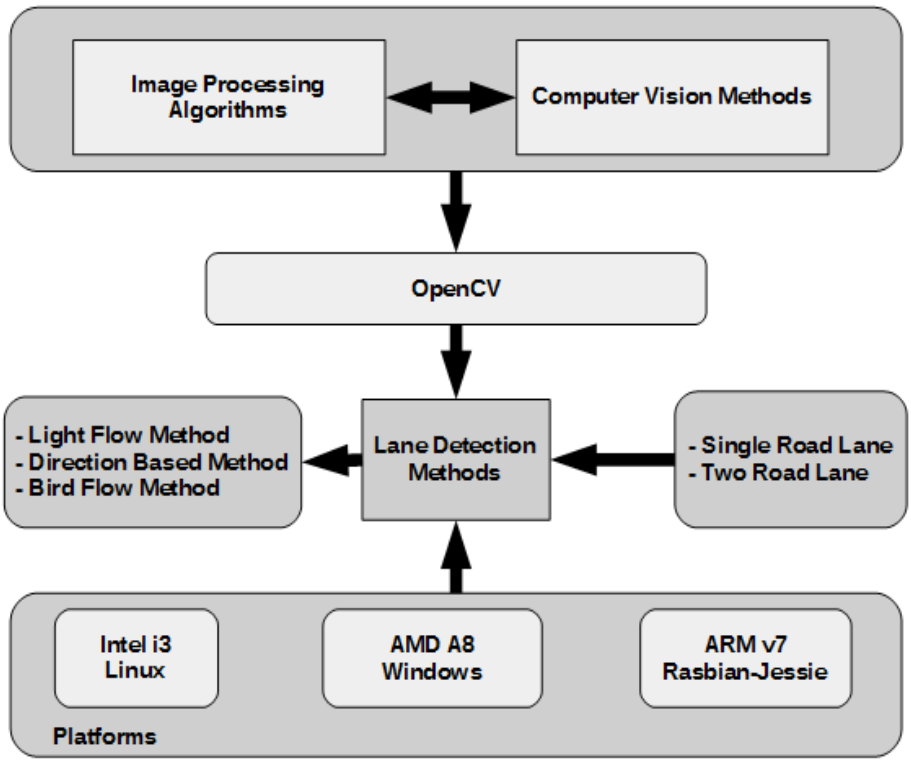

Figure 4.1: Lane Detection Methods architecture

In this thesis project, proposed lane detection methods are implemented in OpenCV based on C/C++ under Microsoft Visual Studio and CodeBlocks IDE and tested on different operating systems and processor including, Windows with AMD A8, Linux based Ubuntu

with Intel i3, Rasbian-Jessie available with AMR v7. The performance evaluation result obtained from these systems have been discussed in chapter 6. Figure 4.1 shows proposed lane detection methods architecture based on image processing algorithms and computer vision methods implemented in OpenCV and tested on different platforms. The main goal of detecting edge or line in an image is to find where the value of pixel intensity gradient g is changing which reliably showing different between two objects in an image space.

In [41] preference concepts is presented which is applied to combine direction, pixel color delta and maximum y-movement width in a preference for moving from measurement point to lane. In this thesis project work, binary mask image, which is converted using range threshold to covert color image in gray-scale image, is used for detecting relevant road lane points.

Figure 4.2 shows, general architecture used in proposed lane detection methods. Each method start with extracting frame from video file and save the captured image in frame. In case there is not any frame available in the video file, method exit with short message indicate that there is no more image frame available.

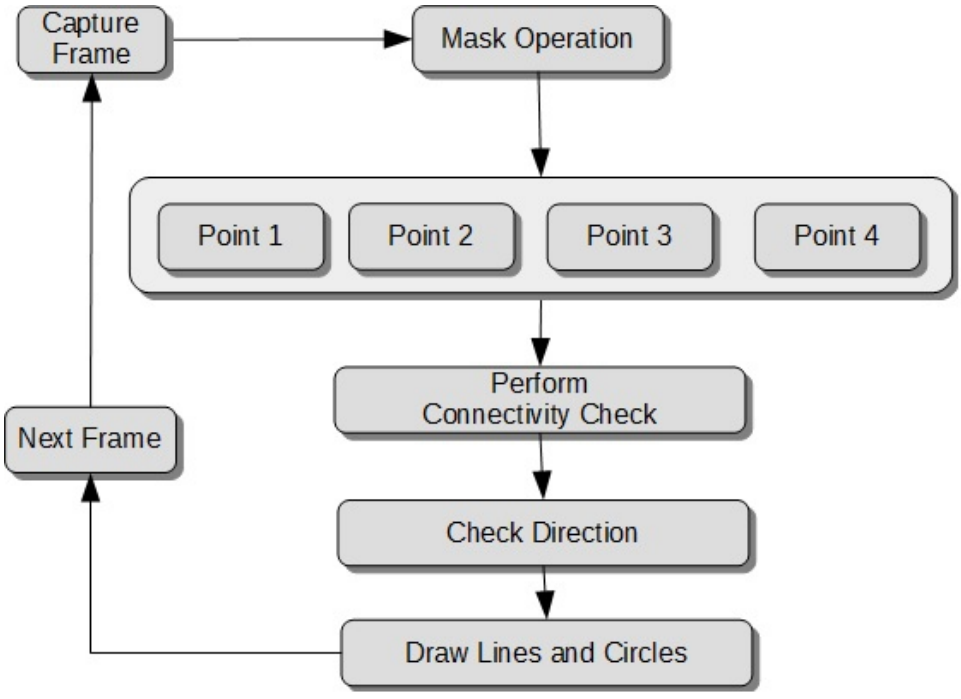

Figure 4.2: Basic concept architecture of lane detection methods

Once an image frame is available and ready for pre-processing step, a mask frame is

created by applying range threshold on input frame.

The basic concept architecture of proposed methods is shown in Figure 4.2, which described the overall flow of the methods. Each method start with extracting new frame from video file and a mask operation is applied on the available frame. Four relevant Points are extracted by applying any proposed method and a connectivity check is performed on each extracted white point. On the base of extracted four points and their position and magnitude direction assistance method is applied for providing indication of road direction. Once connectivity check and direction assistance procedure is completed then lines and circle are drawn between each extracted points indicates as road lane in an image and at the end methods check for next frame and apply same procedure for next available frame. The following section discussed basic architecture concept of lane detection methods proposed in this work**.**

4.1 Light Flow Method

Light flow algorithm is result of evolution of Direction Based Method. The following points are considered in this methods. The algorithm name “Light Flow Method” is given on the base of planar projective transformation [48] of the image. The projection of light looks like planar projection on the road and it cover the white road lane in the light flow with limited left and right width range. It is believed that the continue white lane on the road has some geometrical properties. The white pixels in binary image frame create some kind shapes like circle, lines or counters. The pixel elements of these shapes are connected components which has some boundary width which is used go extract edges of the shapes. The pixel intensity within each component has same intensity value (255 as white) which help to differentiate from background and object/shapes available in binary image.

This method start with retrieving frame from video file and apply mask filter operation with pre-defined filter parameters based on RGB values to create binary mask image as shown in the following code.

```
inRange(image, Scalar(0,210,148), Scalar(255,255,255), mask);
```

The Scalar is RGB color data structure available in OpenCV, inRange methods filter the source image (image) between lower range of color values (Scalar (0,210, 148) to Scalar (255,255,255). The parameters of Scalar data are color values in form of RGB. At the end

of the this operation, output binary image file called mask is created which has only 0 or 255 intensity value for each pixel element according to the filter operation. The binary mask image help to find connected components or objects and detecting edges of the object from an image. After the mask operation, `gonext()` function is performed to extract four relevant points which form the road lane in mask image. The `gonext()` function extract first points after performing connectivity check (connected components as road lane pixels). Once connectivity checked is perform then extracted point coordinates are store in the road lane model data structure which is later used to draw the lines between four points. Once first relevant point is extracted it start searching second point within limited left and right side width. The limited left and right side width is pre-defined assuming that the next road lane points will be existed within limited range of the first point.

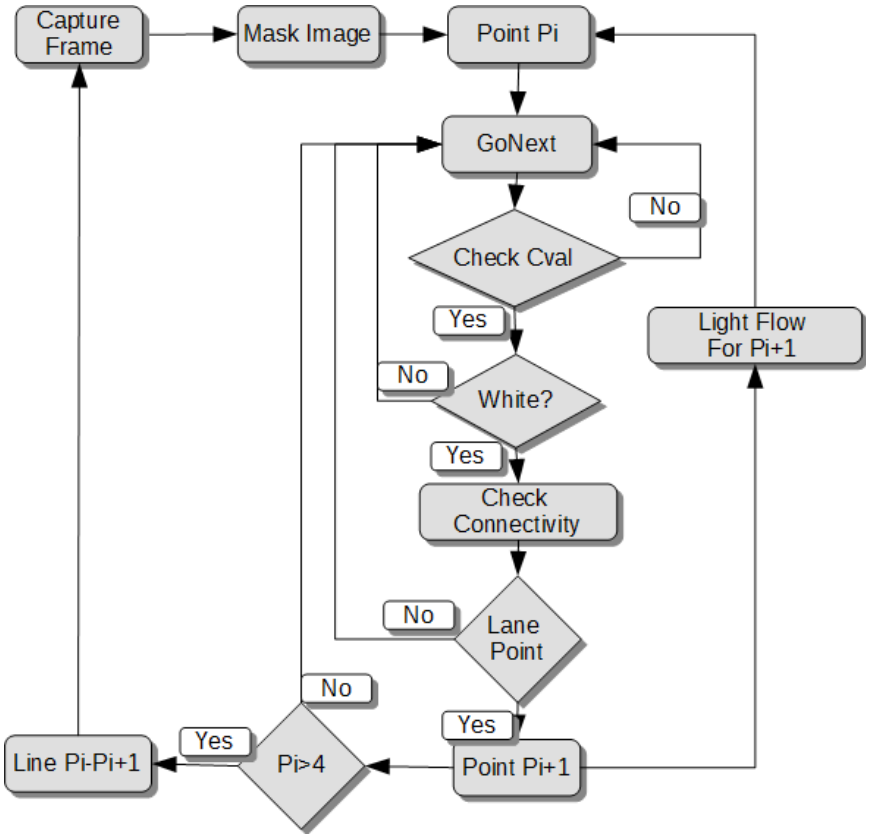

Figure 4.3: Basic architecture concept of Light Flow Method

In case of first image it start traversing for white pixel from right middle of the image. Once four relevant points are extracted in previous image frame it start applying bounded left and right width procedure. Instead of starting searching white pixel intensity value for P2, P3 and P4 at specific `Point (x,y)`, a new upper and lower bounded range in `Y` axes

direction can be granted on the base of information obtained from previous P1 `Point (x, y)`. The basic flow of the Light Flow Method is shows in Figure 4.3.

The functionality of bounded left and right width for P2,P3,P4 help to reduce the computation steps and time and provide robustness in extracting point closed to the previous point. The mask frame is used for extracting four relevant lane points by using *`gonext()`* function available in implementation detail. This function apply recursive operation to extract points on the base of row and column (x axes and y axes values) values given at the starting point of the methods. In case *`gonext()`* function find and relevant points in mask image, then connectivity check procedure is applied to confirm that a set of connected pixel elements form some kind of road lane.

In case the connectivity procedure recognize that current pixel elements is not suitable for lane model, it discard the current point and continue to perform traversing process for searching white pixel elements from mask image frame. Once relevant number of points (in proposed methods, four points are used to make sure that extracted four points lie on the road lane) are extracted from mask image frame, a direction indicator procedure is applied for provide direction assistance to the driver. The direction procedure is used on the base of location of four extraction points and direction can be in any one of the following side straight, left or right direction. The output result of the Light Flow Method on single image frame is shown in the Figure 4.4. The output result of Light Flow Method applied on single image frame shows the four extracted *`Points`* in blue circle reflect as road lane. Four red color circles shown as starting points for each blue circle lane point. The light blue color lines starting from each dark blue color circle, growing to left and right direction, shows the left and right side width for traversing width for next lane point. For example, first blue circle create the boundary width for second blue circle to

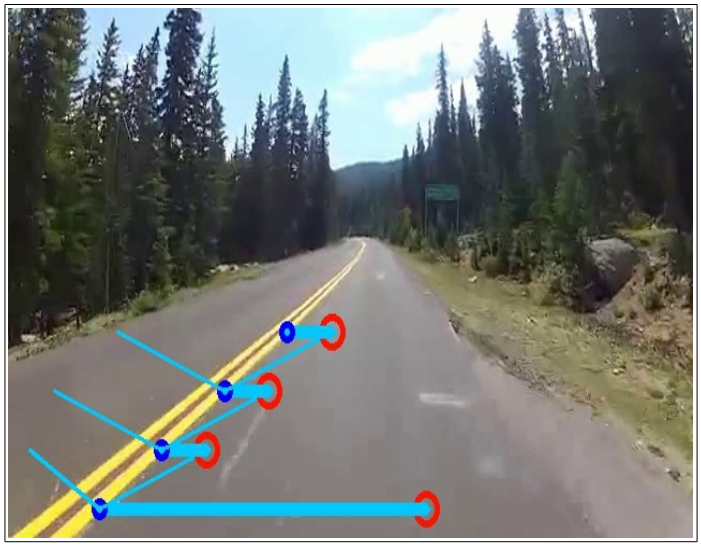

Figure 4.4: Output Light Flow Method

reduce the computation time. The second Points start again from red circle and traverse to left side until it hit with any white pixel. The same procedure is applied for rest blue color circles.

4.2 Direction Based Method

Direction based method apply recursive functionality for extracting four relevant points from mask image frame. First it check the `ydepth` value if is out of image `x` and `y` `axes` range or not.

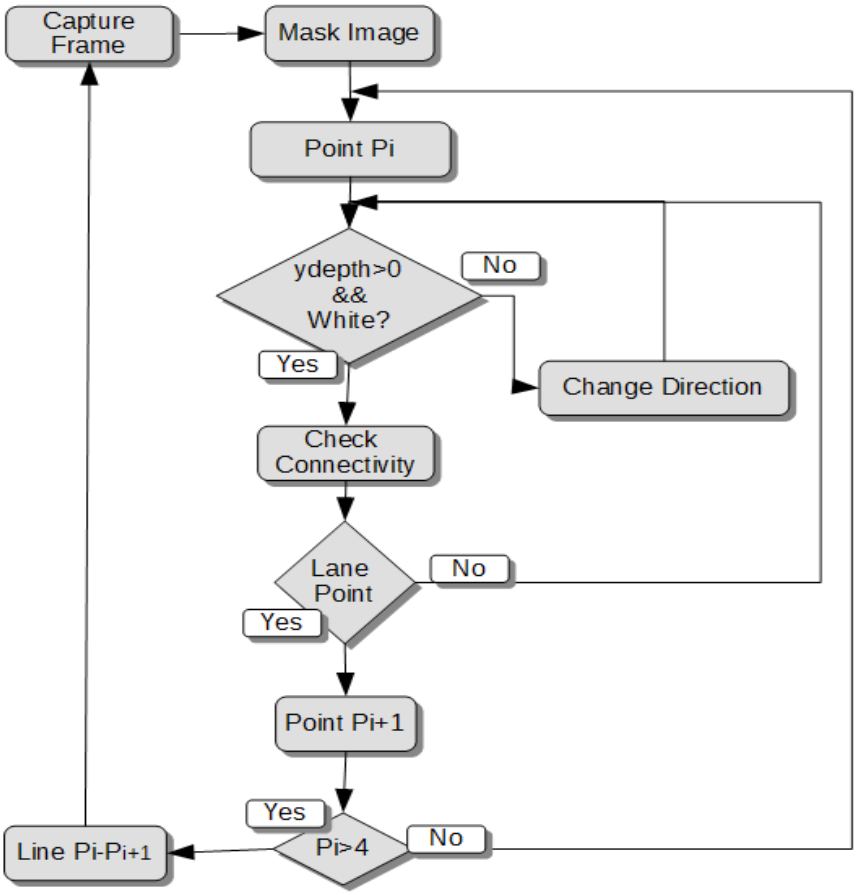

Figure 4.5: Basic concept architecture of Bird Flow Method

Then it perform second step to check if the current pixel element value is within the white range or black. In case it hit with white pixel element values it change the direction to left side for the confirmation of lane connectivity check. After performing connectivity check extracted current `x` and `y` values is updated in pre-defined `Point` data structure. The overview of Direction Based Method is shown in Figure 4.5.

As shown in the Figure 4.5, “Change Direction” steps change the direction on some basic parameters including, the value of current *mask.at.<Point> (x,y)* pixel element, *ydepth* value and current value of *dir* variable. Change direction functionality in this method play an important role for extracting road lane points and to confirm that extracted four points are creating a road lane. Once first point is extracted from mask image, it start from same `x axes` value but `y axes` is increased to make distance between two points. A direction indicator is applied and shown in form of text on output image frame based on the average coordination value of four extracted points. The direction indicator is shown in the text for as “Move Right--”in the following Figure 4.6.

Figure 4.6: Output of Direction Based Method

The connectivity check performed in this method is also based on direction of current white pixel value. It traverse left one pixel element from current x axes and y axes coordinate. In case current values of pixel element is still white, than it traverse two pixels down and left. From there it perform again white pixel element check and in case it hit white pixel value, it traverse two pixel down. The detail information about implementation of connectivity check can be found in next chapter. Figure 4.6 also shows the current point coordinates extracted for each road lane points. For example, Point P1 has 176,332 coordinate value (176 as x axes and 332 as y axes value) which is relevant point for road lane extracted after applying connectivity check.

4.3 Bird flow Method

The following section provide detailed concept overview of Bird Flow Method proposed for two road lane detection (left and right side road lane) problem. This method is extended version of both Light Flow Method and Direction Based Method. This method is used to detection left and right side lane on the road. It apply pixel element traverse procedure as

described in the previous section. It start from the right middle point at constant x axes coordinates value and check for left side white pixel element value until the value of x axes is greater than 20. Once it hit any white pixel element, it start traversing to right side for road lane point. Once in first step, both left and right side relevant point value is found (white pixel element). The point coordinates values of both left and right side is updated in separate pre-defined Point data structure.

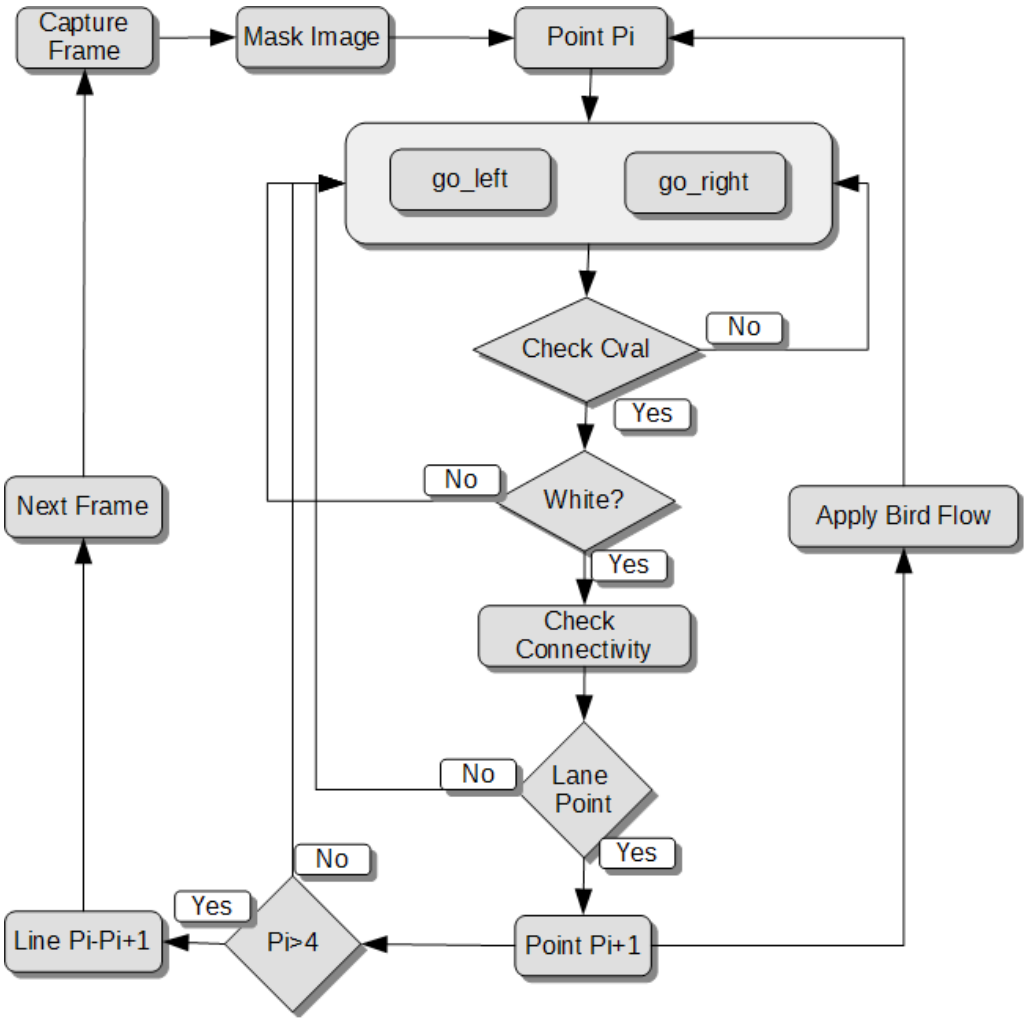

Figure 4.7: Basic concept architecture of Bird Flow Method

The Point data structure is updated only after performing connectivity check to confirm that the extracted points are not single white value but a set of connected component which form a road lane points. The following Figure 4.7 shows basic concept architecture of Bird Flow Method. It take binary mask image as input parameters for detecting road lane points. The `go_left` and `go_right` function are used to update Point data structure for four

road lane on each side.

The function *go_left* traverse to left side for searching white pixel element value. The function *go_right* perform same steps but to the right direction. Once both function hit with white pixel element value they perform connectivity check and then Pi is updated. This method start extracting new image frame and perform same steps for next four Points.

The following Figure 4.8 show output result obtained on single image frame after performing Bird Flow Method. The blue color circle shows four extracted points on left side road lane and green color circles shows four extracted points on right side on the road lane. The middle four shows the starting coordinates at which *gonext()* function start traversing left and right side for detecting road lane in an image.

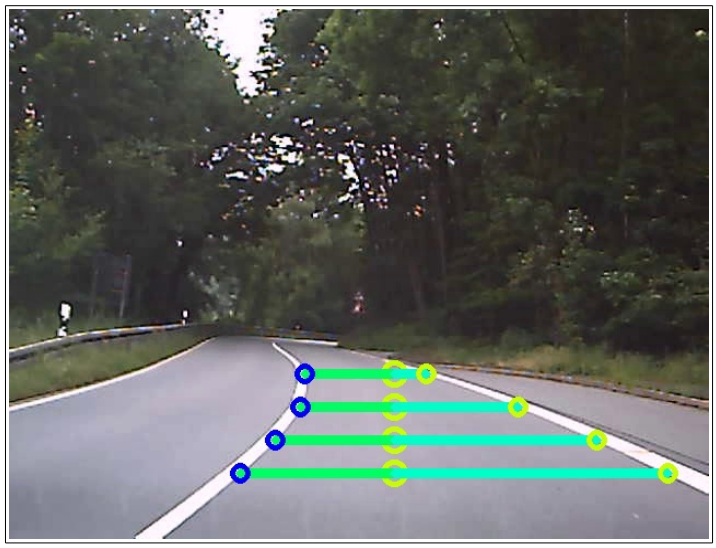

Figure 4.8: Output of Bird Flow Method

It apply connectivity check procedure from Light Flow Method as described in the previous section. The limitations of this method including it perform lane detection steps only on straight lines available on the road. The road lane which are not in continuous form is not best input for this this method. A small modification in the proposed method will help to detect road lane which are not in continue form. The second problem of this method is that, it is possible this method will not detect road lane point on one side but it detect on other side. In this case draw a straight line between extracted four points shows not good result. A minor improvement in this method is to apply length between two points existed on same y axes value and coordination between these two points will help to improve effectiveness for road lane detection problem. Figure 4.8 shows the four green color circles available in the middle of each road lane points. These green color middle circle reflect the starting point for of *go_next()* function. From this point, *go_next ()* function traverse to left and right side which is shown in different color.

The implementation details of these proposed methods has been give in the following chapter which provide concept detail of connectivity check, the procedure steps executed in extracting single and two road lanes in each method.

5 System Architecture and Implementation

> Worry is like a rocking chair; it keeps you in motion, but gets you nowhere...
> --- Chanakya, 370–283 BC

The following section provide implementation architecture detail of the proposed lane detection methods. This chapter provide how relevant four points are extracted from a mask image.

Light Flow Algorithms

For the purpose of lane detection, some suitable points are extracted from the binary mask image which is obtained after applying `inRange` filter operation on input image frame. The steps performed by the Light Flow Method are described in the following section.

Step 1: Applying Mask operation: As discussed in the previous chapter, the mask operation is applied, on captured image frame, using `inRange` filter operation available in OpenCV library. It also possible to apply `inRange` filter operation only on the half of the image (region of interest) to reduce computational operation which make limited computational boundary for the filter operation. In this work, for the safety reason and not to miss the relevant information in the image frame filter operation is applied on whole image frame. Each pixel elements has in binary mask image has 0 or 255 value which form as binary threshold. This binary mask image is used for further processing in each proposed methods in this work.

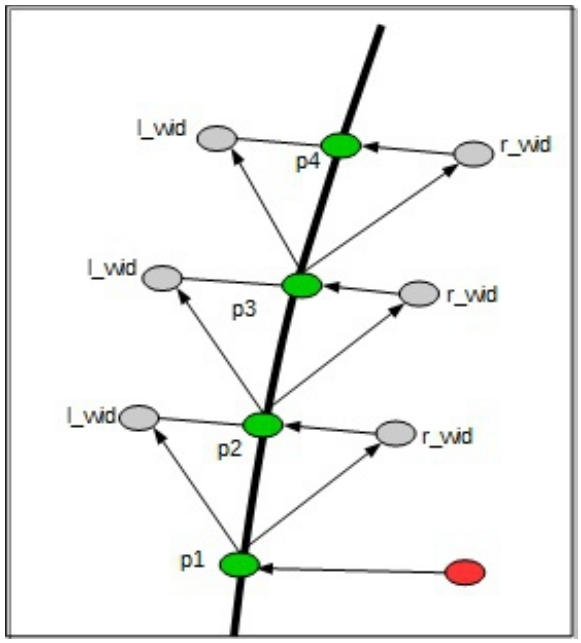

Figure 5.1: Light flow method

Step 2: Extracting four points in the image: The next operation, `gonext()` function is applied on the binary mask image to extract possible first point. This operation receive the following parameters as input. The starting point `cx` for `x` axes and `cy` for `y` axes. The starting point as red circle shown in the Figure 6.2. It traverse from starting point `cx` to the left side till it find first white pixel value. In

case, there is not white pixel value detected on the x axes and *cx* value reach to the left width, it will return Point data structure with the value zero as row and column, for example *Point(0,0)*. In case, it will find any white pixel value, the function will continue to the next step in the method which is described below.

Step 2: In previous operation, once traverse operation find any white pixel value, a connectivity check is performed which make sure that there is not only one or two white pixel but there are more. This operation is applied to make get the confirmation that a set of pixels are connected in some predefined rectangular area. This operation create another mask region of interest in an image with rectangle *rect (row, column, width, height)*. The row and column parameters are the current location of white pixel and width and height parameter is incremented with the value two pixels in the both side (in row and column parameters two point decreased in x axes and two points decreased in y axes). From new x and y point a new region of interest is derived with *width* size 4 and *height* size 4, which create new mask 4x4 matrix. A mask 4x4 region of interest is created from the current white detected point with the width 4 and height 4. This operation is shown in the Figure 5.2. The four connectivity operation return total number of white pixels available in new created mask matrix with the size 4x4. In case total number of white pixel are more then 8 out of 16 elements (4x4) of mask region, connectivity function return the current detected point as final output result to the p1 which is considered as lane point p1 in an image.

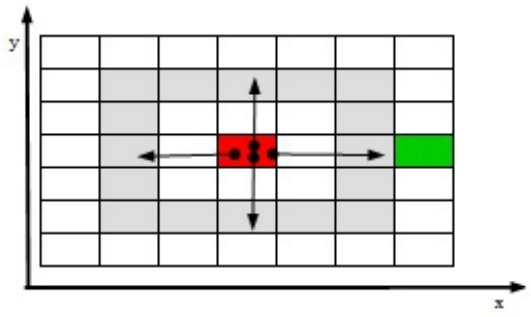

Figure 5.2: Connectivity check for Light Flow Method

Step 2: The next operation depend on the y point obtained in the p1. In case the p1.x is greater then zero which is extracted after applying connectivity operation. The x axes value is updated for the next traverse width for p2 point. It increase the p1.x's value with left width (left width=20) in both right and left side. The value of the left width is like a light flow on the both side for the next point detection. It has some benefits from the previous point detected (p1) that instead of traversing from the starting point which is extreme left side, it should start searching next white pixel value from limited right and left side width. Second it is believed that the next expected point will be available in the small range of left

and right side boundary. The updated left and right side width is shown in the Figure 5.2 marked as *l_wid* and *r_wid*. The method start from left width point and traverse to the right side with right width. In the next operation, *cy* value is updated.

Direction based method

This method is based on recursive checking the current pixel element value from the mask image frame and is performed to extract four relevant points on the road lane. These four extract points forms a detected lane points on a single lane on the road. These four detected points are shown as green color circle in the following Figure 5.3.

The direction based method is the second lane detection method implemented to extract lane points from camera image frame. The points extracted in this method are based on directions which are nothing but possible movement in direction up, down or left. It is believed that the white lane in an image has some properties like, size of the lane, brightness of the color, road lane are painted in some geocentric shapes like curves, horizontal or vertical lines and in general, road lane are painted in white or yellow color. As described in the following Figure 5.3, the method start with constant input variables as x (`cx`) and y (`cy`) axes, direction (`dir`) and y depth (ydepth) values. The starting point of the method is depicted as red color circle. This method perform the following steps to extract four relevant points which can be predict as road lane in an image frame.

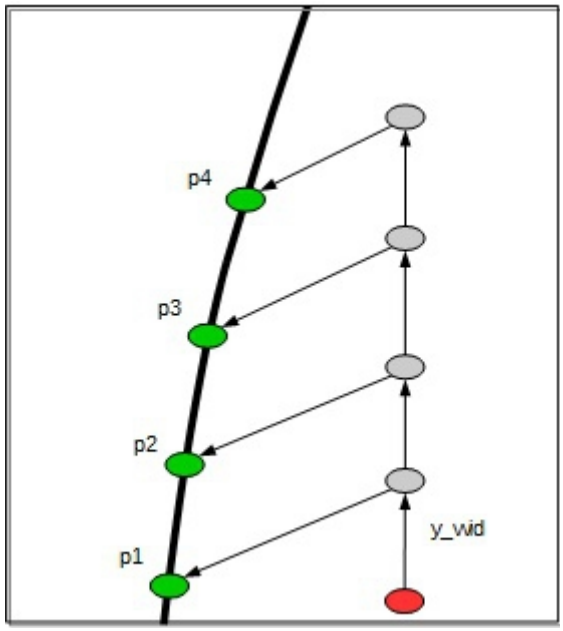

Figure 5.3: Direction Based Method

Step 1: It take x and y starting point, direction, and y depth variable as input.

Step 2: It traverse in the direction input given in the variable `dir`. In this method the first direction input is given as 1 which is called to up direction.

Step 3: While traversing up direction, it constantly check the current point coordinates pixel element values as available at the location current row and columns. In implemented algorithm the location of the current point is given as `mask.at<unit_8>(y ,x).`

Step 4: In case, y axes depth value reached to the zero and not any white pixels element

detected, the function start in direction two which is going down proportional to the left side. This functionality can be achieved by increasing the y axes again and decreasing the x axes value. While traversing to the left and down direction, in case this method extract any white pixel from the image, the method perform shift the traverse in different direction which is explained in the next step.

Step 5: This step is performed only in case of white pixel element value scanned by traversing to any direction. Once it extract any white pixel in an image, it move to the third detection which is change the current location variable as cx and cy to move in two pixel left and one pixel down. This step is performed to make sure that the white pixel detected in previous step is not a single white pixel and the neighboring connected pixels are has white intensity value.

Step 6: In case, after moving two pixel to the left side and one pixel to the down side, this method again find any white pixel elements it changed to the new direction which is called down direction. It move one element down to the pixel element from the previous white detected pixel element.

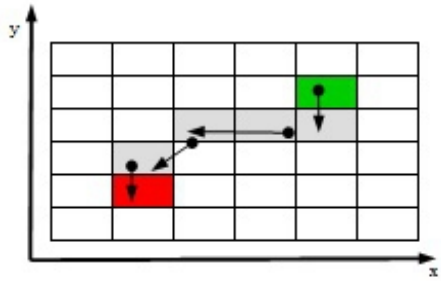

Figure 5.4: Connectivity check for Direction Based Method

Step 7: The connectivity check for Direction Based Method is shown in Figure 5.4 which is used to check and confirm that white pixel extracted at current location is not a single white pixel and the neighbor pixels for a road lane. The connectivity check is performed for each points extracted on road lane marking.

Bird Flow Method

The lane detection methods discussed in the previous section are suitable only for single middle road lane. The bird flow method is designed to detect both left and right side lane on the road. In this method vector based data structure is used for modeling left and right side road lane. The Bird Flow Method perform the following steps for lane detection on the road.

Step 1 Starting row and column parameters are given to *gonext()* function. The starting point in this method is shown in the figure in red circle. It start from the middle point of the image frame and traverse left side and right to extract white pixel element value.

Step 2: In case, white pixel element value found in any side of the lane, the location point

data structure value is updated in the left and right point data structure.

Step 3: In case, white pixel element value found, the left and right, the point data structure is updated method start extracting next left of point from mask image.

Step 4: Once connectivity check procedure is completed, the *y* value is updated for next point extraction. It follow the same previous steps for extracting relevant point value for current Point data structure.

Figure 5.5 shows the structure of Bird Flow Method used to extract four points for lane detection problem. These four points are extracted on both left and right side shown in green color circles. The name Bird Flow Method is given to this method on the base of traversing nature for extracting four points in an image. It traverse left and right from the middle point until it hit white color pixel value left or right side of the road.

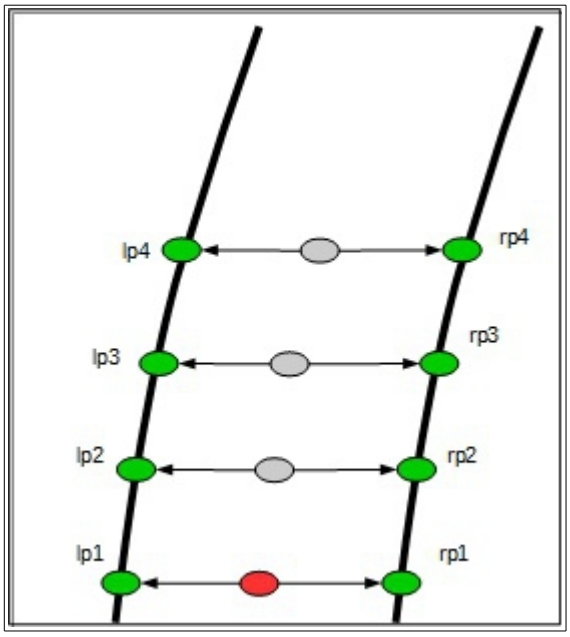

Figure 5.5: Bird Flow Method

For optimization purpose the implementation of Light Flow Methods can be applied in this methods which will reduce more computational steps for extracting relevant four points. In this case, connectivity check procedure need to be further optimized to confirm the distance between relevant left and right side points

6 Performance Evaluation

> Problems are common, but attitude makes the difference!!!
> --- A.P.J. Abdul Kalam

This chapter provide detailed information about experiments setup which is used to test the proposed lane detection methods and discuss the performance result data. It also demonstrate the efficiency of the proposed methods obtained from different hardware and software systems setup.

6.1 Experimental Setup

For the evaluation of proposed lane detection methods, all the algorithms are test on different types of processors architecture running on different operating systems. The following section described the hardware and software architecture used in this project work. The image frame size of lane detection methods tested in this project is *360x640* (360 rows and 640 columns) pixels for Light Flow Method and Direction Based Method and *480x640* pixels for Bird Flow Method.

Microsoft Windows based System

Operating Systems:	Window 8 64 bit
Processor:	AMD A8 APU
Software Module:	OpenCV 3.0, OpenCV 2.9, Visual Studio 2012, CodeBlocks
Compiler:	mingw32-g++, Microsoft ® C/++ Optimizing Compiler Version 18.00.40629 for x86

Linux based System

Operating Systems:	Ubuntu LTE 14.4
Processor:	Inteli3
Software Module:	OpenCV 3.0, CodeBlocks
Compiler:	GNU gcc

Embedded Systems based System - Raspberry Pi

Operating Systems:	Rasbian - Jessie
Processor:	ARM Cortex-A7 CPU 900MHz quad-core
Software Module:	OpenCV 3.0
Compiler:	GNU gcc

6.1 Raspberry Pi

The Raspberry Pi is a small size computer board available with a processor, RAM and some hardware ports. The following list shows the hardware architecture of Raspberry Pi he Raspberry Pi 2 model B [22] [81].

Model:	Raspberry Pi 2 model B
Operating System:	Rasbian-Jessie
Processor:	ARM Cortex-A7 CPU 900MHz quad-core
Memory Size:	1 GB RAM
Hardware Ports:	4 USB ports, 40 GPIO pins, Full 1 HDMI port, Ethernet port, Camera interface (CSI), Micro SD card slot, VideoCore IV 3D graphics core, Display interface (DSI)
Audio Ports:	Combined 3.5mm audio jack and composite video

The run time parameters obtained in number of image frame slots, for example, for first 100 frames and than set these parameters to zero. The next run-time parameters result obtained for 200 frames and than next 300 frames. The number of image frames are increased to gain the average frame per second. The number of frames are increased 100 plus previous slot of frames.

6.2 Metrics for Performance Evaluation

OpenCV offer some functional ties to measure the run time of the application. These in-build functions are g*etTickCount( )* and *getTickFrequency ( )*. The *getTickCount( )* function returns a number of ticks and *getTickFrequency()* function help to convert ticks into seconds. The following method is used to measure the run-time of the lane detection methods.

```
double start_ime = (double)getTickCount();
        // Lane detection method
double end_time = (double)getTickCount()
double total_time =  end_time – start_time ;
cout<< "Run-Time in Seconds :" <<  total_time / getTickFrequency() << endl;
```

The performance result are evaluated on a set of number of frames processed in intervals called slots. These slots are first 100 frames, next 200, 300,400 and 500 frames. At the end

of the result, average frame per second rate is computed for all the proposed methods.

The following section discussed the experimental performance result for proposed lane detection methods.

6.3 Experimental Results

The following section discussed the performance result for each lane detection methods.

Light Flow Method

The following Table 6.1 shows run-time performance result of Light Flow Method executed on different operating systems including Windows, Linux and Rasbian-Jessie which were running on Intel i3, AMD A8 and AMR A7 respectively. The performance result data obtained in slots wise on different number of ranges of image frame. The following table also shows average frame per seconds obtained by dividing total run-time with total number of frames executed on each processor.

Light Flow Method

Run-Time (in Sec.) for Light Flow Method

Frames	Intel i3	AMD A8	ARM v7
100	0.91	4.68	4.27
200	1.79	3.12	8.35
300	2.61	4.67	12.71
400	3.81	6.25	17.88
500	4.93	7.79	21.85
1500	**14.04**	**26.51**	**65.06**
Avg. FPS	**106.82**	**56.59**	**23.05**

Table 6.5: Run-time measurements of Light Flow Method

The run-time parameters are obtained in seconds time unit for each slot. These methods are executed around 20 times to check the consistency of the result. The result shows that each systems take around half second additional time when it is executed for the first time. The run-time parameters are obtained in seconds time unit for each slot. These methods are executed around 20 times to check the consistency of the result. The result shows that each systems take around half second additional time when it is executed for the first time.

After that once same method is executed on the same systems, it reduce around 0.5 second time for first slot of number of frames. The extract time taken in the first execution model

can be due to compiling, building and setting up the image frame extraction module. Once the frame extraction module start working in normal way, it reduces the execution time for each slot.

From the result, it can be conclude that AMR A7 processor perform very low average frame per second (FPS) and desktop processor perform always high number of average frame per second.

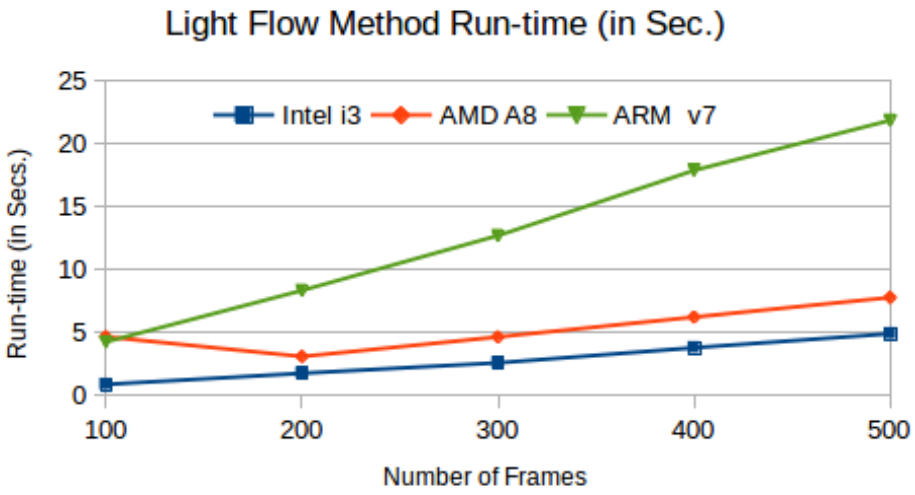

Figure 6.1: Performance result of Light Flow Method

The result from Figure 6.1 shows that Light Flow Methods on Inteli3 processor perform best other then AMD A8 and ARM v7. The result shows that inteli3 processor performance four time better than ARM v7 processor and AMD A8 processor take 4.68 seconds for first 100 frames but it reduces the run-time once number of frame load increases.

The following Figure 6.2 show the output of Light Flow Method performed on single image frame. Four blue color circles show the extracted points from mask image. The left side direction indicator is drawn after manipulating the location of extracted four points in single image frame. The direction indicator is calculated after considering the location and magnitude of four blue color circles. The average frame per second rate for inteli3 is more then 100 frames, for AMD A8 around 60 frames and 23 frames for ARM v7 processor. The average frame per second rate can be improved on ARM v7 processor after applying further optimization in implementation. The last three point extracted are in the limited range of left and right width of first point.

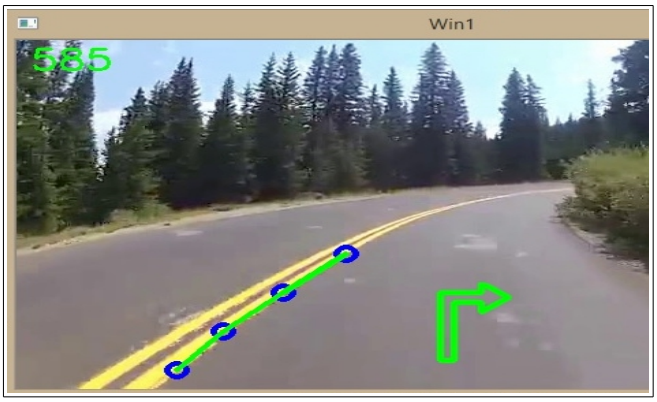

Figure 6.2: Output Result of Light Flow Method

The implementation of direction indicator can also be further optimized by attaching pre-designed small size direction image instead of drawing a new direction curve each time for every new frame. The Light Flow Method also help to detect curves available in image frame automatically because points are extracted on the based of left and right width of previous point.

Direction Based Method

The following section discuss the performance result of Direction Based Method. The implementation of Direction Based Method for lane detection problem user-defined retrieve relevant four points after traversing the pre-defined directions. Total number of steps are counted for each direction and it is realized that Direction Based Method perform more the 1400 steps on single image frame which is the main reason of consuming high run-time for the lane detection method.

Direction Based Method

Run-Time (in Sec.) for Direction Based Method

Frames	Intel i3	AMD A8	ARM v7
100	1.07	4.65	4.77
200	2.18	3.17	9.89
300	3.27	4.70	14.54
400	4.50	6.23	19.89
500	5.82	7.80	25.15
1500	**16.83**	**26.56**	**74.25**
Avg. FPS	**89.14**	**56.47**	**20.20**

Table 6.6: Run-time measurements of Direction Based Method

Further optimization can be applied on this method to decrease the run-time. Table 6.2 shows the run-time performance result obtained after running Direction Based Method on different processor. The average frame per second rate is reduced from Light Flow Method because Direction Based Method perform around 1200 steps more than Light Flow Method. The average rate of frame per second on Intel i3 is around 90 frame, on AMD A8 is around 57 frames and on ARM v7 is around 20 frames. The results shows that this method consume high average rate frame per second than other two proposed methods.

The Figure 6.3 shows the result of Direction Based Method in graphic diagram and indicate that inteli3 processor has better run-time then AMD A8 and ARM v7 processor. It has higher run-time than previous method because it check white color points on the based of directions by performing high number of steps.

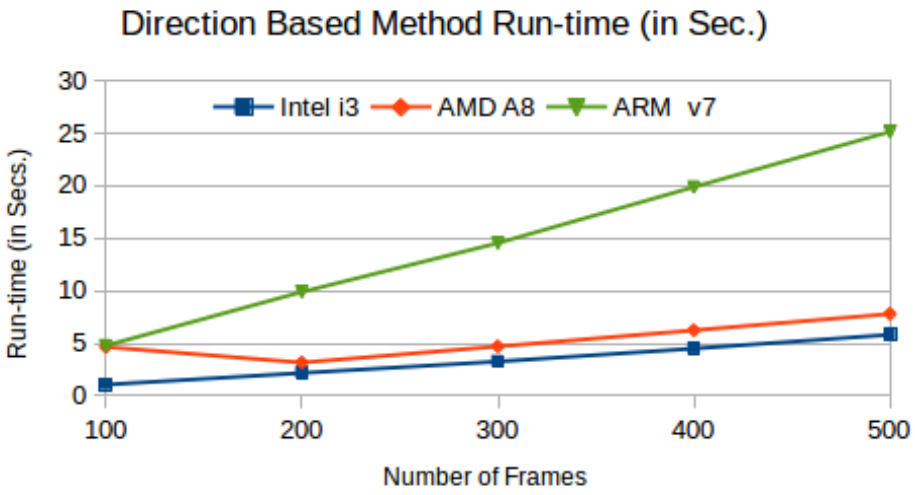

Figure 6.3: Performance result of Direction Based Method

The Direction Based Method the basic method which was proposed for lane detection problem. Light Flow Method and Bird Flow Method are the evolution result of Direction Based Method implemented with minor changes in traversing and locating white pixels values in a mask image. In Direction Based Method, curve direction indicator are provided in text form for evaluation of the systems performance and extracted four points in an image. The Figure 6.4 shows the output result of Direction Based Method obtained on single image frame.

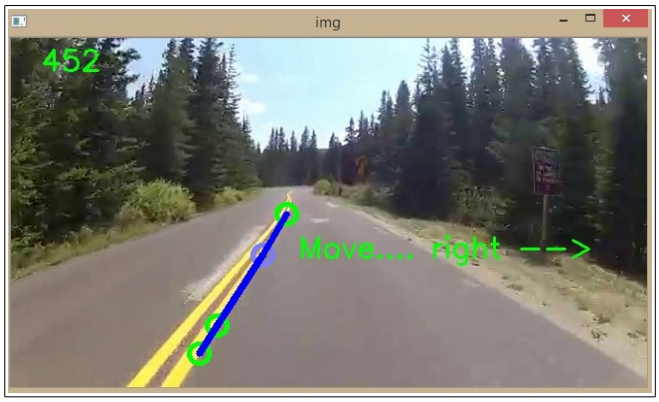

Figure 6.4: Output result of Direction Based Method

Bird Flow Method

The following section discuss the performance result of Bird Flow Method used for detecting left and right side road lane. Bird Flow Method is improved version of Light Flow Methods and perform same steps for detecting left and right side road lane. The following Table 6.3 shows the run-time performance result for different processors.

Bird Flow Method

Run-Time (in Sec.) for Bird Flow Method

Frames	Intel i3	AMD A8	ARM v7
100	1.26	4.40	5.93
200	2.51	3.31	11.82
300	3.77	5.28	17.76
400	5.39	6.31	23.85
500	7.09	7.83	28.78
1500	**20.02**	**27.13**	**88.14**
Avg. FPS	**74.93**	**55.30**	**17.02**

Table 6.7: Run-time measurements of Bird Flow Method

The result shows that it has average frame rate around 75 frames per seconds on Intel i3, 55 frames on AMD A8 and 17 frames on ARM v7 as average frame per second. It has high run-time than other two methods because it perform lane detection methods on both the side for left and right road lane. The Figure 6.5 shows the graph for Bird Flow Method and indicate that Intel i3 processor perform better then other two processors.

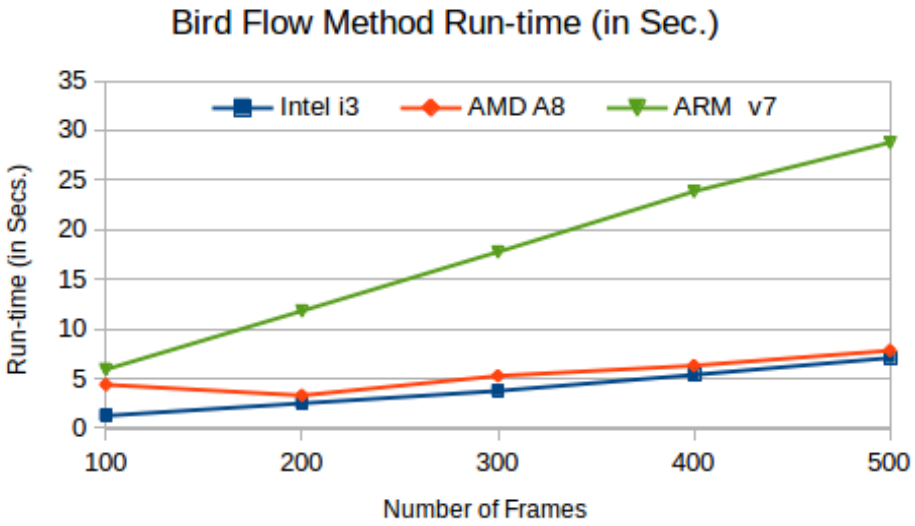

Figure 6.5: Performance result of Bird Flow Method

The system with AMD A8 processor has more run-time for the first set of frames, it reduced in second set of frame even the number of frames are increased (two times increased) from first frame slot. Figure 6.6 shows output result of Bird Flow Method applied on two road lane. The circles in blue and green colors represents left and right side road lane respectively.

Figure 6.6: Output result of Bird Flow Method

The green color circles available in the middle points represent as a starting point for each left and right side road lane. This starting point is constant x and y axes in this method. But for optimization purpose constant *x* and *y* axes can be changed as per the Light Flow Methods.

Average Frame Per Second

The following section provide discussion about number of *average frame per second* measured from all the experimental setup (from different processors) for all lane detection methods proposed in the project work. The following Table 6.4 shows the average frame per second measurement result obtained from different processor. The average frame rate for Light Flow Method lane detection on Intel i3 is 106 frames per seconds which is good frame for real-time application performance. On other hand same method achieve around 23 frames per second on embedded based systems (Raspberry Pi) which is very low due to different architecture of the processor.

Average Frame Rate Per Second

Processor	Light Flow Method	Direction Based Method	Bird Flow Method
Intel i3	107	89	75
AMD A8	57	56	55
ARM v7	23	20	17

Table 6.8: Average frame per second on different processor

From Table 6.4, it can be realize that Light Flow Method perform 47% more number of frames then Direction Based Method on Intel i3 processor. The reason behind 47% frame rate increment in Light Flow Method is the number of pixel manipulation in both the methods. The Direction Based Method perform more number of pixels manipulation in single frame to extract four relevant lane points. Light Flow Method on other two processor including AMD A8 and ARM v7 has 36% and 26% respectively per more number of frame rate over Direction Based Method. The systems with ARM v7 processor has 74 average frame rate for Light Flow Method which is very good rate from point of Embedded real-time performance evaluation. The frame percentage rate of Bird Flow Method on ARM v7 has 17 average frame per second in comparison to Intel i3 which has 23 average frame per second. The Bird Flow Method on ARM v7 has less frame rate from other two methods because it is performed to extract two road lane. Figure 6.7 shows average frame rate per second graph extracted from three experimental setup. From the diagram, it can be realizes that Direction Based Method has same performance result on each systems setup. On other had Light Flow Method and Bird Flow Method has huge difference on each systems setup, where Light Flow Method on Intel i3 has highest performance result from other two lane

detection methods.

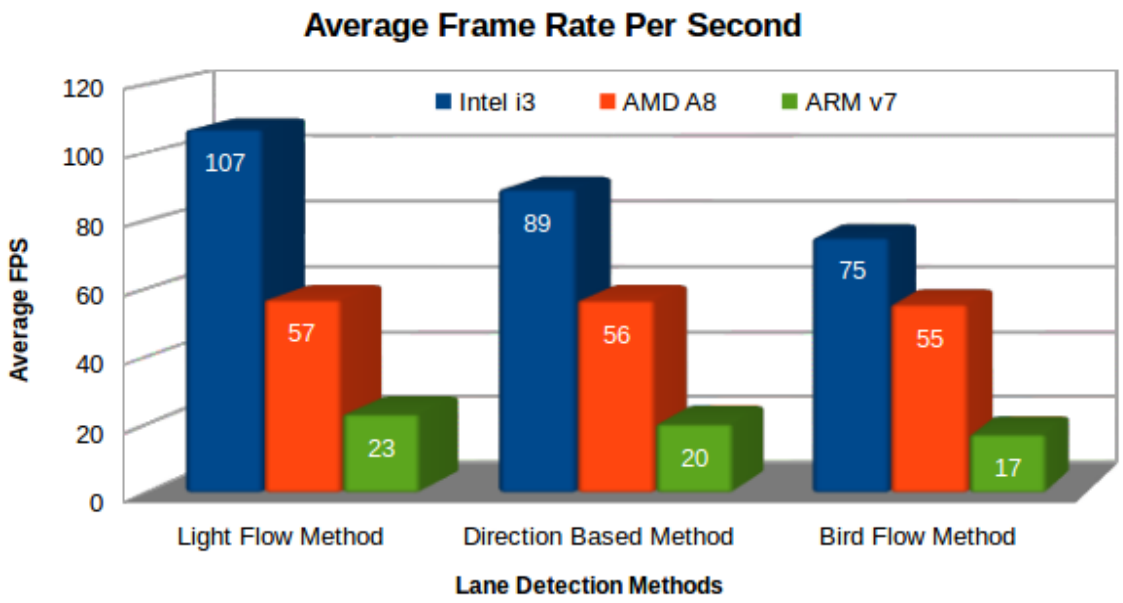

Figure 6.7: Average frame per second of Lane Detection Methods

The systems with AMD A8 performance always in the middle because it is based on Microsoft Visual Studio based compiler implementation. The Linux based operating systems including Intel i3 and ARM v7 use GNU gcc and g++ compiler has different architecture over Windows based system including AMD A8 which has their own MS Visual Studio compiler. From the research it is recognized that Visual Studio 2012 based implementation create some extra linking and object files which create high project memory size over Linux based CodeBlocks IDE. As it can be seen from Figure 6.7, Light Flow Method perform better than Direction Based Method for single road lane detection.

The experiment of optical flow measurement, a method of object motion, has been implemented based on embedded systems processor (Raspberry Pi) using OpenCV library [81] on different image size. The result obtained in that experiments shows that Raspberry Pi processor can achieve one sixth performance in comparison of CPU processor and processing run-time is increase proportion to the image size. In this work, all lane detection methods tested on Raspberry Pi based systems has more one third average frame rate per seconds which is relevant for real time embedded systems platforms

Average Frame Rate per Seconds on CPU and GPU

The following section discussed average frame rate per second of in-build OpenCV

functions obtained on AMD A8-6500 with Radeon (tm) HD Graphics 3.5 GHz available with Radeon™ HD 8570D APU, an integrated graphic processing unit. The implementation of two OpenCV functions including Canny () edged detection and inRange () are tested in Microsoft Visual Studio 2012 running on Microsoft ® C/++ Optimizing Compiler Version 18.00.40629 for x86 and open source CodeBlocks IDE running with mingw32 g++ compiler. In Figure 6.8 shows the results obtained after performing Canny edge detection and inRange function performed on 100 frames under different software and hardware environment including mingw32 g++, VS 2012 compilers and CPU, GPU processors. From the Figure 6.8, it concludes that Windows based compiler has equal average frame rate per second on CPU and GPU for both OpenCV algorithms. On other hand open source compiler mingw32 based GPU processor has higher performance result over CPU processor.

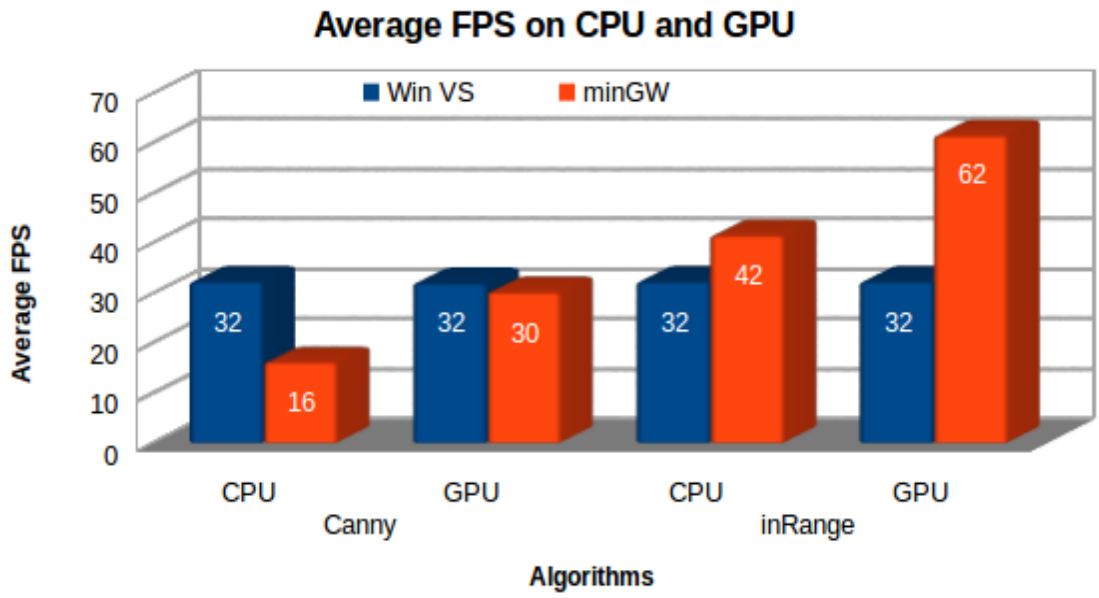

Figure 6.8: Average frame per second on CPU and GPU

In Windows based systems, data communication operations between CPU and GPU has more influence on performance result. As it is shown in the Figure 6.8, Windows based systems has same performance result for both algorithms on CPU and GPU. GPU performance result 32 average FPS also including the data transfer rate between CPU and GPU. To achieve better performance result on GPU, OpenCV implementation should reduces this communication operation and most of the operations on GPU should be

implemented on GPU so GPU can do much work. For example, extracting image from video should be done on CPU, implementation operations should be performed on GPU frame. The problems related to the direct memory access (DMA) has been discussed [73] to increase the performance and improve the access speed of direct memory assess (DMA) buffer by using a synthesis process for the a reconfigurable hardware Zynq based system in an embedded video processing project. FPGA based hardware acceleration to increase performance computation for real-time image processing methods using EO sensors has been proposed at Chemnitz University of Technology in Germany [75]. A road lane detection and tracking has been developed and implemented for hardware acceleration support on heterogeneous platform including CPU, GPUs (NVIDIA GeForce GTX 660 Ti and NVIDIA GeForce GTX 285) and FPGA (FPGA - Altera Stratix-V) hardware components using OpenCV and OpenCL library [79].

7 Conclusion

> "You" are nothing but what your Mind is....
> no other word you need....

This thesis project work explores the underlying current state of the art, concepts, techniques and hardware and software platforms of lane detection systems, an exciting and rapidly growing research area with great potential of causing paradigm shifts in current advance driver assistance system applications, based on contrast analysis of image processing and computer vision methods. Essentially, the aim is to design and develop computer vision based methods for lane detection and tracking problems that are capable of providing road lane direction information and warnings by extracting relevant information about the content conveyed by the image frames in different road conditions.

Image processing and computer vision is a strongly multidisciplinary research area typically consists of algorithms and methods for processing and manipulating image or camera frame data including physical parameters of camera position, by using mathematical and geometric processing methods used to extract and convert raw image data into meaningful information. In addition, more sophisticated methods and algorithms emerge continually to keep up with the demands of real-time performance and safety requirements imposed by the automotive industry in their advanced driver assistance system applications. The large computational demands, implementation and complexity of these sophisticated methods which need to achieve within strict embedded systems requirements, including small in size, low energy/power consumption, and high performance, impose new challenges in the design and development process for embedded systems based platforms. All these parameters of image processing and computer vision based software development process designed for embedded systems platform based application represent a new domain with the name called Embedded Vision Systems (EVS) in electronics, manufacturing and automotive industry. The main aim of embedded vision systems pipeline [20] is to deal with the key problems in the standard design and implementation methodologies, address solutions to challenges arises in computer vision based method applied on embedded systems platform.

The work at the hand developed, implemented and evaluated proposed lane detection methods for advanced driver assistance systems applications based on contrast analysis

using low level image processing algorithms and OpenCV library, on different architectures. This concluding chapter summarizes the contributions, results and learning of the work. For that, it recapitulates the main contents and goals of the particular chapters.

7.1 Summary

In recent years, computer vision based methods are applied in automotive advanced driver assistance applications and these methods are based on image processing algorithms. The computer vision and image processing techniques are designed and implemented while keeping enough resource available to solve problems in their own domain. The nature of available automotive and embedded systems based applications required efficient solution in form of high performance, small size, low cost and hard real-time performance. The following section provide short summary about each chapter presented in this thesis project work.

The first chapter gave an introduction to the thesis work and discussed current problems available in advanced driver assistance systems based on image processing methods. These problems motivate to review and search different solution to improve the performance and efficiency for lane detection methods. Low-level image processing based methods, used for lane detection problems, are described. It also discuss about the current problems and new trends currently passing in the vision-based advanced driver assistance systems including land detection warning and lane departure tracking systems, night vision and vehicle detection systems designed and developed on different hardware and software architecture.

Chapter 2, discussed some basic functions and methods available in OpenCV library which were extensively used in the implementation of proposed lane detection methods. The basic image processing functions and methods available in OpenCV library are already optimized functionalists ready to use on different hardware and software platform.

Chapter 3, provided with the state of the art of lane detection and tracking application problems currently available and implemented on different hardware and software platforms.

The computer vision-based method required high computational power and energy resources and on other hand automotive application systems required to achieve high performance within low range of energy consumption and resource matrices. The vision-based automotive applications already designed with advanced FPGA or GPU based

system architecture also consuming high energy and computational power.

To improve the performance matrices and reduce the energy consumption parameters, OpenCV library implemented in C/C++, can be used with extended image processing and computer vision algorithms for the development and implementation of embedded vision systems. OpenCV library which are designed to provide high performance result in the domain of image processing and vision based systems.. Some real-time open source operating systems already tested with these libraries [12] and the performance result meeting the current hard real-time requirement of automotive systems [13]. Among the different options, computer vision based solutions have gained a strong interest due to their benefits over the electronic and sensor based systems like LiDAR or radar.

Chapter 4, discussed about concept design and methodology used in proposed lane detection methods. Beyond improving the performance and testing on different systems architecture, in this project a new low-level image processing based approaches are discussed and implemented in open source OpenCV library for lane detection and tracking problem. The idea which is implemented in the proposed methods were to reduce computational processing power by eliminating processing area of an image to get relevant information about lane model in the road environment.

These proposed lane detection methods are described in implantation Chapter 5, are tested using single and two road lane models. As discussed in this chapter, three lane detection methods are proposed including, Light Flow Method, Direction Based Method and Bird Flow Method. The Light Flow Method is applied on single road lane and use recursive approach to extract four relevant points from an input image frame. It also use robust method applied at starting point for manipulating next frame, based on the information obtained in previous frame. In this case, it reduce the computational cost to achieve high performance result. The same approach is not applied in Bird Flow Method which is used to extract two road lanes available both on left and right side on the road. This method is suitable for providing guidance to the robot, including manufacturing and electronics based systems where robots are used for different kinds of service, and lane detection problem when only two white road lane are available.

Chapter 6, shows the performance result obtained after performing Light Flow Method, Direction Based Method and Bird Flow Methods on different operating systems including Microsoft Windows, Linux based Ubuntu LTE 14.4 and Linux based Rasbian-Jessie which

were running on AMD A8 APU, Intel i3 and ARM v7 processor respectively. The performance results show that proposed methods achieve around 102 to 23 average frame per second obtained after testing on different processor architecture. A small modification in these proposed method can achieve detecting more then two road lanes in single image frame.

Chapter 7 provide conclusion with summary and future work which can be applied on these proposed lane detections methods.

7.2 Future Work

In this section, a list of task or activities are discussed which can be applied in future on the proposed methods to solve lane detection problems and some other kind of advance driver assistance systems application problems including light detection, object detection in front of moving car.

- The first future activity is to include the extension of camera module for capturing image frame directing from the camera and applying proposed methods for lane detection problems. This future work required a small medication in the proposed lane detection methods which is just one single line of code as shown in Chapter 2, OpenCV basic operations.
- For detecting curves and lines in different shapes, the physical position and distance parameters of camera module and depth information can be used to improve the lane detection methods.
- The lane detection methods has been implemented in sequential based processing mode. A small change in "*for loop"* code for extracting four points can be implemented for parallel and multi-core based processing. OpenCV provide `parallel_for_` loop to implement thread based parallel processing implementation. These parallel "`for loop`" also can be used to detect left and right side road lane in the design and implementation process of parallel and multi-core based process mode.
- In this thesis work, proposed lane detection methods, use the contrast analysis and evaluation of intensity value of pixels for deciding if the current pixel element is in the range of intensity value of road lane model or not. The white and yellow pixel values are evaluated on the base of `x axes` and `y axes` coordinates which are not

sufficient. To improve the accuracy of the proposed methods, the parameters of camera module need to add in the design and evaluation process from camera space including, *x*, *y* and *z* axes coordinates and need to map with image space coordinates including *x axes* and *y axes*.

- OpenCV 3.0 also provide new methodologies for providing hetrogenous computing environment, including UMat data structure, and to implement CPU and GPU based implementation of computer vision applications. In this work, two OpenCV functions called `Canny ()` an edge detection function and `inRange ()` function used to create binary threshold image, has been tested to measure average frame rate per seconds on heterogeneous CPU and GPU based platform. The performance result shows significant changes in the performance result achieved after running the test on 100 frames.
- With small changes these proposed lane detection methods can be used to implement for heterogeneous based platforms which will help to reduce computation time for each method.

8 Appendix

A Installation of OpenCV

1.1 OpenCV on Microsoft Windows based system

OpenCV libraries can be used on Windows based systems under Microsoft Visual Studio and CodeBlocks, an open source IDE for implementing computer vision applications. In next section, OpenCV installation and build process is discussed for both Windows and Linux based systems. Please note that, CodeBlocks is more flexible in use and easy to compile and run OpenCV program. It also provide facilities to make project templates which we can be re-used for further project for implementation, design and testing purpose.

Installing OpenCV in Windows for Visual Studio

Installing OpenCV libraries under Windows to build program with Visual Studio is very simple and easy task.

Step 1: Download the required OpenCV libraries from the web-page.

Step 2: Extract the downloaded folder in default location C:\

Step 3: Add the following setting in the new created project.

1. Create Console Project
2. Add main.cpp file
3. Goto Project Properties
4. In C/C++ General ->Additional include Directories add the following line
   "C:\opencv3\build\include"
   For Example the out will be like following code:
   C:\opencv3\build\include;%(AdditionalIncludeDirectories)
5. Linker->General -> Additional Library Directories add the following lines
   C:\opencv3\build\x86\vc12\lib; %(AdditionalLibraryDirectories)
6. Linker Input - > Additional Dependencies add the following lines
   opencv_ts300.lib; opencv_world300.lib;%(AdditionalDependencies)

Installing OpenCV in Microsoft Windows for CodeBlocks

A) Install MingGW compiler for Windows, which can be downloaded from www.mingw.org

1. Install MingGW compiler for writing program in C/C++ in Windows operating system
2. Install it to the default location, for example, c:\MingGW
3. To compile C/C++ program, select these options “mingw32-base” and “mingw32-gcc-g++”
4. Add c:\MingGW\bin path to the “Environment Variables” in The PATH variable

B) Install CodeBlocks, is an IDE for (Integrated Development Environment) for writing program in C/C++ and others programming languages, which can be downloaded from www.codeblocks.org/downloads/26 available for Windows XP / Vista / 7 / 8.x / 10.

1. Install it to the default location c:\codeblocks\

C) Install Cmake, which is an open-source systems that manages the build process in different operating systems.

1. Install it to the default location c:\cmake\

D) Install OpenCV libraries, which can be downloaded from http://opencv.org/downloads.html

1. Install it to the default location c:\opencv\
2. Open Cmake in graphical-user mode
3. Add c:/opencv/source to the first option “Where is the source code”
4. Add c:/opencv/release to the second option “Where to build the binaries”
5. Click the “Configure” button and select the “minGW Makefiles” for the compiler option
6. Once Cmake finish the configuration option, click on the “Generate” button. The OpenCV build generation process will take some time (From 2 to 4 hours.) depending on the options selected in the build configuration process.
7. Once Cmake finish its process. Set the following environment variables in the PATH variable.
   - c:/opencv/release
8. Open new project in CodeBlocks and set the following variables:
   a) Goto to Build Option
   b) Search directories → Compiler: add “c:/opencv/build/include”
   c) Search directories → Linker: add “c:/opencv/Release/lib”
   d) Linker setting: add “ libopencv_xxx.dll.a” libraries, Here xxx is replaced which type of libraries required as per the systems. The following is a list of example libraries which need to add.

..\..\..\..\opencv3-1-0\opencv\Release\lib\libopencv_core310.dll.a

..\..\..\..\opencv3-1-0\opencv\Release\lib\libopencv_highgui310.dll.a
..\..\..\..\opencv3-1-0\opencv\Release\lib\libopencv_imgcodecs310.dll.a
..\..\..\..\opencv3-1-0\opencv\Release\lib\libopencv_imgproc310.dll.a
..\..\..\..\opencv3-1-0\opencv\Release\lib\libopencv_ts310.a

1.2 Installation of OpenCV on Raspberry Pi

Installing and Building OpenCV on Raspberry Pi

In this project Raspberry Pi has been used for testing lane detection methods to measure performance issues. This section discuss the steps of installing and building OpenCV with version 3 on Raspberry Pi which has Rasbian Jessie Linux based operating systems. Please note that installing OpenCV library take around 4-8 hours. The rest of the section explain how to install and build OpenCV using CMAKE build software.

Step 1:

Configure your Raspberry Pi by installing required packages and libraries used by OpenCV.

$ sudo apt-get update

$ sudo apt-get upgrade

$ sudo apt-get install build-essential git cmake pkg-config

Step 2:

Install required packages

$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev

$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

$ sudo apt-get install libxvidcore-dev libx264-dev

$ sudo apt-get install libgtk2.0-dev

$ sudo restart

Step 3:

Extract OpenCV libraries

$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.0.0.zip

$ unzip opencv.zip

$ wget -O opencv_contrib.zip

https://github.com/Itseez/opencv_contrib/archive/3.0.0.zip

unzip opencv_contrib.zip

Step 4:

Installing OpenCV

```
 $ cd ~/opencv-3.0.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_C_EXAMPLES=ON \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH =~/opencv_contrib-3.0.0/
modules \
    -D BUILD_EXAMPLES=ON ..
$ make -j4  // For installing using 4 cores
$ make
$ sudo make install
$ sudo ldconfig
```

Compiling and Running OpenCV program on Raspberry Pi

This section explain how to compile and run OpenCV program in command line editor

To check the OpenCV libraries use the following command

```
pi@raspberry~ /OpenCVTest$pkg-config --cflags --libs opencv
```

Step 1: Goto the folder where test1.cpp contain for compiling

```
pi@raspberry~ /OpenCVTest$ g++ -ggdb `pkg-config --cflags --libs opencv` test1.cpp .cpp test1
```

Step 2: Use below command to run the program

```
pi@raspberry~ /OpenCVTest$ ./test1
```

B Content in CD

Folder Name	**Description**
Master Thesis Documents	- MasterThesis_Report_S_Kumar.pdf
	- Task_Description.pdf
	- Poster_Master_thesis.pdf
Implementation	1. Light Flow Method - light_flow.cpp
	2. Direction Based Method
	- direction.cpp
	3. Bird Flow Method
	- birdflow.cpp
	4 Image_and_video
	- Image and Video files
Diagrams	List of diagrams created in this project work
References	List of references used in this project work

C Abbreviations

ADAS	Advance Driver Assistance System
AI	Artificial Intelligence
APS	Active-Pixel Sensor
AVB	Audio/Video Bridging
CAN	Controller Area Network
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide Semiconductor
CNS	Car Navigation Systems.
CV	Computer Vision
EVS	Embedded Vision Systems
FPGA	Field Programmable Gate Array
FPS	Frame Per Second
GPS	Global Positioning System
GPU	Graphics Processing Unit
HMI	Human Machine Interface
HT	Hough Transform
ITS	Intelligent Transport System
LDA	Lane Detection Algorithm
LDM	Local Dynamic Map
LDW	Lane Detection Warning System
LLIP	Low-Level Image Processing
ML	Machine Learning
MOST	Media Oriented Systems Transport
OpenCL	Open Computing Language
OpenCV	Open Computer Vision
ROI	Region of Interest
SoC	Systems-on-chip
V2V	Vehicle-to-Vehicle

9 References

1. K.H. Lim, K.P. Seng, L. Ang, "**River Flow Lane Detection and Kalman Filtering-based B-Spline Lane Tracking**", ECU Publication, 2012
2. N. Mattern, "**Multi-Sensor Vehicle Localization in Urban Environments using Image Prediction**", Dissertation, Fakultät für Elektrotechnik und Informationstechnik, TU-Chemntz, Jan 2015
3. Y. W. Seo, "**Detection and Tracking the Vanishing Point on the Horizon**", CMU-RI-TR-14-07, May 2014
4. W. Zou, D. Xu, J. Yu, "**Embedded Vision Positioning Systems Based on ARM Processor**", Embedded Visual Systems and Its Application on Robots, 30-46, 2010
5. S. Tan, J. Mae, "**Real-Time Lane Recognition and Position Estimation for Small Vehicle**", ISSN: 1942-1973, Internet-working Indonesia Journal (IIJ), 2013
6. J. McCall, M. Trivedi, "**Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation**", IEEE Transaction on Intelligent Transportation Systems, Vol. 7, No. 1, March 2006
7. B. Dipert, T. Wilson, T. Droz, L. Riches, G. Agarwal, M. Jacobs, "**Smart In-Vehicle Cameras increase Driver and Passenger Safety**", John Day's Automotive Electronics, Oct. 2014
8. S. Neuendorffer, T. Li, D. Wang, "**Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries**", XAPP1167 (v3.0), Xilinx Inc., USA, June 24, 2015
9. M. Bellino, Y. L. de Meneses, P. Ryser, Jacot, "**Lane Detection Algorithm for an Onboard Camera**", SPIE proceedings of the workshop on Photonics in the Automobile, 2004
10. Embedded Vision Alliance, "**Embedded Vision Enables a New Realm of System Innovation**", Embedded Vision Case Studies, www.embedded-vision.com
11. N.G. Chaitaliya, A.I. Trivedi, "**Facial Feature Point Extraction for Object Identification using Discrete Contourlet Transform and Principal Component Analysis**", Advances in Biometrics for Secure Human Authentication and Recognition, CRC Press, 2014
12. G.Velez, M. Nieto, O. Otaegui, G. V. Cutsem, "**Implementation of a Computer Vision based Advanced Driver Assistance Systems in Tizen IVI**", DOI:10.13140/2.14306.6567, Sept, 2014
13. G. Velez, O. Otaegui, "**Embedded Platform for Computer Vision-Based Advanced Driver Assistance Systemss: A Survey**", Donostia-San Sebstian, Spain
14. A. N. Belbachir, P. M. Göbel, "**Smart Cameras: A Historical Evolution**", Springer Science+Business Media, LLC 2010
15. R. Kolke, C. Gauss, D. Silvestro, "**Reduzierung von Unfällen durch Notbremssysteme bei Pkw**", ADAC Technik Zentrum

16. Z. Nikolic, G. Agarwal, B. Williams, S. Pearson, "**TI Gives Sight to Vision-Enabled Automotive Technologies**", White Papers, Texas Instruments, Oct, 2013

17. S. Matuska, R. Hudec, M. Benco, "**The Comparison of CPU Time Consumption for Image Processing Algorithm in Matlab and OpenCV**", IEEE 978-1-4673-1179-3/12, 2012

18. A. Dubey, K. M. Bhurchandi, "**Robust and Real Time Detection of Curvy Lanes (Curves) With Desired Slopes For Driving Assistance and Autonomous Vehicles**", VNIT

19. R. Laganiere, "**OpenCV 2 Computer Vision Application Programming Cookbook**", Packt Publishing Ltd., May 2011

20. M. Kölsch, S. Butner, "**Hardware Considerations for Embedded Vision Systems**", Embedded Computer Vision, Advances in Pattern Recognition, Springer-Verlag, 2009

21. N. Nai, G.N Rathna, "**Real-Time Face Detection on GPU Using OpenCL**", DOI:15.5121/CSIT.4238, 2014

22. The Raspberry Pi Foundation, www.raspberrypi.org

23. T. Smith, H. Vardhan, B. Malet, D. Mingay, "**ARE WE THERE YET? THOUGHTS ON IN-CAR HMI**", Version 1.0, July 2014

24. Y. Wang, D. Shen, E. K. Teoh, "**Lane Detection using Spline Model**", Pattern Recognition Letters, 21, 677-689, 2000

25. S. Saha, S. S. Bhattacharyya, "**Design Methodology for Embedded Computer Vision Systems**", Embedded Computer Vision, Advances in Pattern Recognition, Springer-Verlag, 2009

26. Tim Moynihan, "**CMOS Is Winning the Camera Sensor Battle, and Here's Why**", Article-246931, PCWorld, USA

27. F. Samadzadegan, A. Sarafraz, M. Tabibi, "**Automatic Lane Detection in Image Sequences for Vision-Based Navigation Purposes**", Geomatics Department, Faculty of Engineering, University of Tehran

28. R. Polikar, "**Pattern recognition in bioengineering**", Wiley Encyclopedia of Biomedical Engineering 4, 2695-2716, New York: Wiley, 2006

29. Y. Zhang, A. S. Dhua, S. J. Kiselewich, W. A. Bauson, "**Challenges of Embedded Computer Vision in Automotive Safety Systems**", Embedded Computer Vision, Advances in Pattern Recognition, Springer-Verlag, 2009

30. Vision Sensor, "**PresencePLUS P4 Sealed OMNI**", Banner Engineering Corp Limited, Minnesota

31. A. J. Lipton, "**We Can Watch It for You Wholesale**", Embedded Computer Vision, Advances in Pattern Recognition, ISSN 1617-7916, Springer-Verlag, 2009

32. S. Evanczuk, "**Top 5 Myths in Automotive Vision: Designing Embedded Vision Systems Is Easier Than You Think**", Technical Article, Avnet, Inc.

33. L. G. Shapiro, G. C. Stockman, "**Computer Vision**", Prentice Hall,

34. G. Hamarneh, K. Althoff, R. Abu-Gharbieh, "**Automatic Line Detection**", Project Report for the Computer Vision Course, Sept. 1999

35. Z. Lin, J. Sankaran, T. Flanagan, "**Empowering Automotive Vision with IT's Vision AccelerationPac**", Texas Instruments, White Paper, Oct. 2013

36. "**Automotive Imaging-Dynamic Vision for the Road Ahead**", OmniVision Technologies, USA

37. Smart Camera, "**NI 177x High-Performance Smart Camera**" National Instruments Corporation, Austin

38. Cooperative Intelligent Transport Systems, "**C-ITS Platform**", Final Report, Jan, 2016

39. A. C. Bovik, "**Introduction to Digital Image and Video Processing**", Elsevier Academic Press, 2005

40. KHRONNOS, "**OpenCL Overview**", www.khronos.org

41. D. Reissner, W. Hardt, F. Strakosch, F. Derbel, "**Energy-efficient adaptive communication by preference-based routing and forecasting**", DOI:10.1109/SSD.2014.6808912, IEEE, 11-14 Feb. 2014

42. G. Raman, D. Reißner, W. Hardt, "**Image Processing Based Evaluation of Chain Elongation for Industrial Carrier System**", Master Thesis, TU-Chemnitz, Sept 2014

43. OpenCV overview and examples, www.opencv.org

44. Elektrobit, "EB Assist ADTF", www.elektrobit.com

45. S. Sivaraman, M. M. Trivedi, "**Active learning for on-road vehicle detection: a comparative study**", DOI 10.1007/s00138-011-0388-y, Springer-Verlag 2011

46. P. Nagendra, "**Performance Characterization of Automotive Computer Vision Systems using Graphic Processing Units (GPUs)**", 978-1-61284-861-7/11, IEEE 2011

47. K. Chitnis, R. Staszewski, G. Agarwal, "**TI's Vision SDK, Optimized Vision Libraries for ADAS Systems**", Texas Instrument, Tech. Report, 2014

48. P. Palašek, P. Bosilj, S. Šegvić, "**Detecting and Recognizing Centerlines as Parabolic Sections of the Steerable Filter Response**", MIPRO, Opatija, Croatia, May 2011

49. T. Bräunl, S. Feyrer, W. Rapf, M. Reinhardt, "**Parallel Image Processing**", Springer-Verlag Berlin Heidelberg 2001

50. K. Dawson-Howe, "**A Practical Introduction to Computer Vision with OpenCV**", John Wiley & Sons Ltd, 2014

51. D. L. Baggio, D. M. Escriva, N. Mahmood, R. Shilkrot, S. Emami, K. Ievgen, S. Saragih, "**Mastering OpenCV with Practical Computer Vision Projects**", Packt Publishing Ltd. Dec. 2012

52. S. Blokzzyl, M. Vodel, W. Hardt, "**FPGA-based Approach for Runway Boundary Detection in High-resolution Color Images**", 978-1-4799-2179-9/14,

IEEE 2014

53. G. P. Stein, E. Rushinek, G. Hayun, Amnon Shashua, "**A Computer Vision System on a Chip: A Case Study From The Automotive Domain**", Mobileye Vision Technologies Ltd. www.mobileye.com, Israel

54. L. J. Belaid, W. Mourou, "**Image Segmentation: A Watershed Transformation Algorithm**", Image Anal Stereol 28:93-102, 2009

55. S. Oka, A. Garg, K. Varghese, "**Vectorization of Contour Lines from Scanned Topographic Maps**", Automation in Construction 22, 192-202, Elsevier, 2012

56. S. Thakker, H. Kapadia, "**Image Processing on Embedded Platform Android**", International Conference on Computer, Communication and Control, IC4, IEEE, 2015

57. X. Miao, S. Li, H. Shen, "**On-board Lane Detection System for Intelligent Vehicle based on Monocular Vision**", International Journal on Smart Sensing and Intelligent Systems, Vol. 5, No. 4 Dec 2012

58. X. Yang Z. Ling, "**Research on Lane Detection Technology based on OpenCV**", 3rd International Conference on Mechanical Engineering and Intelligent Systems, (ICMEIS) 2015

59. C. T. Johnston, K. T. Gribbon, D. G. Bailey, "**Implementing Image Processing Algorithms on FPGAs**", Proceedings of the Eleventh Electronics New Zealand Conference, Palmerston North, New Zealand, pp. 118-123, Nov. 2004

60. R. K. Satzoda, S. Lee, F. Lu, M. M. Trivedi, "**Snap-DAS: A Vision-based Driver Assistance System on a Snapdragon Embedded Platform**", Intelligent Vehicles Symposium, IEEE, Jun. 2015

61. J. Lee, J. Cho, "**Effective Lane Detection and Tracking Method using Statistical Modeling of Color and Lane Edge-orientation**", Advanced in Information Sciences and Service Sciences, Volume 2, Number 3, Sept. 2010

62. H. Estl, "**Paving the way to self-driving Cars with Advanced Driver Assistance Systems**", Worldwide Systems Marketing for Advanced Driver Assistance Systems (ADAS), Texas Instruments, Aug. 2015

63. S. Chai, "**Mobile Challenges for Embedded Computer Vision**", Embedded Computer Vision, Advances in Pattern Recognition, Springer-Verlag, 2009

64. W. Anne, "**Embedded Vision: Which standards are necessary to prepare the sector for the future?**", Artikel Id:10857677, www.ibv.vdma.org, Nov. 2015

65. M. Scarpino "**OpenCL in Action**", Manning Publication Co. 2012

66. A. Bharade, S. Gaopande, A. G. Keskar, "**Adaptive pre processing and image stabilization on embedded platform**", 9th International Conference Industrial and Information Systems (ICIIS), Dec. 2014

67. G. Savitha, P. S. Venugopal, S. devi, N. Chiplunkar, "**An Approach for Object Detection in Android Device**", Fifth International Conference on Signals and Image Processing (ICSIP), 2014

68. A.Prioletti, A. Møgelmose,P. Grisleri, M. Trivedi, A. Broggi,T. B. Moeslund,

"**Part-Based Pedestrian Detection and Feature-Based Tracking for Driver Assistance: Real-Time, Robust Algorithms, and Evaluation**", TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 14, NO. 3, IEEE, SEP. 2013

69. Linley Mobile Conference, "**Enabling Intelligent Vision Processing in Embedded Systems**", Apr. 2015
70. J. Bier, "**Processing Options For Implementing Vision Capabilities in Embedded Systems**", Article, Altera Corporation USA
71. M. Imran, K. Benkrid, K. Khursheed, N. Ahmad, M. O'Nils, N. Lawal, "**Analysis and characterization of embedded vision systems for taxonomy formulation**", SPIE 8656, Real-Time Image and Video Processing 2013, Feb. 2013
72. G. Hegde, N. Kapre, "**Energy-Efficient Acceleration of OpenCV Saliency Computation using Soft Vector Processors**", 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, IEEE, 2015
73. J. Cerezuela-Mora, E. Calvo-Gallego, S. Sánchez-Solano, "**Hardware/Software co-design of video processing applications on a reconfigurable platform**", IEEE, 978-1-4799-7800-7, 2015
74. J. Fritsch, T. Kühnl, A. Geiger, "**A new performance measure and evaluation benchmark for road detection algorithms**", Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference, Oct. 2013
75. S. Blokzyl, M. Vodel, W. Hardt, "**A Hardware-Accelerated Real-Time Image Processing Concept for Hight-resolution EO Sensors**", Deutscher Luft- und Raumfahrtkongress, DocumentID: 281364, 2012
76. W. F. Abaya, J. Basa, M. Sy, A. C. Abad, E. P. Dadios, "**Low Cost Smart Security Camera with Night Vision Capability Using Raspberry Pi and OpenCV**", 7th International Conference Humanoid, Nanotechnology, Information Technology Communication and Control, Environment and Management (HNICEM), IEEE, Nov. 2014
77. P. B. Nayana, S. Kubakaddi, "**Implementation of Hand Gesture Recognition Technique for HCI Using OpenCV**", International Journal of Recent Development in Engineering and Technology, ISSN 2347 - 6435, Vol. 2, Issue 5, May 2014
78. Y. Chen, S. Gaz, B. Zhang, K. Du, "**A Pedestrian Detection and Tracking Systems based on Video Processing Technology**", Fourth Global Congress Intelligent Systems (GGIS), IEEE, Dec. 2013
79. N. Madduri, "**Hardware Accelerated Particle Filter for Lane Detection and Tracking in OpenCL**", Master Thesis at Department of Informatics Julius-Maximilians-Universität Würzburg, Jan. 2014
80. J. Leboeuf-Pasquier, A. G. Villa, K. H. Burgos, D. Carr-Finch, "**Implementation of an Embedded System on a TS7800 Board for Robot Control**", Electronics, Communications and Comnputers (CONIELECOMP), Feb. 2014
81. Y. Sugiki, T. Yamaguchil, H. Haradal, "**Implementation of Optical Flow Measurement System with an Embedded Processor**", 15th International

Conference on Control, Automation and Systems (ICCAS), Oct. 2015

82. A. Fathi, J. Krumm, "**Detecting Road Intersections from GPS Traces**", Sixth International Conference on Geographic Information Science, Zurich, Sep. 2010

83. A. Habibovic, C. Englund, J. Wedlin,"**Current Gaps, Challenges and Opportunties in the Field of Road Vehicle Automation**", F2014-ACD-035, FISITA World Automotive Congress, Maastricht Jun. 2014

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Studentenservice – Zentrales Prüfungsamt
Selbstständigkeitserklärung

Name:	Bitte beachten:
Vorname:	1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.
geb. am:	
Matr.-Nr.:	

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keinem anderen Prüfer als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: ..... Unterschrift: .....

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.