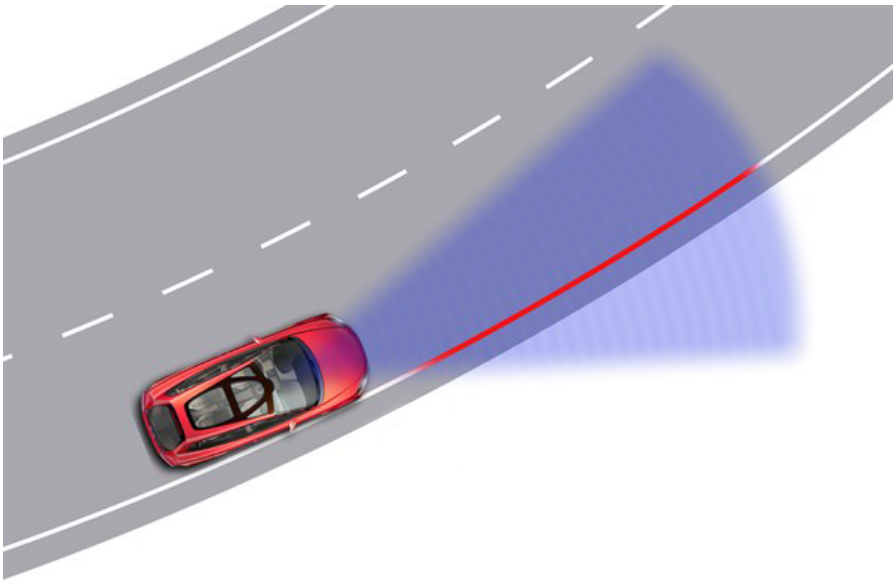
CHALMERS



Integration of a lane detection system in a virtual environment to test and evaluate active safety and autonomous driving.

A research project at Volvo cars to emulate the behavior of a lane detection system.

Master's Thesis in System Control and Mechatronics

Joakim Olsson

Department of Applied Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2015
Master's Thesis 2015:57

Master's thesis in system control and mechatronics

Integration of a lane detection system in a virtual environment to test and evaluate active safety and autonomous driving.

A research project at Volvo cars to emulate the behavior of a lane detection system

JOAKIM OLSSON

Department of Applied Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2015

Integration of a lane detection system in a virtual environment to test and evaluate active safety and autonomous driving.

A research project at Volvo cars to emulate the behavior of a lane detection system

JOAKIM OLSSON

Master's Thesis 2015
ISSN 1652-8557
Department of Applied Mechanics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: + 46 (0)31-772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2015

Abstract

A vehicle's ability to determine its position relative to the road is a crucial feature for active safety functions and autonomous driving. Consequently, autonomous vehicles and current vehicles featuring active safety systems are usually equipped with a lane detection system that estimates the position of the vehicle relative to the road lane markings.

The field of active safety has grown exponentially during the past decade, which in turn has hugely increased the demand for functional testing. Virtual environments are particularly economical, rapid, and safe for functional testing and are today also used to simulate autonomous driving.

The aim of this thesis was to design and implement a model for a visual-based lane detection system to be integrated in the virtual environment used for functional tests at Volvo Cars. A virtual camera model with appurtenant camera algorithms was developed and enabled the computation of a 2D-image through projection of points in 3D-space from the virtual environment, mimicking the behavior of a real visual lane detection system.

To emulate the real performance of the sensor, various software components were modeled and their outputs compared to field data from a production lane detection system. A multivariate analysis was executed to characterize the real system's error and improve the virtual model. The error performance was modeled with a 3-dimensional confidence curve together with a low pass FIR filter to create the extreme boundaries of the sensor. The model of the system behavior was merged with the camera model to create a vision based virtual lane detection system. This virtual system was integrated and verified in the virtual environment for functional test at Volvo Cars.

Acknowledgements

The work presented in this report is founded upon research, hard work and dedication. Despite this, the project would not have been executable without the support of some key individuals and organizations. Consequently I would like to take this opportunity to show my gratitude towards them.

First of all to my two supervisors professor Marco Dozza at Chalmers university of technology and Liu Feng at Volvo Cars corporation for your kind and supportive advisory. Without your guidance the success of the project would not have been achievable.

I would also like to express my gratitude towards Volvo Cars Corporation which has helped me with expertise and provided me the necessary data.

Joakim Olsson, Gothenburg 15/05/18

Contents

Nomenclature

Symbol	Units	Description
$[x_c,y_c,z_c]$	m	Position coordinates in the camera coordinate system.
$[\theta_c,\phi_c,\psi_c]$	*rad*	Orientation angles of the x-,y- and z-axis of the camera frame relative to the vehicle coordinate system.
$[x_v,y_v,z_v]$	m	Position coordinates in the vehicle coordinate system.
$[\theta_v,\phi_v,\psi_v]$	*rad*	Orientation angles of the x-,y- and z-axis of the vehicle frame relative to the global coordinate system.
$[X,Y,Z]$	m	Position coordinates in the global coordinate system.
α	*rad*	Horizontal view angle.
β	*rad*	Longitudinal view angle.
M	*pixels*	Horizontal resolution value.
N	*pixels*	Longitudinal resolution value.
f	m	Focus length.
f_x	m	Horizontal spread of the image plane.
f_y	m	Longitudinal spread of the image plane.
R_x	-	Rotation matrix for rotation operations around the x-axis.
R_y	-	Rotation matrix for rotation operations around the y-axis.
R_z	-	Rotation matrix for rotation operations around the z-axis.
$R_{cam2veh}$	-	Rotation matrix to convert a point from the camera frame to the vehicle coordinate system.
$R_{veh2cam}$	-	Rotation matrix to convert a point from the vehicle frame to the camera coordinate system.
$\bar{P}_c$	m	Point in the camera coordinate system.
$\bar{P}_v$	m	Point in the vehicle coordinate system.
$\bar{P}_g$	m	Point in the global coordinate system.

o_c	m	Location of the camera origin relative to the vehicle frame.
o_v	m	Location of the vehicle origin relative to the camera frame.
u	m	Continous x-position of a point in the image plane.
v	m	Continous y-position of a point in the image plane.
m	int	Discrete pixel position in the x-position of a point in the image plane.
m	int	Discrete pixel position in the y-position of a point in the image plane.
ψ	rad	Yaw orientation of the host car.
$\dot{\psi}$	$\frac{rad}{s}$	Yaw rate of the host car.
V	$\frac{m}{s}$	Host car speed.
t_{ψ_i}	s	Time stamp in the yaw array.
t_{V_i}	s	Time stamp in the speed array.
d_l	m	Left lane range estimation.
d_r	m	Right lane range estimation.
X_L	m	Left lane x-position in the global frame.
Y_L	m	Left lane y-position in the global frame.
X_R	m	Right lane x-position in the global frame.
Y_R	m	Right lane y-position in the global frame.
V_{dir}	m	Vehicle direction vector.
V_{lane}	m	Vector pointing from the position of the vehicle to the estimated lane point.
V_{est}	m	Vector pointing from the position of the vehicle to the lane reference point.
θ_{lane}	rad	Angle between the direction vector and the lane vector.
θ_{est}	rad	Angle between the direction vector and the estimation vector.
a_{L_0}	m	Lateral offset to the left lane represented by the zero-component in the left lane polynomial vector

a_{R_0}	m	Lateral offset to the right lane represented by the zero-component in the right lane polynomial vector
D	-	Data matrix with n columns representing the included dimensions and m rows which represents the observations.
C	-	Covariance matrix of the principal components.
ccv	-	Covariance function.
P_w	-	Pixel width.

Acronyms

ACC	**A**daptive **C**ruise **C**ontrol
CSA	**C**urve **S**peed **A**daptation
ELE	**E**xpedition **L**og **E**xaminer
ELS	**E**xpedition **L**og **S**anner
GUI	**G**rafical **U**ser **I**nterface
GTA	**G**round **T**ruth **A**nalysis
HSI	**H**ue **S**aturation **I**ntensity
LD	**L**ane **D**etection
LDA	**L**ane **D**etection **A**lgorithm
LDS	**L**ane **D**etection **S**ystem
LDW	**L**ane **D**eparture **W**arning
LKA	**L**ane **K**eeping **A**id
SIL	**S**oftware **I**n the**L**oop
PCA	**P**rincipal **C**ompoant **A**nalysis
VCTS	**V**olvo **C**srs **T**raffic **S**imulator

1

Introduction

1.1 Background

Fatal road accidents is the 8th most common cause of death worldwide with over 1.2 million casualties yearly [1]. Research shows that around 93% of all traffic accidents are caused by human deficiencies [2]. The increment in number of road accidents has led to a fast growth in the market of active safety systems, which is why, it is now one of the most resource focused research and development areas in the field of automotive design. A contribution to the expansion is the improved hardware accessories now available on the market such as high performance processors, which enables solid real time computation, and a great selection of sensors to monitor the surroundings of the car. These are the most critical prerequisites in the process of obtaining fully self-driving cars. Active safety does not only generate a safer driving environment for the costumer but also provides the driver with various types of supporting functions which immensely improves the overall driving experience.

One of the most crucial components in the development of active safety is the lane detection system (LDS) which actively finds the lane markers on the road. Lane detection is a well-studied field and has been researched since the mid-1980s [3]. Due to the constraint that the LDS should perform in real time speed a lot of the initial studies conducts some major simplifications and assumptions such as straight road, constant road width etc. to lower the computation pressure on the processor. Even tough this constrain has repressed the field, the exponential growth of processor power the last couple of decades has enabled the use of complex computer vision algorithms and encouraged the studies of lane detection even further.

The LDS is used to determine the position of the host vehicle relative to the road, estimate the heading direction and to give road information such as number of lanes, road width, curvature etc. Functions such as lane keeping aid (LKA), adaptive cruise

control (ACC), lane departure warning systems (LDW) and curve speed adaptation (CSA) all depend on a reliable and accurate LDS.

Figure 1.1: Lane marker detection is a crucial component in the field of active safety [4].

As active safety functions, increase in both complexity and number so does the need of a virtual testing environment to support further development. Functional tests through simulations offers both fast and safe response on the behavior of a active safety function which is therefore a preferable function evaluation method. As hardware tests are both expensive and slow as well as unsafe, they should hence preferably be used in the final stage of the development line. Consequently, Volvo cars has developed a simulation environment called *Volvo Cars Traffic Simulator (VCTS)* which enables assessment of active safety functions and testing of critical traffic scenarios. The ($VCTS$) environment supplies the function developer with quick feedback, which is why the simulator is a key component in Volvos everyday work of creating innovative and intelligent active safety functions. The simulator consists of a huge amount of models, representing different components in the actual vehicle, which all together makes a virtual representation. The more accurate the models mimics the behavior of the component it is representing, the higher accuracy level the overall simulator can obtain. Even though the LDS is a crucial component in the field of active safety, most of the research that has been conducted in the field has focused on real life implementations.

For reasons like this, it is of great importance to have a virtual representation of the LDS that accurately mimics the performance of the real system. The work presented in this report will consequently be focused on the exploration on how a virtual LDS can be constructed found on a real vision based LDS and how to most effectively capture the behavior and performance of the system in its entire area of application. The master

thesis is carried out in cooperation with Volvo Cars to design and implement a visual-based *LDS* to be integrated in the virtual environment used for functional tests at Volvo. Even though the work is focused on the modelling of an *LDS*, it can be applied in a broader field, for any type of system modelling. The work will be restricted to include only camera vision *LDS* which is by far the most common approach due to its simplicity and low hardware cost. Usually multiple cameras are used to capture a broader range spectrum and thereby give the *LDS* preferable properties on both short, mid as well as long range. This report will only focus on single camera setup but the work can easily be expanded to include multiple camera *LDS*.

1.2 Previous research

If the *LDS* system is stripped down to its most fundamental components the task of the system is to capture the 3-dimensional world in its proximity, interpret the data as well as distinguishing the road lanes from the rest of the data set and finally map the positions of the lane markers in 3D-space. One way of capturing the surroundings is by using a LIDAR-system that with great precision can measure both range and light intensity at multiple points in 3D-space. The LIDAR is characterized by high accuracy but to an excessively high cost which is why the LIDAR system is not suitable for mass production. More conventionally a camera is used to transform the 3-dimensional world into 2D-space.

The identification of a lane marker in a 2D-picture is the most complex and time consuming process in an *LDS* system. Most commonly a lane marker is defined as a region with high contrast around its proximity with limited intensity gradients internally. A lot of research includes *Hough transformation* to distinguish the lane pixels from the rest [5] but multiple other approaches exists such as *particle filtering* [6] or *hue-saturation-intensity* (HSI) analysis [7]. As mentioned previously a lot of different models have been developed through the years which all applies different assumptions to simplify this complicated task. In [8] the algorithm is based on a straight line model and assumes that the road have a relative low curve ratio. The algorithm is effective for straight road situations but does not cover the whole spectrum of road geometries such as roundabouts and turns. A different approach is to utilize a parabolic model [9] which has less constraints regarding to group the road geometries into features.

1.3 Objectives and contribution

As previously shown, a great deal of research has been conducted for real life implementations of *LDS*. However, the aim of this project is to contribute with a virtual representation of the *LDS* which in the development process would enable fast and safe response through simulations. The main focus of the research presented in this report will therefore be the development of a virtual *LDS* that effectively can emulate the behavior and the performance of a real *LDS*. The research will, as previously stated, be

focused on visual based LDS. Today a lot of active safety functions utilizes the on board camera which visually monitors the surroundings of the vehicle. The camera is therefore a very important component to be included in functional tests. The thesis will for this reason research the possibility of including a virtual visual component in *Software In the Loop* (SIL) tests.

Apart from trying to implement a virtual representation of the LDS the goal of this research is to analyse the behavior of the LDS and by that pinpoint the factors that is influential on its performance. The knowledge gained from this analysis could aid the field by identifying the factors that is contributing to major errors thereby reducing the LDS performance.

Even though the target of the project is to emulate a LDS, the techniques developed can be generally applied for other types of sensor modelling tasks.

1.4 Approach

As the parabolic lane detection model gives the highest degree of generality as well as geometrically freedom the work presented in this report will be focused on system modelling of a parabolic LDS. Even though a parabolic system will be modelled it can generally be applied to include a broader field of lane detection models. The lanes in a parabolic LDS model is represented by a higher order polynomial which takes the longitudinal range from the host vehicle as the input and gives the lateral offset as the output like in equation (1.2). The parameter a_0 is the estimated lateral offset from the host vehicle to the lane at the present time stamp which is shown in figure 1.2.

$$f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} .... a_1 \cdot x + a_0 \tag{1.1}$$

$$0 \leq x \leq d \tag{1.2}$$

$$\begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix}, d \tag{1.3}$$

By creating a polynomial from the positions of the known feature points in the image the lane markers are transformed from a discrete representation to a continuous which gives the system access to the lane positions at any range. The order of the polynomial can be chosen according to preference where a higher order polynomial can theoretically obtain

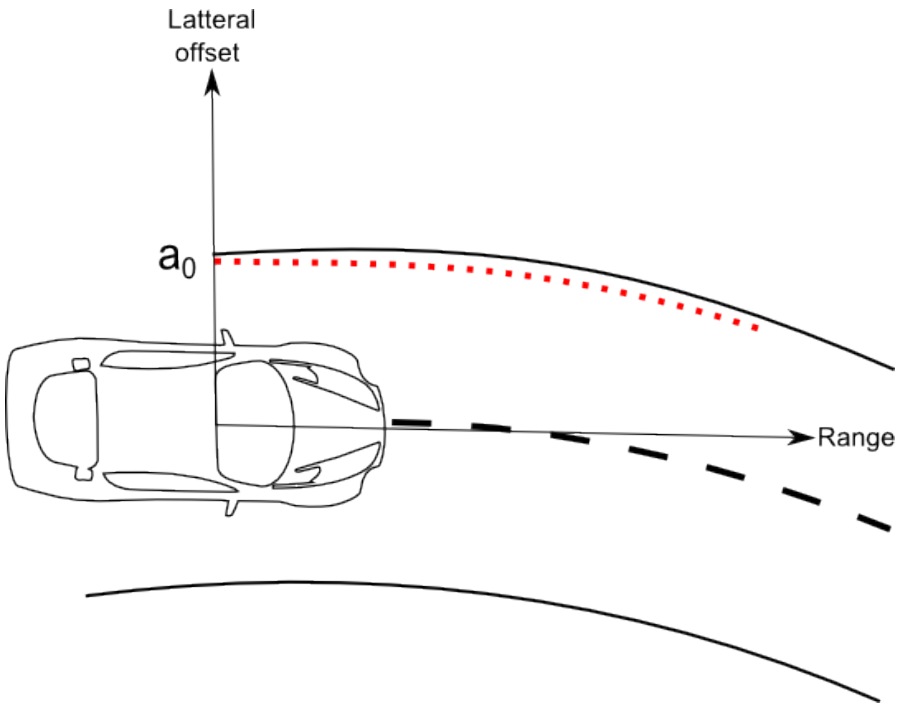


Figure 1.2: The figure shows how the polynomial representation of the lane marker is defined where the x-axis is the range from the origin of the host car and the y-axis is the lateral offset. a_0 is the zero order parameter in equation (1.2) and is by definition the lateral offset closest to the host vehicle.

a higher accuracy and a broader geometry span but to the cost of a higher computation load while lower orders gives the opposite. For this work, a third order polynomial *LDS* will be analysed but the emulation model will be constructed to allow any preferred polynomial order.

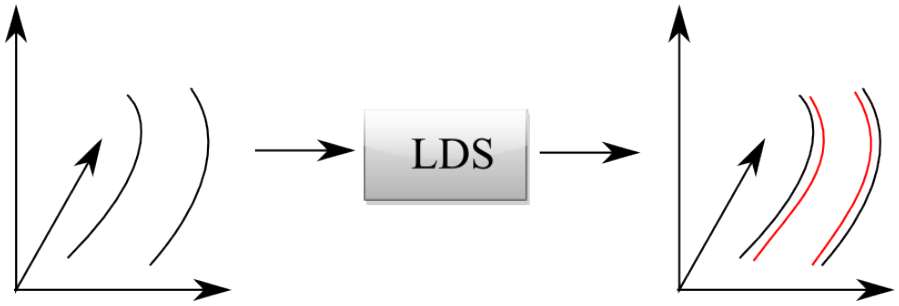


Figure 1.3: Overview of the *LDS* function. The system takes a 3-Dimensional environment as input and produces the position of the lane markers in 3D-space as output.

The objective of the *LDS* can thereby be decomposed into the process of tuning the entries in the parameter vector shown in equation 1.3 to the values that minimizes the error. Like figure 1.3 shows, the system is fed with a 3D-environment, interprets the surroundings and outputs the lane marker positions in the form of the parameter vector at any given time stamp. The 3D-world in the simulation environment is represented by a point cloud scenario. The scenario includes roads, vehicles, pedestrians and animals, which the host car can interact with. The objects are constructed by multiple points in 3D-space, referenced to a common global coordinate system.

An alternative approach to the model structure presented in figure 1.3 is the one in figure 1.4. Here the *LDS*, which serves as a black box, has been decomposed into two subsystems, a virtual camera and a lane detection algorithm block. There are many advantages gained from this type of setup compared to the initial concept. For instance, extracting the camera from the *LDS* will facilitate the black box modelling. A camera is a well-known application which easily can be represented virtually so by separating the hardware function from the software functions, the unknown components in the black box will then be decreased, which in turn simplifies the modelling. The size of a black box should preferably be restricted as much as possible to make it easier to pin point and emulate special features in the system.

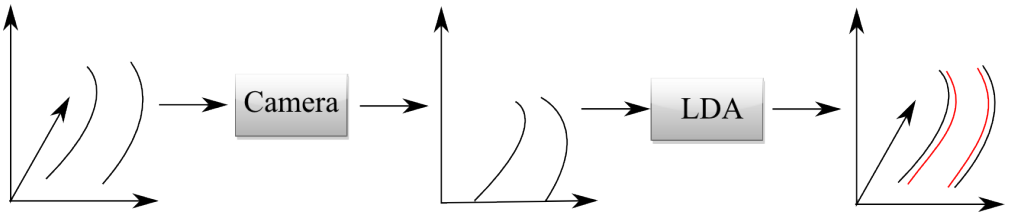


Figure 1.4: Alternative system model approach where a virtual camera is implemented to convert the 3D-environment into a 2-dimensional image which serves as input for the lane detection algorithm.

A second advantage is the gained natural error that will be included in the camera. For instance, the model will naturally emulate discretisation errors due to the fact that the image is represented by a pixel grid and not a continuous space. These types of errors are relatively small and could consequently be complicated to capture in a black box modelling process. As they most definitely will be included in the real *LDS* it is an error that should be included in the virtual system.

The final benefit gained from this setup is the enabling of hardware evaluation. By having a virtual model of the camera the system performance can be tested based on different camera settings. Camera settings that can be tested are position, alignment, pixel resolution, view angle, focus length etc. This will give the virtual *LDS* a great advantage as it can be used to check if a better hardware solution can supply a better outcome or if a cheaper camera can receive similar results. Next chapter will present the virtual camera model with the appurtenant algorithms.

2

Method

This chapter is created to guide the reader through the execution of the project and to describe how the final lane detection model is obtained. The chapter is divided into four sections (*Virtual Camera Model*, *System Performance*, *Data Acquisition*, *Principal Component Analysis*) which all includes multiple subsections.

To create the lane polynomials $f(x)$, a virtual camera model is initially used to project the surrounding 3D-points onto the camera image. The camera then distinguish the detected lane positions and projects them back into 3D-space. The virtual camera model, along with the appurtenant camera algorithms are described in section 2.1.

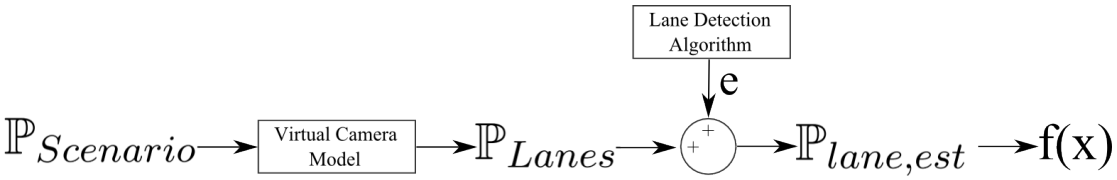


Figure 2.1: Graphical schematics over how the *LDS* converts a point cloud in a simulation scenario $\mathbb{P}_{Scenario}$ into third degree polynomials $f(x)$, representing the estimated lane markers by using a *Virtual Camera Model* and a model over the *Lane Detection Algorithm*.

When the camera model has mapped the detected lane points back to 3D the lane positions $\mathbb{P}_{Lanes}$ are situated almost perfectly compared to the true positions of the lane markers. The only error that has been added in the image process are errors caused by the hardware in the camera. As this alone, does not capture the entire behaviour of the *LDS* a lane detection algorithm block has been added with the purpose of shifting the perfect positioned points to a location where the *LDS* probably would have estimated them to according to the error performance of the system. The analysis of the system performance is described in section 2.2 along with a tool, specially developed to help the user visualize the behaviour of the *LDS*.

Section 2.3 describes how *LDS* field data collected through real life test expeditions, can be extracted and computed to form the input for the virtual *LDS* model with the purpose of mimicking the error performance. The chapter is then finished with section 2.4, which describes a common statistical tool for multivariate analysis which was used to analyse which factors most affects the *LDS*.

2.1 Virtual camera model

As argued in section 1.4 the use of a virtual camera model has multiple advantages. By including a camera, the input to the lane detection algorithm will be in 2D-space which is what the real *Lane Detection Algorithm* (*LDA*) computations are based on. This isolates the real unknown components and thereby aids the modelling of the *LDA* black box.

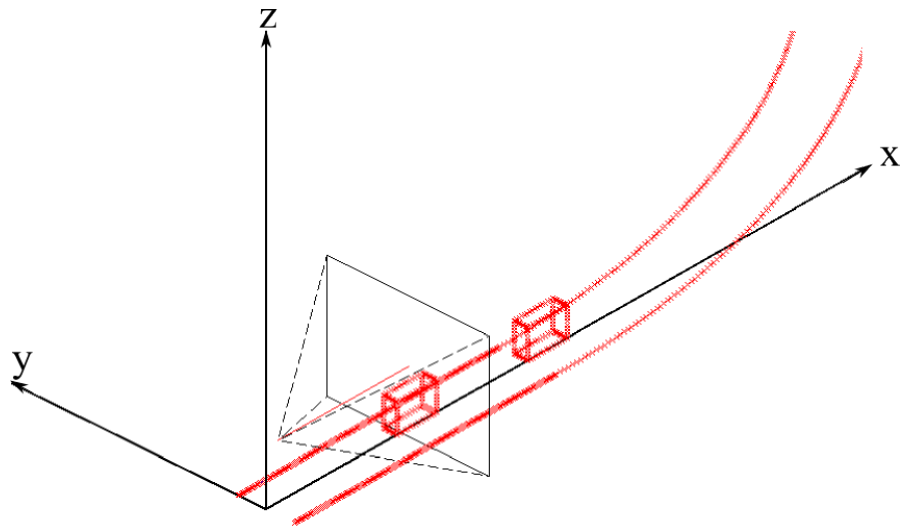


Figure 2.2: An example of a single time instance in a point cloud scenario including a left curved road, two cars (red boxes) and the host vehicle with its field of view. All points are referenced relative to the common global coordinate system $[x,y,z]$.

The camera model interacts with a 3D world and projects points in its field of view onto the camera image. Figure 2.2 shows a scenario where the host car and two additional drivers (red boxes) are driving on a left curved road. All points in the 3D-world are referenced relative to a common global coordinate system. The roads consists of stationary points while objects like cars, animals or pedestrians are allowed motion throughout the simulation.

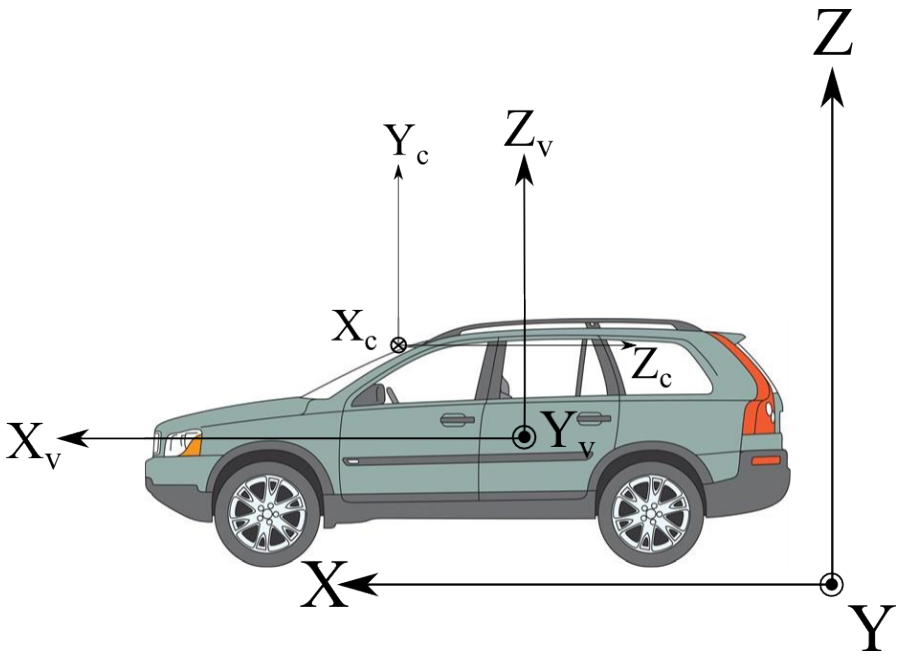


Figure 2.3: The three coordinates systems necessary for the camera model to fully function. The camera coordinate system are denoted $[x_c,y_c,z_c]$ while the vehicle coordinate system is denoted $[x_v,y_v,z_v]$. The camera system is coupled to the vehicle coordinate system while the vehicle system is coupled to the global system called $[X,Y,Z]$

2.1.1 Coordinate systems and transformations

The camera model is utilizing three coordinate systems, $[x_c,y_c,z_c]$, $[x_v,y_v,z_v]$ and $[X,Y,Z]$ which are defined in figure 2.3. All three coordinate systems are right hand oriented Cartesian systems in euclidean space. Coordinate system $[x_c,y_c,z_c]$ defines the position and motion of the camera and is defined relative to the vehicle coordinate system $[x_v,y_v,z_v]$ which in turn is defined relative to the global coordinate system $[X,Y,Z]$.

The z-axis of the camera system is targeted in the negative field of view direction according to the standard convention [10]. Figure 2.4 shows a more detailed declaration of the camera coordinate system and includes the camera image plane. The image plane is where the three dimensional points that are included in the field of view are projected on and represented in 2D-space. The angles α and β are the so called field of view angles which determines the propagation of the image plane. α is commonly considered as a design parameter which comes as a property of the camera while β are determined through equation (2.1) where M, N are the resolution values. The spread of the image plane is calculated with equations (2.2)-(2.3) with the use of the camera focus length f.

$$\beta = \tan^{-1}\left(\frac{N}{M} \cdot \tan(\alpha)\right) \tag{2.1}$$

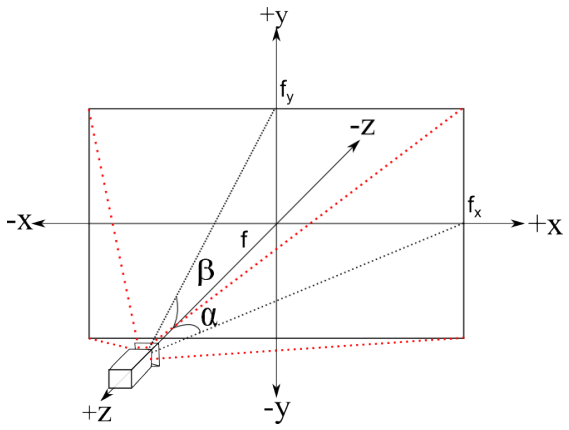
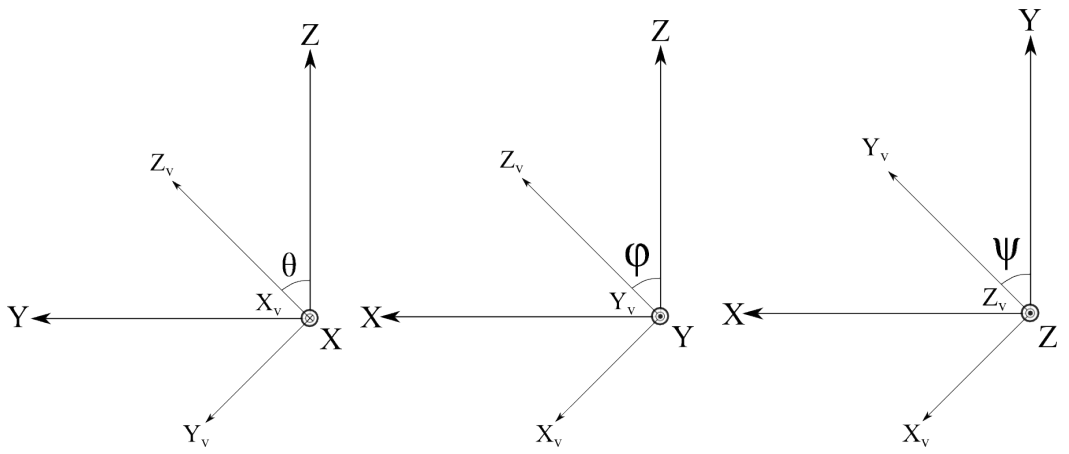


Figure 2.4: Definition of the camera coordinate system including view angles α, β and the image plane stretching from $-fx$ to fx and from $-fy$ to fy. The camera system is faces in the negative field of view direction.

$$f_x = f \cdot \tan \alpha \tag{2.2}$$

$$f_y = f \cdot \tan \beta \tag{2.3}$$

The camera and the vehicle both has six degrees of freedom coupled to the vehicle- and the global coordinate system respectively. The camera has three position coordinate $[x_c,y_c,z_c]$ and three orientation coordinates $[\theta_c,\phi_c,\psi_c]$ which are tied to the vehicle system. Generally the position and the orientation of the camera is stationary throughout a simulation but can be modelled with motions like vibrations. The vehicle has the position vector $[x_v,y_v,z_v]$ and the orientation vector $[\theta_v,\phi_v,\psi_v]$ which are referenced relative to the global coordinate system displayed in figure 2.5. Rotation angle θ describes the rotation of a coordinate system relative to its corresponding reference system around the x-axis while ϕ and ψ denotes the rotations around the y- and the z-axis.



(a) Graphical definition of the θ -angle.

(b) Graphical definition of the ϕ -angle.

(c) Graphical definition of the ψ -angle.

Figure 2.5: A graphical definition of the rotation angles θ_v,ϕ_v,ψ_v where θ_v is a rotation of the vehicle coordinate system around the x_v-axis relative to the global coordinate system while ϕ_v and ψ_v denotes rotations around the y_v- and the z_v-axis respectively.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To translate a point from one coordinate system to another the point needs to be rotated according to the present orientation of its coordinate system. The rotation of a coordinate system in Euclidean space are done with the rotation matrices Rx, Ry, Rz [11]. To define the orientation of a system, Tait-Bryan angles are used with an extrinsic rotation method (X-Y-Z). Due to the fact that rotation in Euclidean space is a non-commutative process, the arrangement of the rotation matrices is of great importance.

$$\overline{P_g} = R_z(-\psi_v) \cdot R_y(-\phi_v) \cdot R_x(-\theta_v) \cdot \left[R_{cam2veh} \cdot R_z(-\psi_c) \cdot R_y(-\phi_c) \cdot R_x(-\theta_c) \cdot \overline{P_c} + o_c \right] + o_v \tag{2.4}$$

$$R_{cam2veh} = \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2}\cos\frac{\pi}{2} & \sin^2\frac{\pi}{2} \\ -\sin\frac{\pi}{2} & \cos^2\frac{\pi}{2} & -\cos\frac{\pi}{2}\sin\frac{\pi}{2} \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.5}$$

Equation (2.4) shows how a point in the camera frame $\overline{P_c}$ is converted into point in the global coordinate system $\overline{P_g}$. The camera point is initially rotated extrinsically to align

with the vehicle coordinate system according to its current state of orientation. As the camera is pointing in the negative field of view direction shown in figure 2.3, rotation matrix $R_{cam2veh}$ is used to rotate the camera frame in the same alignment as the vehicle frame. The matrix is simply obtained by rotation the coordinate system eccentrically 90° around the z-axis and -90° around the y-axis. The location of the camera origin o_c relative to the host vehicle is added and then rotated according to the Tait-Bryan state of the vehicle. Finally the position of the host car relative to the global frame o_v is added. To convert a point in the global space to the camera coordinate system equation (2.6) can be applied.

$$\overline{P_c} = R_x(\theta_c) \cdot R_y(\phi_c) \cdot R_z(\psi_c) \cdot R_{cam2veh} \cdot \Big[R_x(\theta_v) \cdot R_y(\phi_v) \cdot R_z(\psi_v) \cdot [\overline{P_g} - o_v] - o_c \Big] \quad (2.6)$$

2.1.2 Field of View Algorithm

The virtual camera model needs the ability to determine if a point lies inside or outside its field of view to fully function. The most obvious reason for a field of view algorithm is to enable control and restriction over what the camera can see but it is also beneficial to speed up the computation as well as to disable bugs generated through false projection. If points that lies on the positive z-axis (behind the camera) are projected onto the image plane it will be falsely positioned on the inverse part of the image plane which will create an image effect.

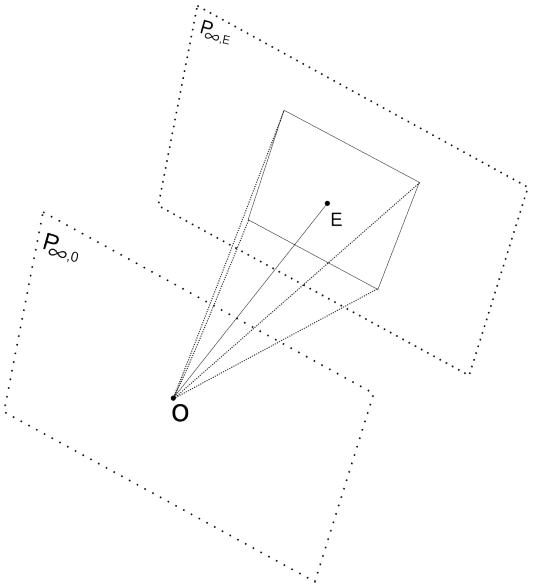


Figure 2.6: The field of view volume which is bounded geometrically by a polyhedron with its central line stretching from the optical center 0 to the end point of the field of view E.

The field of view is geometrically bounded by a four sided polyhedron with a central

line stretching from its optical centre O to the end of sight E (see figure 2.6). In theory the optical endpoint lies on an infinite distance away from the optical centre but as the camera discretizes the continuous world, the range of sight is bounded by the camera resolution. The field of view volume is determined by the field of view angles α and β according to equation (2.1)-(2.3).

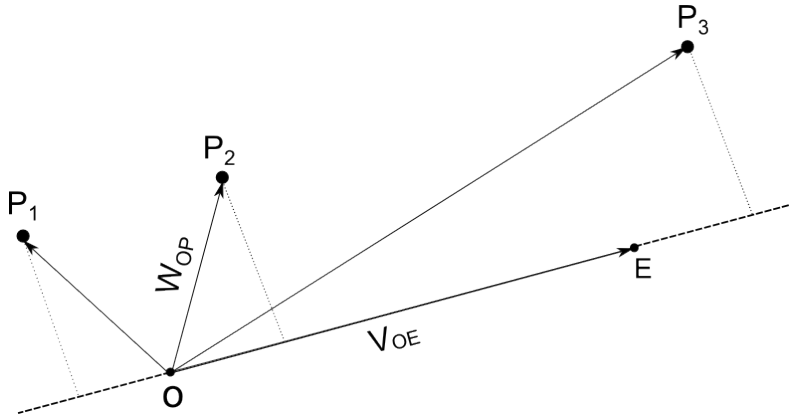


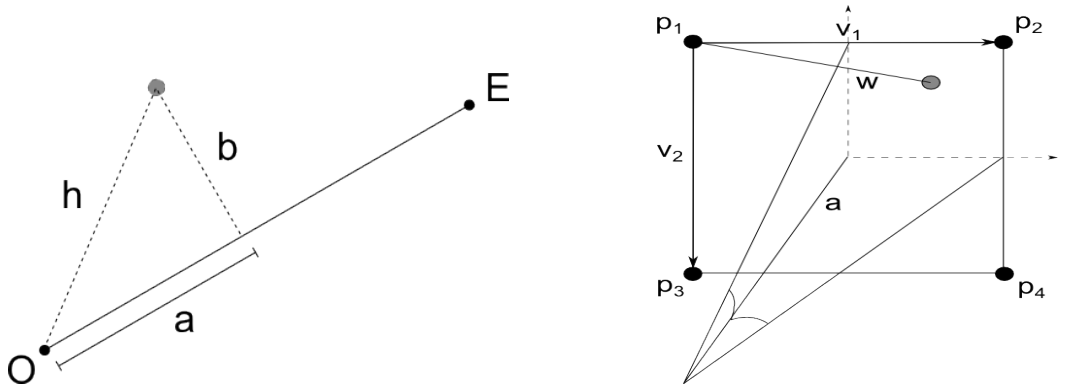
Figure 2.7: A description of how the field of view angle algorithm determines if a points is inside or outside the field of view by checking if the projection of the view vector W_{OP} ends up on the optical view segment V_{OE} or not. Here points P_1 and P_3 are rejected while P_2 is passed on for further investigation.

The first step on checking if a point is inside the field of view volume or not is to look at the projection of the vector between the optical centre O and the point of investigation P denoted W_{OP} onto the optical view segment V_{OE} (see figure 2.7). The vector W_{OP} should pass the condition in equation (2.7) as well as the condition in equation (2.8). Point P_1 in figure 2.7 fails the condition in equation (2.7) as its projection lies on the wrong side of the optical centre giving it a negative projection vector. Point P_3 is projected on the far side of the optical end point which generates a projection vector larger than the optical line segment which in turn contravenes against the condition in equation (2.8). The only point in figure 2.7 that will be clarified by the algorithm is P_2 that passes on for further investigation. Points that pulls through these two conditions are bounded to lie inside the two infinite planes $P_{\infty,O}$ and $P_{\infty,E}$ in figure 2.6.

$$W_{OP} \bullet V_{OE} \geq 0 \tag{2.7}$$

$$V_{OE} \bullet V_{OE} \geq W_{OP} \bullet V_{OE} \tag{2.8}$$

If a point is cleared through tests (2.7) and (2.8) the algorithm needs to check if it is inside the four sided polyhedron. If the point passes this test it can be concluded that the point lies in the camera field of view and should thereby be projected onto the image screen, otherwise it should be discarded. To determine the state of the point, the



(a) (b)

Figure 2.8: (a) Shows the orthogonal triangle that is used to determine the range from the optical center to the point where the projection ends up. (b) Shows the field of view rectangle on range a, its corresponding side vectors V_1 and V_2 as well as the vector from corner 1 to the point that is to be investigated.

algorithm first needs to calculate where on the optical line segment OE the projection ends up. This is done by looking at the simple orthogonal triangle in figure 2.8a. The distance b is the closest range to the line segment and is calculated through equation (2.9) where x denotes the cross product of the two vectors V_{OE} and W_{OP}. The hypotenuse h is the norm of the view vector W_{OP}. Through an elementary Pythagorean Theorem operation the distance a can be found with equation (2.11).

$$b = \frac{\|V_{OE} x W_{OP}\|}{\|V_{OE}\|} \tag{2.9}$$

$$h = \|W_{OP}\| \tag{2.10}$$

$$a = \sqrt{h^2 - b^2} \tag{2.11}$$

Once the distance on the optical line segment has been determined the algorithm can calculate the rectangle that bounds the field of view polyhedron in 2-dimensions at this specific range. The rectangle has corner points $P_1 - P_4$ and the coordinates of the points are displayed below:

$$\begin{aligned}
P_1 &= [-a \cdot \tan\alpha, \;\; a \cdot \tan\beta, \;\; -a] \\
P_2 &= [\;\; a \cdot \tan\alpha, \;\;\; a \cdot \tan\beta, -a] \\
P_3 &= [-a \cdot \tan\alpha, -a \cdot \tan\beta, -a] \\
P_4 &= [\;\; a \cdot \tan\alpha, -a \cdot \tan\beta, -a]
\end{aligned}$$

The final assessment is to check if the point lies inside the bounding rectangle shown in

figure 2.8b. This is done by projecting the point onto the side vectors V_1 and V_2 and check if it fulfils all conditions in equation (2.12).

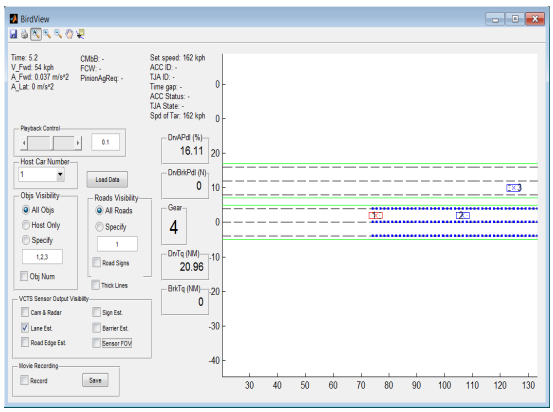
$$W \bullet V_1 \geq 0 \quad W \bullet V_2 \geq 0 \quad V_1 \bullet V_1 \geq W \bullet V_1 \quad V_2 \bullet V_2 \geq W \bullet V_2 \tag{2.12}$$

If a point manage all tests up to this stage, it can be concluded that the point lies inside or on the boundary of the field of view polyhedron and should thereby be further processed for projection onto the camera image.

2.1.3 Obstacles and Ray Intersection

To make the virtual camera model more authentic it is essential to include an algorithm that handles obstacles which blocks the camera view. This is a feature which is very hard to accurately emulate without the presence of the virtual camera which again shows the major benefit of having the model. Figure 2.9 clearly displays the problem that occurs in the absence of an obstacle algorithm. Figure 2.9a shows a real life lane estimation scenario. The view of the host car is blocked by a truck which prevents the algorithm from estimating the lanes further away. Figure 2.9b on the other hand, shows a possible estimation of the same scenario but without any ray intersection algorithm. View blockage is thereby a very important feature to capture which encourage further investigation of the algorithm.

(a) Lane marker estimation in a real traffic situation.



(b) Virtual lane marker estimation without a ray intersection algorithm.

Figure 2.9: (a) The lane estimation from a real *LDS*. Due to blockage, caused by the truck in front of the host car the *LDS* can only estimate the lane markers up to the visible point. (b) Simulation of the same scenario with a virtual *LDS* which lacks the competence of view intersection.

The points in the 3D simulation environment represents light sources that emits rays of light theoretically in all directions. The ray with the trajectory that stretches from the source of emission to the optical centre is the ray of interest and will be projected on the

camera image. The task of the ray intersection algorithm is to check if any of the objects present at the 3D scenario breaks any these paths. Figure 2.10 shows a simulation where an object intersect a ray of light which prevents it from reaching the camera image.

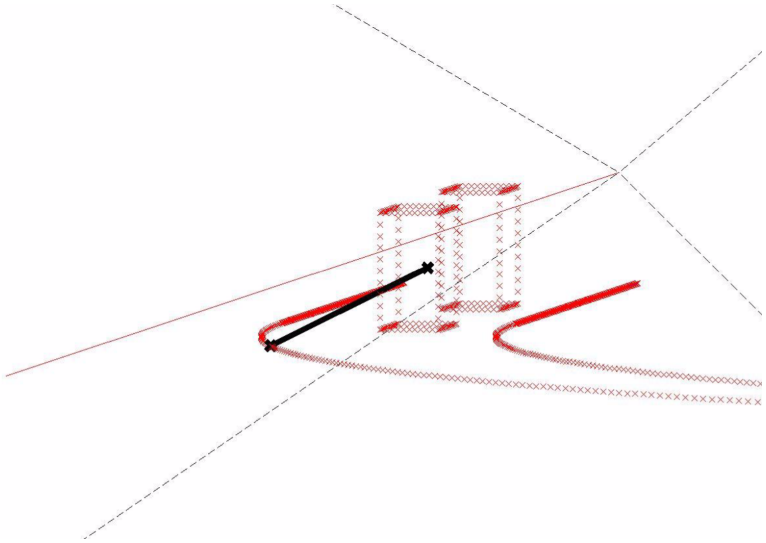


Figure 2.10: A simulated scenario where a car blocks the camera view by intersecting the ray between the point of light emission and the camera origin.

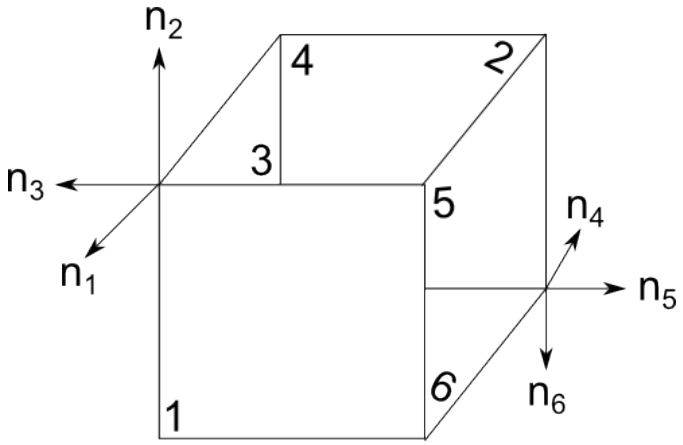


Figure 2.11: The geometrical definition of an obstacle in the 3D-environment. Every obstacle has six sides with six individual norm vectors. Norm vectors 1-3 are situated in corner number 1 while 4-6 are placed in corner 8.

Objects in the 3D simulation environment are represented by boxes like in figure 2.10. The approach of having objects bounded by boxes is conventionally used and helps to simplify the calculations which spares a lot of computational power [10]. More complex rendering algorithms exists which allows a broader spectrum of geometries, but due to

the fact that the simulations should work in near real time speed these algorithms are too heavy computational wise and will thus not be included. Every obstacle, more accurately consists of six sides with six individual norm vectors that defines the plane where the side is fixed onto (see figure 2.11).

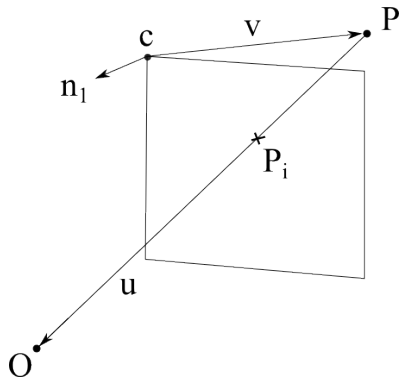


Figure 2.12: A geometrical definition of the intersection algorithm. The ray of light is represented by the vector u which stretches from the source of emission to the optical center. The obstacle plane is defined by its norm vector n which in this figure is situated at the corner point c. Vector v is the vector between the corner point and the source of emission while P_i denotes the intersection point.

To check if an intersection occurs between the plane where the side of investigation are situated and the ray between the point of interest and the camera, equation (2.13) is used. If the condition in equation (2.14) is fulfilled it can be concluded that the ray intersects the plane in the point P_i, otherwise the two points O and P lies on the same side of the plane and will accordingly not be interrupted by it.

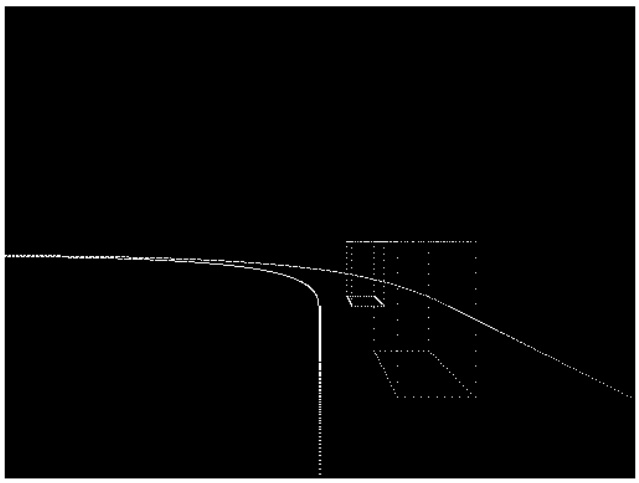
$$s_i = \frac{n \bullet v}{n \bullet u} \tag{2.13}$$

$$0 \leq s_i \leq 1 \tag{2.14}$$

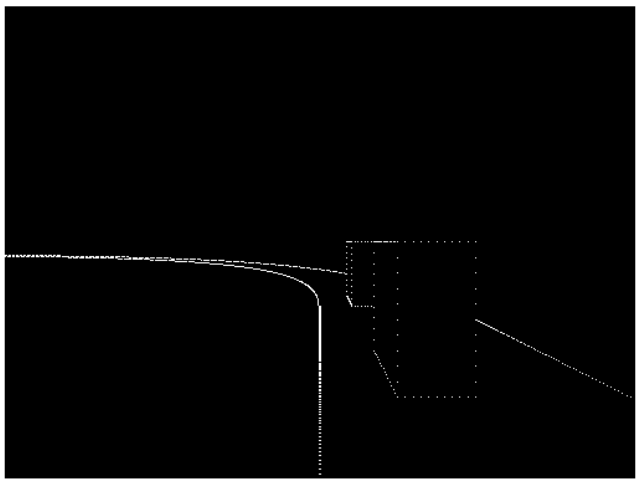
$$p(s) = P + u \cdot s \tag{2.15}$$

If an intersection occurs, the point of intersection P_i is found with equation (2.15) which is the linear equation of the light ray. Once the point of intersection is found it needs to be concluded if the intersection occurs inside or outside the rectangle which defines the side of investigation. This is done with the projection analysis presented in section 2.1.2 (see equation (2.12)).

If an intersection is detected the algorithm should stop the investigation and discard the point, otherwise it should keep screening through the sides of all the objects that lies in the area of interest.



(a) Simulated scenario with disabled ray intersection algorithm.



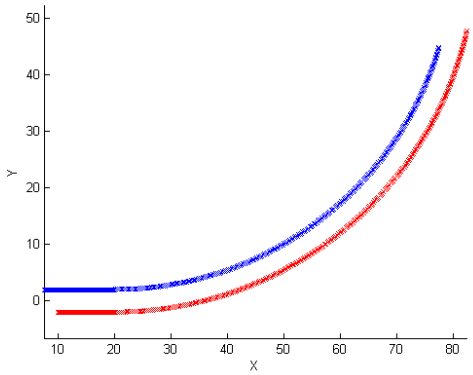
(b) Simulated scenario with enabled ray intersection algorithm.

Figure 2.13: Demonstration of the effect of the ray intersection algorithm. Both pictures shows an identical scenario where two cars are driving in front of the host car and blocking its view. In picture (a) the algorithm is disabled while it is activated in picture (b).

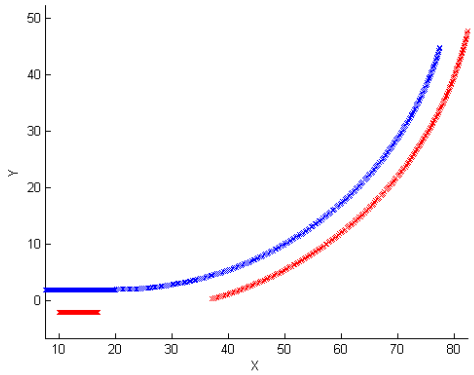
Figure 2.13 shows a simulated scenario where two cars are blocking the view of the host car where the camera is mounted on. Figure 2.13a shows the camera image generated with the ray intersection algorithm was turned off. As a result of the disabling of the algorithm, some points behind the cars are projected on the image even though they

should be blocked. In figure 2.13b on the other hand the algorithm is activated which prevented the blocked points from being projected on to the camera image.

Figure 2.14 demonstrates complications that can transpire from having a *LDS* with absence of a ray intersection algorithm. In figure 2.14a the system produces a perfect lane detection result even though some of the lane marking points on the right lane (red) should not be visible for the system. On the contrary figure 2.14b shows a much more realistic result due to the presence of the ray intersection algorithm and is what would be expected in a real life lane detection situation.



(a) Top view perspective of the detected lane points from the scenario in figure 2.13a where the ray intersection algorithm was dissabled.



(b) Top view perspective of the detected lane points from the scenario in figure 2.13b where the ray intersection algorithm was activated.

Figure 2.14: The resulting lane detection from the scenario in figure 2.13. Figure (a) has a perfect detection even though the two cars are blocking the view of the right lane (red). Figure (b) shows a more realistic result where parts of the right lane could not be detected due to the blocking objects in front of the host car which would be the case in a real life situation.

2.1.4 3D-projection

If a point in the in the 3D environment passes both the field of view test and the ray intersection algorithm it is considered to be a valid point and should thereby be projected onto the image plane. The projection is executed with the linear projection operation displayed in equations (2.16)-(2.17) [12]. Variables u and v denotes the continuous coordinates in the image plane and are plotted in figure 2.15. The image coordinates are bounded by the region R which is described by equation (2.18).

$$u = \frac{x_c \cdot f}{z_c} \tag{2.16}$$

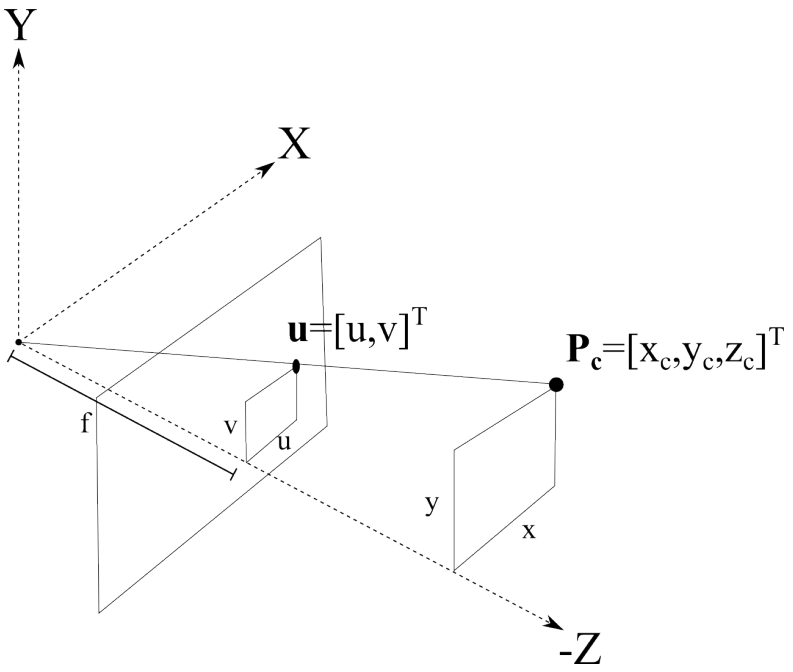


Figure 2.15: A point in the Euclidean space of the camera frame P_c with coordinates $[x_c,y_c,z_c]$ will be projected on the image position **u** with coordinates $[u,v]$ in the intrinsically 2D image plane which is located on a distance f (focus length) from the origin.

$$v = \frac{y_c \cdot f}{z_c} \tag{2.17}$$

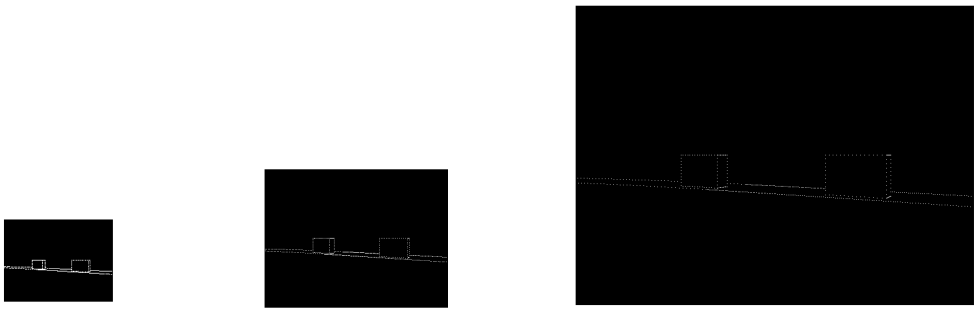
$$R = \Big[(u,v), -f_x \leq u \leq f_x, -f_y \leq v \leq f_y\Big] \tag{2.18}$$

To represent the continuous camera image digitally the image needs to be converted into a corresponding discrete data structure. This is done with a square grid pixel system that is represented by an image matrix I with equivalent size as the resolution values $[M \times N]$. The position in the image matrix is decided by mapping the coordinate to the closest pixel cell with equations (2.19)-(2.20). This will generate a natural discretization error which will be a part of the inaccuracies of the real LDS and consequently this is a feature which is desirable to capture. A decreased resolution implies that the distance the continuous point **u** possibly needs to move in order to fit the discrete structure increases and through that, so does the error. By introducing the pixel error the virtual model gains the advantage of simulating how different hardware setups effects the final result which can give guidance in the decision of the visual component for the real system. This can help the user answer questions like: Is it beneficial to invest in a more expensive camera with higher resolution? Or does a cheaper camera obtain similar results? By having this function the hardware could be tuned to the setup that is most efficient economically but still obtains desired results.

$$n = \left\| \frac{u + f_x}{2f_x} \cdot N \right\| \tag{2.19}$$

$$m = \left\| \left[1 - \frac{v + f_y}{2f_y} \right] \cdot M \right\| \tag{2.20}$$

Figure 2.16 shows how three different resolution settings effects the final image. Other hardware parameters that can be simulated with this model are: Focus length (f), the orientation of the camera on the car or the view angle (α).



(a) 160x120 **(b)** 320x240 **(c)** 640x480

Figure 2.16: A simulated scenario captured with three different resolution values in 4:3 scale.

2.1.5 Inverse 3D-projection

Once the 3D-scenario has been composed down to a 2D-image representation the input for the *LDS* is fully constructed. A real *LDS* analysis the images fed from the camera and then tries to distinguish the lane pixels from the rest.

Once the lane pixels are discovered the *LDS* tries to map the 2D-image positions back into 3D Euclidean space. This is a very complicated task as information has been lost during the transition from the high to the low dimension state. As this transition is an information consuming process, in order to go the inverse direction information needs to be fed. The additional component that is needed is data about how far away from the camera the point is situated. In other words the z-component which is lost during the 3D-projection. Without any reference information this is an impossible task, using only one frame. The most conventional approach for this problem is to analyse multiple frames and compare the movement between the frames with the ego motion of the car. The work presented in this report will not be focused on the process of mapping the 2D-positions back to 3D-space. Instead we will assume that the mapping process is

executed perfectly and then shift the points in 3D-space to match the error performance of the real LDS.

$$u = \frac{2n \cdot f_x}{N} - f_x \tag{2.21}$$

$$v = 2f_y \cdot \Big[1 - \frac{m}{M}\Big] - f_y \tag{2.22}$$

$$P_c = \Big[-\frac{u \cdot z}{f}, -\frac{v \cdot z}{f}, z \Big] \tag{2.23}$$

Once the z-component has been estimated the pixels can be inverse projected with equations (2.21)-(2.23). Equation (2.23) gives the coordinates of the point relative to the camera coordinate system. The point can then easily be converted to the global frame by using equation (2.4). By transforming the points from their original position in 3D-space, to the camera image and then projected inversely back into the 3D-environment, errors due to camera settings are included. The camera model also gives a way of evaluating what parts of the lane markers that are visible by the image sensor which gives a natural estimation of the lane distance. Next chapter will present a way of evaluating the system performance followed by a chapter that shows how this can be implemented and emulated in a virtual system.

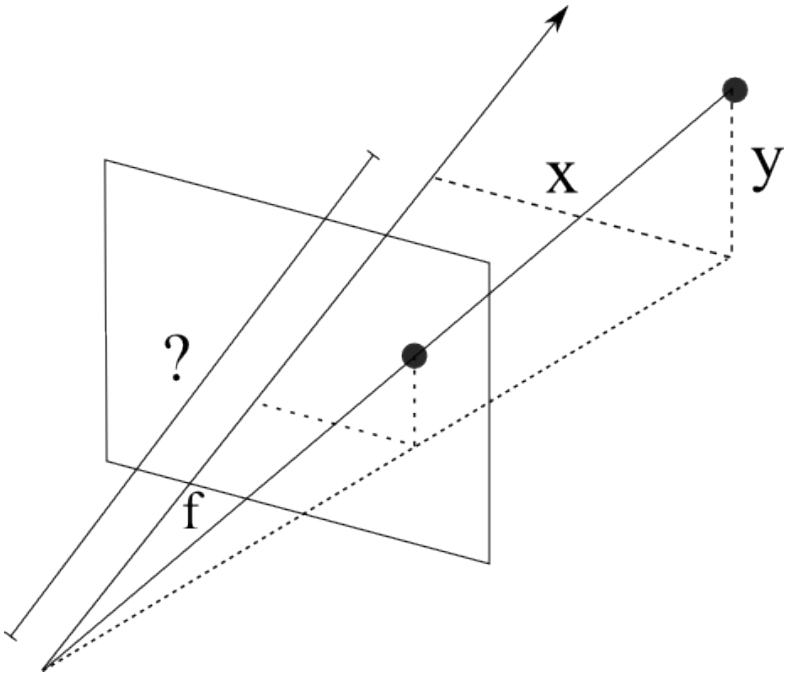


Figure 2.17: An image pixel can easily be projected back to 3D-space once the unknown z-component has been estimated which was lost during the degradation of dimensions.

2.2 System Performance

The camera model, developed in the previous section is used to locate the lane pixels, map them back to 3D-space and by that process include natural errors due to camera settings. This alone gives a model which is close to ideal and thereby fails to capture the real system behaviour. To emulate the *LDS* virtually, knowledge about the system performance needs to be gained. Questions like: How does the system perform in specific situations? , When are the system error high/low?, What state variables effects the system performance? are queries that needs to be investigated in advance to recognize the areas of focus for the modelling process. To get an insight in how the black box *LDS* works, a specialized software tool called *Expedition Log Examiner(ELE)* has been developed, which helps the user to investigate the functionality of the *LDS*.

2.2.1 Expedition Log Examiner

The task of the *expedition log examiner* is simply to process and analyse the data from the *LDS* that has been logged during testing expeditions. With the help of Volvo Cars corporation, over 400 TB of expedition data, corresponding to approximately half a year of effective driving has been provided for the investigation of the *LDS* behaviour. The *ELE* is a matlab based GUI that mainly is intended as a visualisation tool to aid the user in the investigation of the *LDS* performance at any expedition instance.

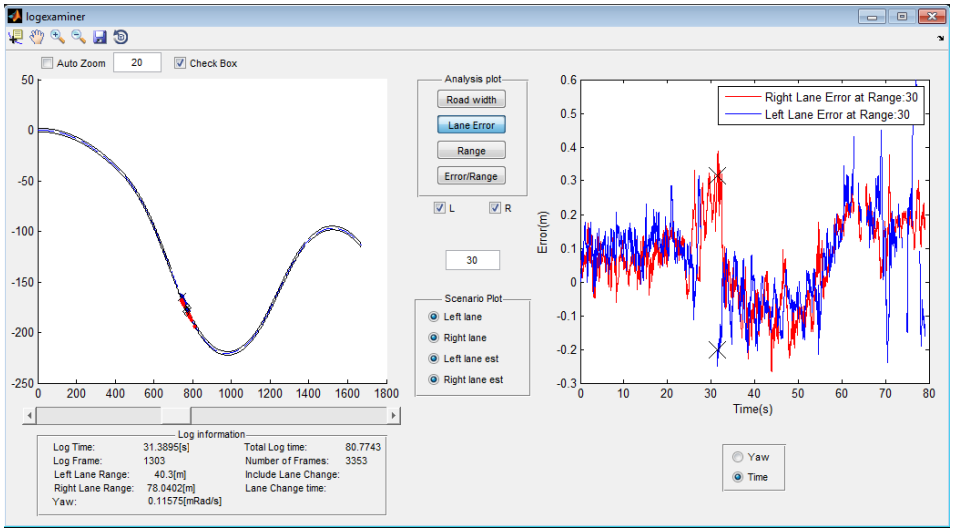
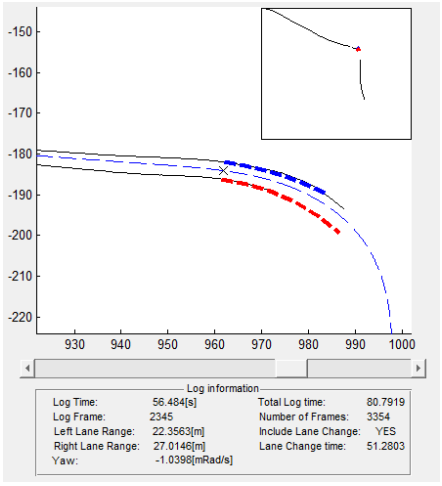


Figure 2.18: The main layout of the expedition log examiner. The program consists of two plot windows. The left window shows the trajectory of the host car for the selected expedition log which guides and helps the user to browse around in the loaded scenario. The right window is the analysis window which includes multiple functions that evaluates and visualizes the performance of the system in different ways.

Figure 2.18 shows the main layout of the *ELE* software. The program consists of two plotting objects. The left window displays the trajectory of the host car for the selected

expedition log and is intended as a tool to guide and help the user to browse around in the loaded scenario. The second window is the analysis window which includes multiple functions to evaluate and visualize the performance of the system.

(a) A captured camera image of a real expedition scenario.



(b) Identical scenario loaded into the expedition log examiner.

Figure 2.19: The main function of the *ELE* describes graphically. Figure (a) displays a camera image captured from a expedition scenario. The data collected during the scenario was fed to the expedition log examiner and the result is shown in figure (b). The dashed blue line represents the trajectory of the host car, the continuous black line is the position of the lane markers while the blue and the red lines are the LDS estimation of the left and right lane markers.

Figure 2.19a shows a camera image captured from a hardware test expedition. The data collected during the expedition was then fed to the expedition log examiner and the results are displayed in figure 2.19b. The blue dashed line is the trajectory of the host car which is calculated through equation (2.24) and (2.25) where ψ denotes the yaw angle while V is the vehicle velocity. All entities are stated in the discrete domain. The vehicle orientation is given by a gyroscopic sensor which outputs the yaw rate $\dot{\psi}[\frac{rad}{s}]$. The yaw rate is then converted to the global orientation variable ψ through the discrete integration shown in equation (2.26).

$$X(k+1) = V(k) \cdot \Delta t \cdot \cos\big(\psi(k)\big) + X(k) \tag{2.24}$$

$$Y(k+1) = V(k) \cdot \Delta t \cdot \sin\big(\psi(k)\big) + Y(k) \tag{2.25}$$

$$\psi(k) = \sum_{i=1}^{k} \psi(i) \cdot \big(t(i+1) - t(i)\big) \tag{2.26}$$

Due to the fact that all of the entities are measured through different sensors with individual capturing frequencies a sensor fusion process has been conducted to synchronize all the data. The sensor fusion is executed with and linear interpolation which is a valid assumption due to the relatively high frequencies of the full set of sensors. This implies a low deviation from a linear behaviour between two data points and thereby small errors received from the linear interpolation. The interpolation is executed through equation (2.27) and displayed in figure 2.20 where the top line represents the velocity data timeline while the bottom line is the yaw-direction timeline. As can be seen both data sets are captured with different frequencies which is why the interpolation process in equation (2.27) is needed.

$$\psi(i) = \psi_{k-1} + \left(\frac{t_{v_i} - t_{\psi_{k-1}}}{t_{\psi_k} - t_{\psi_{k-1}}}\right) \cdot \left(\psi_k - \psi_{k-1}\right) \tag{2.27}$$

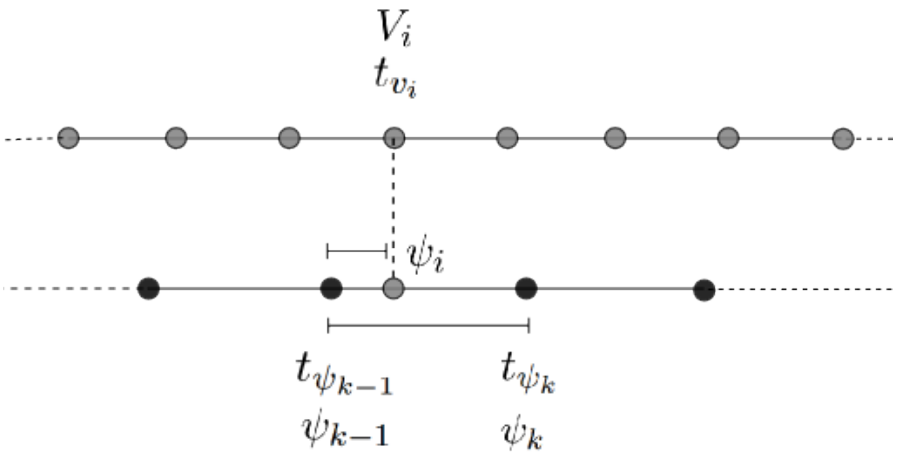


Figure 2.20: Interpolation schematics for the applied sensor fusion where the top line represents the velocity data timeline while the bottom line is the yaw-direction timeline.

The bold red and blue lines in figure 2.19 are the lane marker representations which is estimated by the *LDS* at the selected time stamp. As previously mentioned the *LDS* estimation is given in the form of a 3rd order polynomial as in equation 1.2 and a distance value, d_l for the left lane and d_r for the right. The distance values specifies for what longitudinal range the polynomials are valid. If demanded, the *ELE* will plot out the polynomials which aids the user to investigate the *LDS* behaviour. The information panel below the scenario figure gives the user information about the properties that are listed below:

- Current time stamp t
- Current frame number n
- Left lane distance d_l
- Right lane distance d_r
- Yaw rate ψ
- Total log time T
- Total number of frames N
- Occurrence of lane change
- Lane change time stamp $T_{laneChange}$

To gain perception on how accurate the LDS polynomials are, the ELE software calculates reference lines (black lines in figure 2.19) and plots them at the positions where it estimates that the lane markers are situated. This is done with a Ground Truth Analysis method (GTA). The prediction of the reference lines are based on the assumption that the LDS polynomials are most accurate close to the origin. There are many concepts that supports this assumption, for instance the fact that the pixel discretization error is smallest close to the camera and increases linearly further away from it. It is also common to fit the lane marker polynomials in a manner that minimizes the error to the detected points close to the host vehicle as this information is more important than road information further ahead.

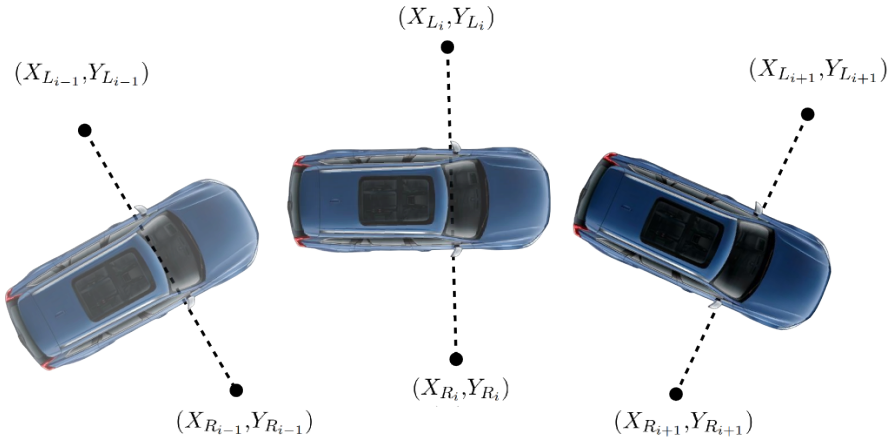


Figure 2.21: The figure is a graphical description of how the GTA-algorithm is utilized to define the reference lines of the lane markers. For every time instance where a unique polynomial prediction is present the left and right lanes can be estimated by shifting the lateral offset parameter a_0 in equation 1.2 perpendicular to the car.

The GTA-algorithm calculates the reference lines by shifting the lateral offset parameters a_0 in equation 1.2 perpendicular to the car for every time instance where a unique polynomial estimation is present. By pre-processing this information for all available camera frames the left and right reference lines can be determined, which is a fairly accurate representation of the ground truth. To achieve higher accuracy a LIDAR system can be used which can estimate a much more detailed position of the ground truth reference line. The coordinates of the lane markers are established through equations (2.28)-(2.29) where $[X_i,\ Y_i]$ is the position of the host car at instance i while a_0 is the

lateral offset represented by the zero-coefficient in the parameter vector. $[X_L,Y_L]$ denotes the position of the left lane but the same equations can be used to calculate the position of the right lane $[X_R,Y_R]$.

$$X_{L_i} = X_i + a_{L_0} \cdot \sin(\psi_i) \tag{2.28}$$

$$Y_{L_i} = Y_i - a_{L_0} \cdot \cos(\psi_i) \tag{2.29}$$

The right window in figure 2.18 is the analysis window which displays the system performance in different aspects. Between the orientation window and the analysis window is the function selection bar where the user can select which aspect *ELE* should evaluate the system performance on. The *ELE* includes four main functions that investigates four key aspects of the system, but the software can be extended to include more tests according to the preference of the user.

2.2.1.1 Road width function

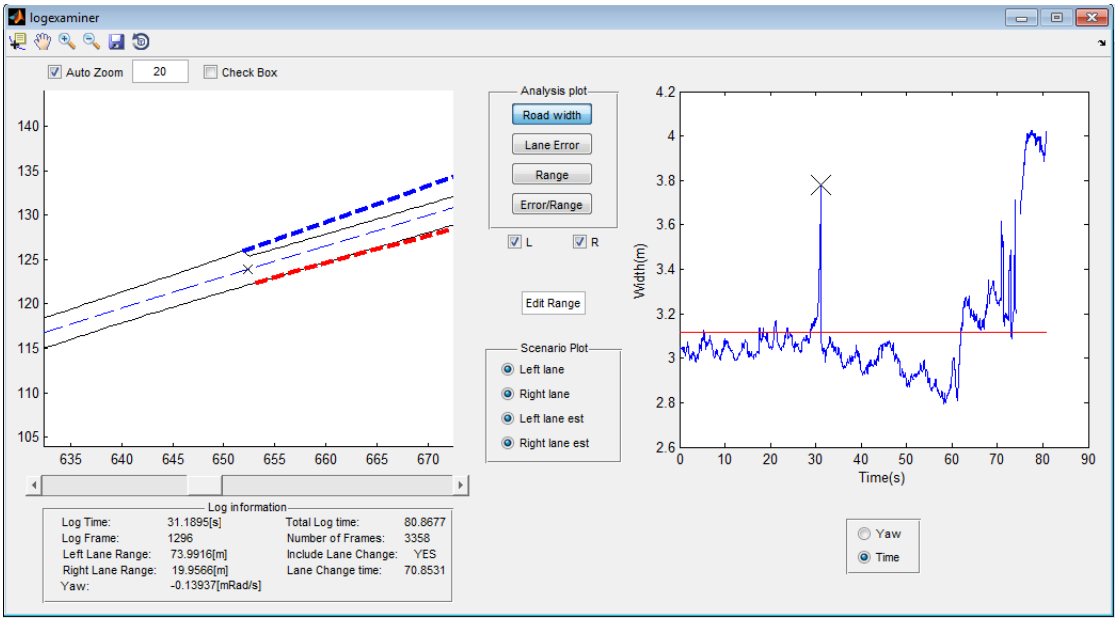


Figure 2.22: Description of the road width function in the *ELE* software. At the selected time stamp the road width is over estimated due to an incorrect prediction of the *LDS*.

The first analysis function is the road width function. As the road width should be more or less constant this functions helps the user to detect anomalies and incorrect estimations. Figure 2.22 shows the road width function in action. At the selected time stamp the road width value is abnormally high because of the sudden lane flip which can be seen in the orientation window to the left. The simulated scenario is taken from

the real life expedition displayed in figure 2.23. In the selected instance, the road is suddenly converted from a single lane road to a double. The LDS then switches its estimation from the continuous left lane to the dashed middle lane which is the cause of this strange road width behaviour. By using the road width function the user can detect similar types of anomalies and then decide whether this type of behaviour should be included in the final virtual LDS model or if it should be discarded.

Figure 2.23: A scenario where a single lane road is converted into two lanes which causes the LDS to change its estimation.

2.2.1.2 Distance function

The second analysis function is the distance function which helps the user to visualize the propagation of the distance variables d_l and d_r (see figure 2.24). As the distance variables are output variables from the LDS it is also of great importance to investigate how they change and what factors they are dependent on. For all the functions there is a switch included which can be toggled to *time* or *yaw* that tells the ELE to sort the data either chronologically by time or by yaw rate. This tool is very helpful, especially for the distance function as it can be used to investigate if high curvature roads affects the distance variables or not. As can be seen in figure 2.24 the range estimation is exposed to extreme fluctuations.

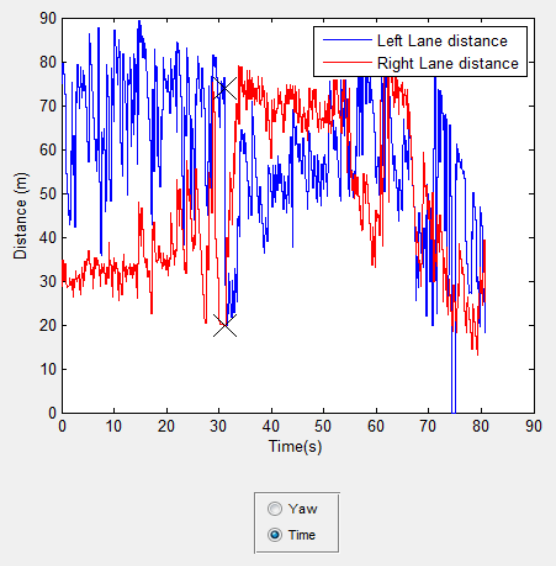


Figure 2.24: The range estimation function which shows the estimated values of the distance variables d_l and d_r.

2.2.1.3 Error analysis function

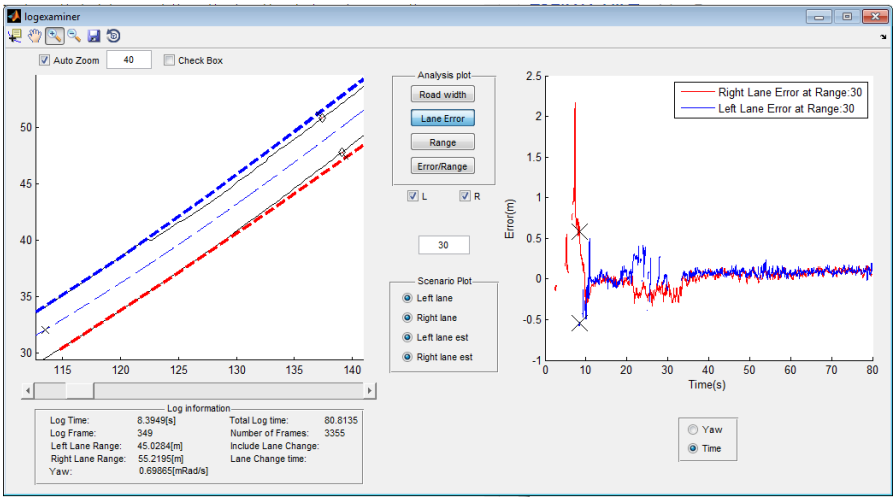


Figure 2.25: The expedition log examiner includes an error function which computes the lane errors at a certain range, specified by the user. The *ELE* also marks the polynomial points and the corresponding points on the reference lines in the orientation window.

The *ELE* also offers the possibility to investigate the errors at different ranges which is done with the error function. The user simply selects the range value where the *ELE* should compute the lane errors and clicks on the range function. In addition to

plotting the resulting errors in the analysis window for all time stamps the *ELE* also marks the estimated point on the polynomial and the corresponding reference point in the orientation window (see figure 2.25).

The errors are calculated according to the error definition defined in the *Error Definition* section below. The error analysis function is a powerful tool and gives great insight of how the system behaves in different situations.

2.2.1.4 Error/Range function

To get an insight of how the distance from the polynomial origin affects the error the *Error/Range* function has been included in the *ELE* software. The *Error/Range* function calculates the error for several predefined range values and displays the current error plot for the selected time stamp and the lanes that the user selects (red=left lane, blue=right lane).

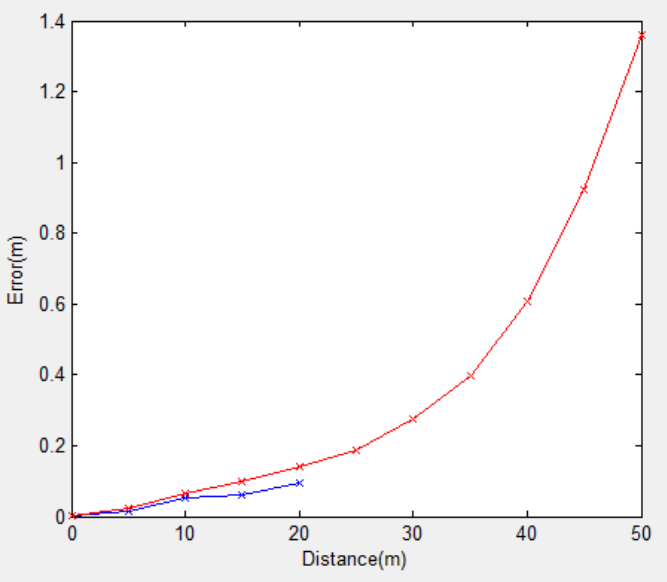


Figure 2.26: The Error/Range function provided in the *ELE* software gives the user an insight on how the error function is affected by the range. The plot shows how the error on the y-axis increases with increased distance for the left lane (red curve) and the right (blue curve).

2.2.2 Error definition

When talking about system performance we more specifically consider analysis of the errors introduced by the *LDS*. For this reason it is of great importance to establish a system error definition. Lane marker system errors can be defined in multiple ways which all have different down/up-sides. As the output of the system is a higher order

polynomial, a natural error definition would be to regard the deviation of the polynomial parameters in vector $\bar{A}$ from the coefficients of the ideal polynomial located in vector $\bar{A}^*$ like in equation (2.30). Even if this method is focused directly on the system output it is not so intuitive and very hard to apply. Deviations in one parameters can cause considerably high errors even if the rest of the parameters are perfectly tuned.

$$\underbrace{\begin{pmatrix} e_0 \\ \vdots \\ e_n \end{pmatrix}}_{\bar{E}} = \underbrace{\begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix}}_{\bar{A}} - \underbrace{\begin{pmatrix} a_0^* \\ \vdots \\ a_n^* \end{pmatrix}}_{\bar{A}^*} \tag{2.30}$$

A superior approach for the error definition is to consider the distance e between the estimated polynomial and the reference line at different ranges x like in figure 2.27. This method is much more intuitive as it is considerably more understandable to visualize, plus the fact that this distance in the end is the critical entity for the active safety functions which is what the virtual LDS should capture. By using this error definition the points detected from the camera model can be shifted from their near perfect positions to match the error of the LDS. When the points finally has been shifted, a polynomial can be constructed using a poly fit algorithm on the manipulated lane points.

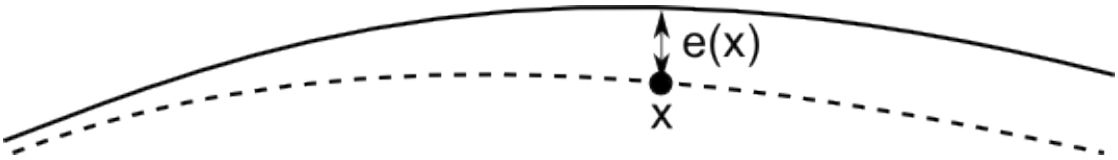


Figure 2.27: Description of the applied error definition which is utilized in the LDS performance analysis. The error e is defined as the distance between the estimated polynomial and the reference line at range x.

Given a polynomial with a coefficient vector A_L defining the left lane or A_R defining the right lane an estimated lane marker point p_{1_L} or p_{1_R} at range x can be found through equations (2.31)-(2.36) where X_v and Y_v are the current positions of the host vehicle. The entity $f(A,x)$ denotes the polynomial function in (1.2) with the coefficient vector A and the range x as input variables. The lateral offset at the current time stamp is represented by the 0-coefficient in each coefficient vectors which is represented by a_{L_0} for the lateral offset to the left lane and a_{R_0} for the lateral offset to the right lane.

$$x_l = X_v - |a_{L_0}| \cdot \sin(\psi) \ , \ y_l = Y_v + |a_{L_0}| \cdot \cos(\psi) \tag{2.31}$$

$$x_r = X_v + |a_{R_0}| \cdot \sin(\psi) \ , \ y_r = Y_v - |a_{R_0}| \cdot \cos(\psi) \tag{2.32}$$

$$p_{1_{Lx}} = x_l + x \cdot \cos(\psi) + \big(f(A_L,x) + |a_{L_0}|\big) \cdot \sin(\psi) \tag{2.33}$$

$$p_{1_{Ly}} = y_l + x \cdot \sin(\psi) - \big(f(A_L,x) + |a_{L_0}|\big) \cdot \cos(\psi) \tag{2.34}$$

$$p_{1_{Rx}} = x_r + x \cdot \cos(\psi) + \big(f(A_R,x) + |a_{R_0}|\big) \cdot \sin(\psi) \tag{2.35}$$

$$p_{1_{Ry}} = y_r + x \cdot \sin(\psi) - \big(f(A_R,x) + |a_{R_0}|\big) \cdot \cos(\psi) \tag{2.36}$$

By using equations (2.31)-(2.36) the LDS estimation of the global position for the left and right lane, denoted $p_{1_L} = [p_{1_{Lx}},p_{1_{Ly}}]$ and $p_{1_R} = [p_{1_{Rx}},p_{1_{Ry}}]$ can be determined for any range x.

When the coordinates of the lane estimation point is determined the objective is to find the corresponding point on the reference line in order to find the lane error. As the reference lines are represented in the discrete domain the method of finding the lane error is not trivial. Just taking the distance between the estimated point p_1 and the closest point in the reference line would not generate a good error prediction as this definition would be dependent on how sparsely the reference points are sampled which is directly dependent on the speed of the host car and the frame rate of the camera. This definition would generate high errors for high speeds due to the increment in the sampling distance between the reference points which in turn would create a system behaviour that is not connected to the real LDS.

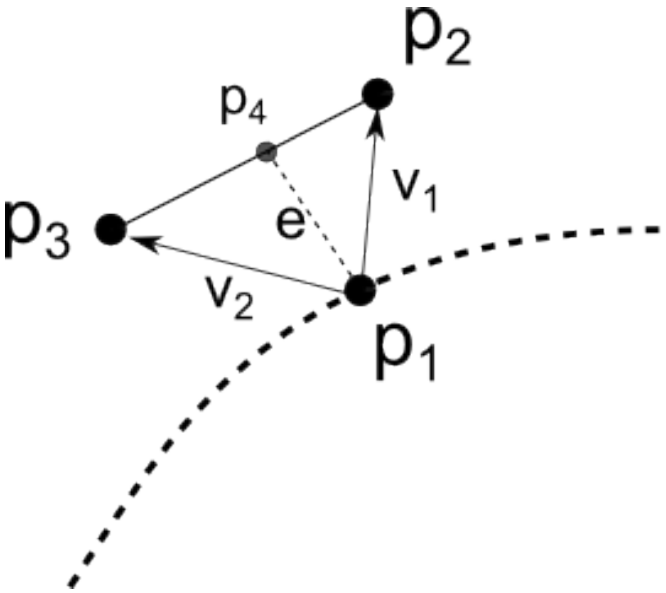


Figure 2.28: A graphical definition of lane error algorithm. The point p_1 denotes the point of investigation on the LDS polynomial while p_2 and p_3 are the closest and second closest points in the reference line array.

Instead of just taking the closest point in the reference point array the lane error algorithm finds points between two reference points by assuming a linear behaviour between the two sampling instances. This assumption is valid due to the fact that deviation from a linear behaviour is only present in situations with high curvature. In these cases the speed of the host car are almost exclusively reduced which implies that the reference points are sampled with higher density which in turn counters this effect. To find a corresponding reference point p_4 (see figure 2.28) to an estimated lane marker point p_1 the closest point in the reference array p_2 is determined. The point in the reference array that lies closest to p_1 is the one that satisfies equation (2.37).

$$\sqrt{(X_{r_i} - P_{1_x})^2 + (Y_{r_i} - P_{1_y})^2} = min\Big(\sqrt{(X_r - P_{1_x})^2 + (Y_r - P_{1_y})^2}\Big) \tag{2.37}$$

The second closest point to p_1 is denoted p_3 and can likewise be found with equation (2.37). These three points together forms the triangle shown in figure 2.29 with side lengths d1, d2 and d3.

$$v_1 = p_2 - p_1 \tag{2.38}$$

$$v_2 = p_3 - p_1 \tag{2.39}$$

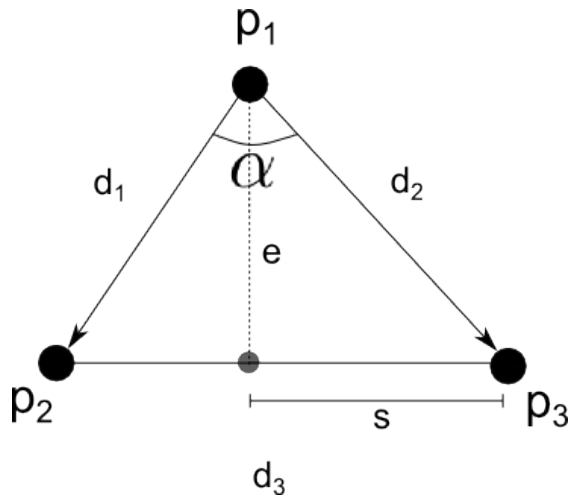


Figure 2.29: The estimated lane point p_1 together with the reference points p_2 and p_3 forms the triangle shown in the figure where the height of the triangle e is the lane error.

The height of the triangle is the lane error e and can be determined through equation (2.40)-(2.41).

$$\alpha = atan2(\|v_1 \; x \; v_2\| \; , \; v_1 \bullet v_2) \tag{2.40}$$

$$e = \frac{d_1 \cdot d_2}{d_3} \cdot \sin(\alpha) \tag{2.41}$$

To determine the point on the reference line which lies between the points p_2 and p_3 equations (2.42)-(2.45) can be used.

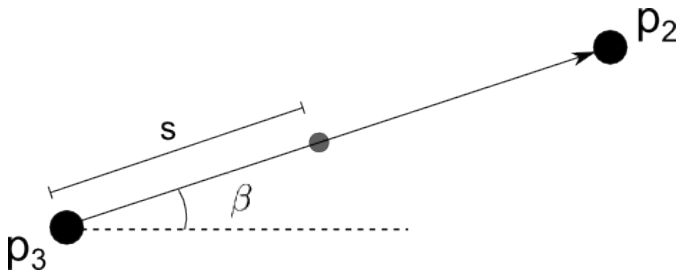


Figure 2.30: A graphical description of how to mathematically determine point p_4 that lies between the two reference points p_2 and p_3.

$$s = \sqrt{d_2^2 - e^2} \tag{2.42}$$

$$\beta = \frac{y_2 - y_3}{x_2 - x_3} \tag{2.43}$$

$$x_4 = x_3 + s \cdot \cos(\beta) \tag{2.44}$$

$$y_4 = y_3 + s \cdot \sin(\beta) \tag{2.45}$$

As this approach only calculates the absolute distance between the predicted point and the ground truth insufficient information is gained to be able to fully shift the lane marker points. This method alone does not provide information about whether the estimated point is located to the left of the reference line or if it is located to the right. To provide the model with this information an algorithm has been developed that determines the location of the error. To compute this the algorithm uses three vectors: v_{est} stretching from the position of the host car to the estimated lane point, v_{lane} stretching from the car to the reference point and v_{dir} which denotes the vehicle direction vector. The vectors are defined in equations (2.46)-(2.48) and displayed in figure 2.31.

$$v_{est} = p_1 - P_v \tag{2.46}$$

$$v_{lane} = p_4 - P_v \tag{2.47}$$

$$v_{dir} = \begin{pmatrix} X_v + \cos(\psi) \\ Y_v + \sin(\psi) \end{pmatrix} - P_v \tag{2.48}$$

When the three vectors v_{est},v_{lane} and v_{dir} are computed the angles of v_{est} and v_{lane} relative to v_{dir} can be determined through equation (2.49)-(2.50). The lane error e^*

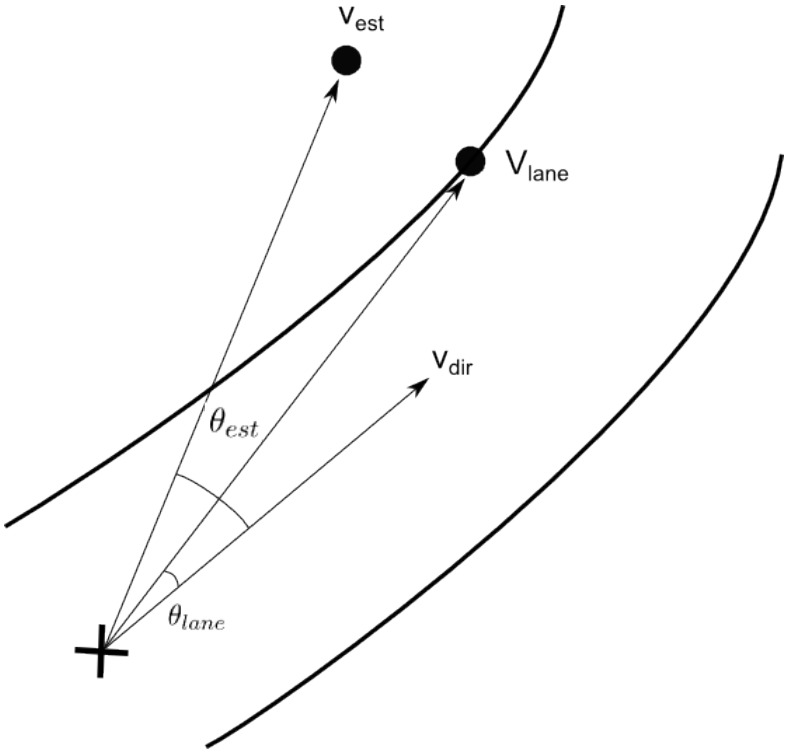


Figure 2.31: Graphical definition of the algorithm that decides on which side of the reference line the estimated lane point lies. If θ_{est} is larger than θ_{lane} the point lies to the left of the lane, otherwise it lies to the right.

is located to the left of its reference line if $\theta_{est} \geq \theta_{lane}$ according to equation (2.51), otherwise it is located to the right. Errors located to the left is denoted with a negative sign while errors positioned to the right assumes positive values.

$$\theta_{lane} = cos^{-1}\Big(\frac{v_{lane} \bullet v_{dir}}{\|v_{lane}\| \cdot \|v_{dir}\|}\Big) \tag{2.49}$$

$$\theta_{est} = cos^{-1}\Big(\frac{v_{est} \bullet v_{dir}}{\|v_{est}\| \cdot \|v_{dir}\|}\Big) \tag{2.50}$$

$$e^*(e,\theta_{est},\theta_{lane}) = \begin{cases} -|e|, & \text{if } \theta_{est} \geq \theta_{lane} \\ |e|, & \text{otherwise} \end{cases} \tag{2.51}$$

2.3 Data Acquisition

To construct the virtual *LDS*, data needs to be gathered to serve as input for the system model. As the expedition log examiner only serves as a visualisation tool to aid the user to analyse the behaviour and the performance of the *LDS* the development of a second software is required.

The additional software, called the Expedition Log Scanner (*ELS*) was constructed to scan through the expedition logs, extract and then form the data which would serve as the foundation for the virtual *LDS* model. The user simply designates which dimensions that is desired to be included in the *LDS* model and selects the expedition logs that is to be examined and then executes the program with the *scan button* (see figure 2.32). The *ELS* then scans through the selected logs and calculates the system errors according to the error definition in section 2.2.2. The error is computed for all selected dimensions and all time instances. The results from every expedition can be displayed in the plot window by either enabling the *Left Lane*, *Right Lane* or the *Trajectory* button below the window. If the *save images* button is enabled the *ELS* will save both the *Left Lane* and the *Right Lane* picture as well as the *Trajectory* image. The images are helpful as they displays the results of the extraction which aids the user to pick out logs with unreliable data. However, the save image process will slow down the scanning speed immensely.

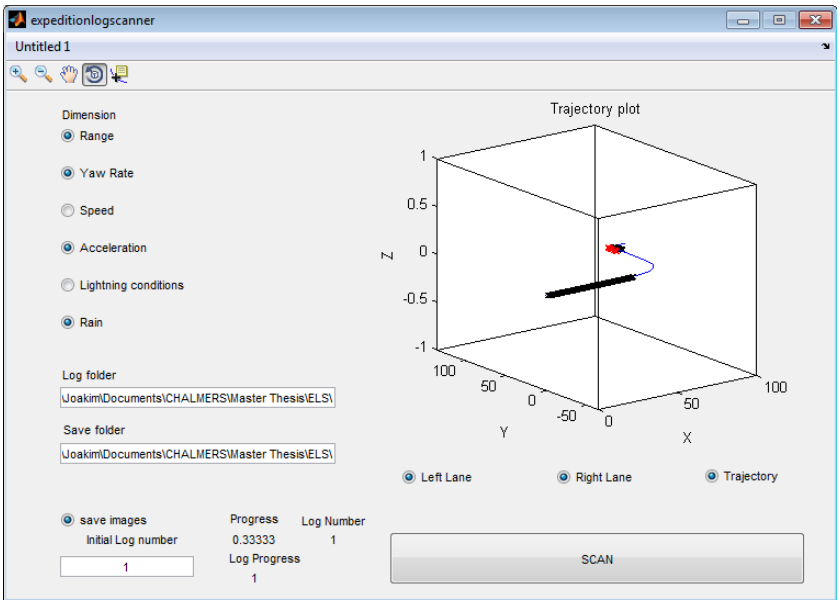


Figure 2.32: Grapghical layout of the Expedition Log Scanner (*ELS*). The user selects the dimensions that is to be extracted and specifies the location of the folder that contains the expedition logs a swell as the folder where *ELS* should save the data. If desired, the *ELS* also displays the results of the extraction and saves the images to the save folder. In this figure the *ELS* displays the trajectory of the host car with detected positions of the left and right lane which aids the user to analyse the validity of the specific log.

The ELS software includes in total 6 different dimensions. If the range dimension is selected, ELS applies a specific function that evaluates the system error for every range value r, specified in the range vector $\bar{R}$ and for every time stamp. By default the *evaluate up to range* function is activated. This function only allows ELS to compute the lane errors up to the polynomial ranges d_L and d_r which are estimated by the LDS. The range vector is by default set to [1:2:100], but can be specified according to the preferences of the user.

Other dimensions that is important to analyse are the lightning and weather conditions. These are the dimensions that can have a direct affect on the image quality which in turn can effect the quality of the lane estimation. The lightning conditions are represented with a Boolean variable that states if the expedition was carried out during daytime or at night. The weather condition variables checks the occurrence of rain. To determine this condition the ELS checks if the wipers on the front window are activated. If true, the wipers are most likely active due to rain and the ELS then assumes this weather condition.

Besides the range dimension and the camera conditions the ELS also offers the opportunity to extract vehicle states. The available states are yaw rate $[\frac{rad}{s}]$, speed $[\frac{m}{s}]$ and acceleration $[\frac{m}{s^2}]$. The dimensions that are included in the Expedition Log Scanner are stated in the list below.

- Range $[m]$
- Yaw Rate $[\frac{rad}{s}]$
- Speed $[\frac{m}{s}]$
- Acceleration $[\frac{m}{s^2}]$
- Lightning condition
- Weather condition

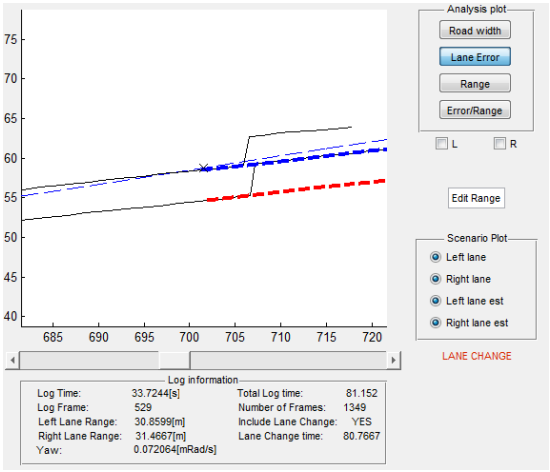
The output of the ELS is the data matrix D which is displayed below. The matrix consists of n-number of columns, representing the selected dimensions and m-number of rows which represents the number of data points. The data matrix for every log will be saved separately in the location specified in the save directory field.

$$\begin{matrix} & \overbrace{\quad\quad\quad\quad\quad}^{n \text{ Dimensions}} \\ D = & \begin{bmatrix} d_{11} & d_{12} & d_{13} & \dots & d_{1n} \\ d_{21} & d_{22} & d_{23} & \dots & d_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_{m1} & d_{m2} & d_{m3} & \dots & d_{mn} \end{bmatrix} \end{matrix}$$

The final stage in the data acquisition process is to merge all of the extracted data matrices into one unified matrix. To enable this process a merge script called the *mergeData.m* has been developed. The *mergeData* script takes a folder directory as input and then scans this directory for extracted data matrices. The gathered matrices are then merged into one large matrix which is saved into the same folder directory.

The *ELS* also includes a secondary task which is to filter out data from abnormal an unwanted scenarios which should not be captured in the virtual lane detection model. An abnormal situation which causes anomalies in the error estimation occurs when the host car is executing a lane change. This complication is displayed in figure 2.33. In (a) the host car is executing a lane change where consequently the original left lane will eventually become the right lane while the inner road edge will become the left lane estimation. The scenario was inserted into the expedition log examiner which is shown in (b). This causes abnormally high error values due to the fact that the left lane (blue line) is evaluated relative to the new left lane (the road edge) while the right lane error is computed relative to the old left lane.

(a) Expedition scenario where the host car is executing a lane change from the right lane to the left. (Displayed lane estimation is not authentic)

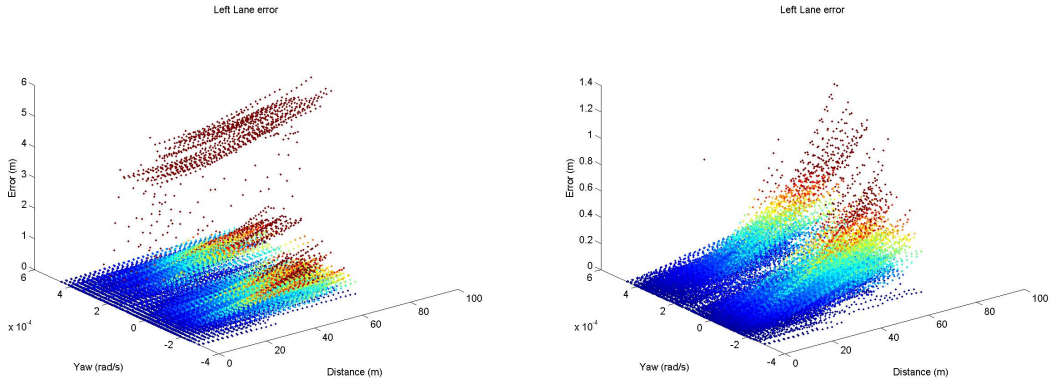


(b) The expedition scenario shown in (a) loaded into the expedition log examiner which shows complications caused by the lane change.

Figure 2.33: Undesirable errors can be caused by extreme maneuvers like lane change. In (a) the host car is executing a lane change. The scenario was then loaded into the expedition log examiner which is shown in figure (b). Consequently the left polynomial error is estimated relative to the lane edge on the second lane while the right lane is estimated relative to the mid marker which previously was the left lane estimation.

The result of the false lane estimation caused by the lane change is plotted in figure 2.34a. The figure shows the 3-dimensional error profile for the scenario displayed in figure 2.33 with range $[m]$ and yaw rate $[\frac{rad}{s}]$ as input dimensions. In absence of a filter that extracts these scenarios the lane estimation errors from the lane change will end

up as a ceiling in the error profile with extremely large values. If this type behaviour is not filtered away from the final input data to the *LDS*-model, there is a risk that these error profiles will affect and impoverish the model. Figure 2.34b show the resulting error profile when the *ELS*-software has executed the lane change filter process. As can be seen the error behaviour is now totally excluded and only the real system behaviour is present.

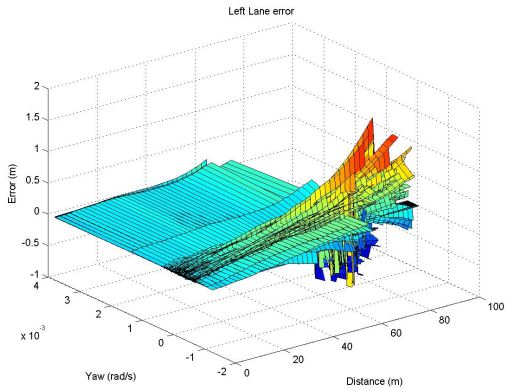


(a) Error profile with included lane change errors.

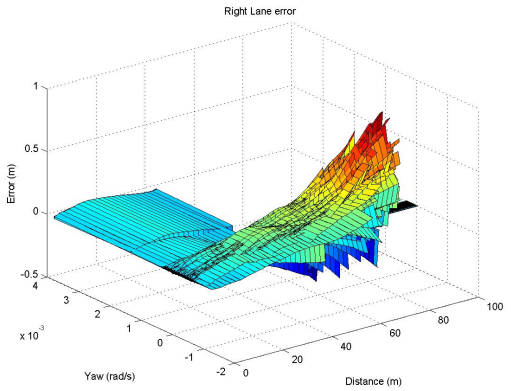
(b) Error profile with excluded lane change errors.

Figure 2.34: The 3D- error profile with respect to range $[m]$ (x-axis) and yaw rate $[\frac{rad}{s}]$ (y-axis) for one specific expedition log. The image displays how the *ELS* software filters out anomalies and abnormal scenarios caused by actions like lane change. In figure (a) no filtering has occurred which thereby causes this strange high error region while in (b), this region has been excluded by the *ELS* filers.

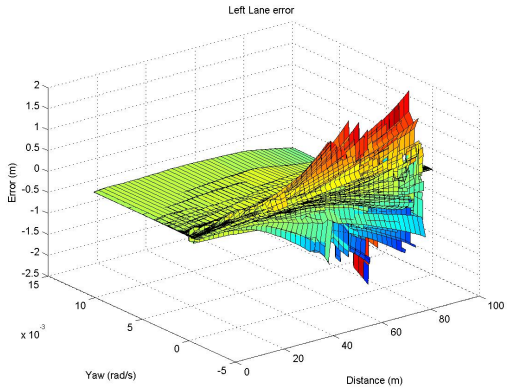
With the *ELS*-software the user can construct and form the error data according to the desired preferences. The error data can then be used as input data for the *LDS* model. It also gives visual aid which helps the user to understand how the system behaves and to pinpoint what dimensions it is affected of. Figure 2.35 on the next page displays the system behaviour for three different scenarios. Figures on the left side represents the left lane estimation while the figures on the right are the right lane estimations. The error profiles are plotted in respect to range $[m]$ (x-axis) and yaw rate $[\frac{rad}{s}]$ (y-axis). As can be seen in the figure a trend of increasing lane error with increasing range values is present, more on this under the Result section.



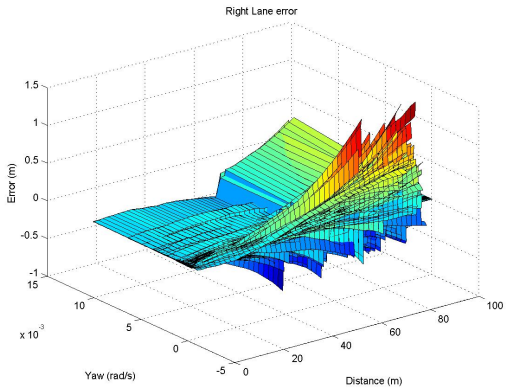
(a) Left lane error profile for scenario 1.



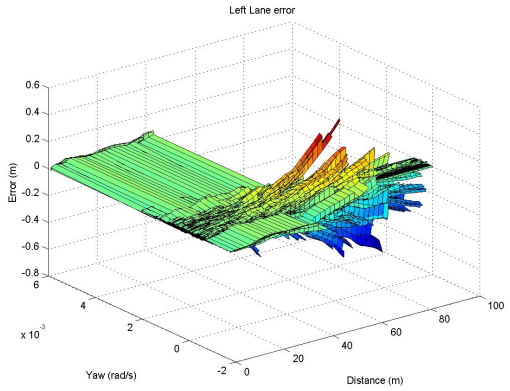
(b) Right lane error profile for scenario 1.



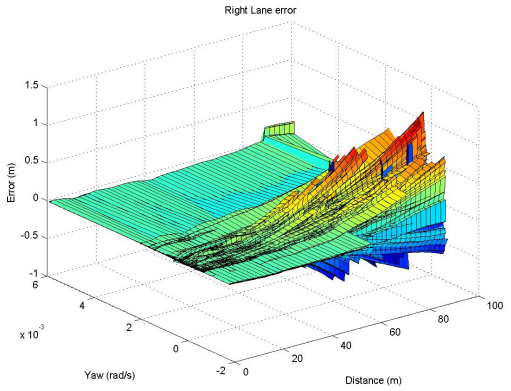
(c) Left lane error profile for scenario 2.



(d) Right lane error profile for scenario 2.



(e) Left lane error profile for scenario 3.



(f) Right lane error profile for scenario 3.

Figure 2.35: The 3-dimensional error profile for 3 different scenarios with range $[m]$ on the x-axis, yaw rate $[\frac{rad}{s}]$ on the y-axis and the error values $[m]$ on the z-axis. As can be seen in the figure there is a trend of increasing error with increasing range.

2.4 Principal Component Analysis

As the relevant dimensions which are influencing on the LDS is unknown, a multivariate analysis is required to determine which variables the virtual model should take as input. The expedition log examiner gives the user an intuition on how different types of variables affects the LDS but does not give conclusive perception. There are multiple types of statistical techniques which can aid in this process but to gain knowledge in this specific matter a *Principal Component Analysis* (PCA) has been conducted.

Principal component analysis is a well-established and developed tool in the field of statistics used for multivariate problems. The method implies a linear orthogonal transformation and was initially introduced by Karl Pearson in 1901 [13]. The PCA takes a set of data observations with any number of possibly correlated dimensions as input and then computes a new set of uncorrelated orthogonal variables called *principal components* [14]. The number of principal components is always less or equal to the number of initial dimensions. For this project the principal component analysis was utilized for dimension reduction which is one of the most common purposes of the method.

Geometrically the PCA tries to enclose the given n-dimensional input data set with an n-dimensional ellipse composed of orthogonal Eigen-vectors. The dimension that contributes with the highest variance will be the primary principal component and the component that explains most of the correlations in the original data set. The primary principal component is followed by the secondary component which stands for the second largest variance and so on. In this way, the principal component analysis lists the input variables according to their relevance in the original data-set.

The initial step in the Principal component Analysis is to subtract the mean $\bar{d}$ from every dimension vector in the data set. This process is followed by the computation of the covariance matrix C [15]. The covariance matrix contains all covariance values calculated for all included dimensions with equation (2.52).

$$ccv(d_i,d_j) = \frac{\sum_{k=1}^{n} \left(d_{i_k} - \bar{d_i}\right)\left(d_{j_k} - \bar{d_j}\right)}{\left(n-1\right)} \tag{2.52}$$

$$C = \begin{bmatrix} ccv(d_1,d_1) & ccv(d_1,d_2) & \dots & ccv(d_1,d_n) \\ ccv(d_2,d_1) & ccv(d_2,d_2) & \dots & ccv(d_2,d_n) \\ \vdots & \vdots & \vdots & \vdots \\ ccv(d_n,d_1) & ccv(d_n,d_2) & \dots & ccv(d_n,d_n) \end{bmatrix}$$

When the covariance matrix C has been calculated the only remaining process is to identify the Eigen-values and -vectors of the matrix. The eigenvalues gives information about the variance of each dimension which in turn declares how much a specific dimension contributes to the variance in the original data set. The Eigen-vectors are the principal components which all together constructs the n-dimensional ellipse that encloses the original data set in $\mathbb{R}^n$. As the covariance matrix is a symmetric matrix and due to the fact that the principal components are Eigen-vectors to the matrix the components are all orthogonal and thereby linearly independent to each other [16].

3

Results

The following chapter will include the discoveries and results, concluded from the research in this project. The results are based on the input data matrix D formed through the process described in section 2.3, *Data Acquisition*. The data was collected from expedition logs spanning over 30 hours. The final data matrix has 7 columns representing the input dimensions *range*, *yaw rate*, *speed*, *acceleration*, *lightning conditions*, *weather conditions* and the resulting *error*. The matrix also consists of over 94 million rows which corresponds to the collected observations.

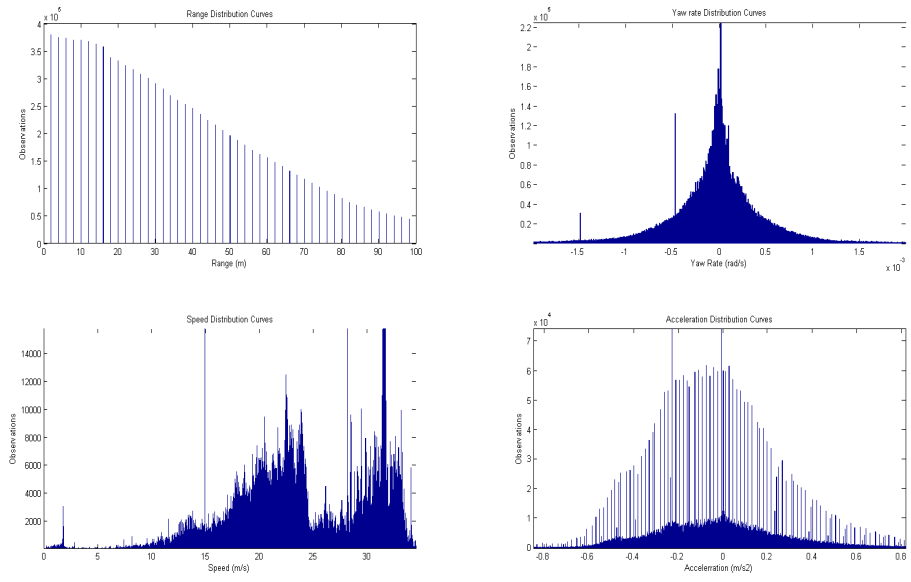


Figure 3.1: The figure shows the distributions curves for the input data.

Figure 3.1 shows the distribution of the input data set. The top left sub figure shows how the range dimension is distributed. As can be seen the range dimension is discretized from 0-100 with an interval of 2 meters as these values where specified in the range vector R during the ELS data acquisition. The reason to why the range dimension is not evenly distributed from 0-100 is because of the *eval up to range* function in the ELS software which prevents error estimations above the limit where the LDS has declared the polynomial to be valid.

The yaw rate is evenly distributed around 0 which is reasonable as this indicates that roughly the same amount of left turns as right turns has been executed while the majority of the expedition time where accomplished on straight roads.

The error distribution curve is plotted in figure 3.2. The results indicates that most of the time, the LDS is producing accurate estimations with low error values.

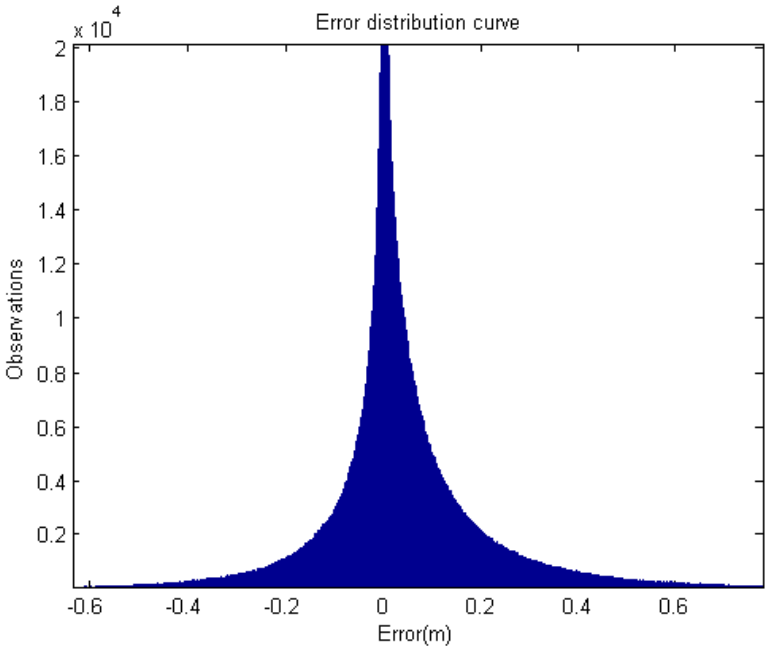


Figure 3.2: Distribution curve of the recieved error values.

The chapter begins with section 3.1, describing the results from the multivariate analysis, which was conducted to investigate which dimensions that constitutes a dominant influence on the LDS.

The multivariate analysis section is followed by section 3.2 which illustrates the results from the analysis of the data, that was created through the data acquisition process. The section covers analysis of the data density as well as identifies the operating points of the LDS.

In section 3.3 the error performance of the real LDS is modeled based on the data and foundings from the two previous sections.
The result chapter is finished with section 3.4 which describes the architecture of the final virtual lane detection model which binds together all results, discovered from the thesis.

3.1 Multivariate Analysis

Prior to the modelling of the virtual model a multivariate analysis was applied to determine which dimensions that should be included in the final system model. The analysis was executed with the *Principal Component Analysis* technique which is described more thorough in section 2.4, *Principal Component Analysis*.

The data matrix D with its 6 input dimensions was fed to the PCA-function and the result of the principal component analysis is displayed in table below.

Dimension	Explanation (%)	Variance
Range	93.83	630.9
Speed	6.14	41.3
Acceleration	0.02	0.13
Lightning Conditions	0.003	0.019
Yaw Rate	0.00	0.00
Weather Conditions	0.00	0.00

Table 3.1: The table shows the results of the principal component analysis with the received variance for every input dimension and the percentage of the variance share.

The results displayed in table *3.1* indicates that the majority (94%) of the variance can be described with the *range* dimension. This means that if only the range dimension would be included in the final LDS-model, the model would still be able to capture 94% of the system behaviour under the assumption that all of the necessary dimensions has been included and enough data has been captured for all of the dimensions in the PCA.

One reasonable explanation to why the range variable is so influential on the lane detection error performance could be that the discretization error due to the pixel representation in the camera increases linearly with range(x) according to equation (3.1) where f denotes the focal length while P_w represents the pixel width. The error is plotted against the range in figure 3.3 with red markers. The continuous black line in the bottom of the figure represents equation (3.1). This alone does not explain much of the error profile but as this only represents a single pixel, and a lane consists of multiple pixels, this can

be a plausible explanation. The pixel errors could also explain why the *LDS* produces an area of errors rather than a line. Equation (3.1) only gives the maximum possible error due to the discretization of the continuous space. The discretization error can thereby assume any value between zero and the maximum error in equation (3.1). This probabilistic effect therefore contributes to the area error profile.

$$e_d(x) = P_w \frac{x}{f} \tag{3.1}$$

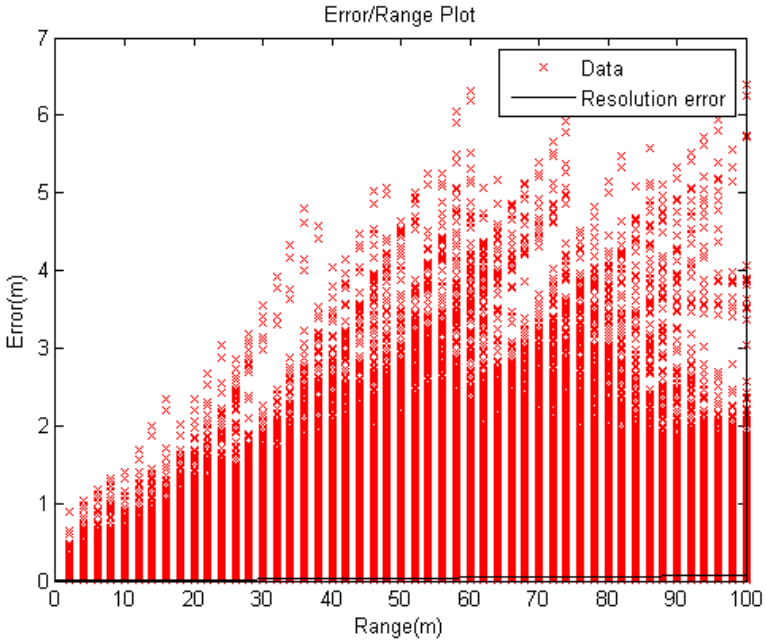


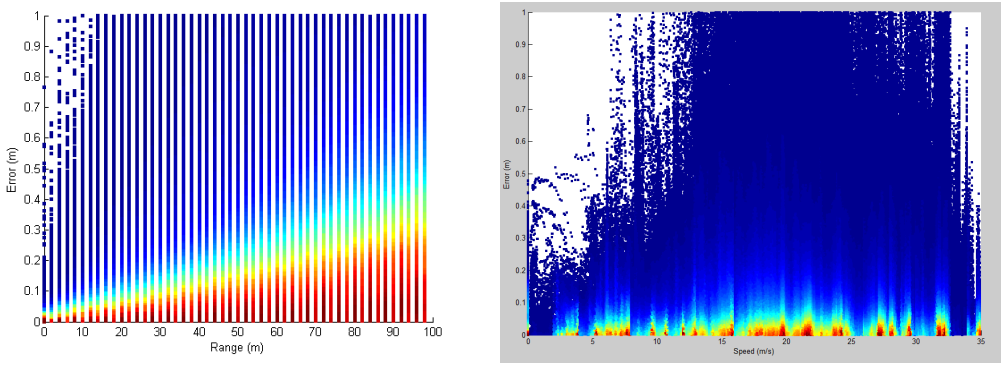
Figure 3.3: The system error profile in respect to range. The black continuous line represents the maximum discretazation error for one pixel which is stated in equation (3.1).

With support of the results from the Principal Component Analysis, it can be concluded that the original 6-dimensional model can be reduced into 2-dimensions (*Range*, *Speed*) without any noticeable effect on the system performance.

3.2 Field Data Analysis

The correlation between the input variable and the error values for both the range- and the speed-dimension is not deterministic as both of the error profiles will form an area like in figure 3.3 rather than a line. Consequently, the 3-dimensional error profile will be represented by a volume instead of a surface (see figure 2.35).

This fact complicates the data analysis process as it is not possible to draw any conclusions about the error performance by only looking at figure 3.3. Most of the plotted data is just single occurrences which does not say anything about the real sensor performance but blocks the view of the more frequently occurring error values. To handle this problem the error profiles has been plotted with a *density scatter plot* function and the results are displayed in figure 3.4. The density plots are utilized to get a perception regarding the input and output data. It is of great importance to be certain that the data is reliable before used to create a model, which is the reason why the density plots are utilized.

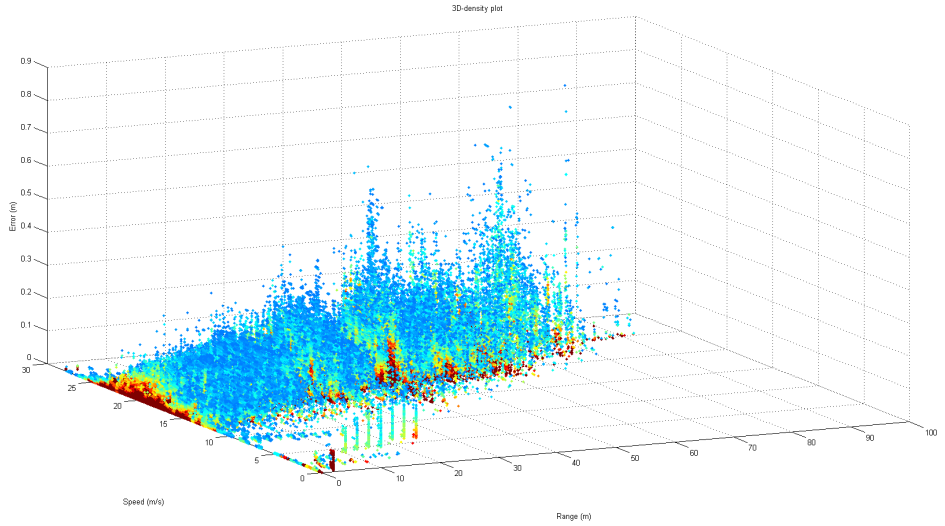


(a) Error density plot for the range dimension.

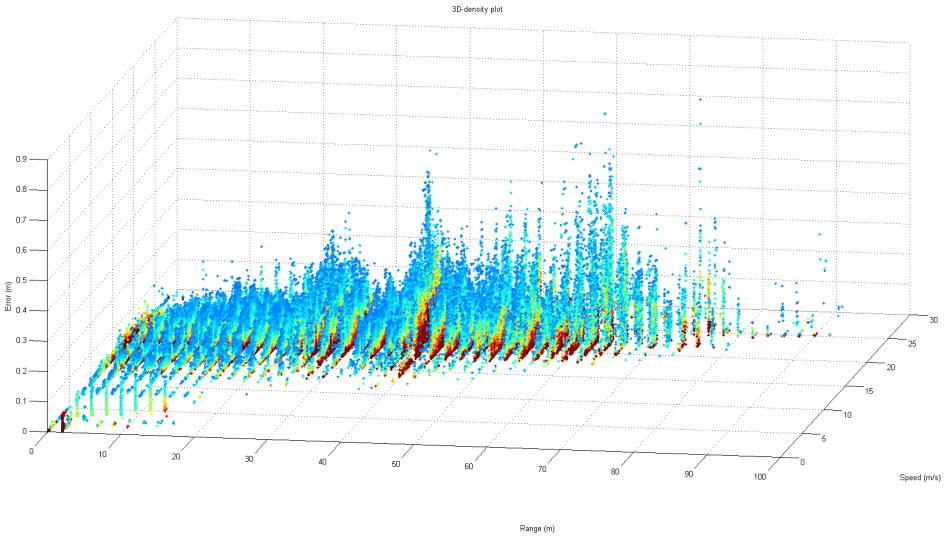
(b) Error density plot for the speed dimension.

Figure 3.4: Error density plots for the range variable as well as the speed dimension. High color temperature indicates high data density and vise verse.

As can be seen most of the data is concentrated in the low error region, which is supported by the histogram results in figure 3.2. We can from these results draw the conclusion that usually, the *LDS* executes with high performance meaning that it successfully estimates the lane markers with low error influence. Even though these density plots gives a perception on how the *LDS* performs regarding the *Range* and the *Speed* dimension, it does not provide the complete picture of the *LDS* performance. The problem is that both dimensions are inter correlated and should therefore be plotted against each other.



(a)



(b)

Figure 3.5: A 3D-density plot in respect to the 3 dimensions *Range*, *Speed* and *Error* created with the *scatterplot3D.m* script. Red color indicates regions with high density and the density decreases with decreased color temperature.

To solve this problem a special software has been developed for this project called *scatterplot3D.m*. As the name reveals the task of the *scatterplot3D.m* script is to make a density curve in 3D. The software encloses the data space with a cube divided into multiple sub cubes. The script then searches through the whole space and calculates how many data points that occurs in every sub cube. The resolution of the sub cubes can be

specified by the user. When every data point has been counted into its cube the software calculates the share every cube represents in respect to the whole space or to a single dimension. Figure 3.5 shows the resulting density plot from when the *scatterplot3D.m* was executed on the input data set.

Once again it can be seen that, the majority of the data ends up in the low error region with less and less data occurrences in the direction of the positive error axis. The figure also highlights that a blind spot exists in the data set at high range and low speed. To further investigate this issue a density plot over the range/speed data was created, which is shown in figure 3.6.

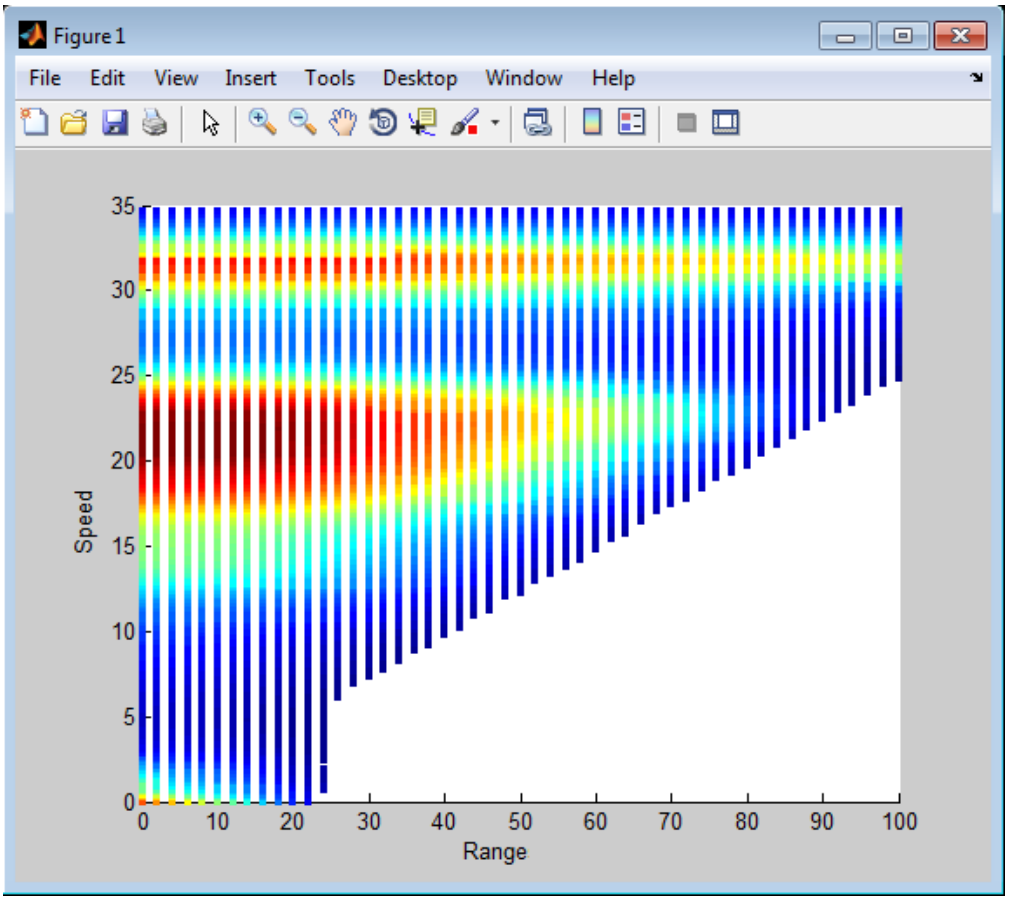


Figure 3.6: A density plot of the Range/Speed data. For high range and low speed there is a blind spot due to the fact that this is an extremely unusual combination.

The density plot confirms the fact that a blind spot exists in the *low speed/high range* region. The reason for the blind spot is that this specific range/speed combination is extremely rare and was not present during the expedition logs that was collected for the input data set. At low speeds, the *LDS* does not estimate the polynomial far away

from the host car as this information is redundant. On the other hand, at high speeds scenarios like when driving on a highway, the *LDS* makes far estimations as information further ahead is more important in these situations. To confirm this conclusion, another density plot was created (see figure 3.7).

The figure shows the value of the *LDS* polynomial distance variable d in respect to speed of the host car. Graphically, there is a perfect match between the missing piece in this density plot and the blind spot in figure 3.6. Data is not collected in the blind spot because the *LDS* does not have a valid representation in this area and the *evalup2range* function therefor prevents *ELS*-software to compute data in this region. The upper limit of the polynomial distance variable seams to follow a strict linear behaviour judging from the results in figure 3.7. This blind spot does not constitute any problem for the virtual lane detection sensor as the real *LDS* does not not operate in this region. The virtual *LDS* should therefor also refrain from operating in this area.

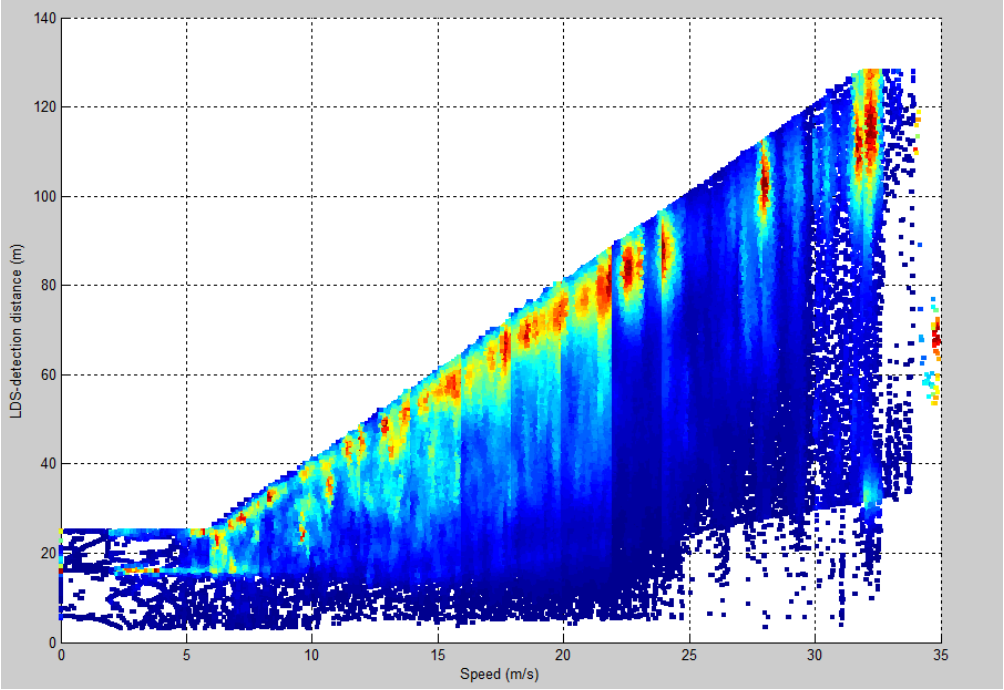


Figure 3.7: A density plot over the *LDS* polynomial distance variable d in respect to the speed of the host car. The distance variable denotes how far the *LDS* estimates the polynomial to be valid. The graph indicates that there is a linear relationship between the speed and the distance variable.

3.3 Lane Detection Algorithm

The major complication regarding the modelling process is the fact that the system behaviour is non-deterministic. This fact renders the 3D error profile to occupy a volume in the range-speed-error space instead of a deterministic surface (see figure 3.8).

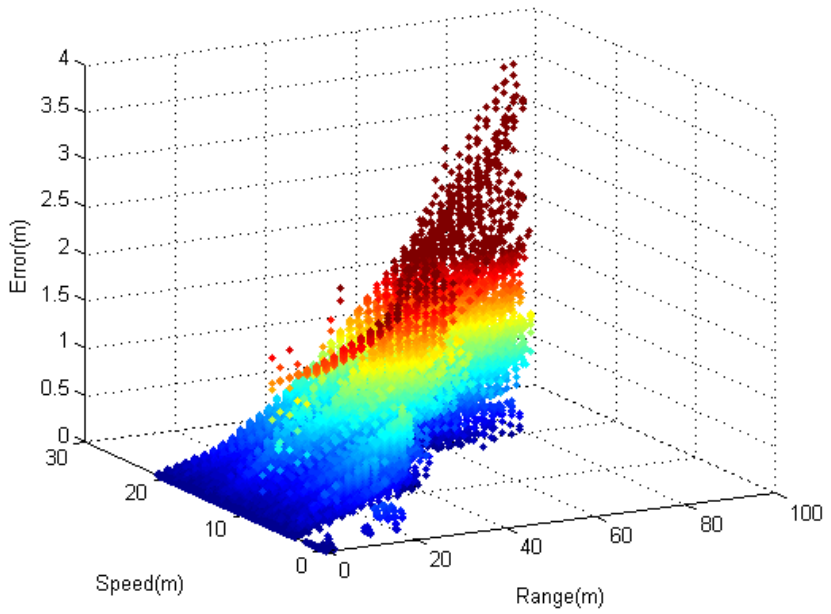


Figure 3.8: The 3D-error performance in respect to *Speed* and *Range*. The system performance is non deterministic which renders the LDS profile to occupy a volume instead of a deterministic surface.

To construct a model which represents the error volume is impossible with simple modelling tools like a nonlinear or linear regression models. A possible solution would be to use the already gained linear model from the principal component analysis but as this model only spans a plane, the model would not be very accurate.

As the LDS includes some randomness which can not be captured, some sort of statistical probabilistic model needs to be applied. To get a perception on the behaviour of the system, confidence level curves of the error profiles was created (see figure 3.9-3.10). Figure 3.9 shows the confidence curve over the range/error distribution. The curve was created by dividing the input into intervals and then adding values to each interval until a certain level of confidence had been reached. The division into intervals comes naturally for the range variables as it is already discretized into sub groups. The confidence level analysis produces the boundary curves for where we can be sure that a certain amount of data (up to the confidence level) is gathered under. As can be seen in figure 3.9 the confidence level curves seems to approach a linear behaviour for 97% of the data.

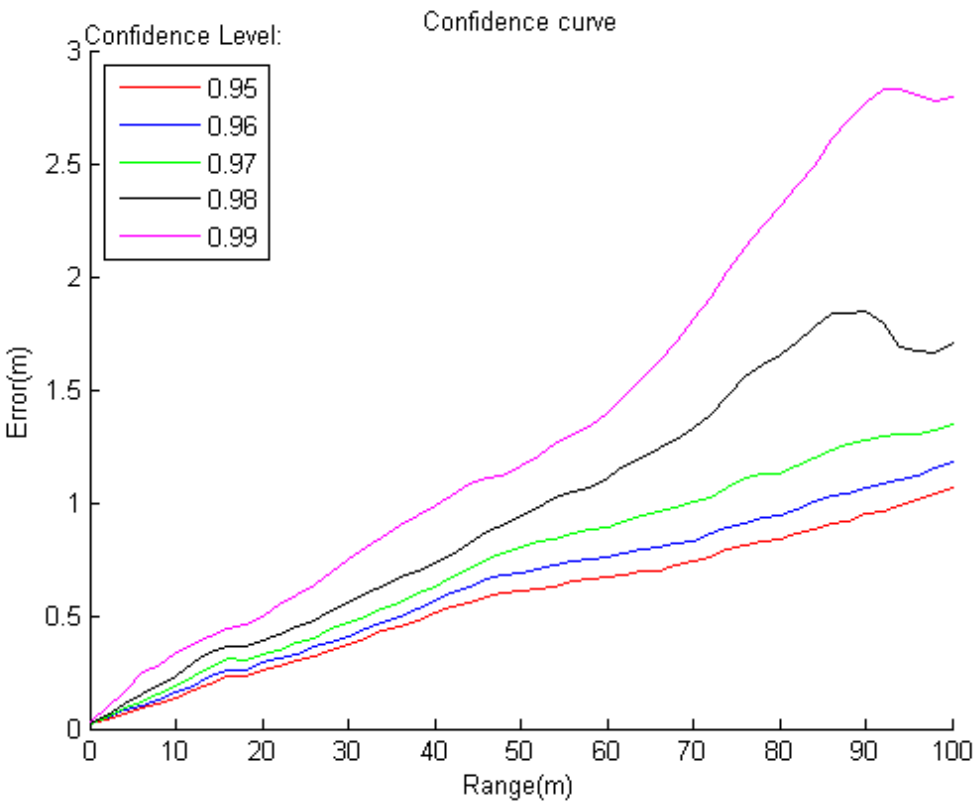


Figure 3.9: Confidence level curve for the range/error profile.

For confidence levels above 97%, the curves are nonlinear at high range. One explanation for this is the reduction of data density in this region. Single occurrences with high error values will consequently be more influential where the data density is low which could be the reason for this nonlinear behaviour.

An identical confidence level procedure was executed on the speed/error data and the results are plotted in figure 3.10. The speed domain was divided into multiple intervals and the confidence level for every interval was calculated similar to the procedure in figure 3.9. The difference between the speed domain and the range domain is that the range domain is naturally divided into sub intervals from the start (see figure 3.4a) while the speed data is distributed irregularly stretching from its min to its max value. The consequence of this irregularity is that the confidence curve is affected by strong fluctuations. The reason for the fluctuations is randomness in the data density for each bin. One bin could potentially include a huge amount of data while the neighbour bin could be almost empty due to the fact that the data isn't evenly distributed. As previously argued, bins with low data density will be more sensitive to high valued occurrences and could therefor give a peak in the confidence curve. To solve these

fluctuations a low pass FIR filter was applied to the original data and the results are displayed by the red curve in figure 3.10. A trend that is noticeable in the confidence level curve is that low speed values gives low errors.

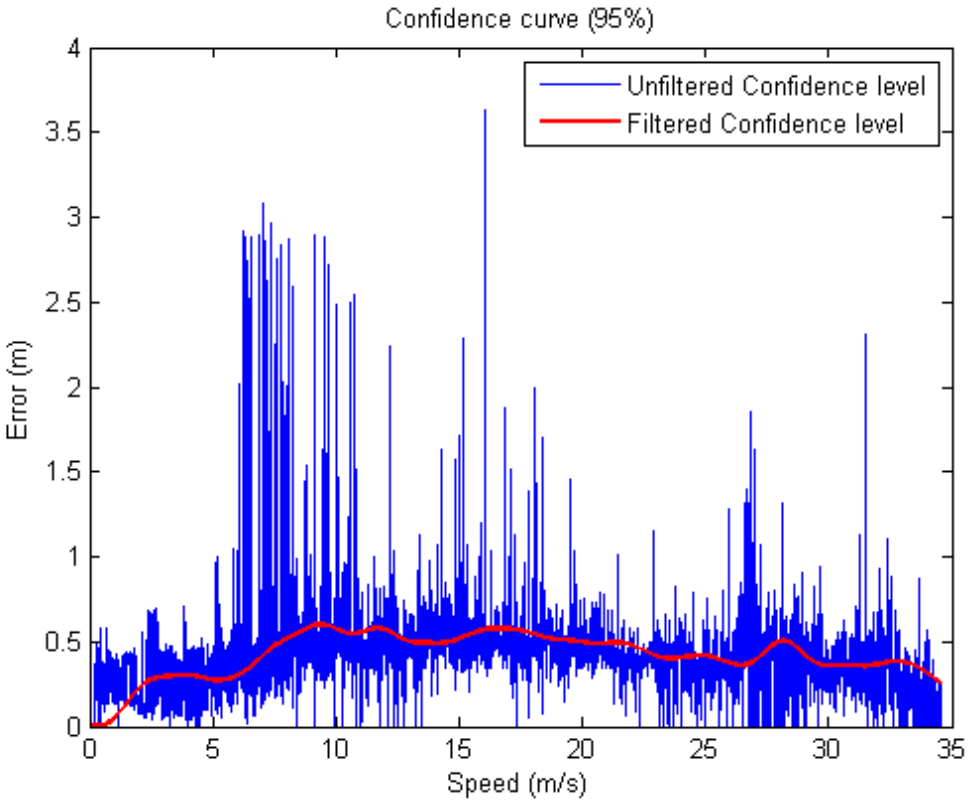
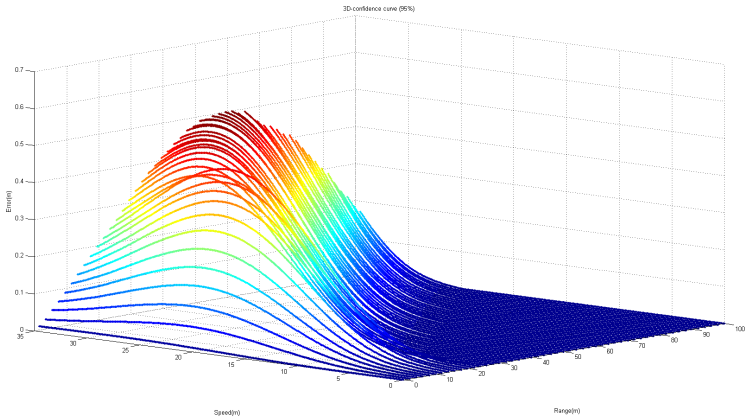
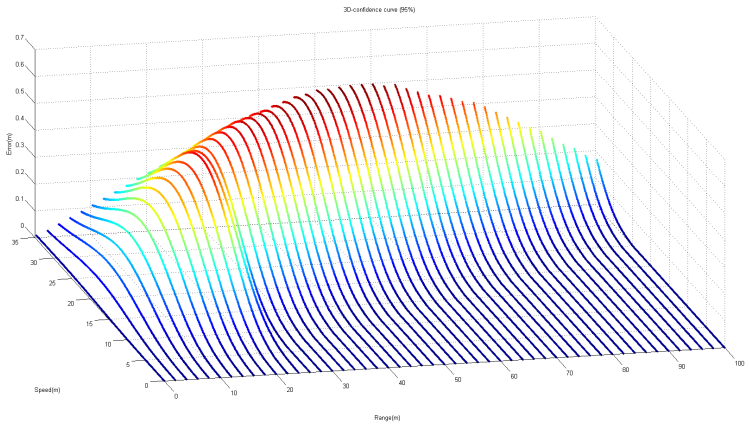


Figure 3.10: Confidence level curve for the speed/error profile. The blue line is the original confidence curve which is influenced by a lot of fluctuations. A low pass FIR-filter was applied to the fluctuating signal and the results are plotted in the red curve.

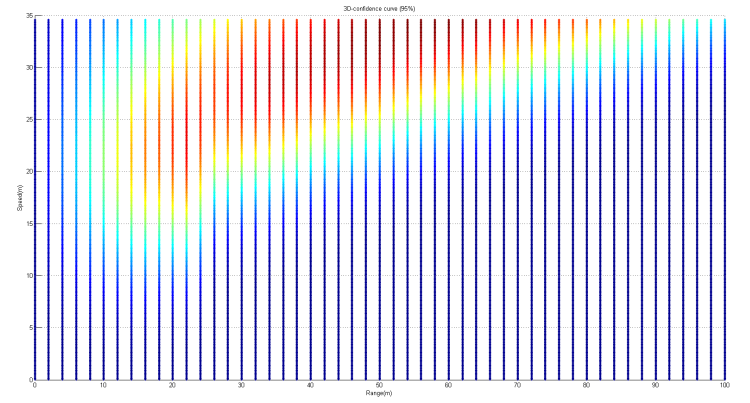
Even though the 2-dimensional confidence level curves gives some perception on the system performance, no certain conclusions can be drawn as these two variables are inter correlated. To deal with this complication a rather unique solution has been developed that utilizes the confidence levels in 3-dimensions. A special script called *conf3D.m* which executes this procedure has been developed. The script divides the range/speed data domain into a grid space and calculates the confidence value for every grid piece. As the 3D-confidence level curve also is sensitive to the grid resolution, the unprocessed output will be strongly influenced by fluctuations. The solution to this issue is, as previously, to apply a low pass FIR-filter to cut away the fluctuation peaks.



(a)



(b)



(c)

Figure 3.11: The 3D-confidence curve of the model with a confidence level of 95%. The model indicates that the system produces increased error values with increased speed and range values.

The results of the 3D-confidence curve model is displayed in figure 3.11. As can be seen in the figure, the 3D-confidence curve combined with the low pass FIR filter gives remarkable results. The system error profile is now represented by single surface instead of a volume. The surface marks the 95% confidence level. We can thereby conclude that 95% of the LDS errors, lies below this confidence surface. Different models with different confidence levels can be modelled in the same way which gives the user the opportunity to choose the model that has the preferred confidence level. It is worth to note that the 3D-model in figure 3.11 also includes the previously mentioned blind spot in the region of low speed and high range. The model is thus not valid in this lineally bounded region.

To convert the surface into the form of an equation a nonlinear regression model can be applied. As the surface has a rather complicated shape it is very hard to represent the whole surface with a single regression model. The parametrization of the model would therefor infuse inaccuracies due to deviations from the true surface which is not preferable. Instead of representing the system with a nonlinear regression model which give unwanted errors the model could be represented by a 2-dimensional look up table. For data inputs that lies in between the grid points linear interpolation could be applied to increase the accuracy even further.

3.4 LDS model architecture

The purpose of this section is to demonstrate the final architecture of the virtual *LDS*, developed in this thesis. Figure 3.12 is a block diagram representation of the final model including the virtual camera model and the adjustment for the system performance.

The *simulation environment* block represents the entire simulation which interacts with the virtual *LDS*. The simulation environment provides the camera model with the virtual 3D-world as well as the current states of the host vehicle. The 3D-world is represented by a *3xN* sized environment vector which includes the positions of all the points in the entire virtual environment. The state variables that are provided to the camera are the orientation of the host vehicle $[\theta_v,\phi_v,\psi_v]$ as well as its global position $[X_v,Y_v,Z_v]$. Based on these input variables, the camera model then calculates the current 2D-image and from that the positions of the left and the right lane markers. The positions are gathered in two separate vectors containing the locations of the left and the right lane markers.

The lane position vectors, along with the current speed of the host car are then filtered through the *range/speed* filter block which prevents the virtual *LDS* from operating in the area where the real *LDS* does not operate based on the findings in section 3.2. The filter block analyses the range of every lane point in the lane vectors and compares that to the current speed of the host car. If a point is located in the restricted area it is extracted and discarded from the lane position vector. Without this block, points located in the restricted area will be further processed by the *LDS* performance block which in turn will apply no error to points in the area. This could lead to optimistic estimations of the lanes, which consequently could be critical in evaluation of active safety functions.

The remaining lane points are then further processed and shifted in space according to the error value they receive from the *LDS* performance block. The error value is calculated according to the speed of the host car and the range of the lane point with the 2D-look up table developed in section 3.3. To convert the adjusted lane positions to a polynomial representation a conventional polynomial fit tool can easily be used which fits a 3rd degree polynomial lane position vectors. By doing this, the lane vectors which are have a discrete representations are converted into a continuous media giving the user the opportunity to calculate the lane positions at any location of the lane estimation.

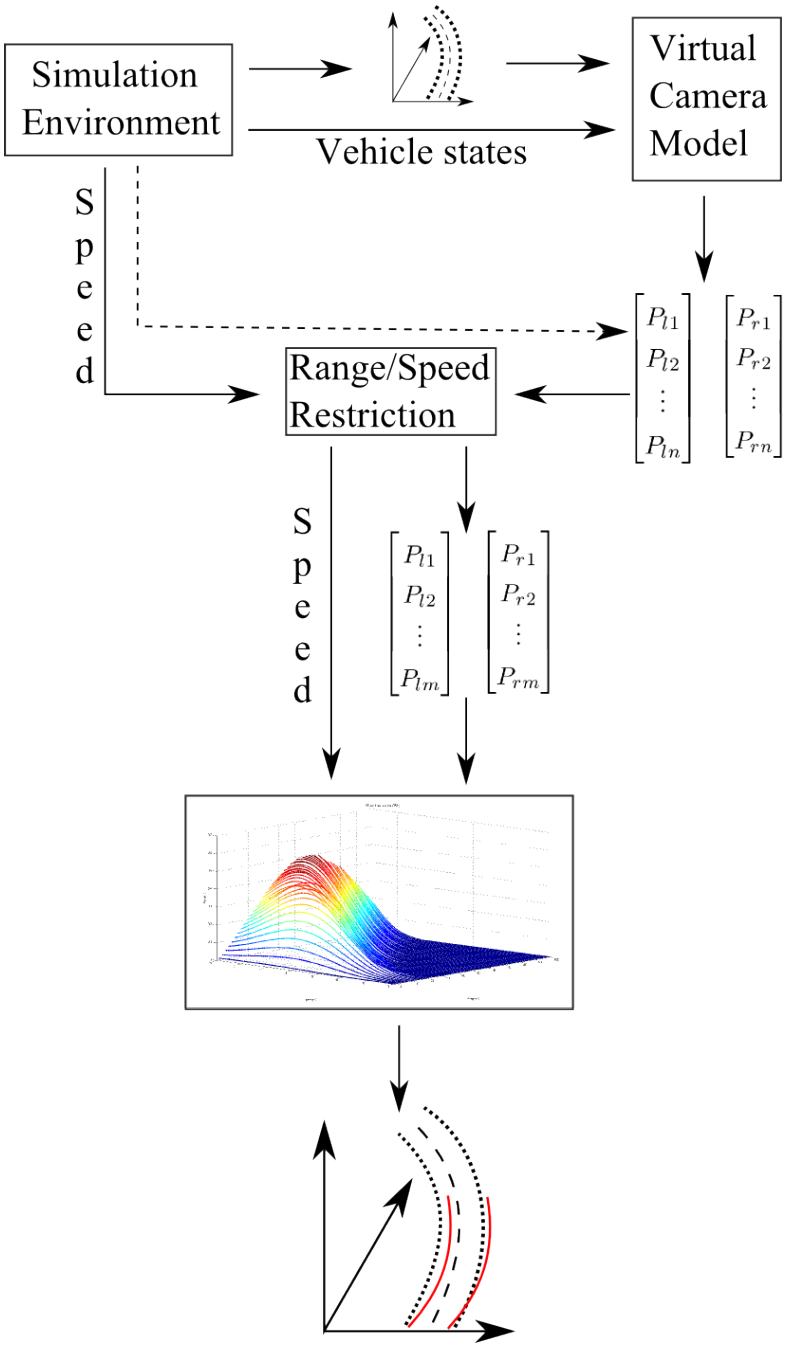


Figure 3.12: A block diagram, explaining the final structure of the virtual *LDS*. The simulation environment supplies the camera model with the virtual world as well as the state of the host car. The camera model is the processing the virtual world and calculates the positions of the left and the right lanes, saved in two separate vectors. The lane position vectors, together with the current speed of the host car are processed through the range/speed filter which prevents the virtual *LDS* from operating in areas where the real *LDS* isn't operating. The remaining lane positions are then adjusted with the 2D-look up table that represents the sensor performance.

As the camera model can be heavy computationally a alternative approach has been developed which overrides the visual component. The simulation environment could provide the virtual LDS directly with the position vectors of the left and the right lane. By extracting the points in the virtual environment that represents the lanes up to a certain range these vectors can be created without the camera model. The range would be determined by the current speed of the host car according to the findings in section 3.2. By doing this, computational power is saved to the cost of the lost perks which the camera model provides. The lane positions would not be influenced by errors introduced by the hardware setting of the visual component which is natural error that preferably should be included in the virtual LDS. By overriding the visual component there is also no way for the system to prevent lane estimations due to blocking, which is achieved by the ray intersection algorithm. This could be a critical function as in absence of it, the simulation environment could be given more information than it would have been provided in a similar real life situation.

Figure 3.13 shows a schematic overview of how the virtual LDS interact with the simulation. The simulation environment block provides the LDS with the position and the orientation of the host car, the current speed as well as the virtual 3D-environment and the LDS computes the left and the right lane polynomials.

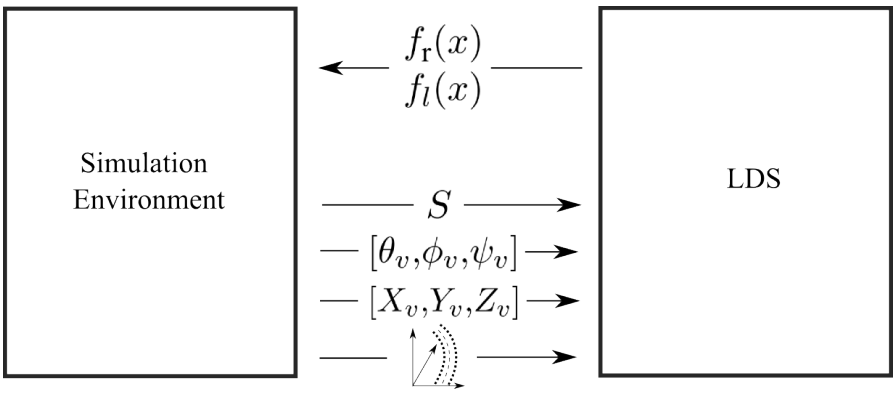
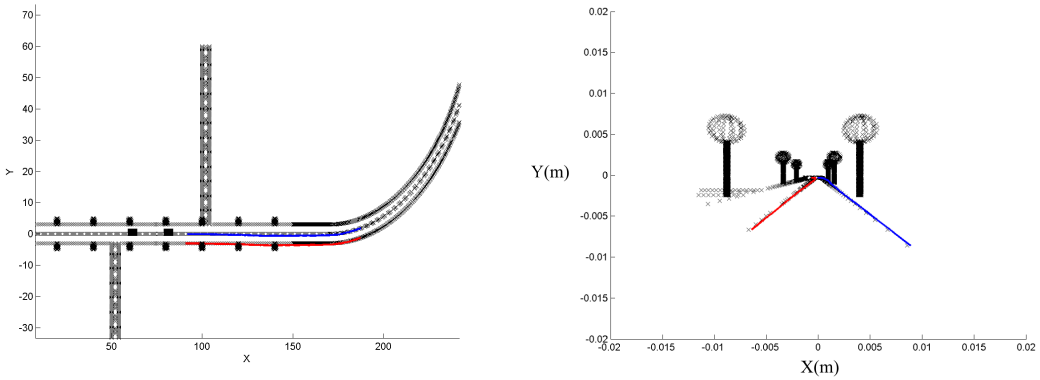


Figure 3.13: Schematics over how the LDS model interacts with the simulation environment. The simulation environment provides the LDS with the position and the orientation of the host car, the current speed as well as the virtual 3d-environment. The LDS then calculates the two polynomials representing the left and the right lanes and outputs them to the simulation environment.

The virtual *LDS* model was finally implemented in a virtual environment using the architecture defined in figure 3.12 and verified through simulations. Figure 3.14 shows a visualisation from the results from the lane detection in the simulation. Figure 3.14a shows a overview of the lane detection, where the blue line represents the left lane estimation while the red line represents the right. Figure 3.14b displays the same scenario from the perspective of the camera where the estimated 3D-point of the lanes, where projected onto the camera image.



(a) **(b)**

Figure 3.14: Images created through simulation where the final virtual *LDS* model was implemented. (a) Shows the resulting lane detection from a overview perspective. (b) Shows the resulting lane detection from the view of the virtual camera.

As the real *LDS* operates in a non deterministic manner, their is no way of accurately validate the model by running the virtual model parallel with the real system an expect identical estimations. Instead, real life scenarios where compared with similar scenarios in the simulation environment to conclude that the virtual *LDS* behaves similarly to the real *LDS*.

4

Discussion

The purpose of this chapter is to summarise the project and to guide the readers with interest of further development. The knowledge gained from the research will be stated in the Main conclusions section among with opinions about alternative approaches.

The Main conclusions section is followed up with the section about further research which is intended as an encouragement for further development in the field as well as to pinpoint the areas of focus. Here, suggestions about factors that will improve the model additionally will be stated.

4.1 Main Conclusions

Through the work, conducted in this thesis a visual based virtual *LDS* was developed and evaluated using two main blocks, a *camera model* and a black box *lane detection algorithm*. The two blocks where then proven to be combinable to create a final virtual *LDS*. The system was then verified by integration with a virtual simulation environment.

By creating a visual component and including that in the virtual *LDS* model, multiple benefits were gained. Instead of just treating the *LDS* as a single black box system, the camera component was extracted from the black box, and modelled separately. This gave among others, the ability to test different types of hardware settings. The hardware evaluation feature can help the user to tune the real hardware setup into the gear that is the most favourable solution regarding, economics, performance etc. The hardware evaluation area is a very promising area with a lot of potential but it needs some further investigations and development before it can be utilized in an effective manner. As the virtual camera model, developed through this project is relatively simple in relation to modern visual systems some comparison is needed to evaluate the validity of the virtual camera model. Even though the entire hardware performance most probably can not be captured with the model, the camera provides another key feature to the *LDS*. The

ray intersection algorithm, included in the model gives the opportunity to have segment interruption due to view blockage which otherwise would be really hard to program. This component makes the virtual LDS much more realistic by enabling smart estimation of the the length of the lanes. Even though the algorithm increases the performance of the system, it requires higher complexity of the simulations environment as all objects needs to include entities about their spacial representations such as their 8 sides and appurtenant norm vectors. Another downside with the algorithm is that it was shown to require a lot of computational power which rendered the simulation to slow down. The virtual camera model was also shown to introduces natural errors due to pixel discretization which is a crucial component in the system performance. This feature adds the randomness to the virtual model due to the fact that points in continuous space needs to move different distances dependent on where they are situated to be able to fit the pixel representation. This is an error which is really hard to model naturally. In addition to all of these benefits gained from the camera model the model also enables the user to save to processed virtual images and use them for visualisation purposes. The extra gained visualisation tool can be highly helpful for simulation analysis. Apart from all the advantages with the virtual camera, the model was shown to slow down the simulation process. The camera is therefore not suited for real time simulations. The model was developed to be optimal in the aspect of computational power but point cloud scenarios with high point density can all the same force the camera model to slow down the simulation speed. To overcome this problem, the system has been designed with the possibility of overriding the virtual camera model when using the LDS in the way described in section 3.4. This gives the user the possibility of disabling the camera to gain extra computational speed.

The research, conducted on reveling the black box LDA gave some insight in the functionality of the lane detection algorithm and knowledge about how the system operates. In an early stage, it was concluded that the LDS operates in a non-deterministic manner regarding the error performance. This is most likely explained by the fact that the discretisation error, introduced by the camera model adds some sort of random behaviour which causes the error performance occupy a volume instead of deterministic surface.

Another conclusion that can be drawn after the black box modeling process are the results from the multivariate analysis. The multivariate analysis gave proof that the LDS is mainly influenced by the two dimensions *speed* and *range* while other dimensions like *yaw rate* and *acceleration* are redundant. One explanation to why this is the case is that the range is the key aspect to the error in the LDS. The error related to the range variable is most likely based on two factors. The first is the fact that the error, infused by the pixel representation increases linearly with range from the optical center according to equation 3.1 presented in section 3.1. The equation shows the maximal distance a continuous point on the camera image needs to be moved to fit the discrete pixel representation. It was also proved that this alone does not explain the entire error profile, but as this only displays the possible error for one pixel a higher error would be expected for a lane which consists of multiple pixels. A second factor that also explains

the error/range relation is the fact that the real LDS does not far from perfectly, pin points the locations of the lane pixels. A false estimation will, with the same affect as in equation 3.1 grow linearly with range. So why does the speed dimension affect the error performance of the LDS? One explanation is that the speed of the host car affects the LDS ability of estimating long range lanes. As shown the thesis, the LDS estimates low range polynomials for low speed while higher speeds enables estimations further away. As the range, through this relation is correlated to the speed, the speed in turn has an affect on the error performance of the LDS. One alternative approach to the one presented in this report would be to only use the speed as a range estimator variable and in turn only use the range dimension as the input variable for estimating the error and by that create a 2D-model instead of the 3D-model, presented in this report. Another surprising result gained from the multivariate analysis conducted in this thesis was that yaw rate had very little affect on the error performance of the LDS. The initial hypothesis in the beginning of the project was that yaw rate would aggravate the lane estimation process. High yaw rate values would imply high road curvature which would make the estimation much more difficult due to the higher geometrical complexity. One reasonable explanation to why this phenomena does not occur is that the yaw rate is measured as a state of the car at the time instance when the polynomial is estimated. This does not say anything about the future road curvature even though the lane estimation further away is dependent on future curvature values. The measuring technique would be accurate in a scenario where the vehicle is in a curve with constant curvature but not in situations where the host vehicle is driving on a straight road with an oncoming curve. To further investigate this problem, the ELS could be updated to calculate future yaw rate values. An experiment where pure curvature situations are included could then be conducted to investigate how the system behaves.

By analysing the performance of the lane detection algorithm it was shown that a strong correlation between the current speed of the host car and the distance which the LDS estimates the lane polynomials to be valid exist. The correlation between the two variables is most likely caused by a software boundary programed by a function developer. The boundary most likely exist due to the fact that for low velocities, long range information is not necessary for the on board active safety functions as it is only information about the proximity of the vehicle which is important. For higher velocities, information further away from the host vehicle is more critical which therefore needs to be accessible by the active safety functions. The behaviour was extracted and included in the final virtual LDS model.

Finally it was showed how a rather unique technique using 3D-confidence level curves with applied low pass filters could be utilized to model the system performance. The technique was encouraged by the fact that the error profiles was influenced by randomness which made the error values occupying a nondeterministic volume. As normal mathematical modelling techniques could not be applied to model a nondeterministic space, the 3D-confidence level curve approach was successfully developed. As this approach is sensitive to areas with low data density, the technique could be devious if a

data distribution analysis has not been applied in prior. As the resulting model is a statistical model which only captures the extreme performance of the system it will never have the ability of being deterministic. The virtual model can thereby never be executed in parallel with the real system with expectancy of perfect correlation between the two. The task of creating a fully deterministic model is not possible so the purpose of the LDS is more accurately to capture the average behaviour or the extremes.

The two main blocks were successfully combined and verified by integration with a virtual environment. The final virtual LDS model was shown to generate multiple advantages such as the ability of being automatically tuned. If the real LDS would be updated, either through software or through hardware improvements, the virtual system can be tuned to run according to the new system performance. The user would simply evaluate the updated LDS by collecting field data through expeditions and then feed the data to the programs that was developed in this thesis. This feature could possibly be used for more then just preserving the relevance of the system. As the performance of the virtual LDS adapts to the received field data, the user can apply this method to create multiple models which could be used for different scenarios. Potential scenarios could for example be, weather conditions, specific traffic scenarios or lightning conditions. The data, extracted for this thesis did not include enough field data influenced by bad weather conditions or expeditions carried out during night time to be able to draw any definite conclusions about their relevance. Based on the perception regarding the behaviour of the LDS that was gained through this project, if these variables where to be included they would most definitely benefit from being modelled into separate models rather than to be included as input variables. The user can then create a library of models including a night model, a rain model etc to more accurately emulate the wanted performance of the virtual LDS.

4.2 Further Research

The errors are calculated relative to the ground truth line which is computed by preprocessing the polynomial estimations and taking the 0-range predictions for all polynomial instances. The ground truth lines are thereby constituted on the assumption that the system is making absolute accurate predictions in the proximity of the car. This kind of nested error definition where the LDS is evaluated relative to itself is ideally not preferable. A better solution would be to evaluate the performance of the system relative to a well know and independent system like a $LIDAR$. The $LIDAR$-system provides high level of accuracy and is uncorrelated to the LDS. The best solution would therefore be to run the $LIDAR$ and the LDS parallel while collecting data and then base the ground truth analysis on the $LIDAR$ output. This approach would provide data closer to the truth which in turn would generate a more accurate model.

A deeper data distribution analysis similar to the one executed in section 3.2 would potentially benefit the LDS model. By gather data to fill in the gaps in the distribution curves to evenly distribute the input data, the distribution effect can be totally excluded

which would isolate the pure effect of the model. The LDS model in this thesis, designed with the 3D-confidence level method could potentially be affected by the difference in data density in the scope of the system. To effectively collect data in the distribution gaps it is suggested a script that scans through expedition logs and finds occasions where these specific situations occurs is developed. It would also be beneficial to collect field data, including extreme situations such as extreme weather or night time driving. As the field data collected for this thesis did not include enough data, that was influenced by this conditions no absolute conclusions can be draw about their affect on the LDS.

An additional area where further research can be applied, is the final confidence curve model. The model covers the top surface which represents the extremes. This is not always wrong when testing active safety functions. If the extremes are covered by the active safety features, so are the less severe situations, but to develop a more realistic LDS the system should be modelled to cover the volume which is occupied by the error profile. One approach to model the volume is to discretise the volume with multiple confidence level surfaces. The LDS model should then jump between these confidence levels and a probabilistic variable should decide how frequently the LDS should be at every level. In this way, the model will obtain a dynamic behaviour which will bring the virtual LDS one step closer to a to the real LDS.

Bibliography

[1] M. P. Tami Toroyan, Kacem Iaych, Global status report on road safety 2013, *Supporting a decade of action*, World Health Organisation.

[2] NHTSA, National motor vehicle crach causation survey, National Highway Safety Administration.

[3] A.Zapp, E.Dickmanns, A curvature-based scheme for improved road vehicle guidence by computer vision, SPIE conference on mobile robots.

[4] Volvo Cars media, `https://www.media.volvocars.com/global/en-gb`, accessed: 2015-04-26.

[5] A. K. Bin Yu, Lane boundary detection using multiresolution hough transform, Michigan State University.

[6] M. C. David Schreiber, Bram Alefs, Single camera lane detection and tracking, IEEE Conference on Intelligent Transportation Systems, Vienna.

[7] S.-J. T. Tsung-Ying Sun, V. Chan, Hsi color model based lane-marking detection, IEEE Conference on Intelligent Transportation Systems, Toronto.

[8] Y. Z. R. Wang, Y. Xu, A vision based road edge detection algorithm, Military Transportation College.

[9] K. Kluge, Extracting road curvature and orientation from image edge points without perceptual grouping into features, University of Michigan.

[10] E. H. Tomas Akenine-Moller, N. Hoffman, Real-Time Rendering, A K Peters,Ltd, Natick, Masssachusetts, 2008.

[11] J. B.Kuipers, QUATERNIONS and ROTATION SEQUENCES, *A Primer with Applications to Orbits, Aerospace, and Virtual Reality*, Princeton University Press, 41 William Street, Princeton, New Jersey, 2002.

[12] R. B. Milan Sonka, Vaclav Hlavac, Image Processing Analysis, and Machine Vision, Cengage Learning, 200 First Stamford Place, Suite 400 Stamford, CT 06902 USA, 2008.

[13] K. Pearson, On lines and planes of closest fit to systems of points in space, FRS, University College London.

[14] H. Abdi, J. Williams, *Principal component analysis.*

[15] L. I. Smith, *A tutorial on Principal Components Analysis*, `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf` (February 2002).

[16] Principal component analysis, `http://en.wikipedia.org/wiki/Principal_component_analysis`, accessed: 2015-05-13.

A

Software Description

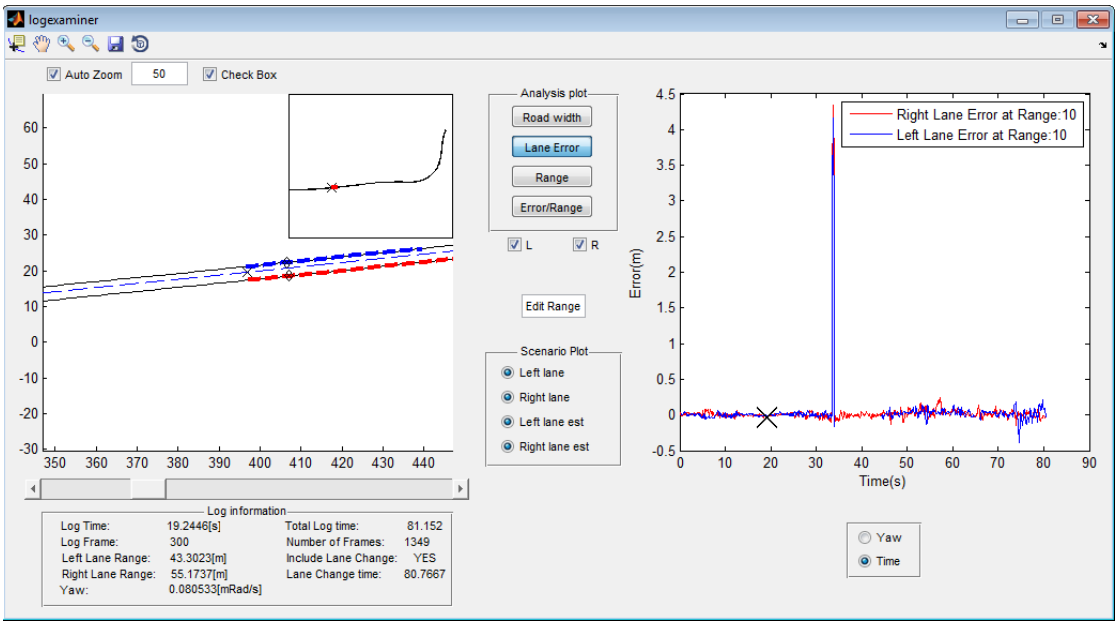
The following chapter is intended as a manual for all the software's that has been developed during the project. The functionality of each program is describes as well as guidelines to how the user should handle the software.

A.1 ELE

The purpose of the Expedition Log Examiner is to help the user to analyse the performance of the lane detection sensor. The function takes a data structure, collected from an expedition as input and then computes and evaluates the sensor performance during this expedition. The performance is then visualized for the user in the *ELE* graphical user interface window.

In order to initialize the Expedition Log Examiner, in prior to starting the software the user needs to load the expedition log data that should be investigated so the structure element called LOGDATA is available in the matlab workspace. When the data successfully has been inserted into matlab the *ELE* is ready to be executed. Starting the expedition log examiner will open the graphical user interface window which is displayed in the figure below. The program needs about 3-5 seconds to initialize without any input from the user. Giving the *ELE* inputs during the initialization process could render the program to give out error messages which would stop the logexaminer. The software will indicate when it is fully initialized by blinking the function buttons.

The user interface consists of two plot windows with appurtenant control buttons. The left window is the navigation window which helps the user to navigate through the selected expedition log with the help of the slider control below the plot window. The trajectory of the car will be visualized and the user can specify if the ground truth line and the lane estimation should be plotted for the left and the right lane. The user can also control zoom of the navigation window by specifying a value in the zoom text box. If



the auto zoom tick box is disabled, *ELE* will display the whole trajectory of the selected expedition log. Next to the zoom box is the check box which enables a small window that displays the whole trajectory during zoom operations. For further navigation an information panel has been added below the navigation window. The panel gives the user information about the current state of the vehicle as well as general information about the selected expedition log.

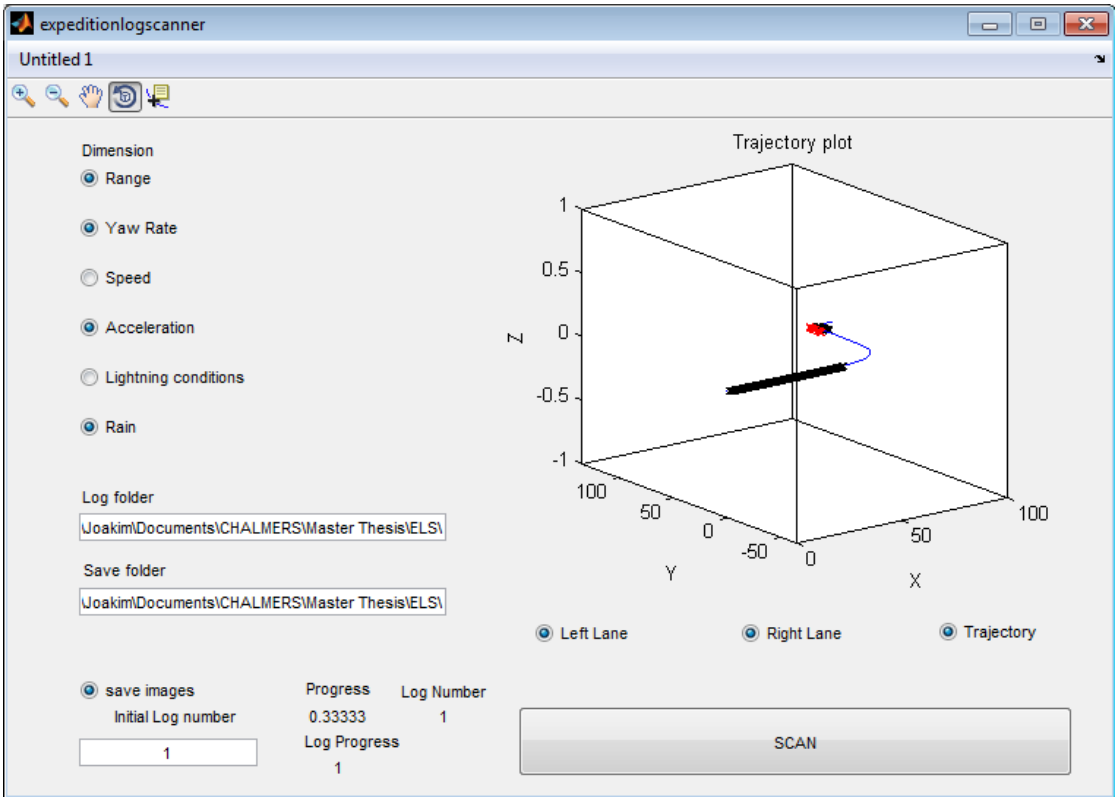
The second plot window is the analysis window which displays the results of the sensor evaluation. The user has the option to choose analysis methods from four different functions, the *road width* function, the *lane error* function, the *range* function and the *Error/Range* function. When the user has selected a function, *ELE* will evaluate the sensor performance of the lane detection sensor and display it in the analysis plot window. *ELE* will also mark the selected time instance selected in the navigation window with a black cross in the analysis window so the use knows exactly where in the analysis results a specific time instance is located. The user can also choose if the analysis data should be sorted chronologically or sorted by the yaw rate. This is done with the panel below the analysis plot window.

A.2 ELS

The *Expedition Log Scanner* is a GUI-based tool which aids the user to extract and compute data from expeditions logs which serves as the foundation for the *LDS* performance analysis. The processed data will be saved in individual data matrices, separately for every expedition log that has been fed to the program.

The user specifies, the location of the log data in the *Log folder* window and the location where *ELS* should save the processed data in the *Save folder* window. The data in located in the Log folder must include a variable called LOGDATA.

The user can also specify which dimensions the *ELS* should evaluate the sensor against by marking the wanted dimensions in the radio button list. The included are: *Range*, *Yaw Rate*, *Speed*, *Acceleration*, *Lightning Conditions* and *Weather conditions*.



To help the user to analyse the processed logs, the performance of the sensor can be displayed in the plot window on the right hand side. The user can choose from displaying the performance of the sensor on the left lane, right lane or displaying the trajectory with the lane estimations. The images can also be saved to the save folder by turning on the *save images*. This helps the user to sort out the logs which are not valid and should not be included in the sensor evaluation. Note that the save process slows down

the extraction procedure.

By altering the *Initial Log number* input the user can control where the extraction should start. This could be helpful for example if the ELE was terminated during extraction. Instead of starting from the beginning of the log list the ELE can start where the process was terminated.

When everything is defined according to the preferences of the user the software is ready to be executed by pressing the SCAN button.

A.3 mergeData.m

The *mergeData* script was developed to merge the data extracted from the expedition log scanner into one unified matrix as this is much more suitable for further analysis. In prior to call the function, the user should gather all of the *.mat* files that should be merged into one or multiple folders. The directories to the folders then serves as input to the mergeData function. The location of the folders should be specified in a char array.

The software does not merge matrices of different size to avoid dimension mixing. If this different size values are detected the software will give out a warning message and ignore the matrix. Given that the function is included in matlab's search path the software is called with the following command:

```
D=mergeData(folders)
```

A.4 scatterplot3D.m

The *scatterplot3D* was created to enable visualization of data density in 3-dimensions. The software divides the space represented by the input data into a 3D-grid and then computes the data density in every grid element. The user specifies a **Nx3** sized matrix containing the data that is to be analysed and the function will calculate the corresponding density matrix.

```
scatter3D(D,res)
```

A.5 ldsTune.m

The purpose of the *ldsTune.m* script is to tune the lane detection sensor model according to the input data, fed to the function. The function calculates the 3-dimensional confidence curve on the input data specified in matrix D. The matrix should include 3 dimensions and N observations in order for the program to function. The function is executed with either of the commands, stated below.

```
ldsModel=ldsTune(D,conf)
```

```
ldsModel=ldsTune(D,conf,res)
```

The *ldsTune* script calculates the confidence surface for the input data up to the confidence level that is specified in *conf*. The user can also alter the resolution of the 2nd input dimension by stating a resolution value in *res*. By default, ldsTune will give the resolution value 500 which means that the speed dimension will be sorted into 500 bins.

The output of the function is an LDS model in the form of a 2D-look up table with appurtenant X and Y values. The output will be saved in the form of a struct.

B

Software Schematics

Figure B.1 below shows the schematics over how the software's, developed in this thesis should be utilized and how they should interact with each other. The main input for the software system is field data, collected through test expeditions.

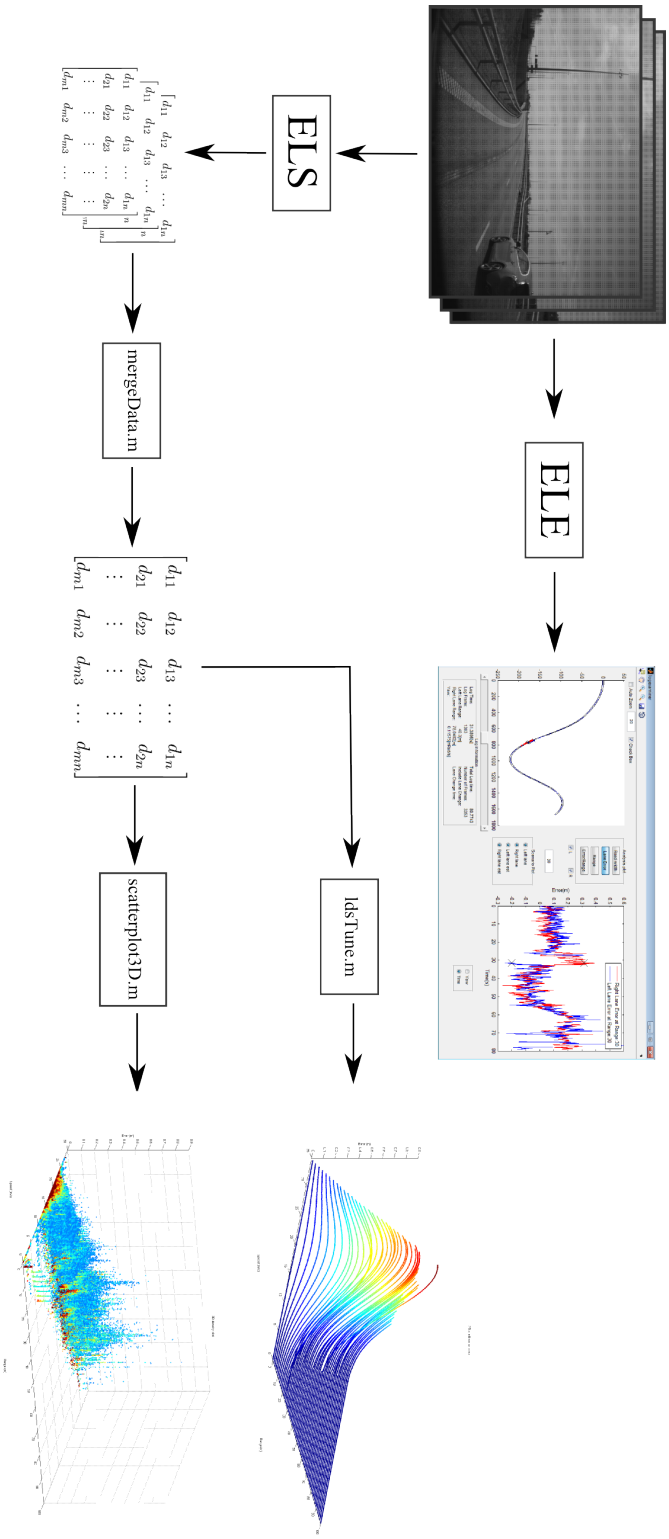


Figure B.1