CHALMERS

Lane Departure Warning and Object Detection Through Sensor Fusion of Cellphone Data

Master's thesis in Applied Physics and Complex Adaptive Systems

JESPER ERIKSSON
JONAS LANDBERG

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

MASTER'S THESIS IN APPLIED PHYSICS AND COMPLEX ADAPTIVE SYSTEMS

Lane Departure Warning and Object Detection Through Sensor Fusion of Cellphone Data

JESPER ERIKSSON
JONAS LANDBERG

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2015

Lane Departure Warning and Object Detection Through Sensor Fusion of Cellphone Data
JESPER ERIKSSON
JONAS LANDBERG

Master's thesis 2015:03
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Example output from the model

Chalmers Reproservice
Göteborg, Sweden 2015

Lane Departure Warning and Object Detection Through Sensor Fusion of Cellphone Data
Master's thesis in Applied Physics and Complex Adaptive Systems
JESPER ERIKSSON
JONAS LANDBERG
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

This master thesis focus on active safety for the automotive industry. The aim is to test an inexpensive implementation of some common functions realized using a cellphone to gather data. A Matlab Simulink model is developed for the purpose, and then the agility of the model is tested by generating c code and running it on a single board computer.

A robust lane detection algorithm is developed by using Hough lines. To better cope with curves in the road, the Hough lines are combined with a parabolic second degree fitting. The Hough lines are also used for a Lane Departure Warning system. Using edge filtering and connected component labeling an obstacle warning is implemented.

Overall the model works well and is fast enough to meet the real time requirements when run on a computer. On the Raspberry Pi 2 chosen as the single board computer the processing is unfortunately not quite fast enough for high speed driving. However when the object detection is removed the Raspberry Pi 2 meets the real time requirements as well.

Keywords: active safety, cellphone, digital image processing, Hough lines, Hough transform, image processing, lane detection, lane departure, LDW, Matlab, object detection, object tracking, obstacle detection, Raspberry Pi 2, Simulink, single board computer

Preface

As the final part of our M.Sc. degree at Chalmers University of Technology, this report represents the result of our master's thesis work carried out at ÅF AB.

ÅF is a large consultant firm, with a lot of representatives working in the automotive industry. As such, their combined experience and knowledge within the area of active safety is vast. Many of the big automotive companies are competing to be among the first to release fully autonomous vehicles, and ÅF as a consultant company wants to be an active part in the success of such vehicles. However, many active safety systems are based on equipment expensive both financially and computationally. ÅF is therefore interested to see if cheaper equipment such as a cellphone can be used to obtain safe results, preferably in such a way that it can be implemented on an electrical test vehicle.

Acknowledgements

We would like to extend our sincerest gratitude to Erik Karlsson at ÅF AB for coming up with the idea for this thesis work, and for his guidance and motivational work as our immediate supervisor. Further, we would like to thank Dr. Krister Wolff, our examiner at Chalmers University of Technology.

We would also like to thank ÅF consulting company for supporting this project, contributing both to the success of this thesis and to our personal development.

Patrik Järlebrantd, Måns Larsson and Louise Kempe are all heroes for stepping through roughly 2x400 frames of video manually labeling 4 points in each frame. No quantitative data would have been obtained if it weren't for you!

Finally, we would like to thank our collegues at ÅF for all the coffee breaks, lunch discussions and positive feedback throughout the work.

Nomenclature

A small glossary of words and abreviations used in this thesis. The page reference is a guide to where the glossary might be used or explained in more detail.

AUTOSAR (AUTomotive Open System ARchitecture), *A global development partnership of vehicle manufacturers, suppliers and other companies from the electronics, semiconductor and software industry designed to eliminate the barrier between hardware and software* [1], page 2
EDF (Edge Distribution Function), *A one dimensional function describing the number of edges in each direction in the image*, page 5
FPS (Frames Per Second), *Number of frames displayed each second*, page 3
GPIO (General-Purpose Input/Output), *A generic pin on an integrated circuit whose behavior, including whether it is an input or output pin, can be controlled by the user at run time*, page 15
GPS (Global Positioning System), *Space based location system that provides details regarding current position if there is an unobstructed line to at least 4 GPS satelites*, page 2
IPM (Inverse Perspective Mapping), *The mapping of an image to a top-view representation*, page 5
LBROI (Lane Boundary Region Of Interest), *Number of frames displayed each second*, page 5
LDW (Lane Departure Warning), *A warning triggered when vehicle is about to leave the traveling lane*, page 2
OBD-II (On-Board Diagnostics version 2), *An output plug in modern cars providing information from the internal network of the car*, page 8
Pixel (-), *Unicolor segment of an image*, page 3
ROI (Region Of Interest), *The part of an image expected to contain all the wanted information*, page 3

Contents

1 Introduction

In the latest years, the automotive industry has quickly developed towards safer vehicles. Active safety such as adaptive cruise control, automatic brake systems and assisted road tracking are existing techniques used by most manufacturers. In the future the vehicles are moving towards even more technology in the cars since more functionality and faster and better data processing is constantly being developed. Data for these functions are captured using different types of sensors in the car.

Most existing techniques are currently quite inaccessible and expensive which means that vital safety features is limited to high end platforms. There is therefore a demand for cheaper techniques with similar performance. As one of the leading consultant companies in Sweden, ÅF wants to see if it is possible. They also want the developed techniques to be deployed to an electrical test vehicle, Figure 1.1, to gain experience and knowledge of how the algorithms can be implemented.

Figure 1.1: *ÅF's electrical test vehicle.*

1.1 Purpose

The purpose of the thesis is to investigate the possibility to use simpler sensors like the camera and accelerometer in a cellphone as input to safety functions. The main points in this thesis can be summarizied as:

1. Can the simpler sensors in a cellphone be good enough to get reliable data for safety functions if the data is fused together?

2. Can the performance be improved by introduction of data from other sources, the OBD-II plug in the car for instance?

3. Is it possible to design efficient algorithms such that they can be computed on a single board computer such as a Raspberry Pi 2? The Raspberry Pi is a small low cost computer with 1 gb RAM and a 900 MHz quad core processor [2].

4. Will the developed functions perform on par with the state of the art implementations existing today when considering safety, stability and speed of the algorithm?

5. Can it be implemented on ÅF's electrical test vehicle?

1.2 Contributions

The contribution of this thesis is mainly to the field of active safety. More specifically, it demonstrates the steps needed to create a simple and efficient model for lane departure warning and obstacle warning. The model is implemented on a cheap single board computer, demonstrating that active safety does not need to be complex or use expensive hardware to be useful.

1.3 Limitations

This thesis work includes modeling lane and object detection as well as handling all the data and calculating the prediction. The model is implemented to work with a data stream that can be either replayed or a live feed. Data from the OBD-II of the car can also be handled by the model, as well as data from sensors in the cellphone, e.g. GPS or gyroscope. The work is focused primarily on handling data from near optimal conditions, not for instance smaller roads with no lane markings or snowy weather conditions. The model is developed using methods inspired by the AUTOSAR standard, and implemented such that additional functions easily can be developed and added in the model.

No cellphone app was developed, to collect the replay data. Instead existing apps were used to gather OBD-II data that was used as inputs to the model. This limited the project from any live usage of the model with OBD-II data. The model is developed to handle slight differences in the position of the mounted cellphone, and its camera. However, it should be close to center of the front window, as high as possible, and facing straight ahead. If the position deviates too much from this, the model will not perform as expected.

1.4 Function Definitions

The focus of this thesis is on active safety, and as such there is a wide range of functions that could be developed, the following were chosen.

A **Lane Detection** algorithm is developed, described in section 2.2. This function uses image analysis and GPS data to find the lane in which the vehicle is traveling by implementing a Hough transform in near field and a second degree polynomial fitting for the far field.

Lane Departure Warning, LDW, is a common feature in modern cars. With knowledge regarding where in the image the vehicle is traveling, it is also possible to detect when the vehicle is about to leave its lane. This functionality is developed in this thesis as well, described in section 3.5.

As a mean to help the driver **Object Detection** was developed, as described in section 2.3. It localizes objects ahead of the vehicle in the traveling lane. With information about the detected objects, different warnings or safety functions such as applying the brake or an audio alert can be developed as well.

To test the mobility and agility of the complete model it is ported to a **Raspberry Pi 2** and run in full scale using a Raspberry camera as input.

1.5 Related Work

In the field of active safety, a lot of work has been done already. Many different safety functions are in use today, as illustrated by Volkswagen's description of their proximity sensing [3] or Volvo's new *IntelliSafe* [4]. Academic articles often focus on one safety function however, and for that reason each function developed in this thesis has a dedicated section in the theory chapter describing work within respective field. Models that can detect the road are described in section 2.2.1 and object detection functions are described in 2.3.1.

1.6 Reading Guidance

This report is divided into three major chapters; Theory, Chapter 2, Method, Chapter 3, and Results and Discussion, Chapter 4.

In the theory chapter, important underlying theory is explained in order to give the reader enough knowledge to completely comprehend this thesis work.

Following the theory chapter, the method chapter thoroughly explains the process used in this work. This chapter describes in detail the implementation of our model and validation of its functionality.

The Results and Discussion chapter feature the obtained results along with our comments and reflections regarding these results.

The final chapter of this report presents the conclusions drawn from this work, and a few suggestions for future work.

2 Theory

In this chapter a thorough explanation of theory needed to understand this thesis will be presented. To start of, some basic digital image processing concepts vital to this work is described. Anyone with experience in image processing can most likely skip this section. After that more purpose specific theory such as lane and object detection are treated. The final section of this chapter discusses applicable safety standards and how they were implemented in the work.

2.1 Image processing Basics

In this section some basic image processing concepts are presented, described in much more detail in [5].

2.1.1 Pixels and Colors

A video is a constant stream of frames, in this thesis 30 frames per second, fps, unless otherwise specified. Each of these frames is an image, consisting of different small parts called pixels. Each pixel represents a small part of the image with a solid color. To extract this information it is common standard to divide the information into three color levels: red, green and blue. For each color, the amount of that color in the pixel is described as a number. This means in practice that each frame can be represented by a $[m, n, 3]$ matrix, where m is the width and n is the height of the image in pixels, and each of the layers in the third dimension represents the colors.

2.1.2 Filtering

In order to manipulate the image matrix described above, different filters can be applied. For instance, one can soften the image by averaging pixel values in the same neighbourhood.

Mathematically, a filter can be represented as a matrix. This matrix is then applied to the image matrix either using correlation or convolution. When correlating the filter, each pixel value is obtained by adding or subtracting neighbouring pixel values according to the matrix values. As an example, the matrix

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

would mean that each pixel value would equal the original pixel value plus the value of the pixel to the right subtracted by the pixel to the left.

Convolution on the other hand can most easily be described as a matrix multiplication in Fourier space, and so requires the image and filter matrices to be transformed to Fourier space first, and then the resulting image to be transformed back.

2.1.3 Sobel Filtering

The Sobel filter is commonly used within image processing for edge detection, as the gradient of the pixel intensity tends to be high on sharp edges. It is also fairly robust to noise, as it uses a weighted average over 3 rows/columns.

The standard Sobel filter can be described as a derivation of the image along a certain axis. The filter has the mathematical form

$$\mathbf{G}_x = \pm \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \mathbf{G}_y = \pm \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{2.1}$$

Note the $\pm$, which is only important if one cares about in what direction the pixel intensity is changing, rather than the magnitude of the gradient. The magnitude $|I'|$ and direction $\angle I'_{xy}$ of the total gradient can be

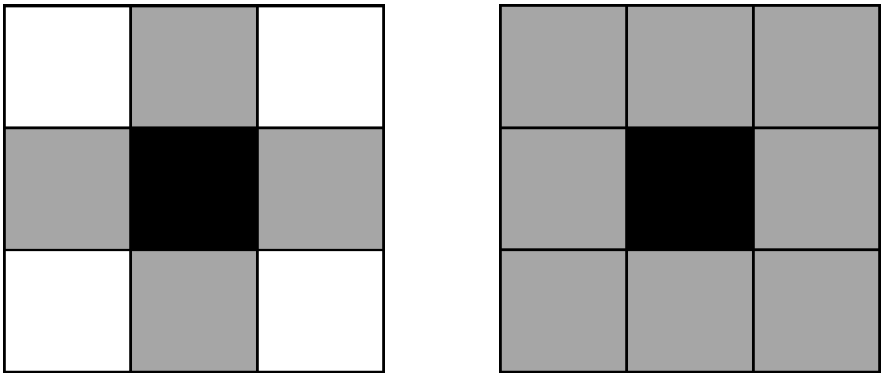


Figure 2.1: *The black square represents the current pixel, and gray are the potential neighbours. To the left 4 neighbours are considered, and to the right 8.*

calculated as

$$\begin{aligned} |I'| &= \sqrt{{I'_x}^2 + {I'_y}^2} \\ \angle I'_{xy} &= \arctan2\left(\frac{I'_y}{I'_x}\right) \end{aligned} \tag{2.2}$$

where I'_x and I'_y represents the image convoluted with G_x and G_y respectively.

2.1.4 Valid Numbers and Padding

For any filter larger than 1x1, caution is needed on and close to the edges of the image. Correlation requires specified values to be assigned to pixels "outside" the image, in order to calculate the result near the edge of the image. Another option is to simply ignore the edges and just use valid numbers, which instead will lead to a smaller resulting image. What option is used depends on the desired form of the result.

2.1.5 Otsu's Method for Autotresholding

Thresholding is a method to go from a greyscale image to a binary image by replacing each pixel with a value below a threshold with a black pixel and the ones above with a white pixel. To decide what the threshold should be, different methods can be used. In SIMULINK there is an *Autothreshold*-block shipped with the *Computer vision toolbox* that uses Otsu's method, therefore this functionality was not developed again for this thesis.

Otsu's method is a method for automatically choosing a threshold for any given image, by minimizing the weighted intra-class variance [6],

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \tag{2.3}$$

where weights ω_1 and ω_2 are the probabilities of each grouping of the pixels.

2.1.6 Connected Component Labeling

In image processing a common task is to group foreground pixels belonging to the same object or cluster [7]. Two pixels are classified as part of the same object if there is a path connecting the two pixels with only foreground pixels [8]. Usually either 4 or 8 neighbours are considered, which can be described as allowing diagonal paths or not. An illustration of 4 and 8 neighbours are shown in Figure 2.1.

A more strict mathematical definition for connectivity would be as follows. Assume P and Q are two pixels in an image. P is considered connected to Q if there exists a set of points $P = P_0$, $P_1, P_2, ..., P_n = Q$ where P_i is a neighbor of $P_{i-1}\, \forall\, 1 \leq i \leq n$ [9].

Connected component labeling is the process of labeling each set of connected pixels in an image with a specific label. The result of this will be that each connected component, i.e. set of connected pixels, has a unique label [8] and can hence be easily identified in the image.

A multitude of different algorithms to label all connected components in an image exist. Many use a recursive algorithm such as the one defined by Stockman et al in [8]. Another possibility is to use a Union-Find method based on a tree structure as described in [8]. The most straightforward way to implement the labeling

however is in a sequential manner, as proposed by Rosenfeld and Pfaltz as early as 1966 [9]. Specifics about the sequential algorithm used in this thesis can be found in section 3.6.1.

2.2 Lane Detection

The feature that was designed first in the developed algorithm was the lane detection. The main reason that the lane detection is needed is to find a safety zone in front of the vehicle, but it can also be used to estimate the travel path of the moving vehicle or to implement an LDW. Lane detection can be implemented in many different ways, some of which are described in Section 2.2.1. The Hough transform that has been chosen for this project, are described further down in Section 2.2.2.

2.2.1 Related Work

A lot of different techniques have been used to detect road lanes in the last 20 years and it is hard to categorize the different techniques because they are influenced by each other and many times combinations of several techniques have been used. Here follows a brief introduction to some of the most used techniques.

One method is to use a top-view image [10] called an Inverse Perspective Mapping, IPM, that is computed from the image acquired with the camera in the car. An IPM utilizes the position and angle of the camera as well as the focal length of the lens in order to transform the image from image world to real world. The distance measures and image quality are highly dependent on accurate data regarding the position. The benefits of this method is that it is possible to get all distances and shapes in real-world units, such as the width of the lane in meters and similar.

Using deformable models is another well-used technique for lane detection [11–15], where the detected lane points are fitted to a mathematical model. The models can use straight lines, parabolic curves or different types of splines to represent the lane, all with different pros and cons. These models tend to be slower and more complex, which is a drawback for this application. However it has the potential to very accurately describe curvatures and other non-straight roads.

A color-based method is used in [16] to find the lanes, which is fast and efficient but have some problems when lights are reflected in the lane, or shadows cover part of the image. In [17] an Edge Distribution Function, EDF, is used to estimate the lane direction and determine if the car is leaving the lane. The EDF is a one dimensional function that estimates the angle of the lane boundaries by voting in the image. This method is based on a linear model computed by Hough transforms, which comes with inherent problems in curves, as it tracks straight lines only.

Some methods [18, 19] are using a hypothesis of the lane that is then controlled. The two referenced articles use different techniques for the hypothesis model, a Neural Network [18] and Haar-like features filter[19]. To achieve good results with these methods a large database of features is needed along with a longer training period in the case of a neural network.

Most implementations utilize a Region Of Interest, ROI, to speed up the calculations. In essence, this means that one limits the part of image to be analyzed to avoid processing the entire image. A more extreme way of doing this is to define a Lane Boundary Region Of Interest, LBROI. Here, the processed part of the image is narrowed down to a small region close to where the lane boundary is expected to be found, as suggested by [15].

2.2.2 Hough Transform

The Hough transform can be used as a method to detect lines in a picture, and is described in much more detail in [5]. Here we give a short summary of the method.

A line can be mathematically described as

$$\rho = x \sin\theta + y \cos\theta \tag{2.4}$$

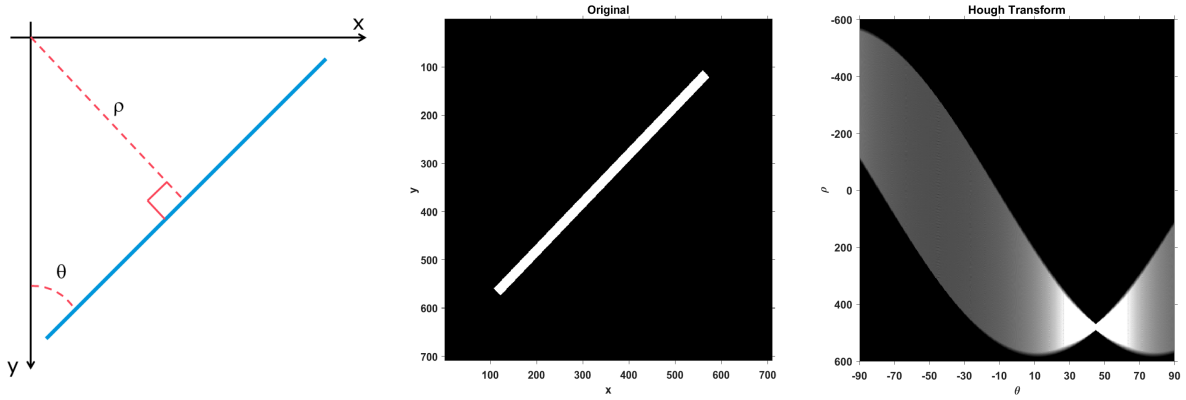
where (x, y) represent a pixel position in the image, θ is the angle of the line and ρ the distance to the origin. Further, there is an infinite number of lines that passes through each point (x, y) as ρ varies from 1 to the diagonal of the image and θ can take on an infinite number of values in the range $[-90°, 90°]$, see Figure 2.2a. In order to achieve decent computational time, ρ and θ are discretized when using the Hough transform.

The output of the Hough transform is a 2D-matrix A. $A(i, j)$ is calculated by summing the number of values along the line represented by $\theta(i)$, $\rho(j)$. Instead of the image one usually uses a binary input, so that

every foreground pixel has the value 1 and every background pixel 0. This reduces computational time, and can be used to study certain features of an image. As an example, if the binary image is obtained by thresholding the Sobel gradient magnitude image, only edges will be used in the Hough transform.

The Hough transform can as described here be used to find lines in an image, by identifying peaks in the Hough transform matrix. The peak value equals the number of active pixels along the represented line, and so a straight line will get a higher than average value.

The result of a Hough transform can be seen in Figure 2.2c, where a simple line has been run through the transform. From the figure, we can see that lines with $\rho \approx 500$ and $\theta \approx 45$ is represented with the highest intensity, which corresponds well with the actual line.



(a) *Illustration of how ρ and θ are represented in the image plane.*

(b) *A sample input for the hough transform.*

(c) *The resulting matrix of a Hough transform when using* 2.2b *as input.*

Figure 2.2: *Descriptive images of the Hough transform process*

2.3 Object Detection

When the current lane has been detected, the next safety feature to be implemented is an object detection. If successful, warnings or instructions to the driver, whether virtual or human, can be sent to avoid colliding with objects determined to be potentially dangerous.

2.3.1 Related Work

A lot of work regarding object detection has been done in the last decades. Commonly, a stereo camera is used in order to obtain a depth in the image [20]. Some implementations prefer a single camera coupled with a radar or sonar system in order to obtain this depth [21].

In the case of our system, a single camera is used for obstacle detection. In [22, 23] a mono-camera setup is used to track objects. Both articles focus on reversing a vehicle, and as such they are operating at low speeds, $\lesssim 20\,\text{km/h}$. In our model, speeds up to $120\,\text{km/h}$ are considered, hence these models at the very least need to be modified.

A common way single camera setups determine distance to objects is through an inverse perspective mapping, as described in section 2.2.1. Due to the nature of this transform however, it is most accurate for short distances, far away the image is to low in resolution to be transformed using an IPM.

One way to find objects in an image is to use image differentials. This method is based on the fact that everything moving at a different speed than the vehicle will be in different positions in two consecutive images. If one subtract consecutive images everything that is left will be objects moving relative to the vehicle velocity. The downside of this method is that there will be a lot of noise. Objects such as lane markers, safety barriers and everything else rooted will be moving compared to the vehicle. As an extension one can use the known vehicle motion to calculate what should happen to the image, using so called optical flow [24–26]. Comparing this to the actual new image will show everything that is moving compared to the vehicle, but is not standing still. Unfortunately elements such as focal length, angle and position of the camera will again be vital data to achieve stable results. Accurate data regarding velocity and acceleration of the car is equally important.

2.4 Safety Standards

When discussing safety an important aspect to adress is the meaning of safe. Some things are obvious, such as stopping before hitting an object. Others are not so obvious, such as recommended distance to keep when driving on the highway. In this section standard values for these cases are presented, and where applicable equations to describe the fundamental reasoning behind these standards are described.

In many cases, values have been determined by the writers using *"common sense"*, as no strict recommendations or laws was found to support the claims.

2.4.1 Working Frames per Second

To make an informed decision regarding how many frames per second the model must treat to obtain satisfying results, one should consider how far the vehicle travels each frame. This can be determined depending on the vehicle speed v as in equation (2.5).

$$\Delta d = \frac{v}{\text{fps}} \tag{2.5}$$

where Δd is the traveled distance. When the car is traveling in $120\,\text{km/h} = 33.33\,\text{m/s}$ we get

$$\Delta d = \frac{33.33}{30} \approx 1.11\,\text{m}$$

if the camera captures 30 frames each second. This is the maximum speed considered in this thesis, as no Swedish highways allow traveling faster. Since equation (2.5) is just a linear correlation, it is easy to see that only using every second frame would mean 2.22 m instead and etc.

Based on observations every fifth frame, or every 5.56 m, is considered enough for the lane detection. On the other hand, for obstacle detection every millisecond earlier that the object can be detected is important. Therefore, this should be done as often as possible without loosing the realtime requirements, i.e. the model must have sufficient time to finish one frame before it must start on next frame.

2.4.2 Definition of Safety Zone

In Sweden a common rule of thumb for distance keeping [27] is that one should keep the same distance in meters as the driving speed in km/h. So when driving 90 km/h one should keep a minimum distance of 90 m to other cars in the same lane. Unfortunately with the angle the video is captured, it is very hard to separate 50 m from 100 m, see Figure 2.3. In this image an attempt to count the number of dashes in the dashed lane marker, could be made to determine how far away the car in front is. However, one would soon realize that only a few pixels difference on where objects far away is could translate to an extreme distance. For this reason it was determined in this thesis to always look for objects all the way to the horizon.

Figure 2.3: *Example images depicting how hard it might be to determine distances in a frame from the video captured. In the right image, the black car (in the same lane) and the white car (in the left lane) are almost next to each other but in reality the white car already passed the black. But how far ahead of the black is the white?*

3 Method

In this chapter the methodology of this thesis work will be described in detail. Throughout the thesis work, MATLAB 2015a has been used, along with corresponding versions of SIMULINK and toolboxes, mainly the *Computer Vision System* toolbox and the *Raspberry Pi support package*.

In general, parameters and thresholds have been determined by visual inspection of the results while varying the values unless otherwise specified. All used values for thresholds can be found in Appendix B.

3.1 Gathering Data

Data needed to develop and evaluate the model was gathered using the android app *Torque*, with the plugin *Track recorder* to enable saving video together with data. The cellphone was also coupled with an OBD-II device via bluetooth, making it possible to save data from the car's built in systems. While driving, the app was set to save video and data from the phone sensors and OBD-II plug each 0.1 seconds. Note that GPS data in the cellphone is not updated as often. The video was recorded at 1080p and a frame rate of 30 frames per second.

The cellphone was mounted to the front window using an ordinary smartphone car mount, see Figure 3.1. For this specific project a model produced by RAM MOUNTS was used. To get a video that could be used in the model, it was important to place the cellphone in a proper way. The developed algorithms can not handle too much of the hood nor no hood at all obstructing the frames. Therefore the camera had to be directed so that more then half of the image was of the road. It was also important to center the camera in the car to get the lane departure warning to work in a desirable way. During the project the data gathering was done with a Volvo V60 but as long the smartphone is placed to fulfill the criteria above the type of car will not impact the results of the model.

Figure 3.1: *The phone used to gather data mounted in the front window of the car.*

3.2 Model Outline

The different functions of the model are implemented as separate algorithms to make the model both easy to extend and to understand. An outline of the model implementation can be seen in Figure 3.2. Before the actual model is run, a startup mode is engaged. When in startup, the model automatically finds the hood of the car, and adjusts what section of the video that should be searched for lines accordingly. The hood is found by looking for horizontal lines in the hough transform.

To reach the goal of an agile and easily adjustable model, the inputs defined in the flow chart in Figure 3.2, can be used with saved data or connected to a live feed. Similarly, the output could save to a file or show a video of the results during execution, depending on available hardware.

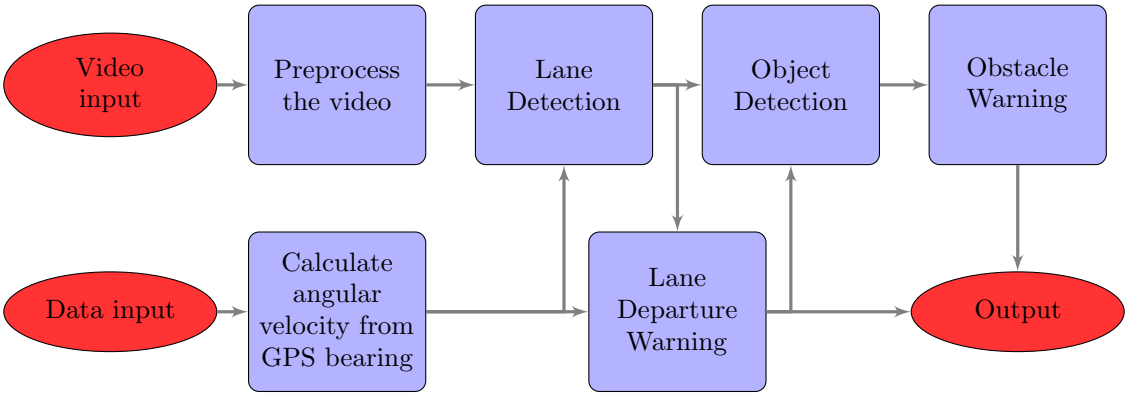


Figure 3.2: *Flowchart of the model outline. Blue boxes represent different algorithms implemented and red ellipses inputs and outputs.*

3.3 Preprocess Video

The first module in the main model is a preprocess where the video is processed to speed up the computations in the algorithms. The video was converted to 640x360 pixels in size to reduce the computational domain. If the video used as input is in color it is also converted to a gray-scale.

Within this resized video a section is chosen where all relevant data is expected to be found, selected by the startup algorithms. This section is then filtered with a Sobel filter to get the amplitude of the gradients, which is then thresholded. The threshold is set using Otsu's method, minimizing equation (2.3), to choose threshold for each image. The resulting binary edge image can be seen in Figure 3.3.

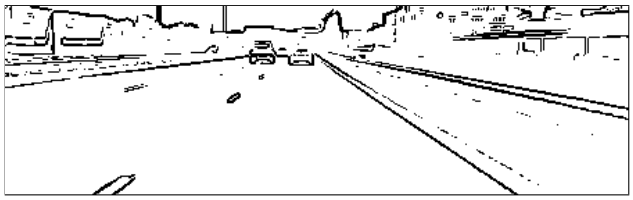


Figure 3.3: *An example of the preprocessed video. Black represents a foreground pixel while white pixels are assumed to be background.*

3.4 Lane Detection

In this section the methods used to locate a lane is described. The ultimate goal of this is to generate a well-defined zone in which obstacles, other cars and similar are considered dangerous or potentially dangerous. The results can also be used for lane departure detection or vehicle travel prediction. The lane detection is divided into two regions, near field and far field as seen in Figure 3.4, for which different methods have been used to find the lanes, similar to [15]. In the figure it can also be seen that instead of a normal coordinate system the x-axis is located in the center of the image where near field and far field are separated, and the y-axis is pointing downwards.

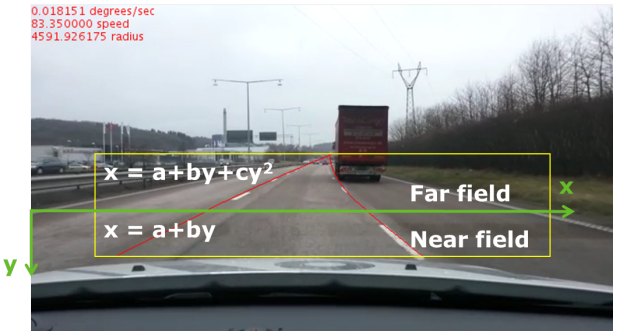
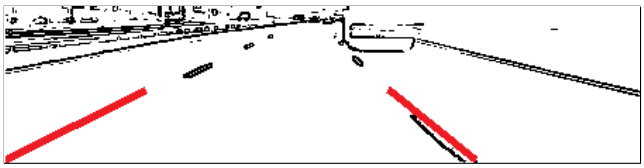


Figure 3.4: *The regions for lane detection.*

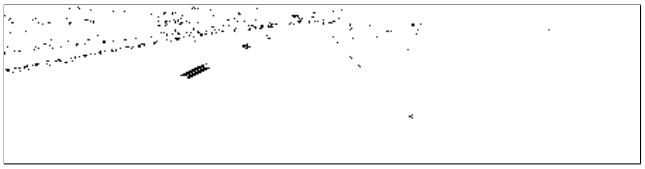
3.4.1 Near Field

In the near field region the lanes are assumed to be straight lines and the lower part of the binary edge image from the preprocess is used as input to a Hough transform as seen in Figure 3.5a. Since the value in the Hough transform matrix represents the length of the line, a comparison with a threshold is done and only lines over a certain length is kept. From the thresholded Hough transform matrix the strongest lines are extracted and assumed to be the lane markers closest to the car, which are marked in Figure 3.5b. If no line is found, the last line that was found is used instead.

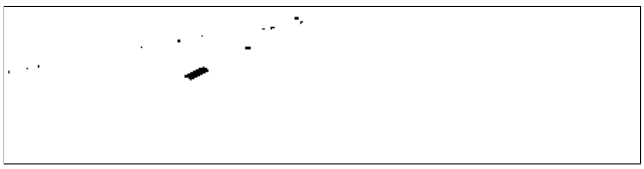
(a) *Binary edge image from preprocess.*



(b) *Frame with the chosen Hough Lines.*



(c) *Edge image where undesired edges for left lines are removed.*



(d) *Final edge image used to find curved lines.*

Figure 3.5: *Binary frames used in lane detection.*

3.4.2 Far Field

When the lines in the near field region have been calculated, the lines for the far field can be determined. To get better results the binary edge image is calculated separately for the left and right line. In Figure 3.5c it can be seen how undesired edges of the left line are removed based on their gradient angle, and in Figure 3.5d a median-filter has also been applied to remove some noise. Those new edge images are then used for the far field. Since the lanes can be curved in this part of the image, a second degree curve is used to describe the lane instead of the straight lines. In the given coordinate system the lanes are given by the following equations

$$\begin{aligned} x &= a + by, & y &> 0 \\ x &= a + by + cy^2, & y &\leq 0 \end{aligned} \tag{3.1}$$

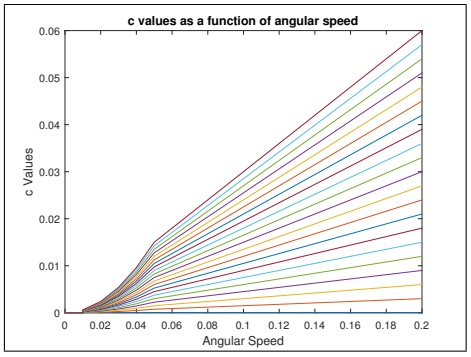
From the near field calculations the coefficients a and b are determined, which leaves the constant c, the curvature constant. To determine c a linear optimization is used, where the x-values for all y-values in the far field are calculated for a number of c-values. The calculated points are then compared to the filtered image described above. Then the c-value with highest number of matching points is chosen. To make the algorithm more robust, points outside the LBROI was not counted. The LBROI was determine as 7 px in each x-direction from the last calculated line.

Angular Speed and Curvature Constant

To decrease computational time the considered c values are limited, and based on data from the GPS bearing. An approximation of how much the car is currently turning is calculated by using a discrete derivative of the bearing data. The result gives numbers approximate in the range -0.2 to 0.2, where 0 represents a straight road, -0.2 a tight left turn and 0.2 a tight right turn. This angular velocity, ω, is then combined with a standard c vector, $c_{vec} = [0 : 0.015 : 0.3]$, according to equation (3.2). The less the road is turning, the smaller c values leading to less curvy lines.

$$c(\omega) = \begin{cases} \omega^2 \cdot \dfrac{c_{vec}}{\max(0.05, |\omega|)} \cdot \text{sign}(\omega) & \text{if } \omega \leq T_{curve} \\ 0 & \text{if } \omega < T_{curve} \end{cases} \tag{3.2}$$

where *sign* is the sign function, returning -1 or 1 depending on the sign of ω. T_{curve} is a threshold, by tests of the method found to work well with $T_{curve} = 0.01$. The result of this function can be seen in Figure 3.6a. An example of the lines that this c represents are shown in Figure 3.6b. The most curved line represents the largest c.



(a) *The c values used for far field curved line detection, based on the calculated angular speed. To stop the values from escalating for high ω the equation is linear for $\omega > 0.05$.*

(b) *Illustrations of the curvature constant's impact on the curved lines, here depicting all possible choices in a frame.*

Figure 3.6: *Curvature constant illustrations.*

3.5 Lane Departure

The implementation of lane departure warning in this thesis is based on the methods developed in [15, 17]. This method is based on the hypothesis that there should be a symmetry between the left and right lane boundaries in the near field of the image, when a car with a centered camera is positioned in the middle of a lane, see Figure 3.7. As the car moves towards one of the lane markers, this symmetry will change and the two angles of the left and right lines will no longer be equal in size. When the difference between the two angles are larger than a predefined threshold the LDW is activated. This can be summarized as in equation (3.3).

$$LDW = \begin{cases} \text{LDW left} & \text{if } \theta_L - \theta_R < -T_{LDW} \\ \text{LDW right} & \text{if } \theta_L - \theta_R > T_{LDW} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where θ_L is the angle of the left line, θ_R the angle of the right line and T_{LDW} the set threshold.

Since this method is based on the near field of the image, the lines from the Hough transform can be used. The nature of the Hough transform means that θ_L and θ_R are already calculated in another part of the model. This means that the only thing left to do is calculate the difference and trigger the LDW as in equation (3.3). An example of a triggered LDW can be seen in Figure 3.8.

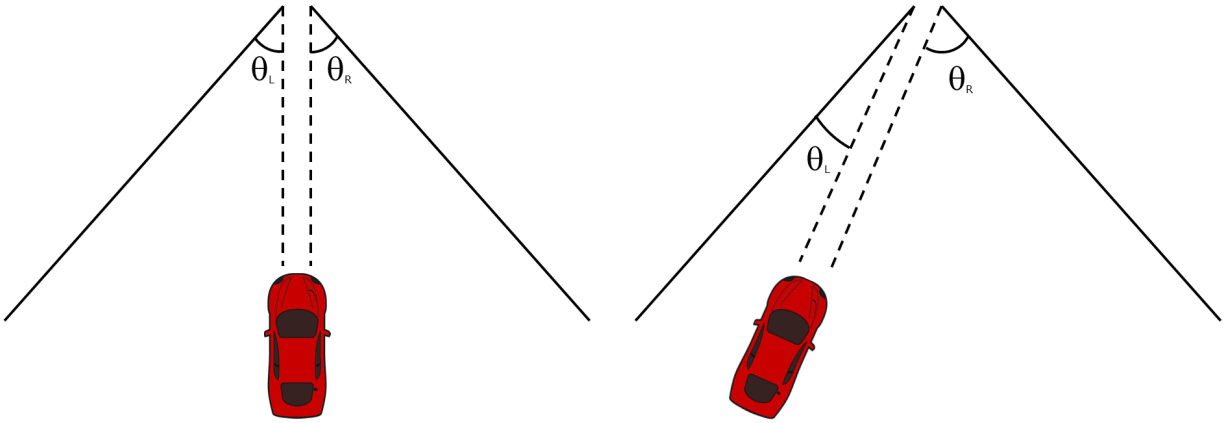


Figure 3.7: *Illustration of the symmetry between left and right lane boundary when the vehicle is placed in the middle of the lane to the left. On the right side, the symmetry is broken as the car is moving towards the left edge.*

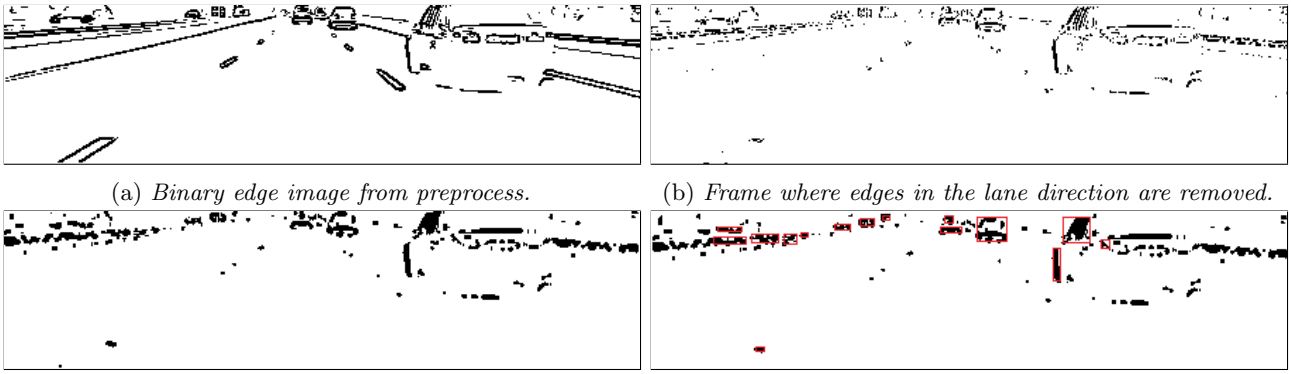
Figure 3.8: *An example depicting the activated LDW, here crossing the lane to the right. The blue circle in the top right corner is the LDW indicating a right turn.*

3.6 Obstacle Detection

This section will describe the algorithm that is used to detect obstacles in front of the vehicle. The method is divided into different parts and starts with filtering of the input image, where ground and lane edges are removed. After filtering, a blob detection algorithm is applied on the image. These processes are described in section 3.6.1. Then the detected blobs are merged together to larger areas as described in section 3.6.2. Finally, too small objects are ruled out as obstacle candidates before issuing a warning.

3.6.1 Filtering and Blob Detection

The obstacle detection starts with the same preprocessed edge image as the lane detection and can be seen in Figure 3.9a. The first step is then to remove edges that are in the same direction as the lane that already has been detected. This is done to avoid detecting the lane markings as obstacles and can be done because obstacles that rises above the road will have vertical edges that are not removed. The effect of this is shown in Figure 3.9b. To make the obstacles more distinct an averaging filter is applied on the new image, as shown in Figure 3.9c.



(a) *Binary edge image from preprocess.*

(b) *Frame where edges in the lane direction are removed.*

(c) *Final edge image used for object detection.*

(d) *Binary image where detected objects are marked with red squares.*

Figure 3.9: *Binary frames used in object detection*

When the filtering is done, the binary video is run through the developed connected-component labeling algorithm, which is used to detect a rectangle around each blob with an area larger than a threshold. For this thesis, 25 pixles were used. The algorithm uses a single pass to scan the image and label each connected region with the same label. To decide which region label to use, the labels of already visited neighbours of the current pixel are scanned. The one with the lowest value is assigned to the current pixel. If any neighbour has any other label value, all pixels with that label are replaced with the same label as the current pixel, since they are now connected to the same region. The algorithm is presented in Algorithm 1. To speed up the detection the region where the blobs are detected is determined by the safety zone. The final result is shown in Figure 3.9d.

```
Data: binaryVideo, leftLine, rightLine
Result: labels
currentLabel = 1;
videoData = binaryVideo(min(leftLine):max(rightLine));
for each row in videoData do
    for each col in videoData do
        if foreground pixel then
            testValues = videoData(row-1,col-1:col+1) ∪ videoData(row, col-1);
            labels(row,col) = min(testValues);
            if labels(row,col) equals 0 then
                labels(row,col) = currentLabel;
                currentLabel = currentLabel +1;
            end
            if testValues ∩ (0, labels(row,col)) not equals ∅ then
                labels(labels==testValues ∩ (0, labels(row,col))) = labels(row,col) ;
            end
        end
    end
end
```

Algorithm 1: The algorithm used to detect blobs in the video, so called connected-component labeling.

3.6.2 Merging Blobs

The detected blobs close to each other in the image are considered to most likely be the same object detected twice, and should be merged to one blob. An embedded MATLAB function was implemented to solve this. It works in iterations, and a maximum number of iterations can be set to keep computational time reasonable. Each iteration, the function calculates the distances between all blobs by using the center of the blob. It then finds all distances smaller than a set threshold, and starts merging one by one of the ones close to each other. Each iteration a specific blob can only be target of the merge once. If many blobs are located close to each

other, multiple iterations of the function is needed to merge them all. In the actual merge of two blobs, the new center of the blob is calculated as the average of the two merging blobs' centers. The new outer frame of the blob is the min and max corners from the two merged blobs.

After the blobs have been merged, a control of the size of the new blobs is included to remove some more noise. The obstacle close to the car will be much larger in the image then obstacles further away. The control equation (3.4) is used to determine which the minimum size A_{min} of an object should be to trigger a warning. The equation has been found efficient by trial and error.

$$A_{min} = (y - y_h)^3 \cdot 10^{-4} + (y - y_h)^2 \cdot 10^{-2} + 120; \tag{3.4}$$

where y is the pixel distance from the top of the image and y_h is the top of the zone, expected to be close to the horizon. A sample of this function can be seen in Figure 3.10. When the zone is changed, only the x axis will change. The shape of the function will be the same.

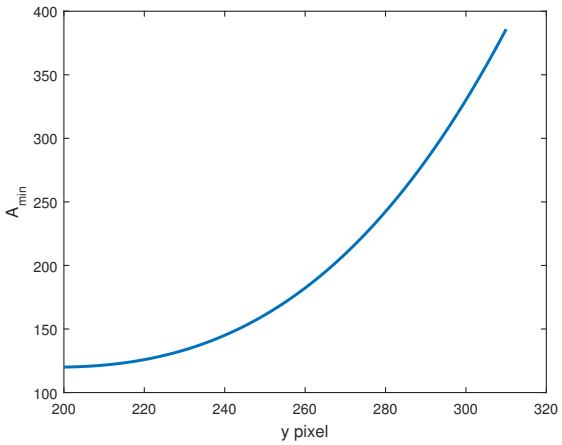


Figure 3.10: *Example of the function used to determine the size of an object.*

3.6.3 Line Classifiers

In order to avoid detecting lines as objects, an attempt at classifying lines was done. The classifier was based on the area of the video where the Hough lines are found. A vector with six values was stored, the max and min values of each color.

For each object the same values are calculated and compared to the classifier. If no value differentiates more than a threshold, the object is considered a line and disqualified from the obstacle warning process.

3.6.4 Obstacle Warning Algorithm Triggers

The obstacle warning can be triggered by a few different scenarios. First of all, and perhaps the most critical, the warning is triggered if an object is found too close to the hood. To control this, the algorithm determines which object is closest to the hood, and if it is closer than a certain threshold $T_{obsClose}$, it triggers the warning.

Further, if an object is getting closer to the hood, the warning is triggered as well. The implementation of this simply consists of tracking the object closest to the hood. If it moves more than a threshold $T_{obsFast}$ in one frame step, or more than another threshold $T_{obsSlow}$ in 3 of the last 5 frame steps, the warning is triggered. The threshold values working best in this application were selected as

$T_{obsClose}$	60 px
$T_{obsFast}$	10 px
$T_{obsSlow}$	2 px

When a warning is triggered it can produce a warning in two different ways depending on how the driver behaves. If the driver is already braking, a weaker warning will occur. If the driver is accelerating when the warning is triggered it will give a stronger warning. To check which warning that should be triggered the

acceleration data from the GPS is used. As can be seen in Figure 3.11, the accelerometer does peak when the driver is brakeing.

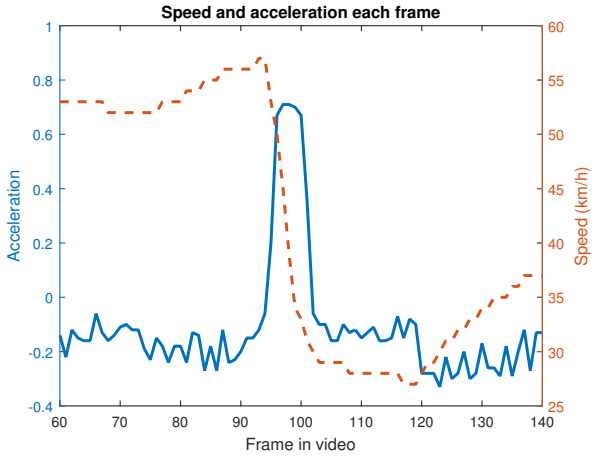


Figure 3.11: *Example of how the accelerometer reacts when a brake occurs.*

3.7 Raspberry Pi 2

In this section the specifics regarding porting to a single board computer are explained. For this thesis a Raspberry Pi 2 was chosen. For the technical data of the Raspberry Pi 2 refer to appendix A.1. An image of a Raspberry can be seen in Figure 3.12.

Figure 3.12: *The Raspberry Pi 2*

3.7.1 Generating Code

Porting the code to a Raspberry Pi 2 was done in SIMULINK by building to a real time target. *Simulink support package for Raspberry Pi* can automatically generate code in the C language from a SIMULINK model that can be built and run on the single board computer. The support package also contains predefined methods for handling the standard onboard camera and setting the General Purpose Input/Output, GPIO pins of the Raspberry.

3.7.2 Inputs

The camera connected to the Raspberry takes images at a maximum of 10 fps, using a format of 4x3, as opposed to the cellphone camera which uses a 16x9 format at 30 fps. To handle this, the preprossesing was changed in the Pi version such that the video was cropped to a correct format. Parameters such as the number of frames to average over must also change to correspond to the different fps, as well as how often the model is run. Unfortunately no GPS or OBD-II data was used with the Raspberry, as it does not natively support Bluetooth nor have any built in gyroscope or GPS.

3.7.3 Outputs

Since the Raspberry does not have any screen or monitor connected, it is not as straightforward to validate the results. One option when replaying already collected data is to save the position of the calculated objects and lane lines in a text file, which can be analyzed after the run. When using a live feed however, it is not possible to save the video to file as well, since this would very quickly fill the memory card used with the Pi.

The solution to the live feed output dilemma was to connect a few LED lights to the GPIO of the board, and turn on different lights in different situations. For instance, a red light is lit when there is an object detected in the safety zone. Two blue lights are used to signal when detecting a lane departure to the left or right respectively.

3.8 Validation of Results

3.8.1 Test Videos and Gold Standard

To generate quantitative results regarding the accuracy of the near field lane detection, i.e. the Hough lines, a set of lines picked by humans were collected, henceforth referred to as the gold standard[1]. Each evaluator was told to mark the straight lines corresponding to the current lane boundary for each frame in the video, using a small GUI created for the purpose, Figure 3.13. The average of all individual choices was then computed for each frame, and considered the gold standard. As a means to evaluate the gold standards validity, the standard deviation for each frame was computed as well.

Figure 3.13: *The GUI used by the testers to determine the lane by dragging the blue stars in the image.*

To evaluate the performance of our model two videos were used. The first one is a 30 second clip in heavy rain, with wipers on. We start on a larger road and then take an exit, under a bridge and into a fairly sharp turn. At one point the driving line divides into two lanes, one exiting the freeway and the other continuing. This video is referred to as *rainyDay*. The second video is in decent weather with overcast, driving on a large three lane road. In this video the vehicle changes lane three times. This video is referred to as *laneDep*.

3.8.2 Comparison Between Algorithm and Humans

To be able to evaluate the output data from the algorithms a validation script was created. In that script the output of the algorithm was compared to the gold standard mentioned above. The script reads two saved *.mat* files, both the gold standard for a specific video and the model generated data points. It then computes distance between lines as well as the difference in angles between the two versions.

[1]Gold standard is defined by the oxford dictionary as *"A thing of superior quality which serves as a point of reference against which other things of its type may be compared"* [28]

4 Results and Discussion

In this chapter all obtained results are presented and discussions regarding the results are included as well. The chapter is divided into sections starting with lane detection for near field and far field respectively. The next sections present the results from the obstacle detection and lane departure warning. The last section in the chapter is an evaluation of the algorithm speed.

4.1 Lane Detection - Near Field

To compare the model results with the golden standard, each endpoint of the two near field lines were compared to the corresponding points in the golden standard. The absolute distance between the model line and picked line were calculated for each frame. A low distance value can be interpreted as a successful line detection. The result can be seen in Figure 4.1.

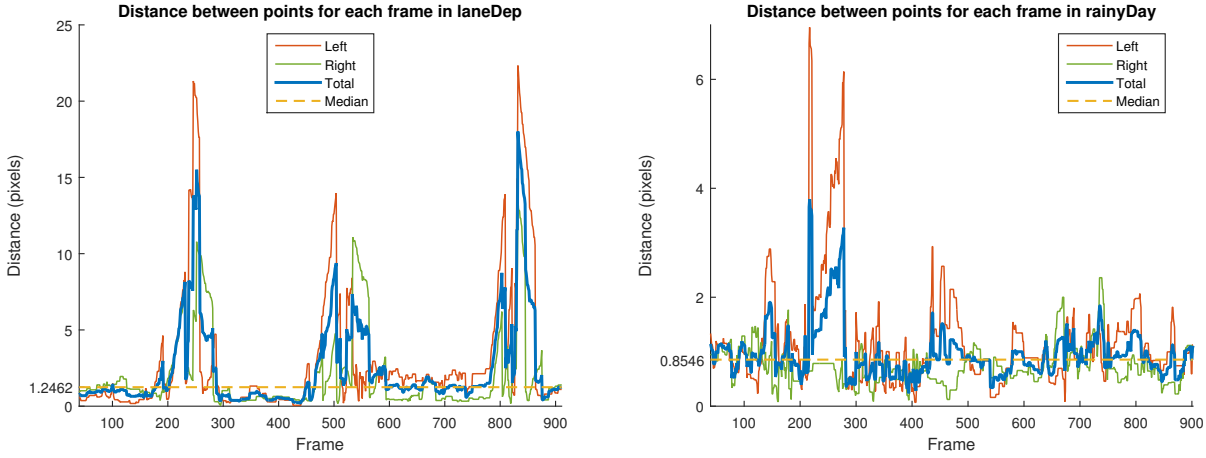


Figure 4.1: *The Distance between the gold standard and the line picked by the model. To the left, the result from video laneDep can be seen, and to the right rainyDay.*

The model line angles were compared to the gold standard as well, which can be seen in Figure 4.2.

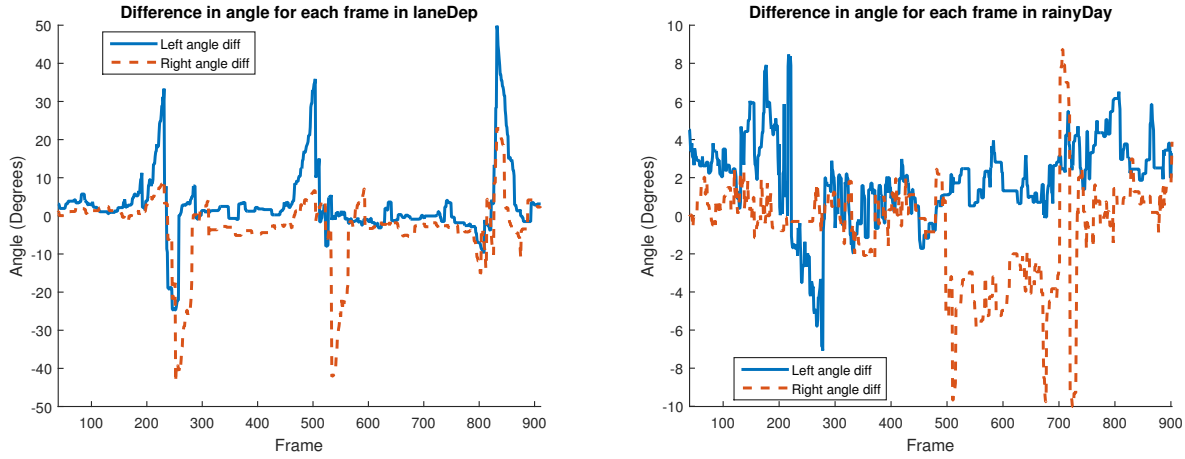


Figure 4.2: *The Distance between the gold standard and the line picked by the model. To the left, the result from video laneDep can be seen, and to the right rainyDay.*

Both Figure 4.1 and 4.2 show that the model performs very accurately for most frames. It is also very obvious from these graphs where in the video the lane changes are happening. In the *laneDep* video, the three peaks in the distance measure each represent a lane change. These three peaks corresponds well with the peaks in Figure 4.4, telling us that not only the developed model, but also humans, have trouble determining where the *"current lane"* is situated during these lane changes. The median in both *rainyDay* and *laneDep* are close to 1 pixel, which has to be considered very accurate.

In the right graph in Figure 4.1 we can see that the left line deviates between frame 200 and 300, while the right remains low. This is because in this part of the video, the current lane is split in two. The right line remains solid but the left line becomes two lines. Similar to a lane change this is hard to handle for both humans and the algorithm, as the same behavior is present in Figure 4.4 as well. The lane split is clearly visible in Figure 4.3.

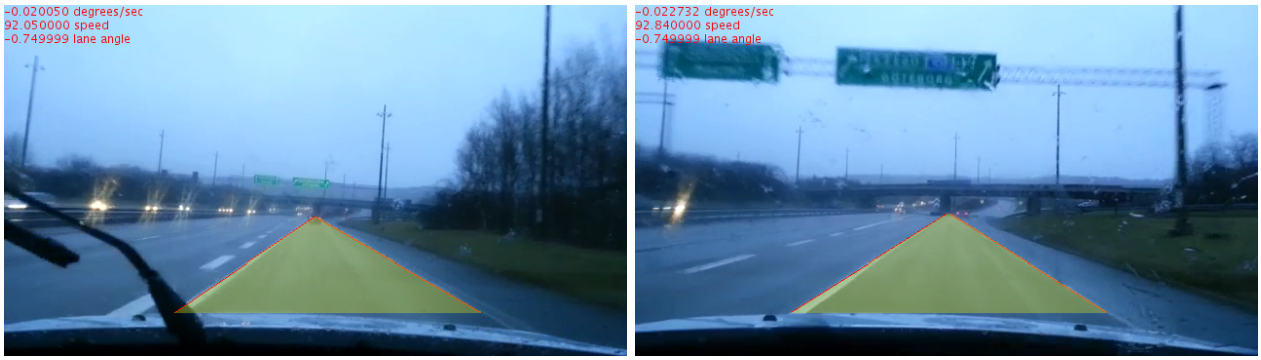


Figure 4.3: *Frame 210 and frame 288 of the rainyDay video used for evaluation. The splitting of the left side lane marker is clearly visible.*

4.1.1 Gold Standard

To justify the use of a gold standard the standard deviation of the picks the testers made in the two videos have been calculated and can be seen in Figure 4.4, illustrating in pixels how much the testers disagree. The figures are based on five testers' picks. For most part of the video the testers agree, and the standard deviation is only a few pixels. However, during lane changes the deviation increases, since it is hard to know what lines to choose as the current lane boundaries when the car is in between two lanes. Even for a human, it is hard to determine when the lane change actually occurs, which makes it almost impossible for an algorithm. The algorithm can be run with a specified rule determining when to classify it as a lane change, but odds are that not all drivers will agree with that rule.

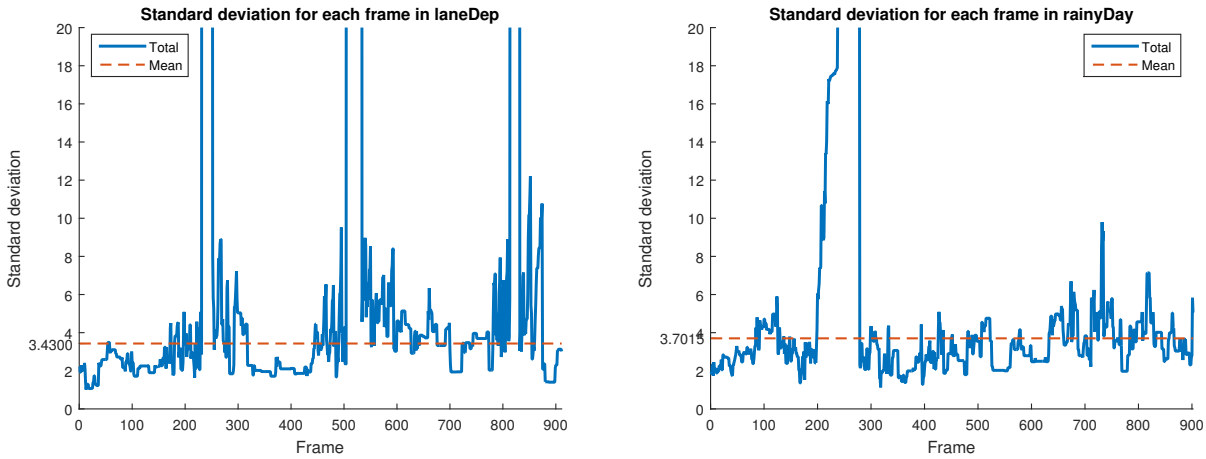


Figure 4.4: *The standard deviation over each frame based on each test persons' choice.*

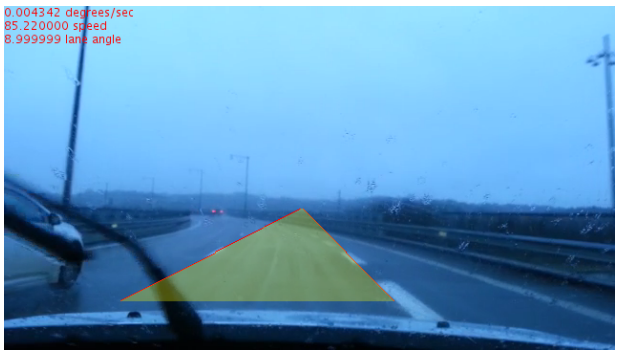
4.2 Lane Detection - Far Field

No quantitative measure for the accuracy of the far field detection has been made. This since it would require a lot of extra points per video frame, which was deemed not possible within the time frame of this work.

By watching the output from the model one can however qualitatively evaluate the algorithm. The curve fitting is not very robust, and highly dependent on the result of the Sobel filtering. In most turns, the algorithm finds the correct turn for a few frames, as can be seen in Figure 4.6d, but also outputs curvatures that are completely wrong in other frames.

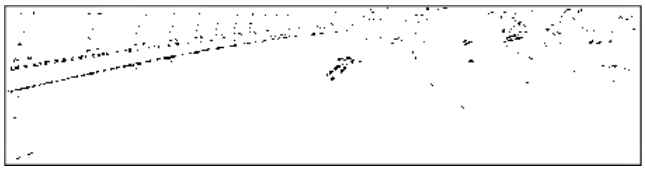
In order to make this part of the model faster and more robust, an approximation of the current angular velocity is calculated using the bearing data from the GPS of the phone. This does introduce a few issues however. One issue is that the update rate of the GPS is slower than the video fps, which means that the calculated angular velocity is in fact an averaged value. The main problem when using the bearing however is the natural delay that arise when it is used. The GPS is only updated ever so often and can not detect a turn until after it has started. In Figure 4.5a we can see that the road is clearly turning to the left, whilst the GPS data indicates a slight right turn. One approach to this problem could be to use the steering wheel angle. Unfortunately, that data is not available as an OBD-II signal, and therefore this thesis does not include any work using that.

In the far field issues arise where the side of the road contain lines in approximately the same direction as the lane marker. An example of this can be seen in Figure 4.5b, where the crash barrier next to the road is clearly visible next to the lane marker in the corresponding Sobel output, seen in Figure 4.5c.



(a) *Example frame where the road is turning but the GPS bearing derivative is claiming something else. Notice that the value described as degrees/sec is relatively small and positive (0.004), which indicates a slight turn to the right.*

(b) *Example frame where the detected lane is decent, but the crash barrier is impacting the results slightly.*



(c) *The binary filtered sobel video used in the calculations to generate* b. *To the left in this image, both the edge from the road lane marking and the crash barrier are clearly visible, which can be an issue for the algorithm.*

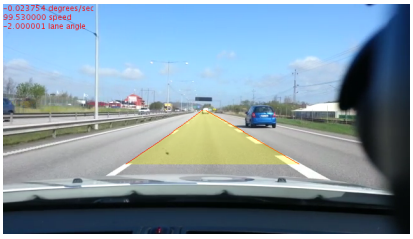
Figure 4.5: *Various results of the lane detection in the far field.*

In Figure 4.5c one can clearly see that the lines are very noisy close to the horizon, which also adds to the difficulty in finding the perfect curve fit. There are two main explanations to this. The first is the most obvious, the camera resolution does not allow for entire pixels to represent the line close to the horizon. The second is due to the filtering applied to the Sobel video. All gradients that are not close to the Hough lines' angles are removed, which will also remove part of the wanted edges in this case because of the relatively tight turn. As we can see in Figure 3.5d, the visible edges are more solid when driving on a straight road.

One general observation is that the curve fitting works slightly better on a clear day than a rainy day. This is not surprising, as the rainy day videos are a lot grayer, with less contrast and color variation. This in turn

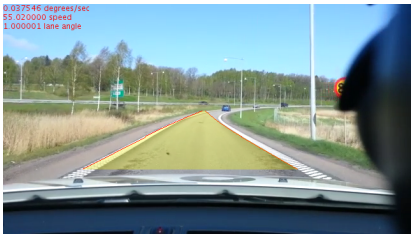
leads to less sharp edges resulting in a blurry Sobel video compared to a clear day.

More results of the curved line fitting can be seen in Figure 4.6.

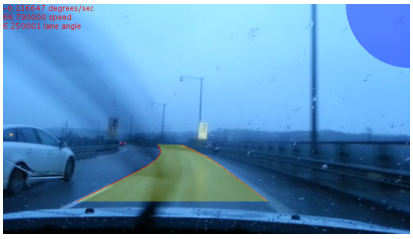


(a) *When driving on a straight road the far field is just the extension of the straight lines from near field detection, which is fairly robust.*

(b) *The right line does not trace the curve of the line, but the safety zone is decent anyway.*

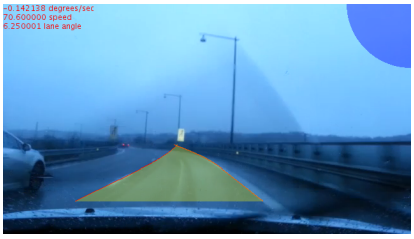


(c) *A curve farther ahead in the driving lane is not detected by the algorithm since the GPS has not reacted yet.*



(d) *Good result despite the rain.*

(e) *Similar to* b *but in rain.*

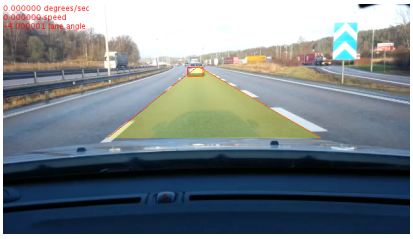


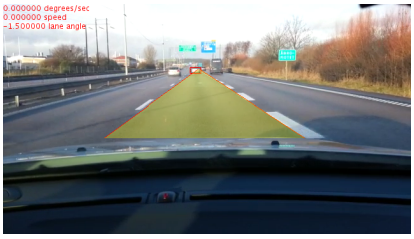
(f) *The algorithm is not detecting the left line in the turn perfectly.*

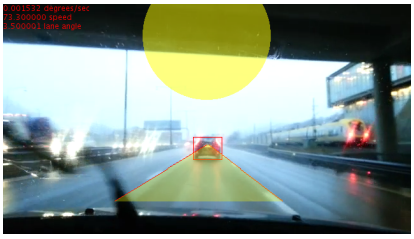
Figure 4.6: *More results from the lane detection in far field region. Notice the marker in the right corner indicating a false LDW triggered in* d *and* f.

4.3 Obstacle Detection

In general the obstacle detection algorithm developed works well when observed while testing. In Figure 4.7 a few examples where the object detection is detecting real objects are shown.







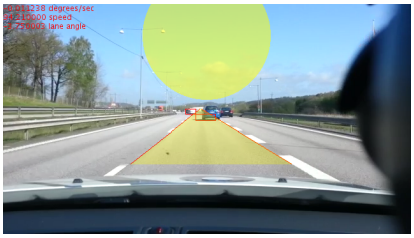


Figure 4.7: *Examples of where the object detection is working as expected. The big yellow circles in two of the images are the weaker version of the triggered obstacle warning.*

When running the model some issues are observed. The algorithm as it is developed is tuned to detect too much rather than too little, to avoid missing something. This means unfortunately that it also detects objects that should not be detected, which may lead to an unwanted triggering of the obstacle warning. Below a few different scenarios where the algorithm does not work as good are discussed.

No sideline

Both videos tested with the gold standard are on highways with lane markers visible. When the lane markers disappear, the model has nothing to track. An example when this happens is shown in Figure 4.8. As we can see in the right image in the figure, there are no edges appearing in the Sobel video meaning the algorithm can not find the lane. Because of this, there is a false obstacle warning triggered in the frame, since the algorithm is tracking something on the sidewalk.

Figure 4.8: *An example where there is no side line markers, and so the lane is not detected accurately. The big red circle in the top of the image is the obstacle warning that has been triggered.*

Shadows and Lane markers

In Figure 4.9 one can clearly see how shadows are detected by the object detection algorithm. The shadow of a car is included in the detected object, which leads to a larger object than expected. However, this is not a major issue since the car should be detected anyway.

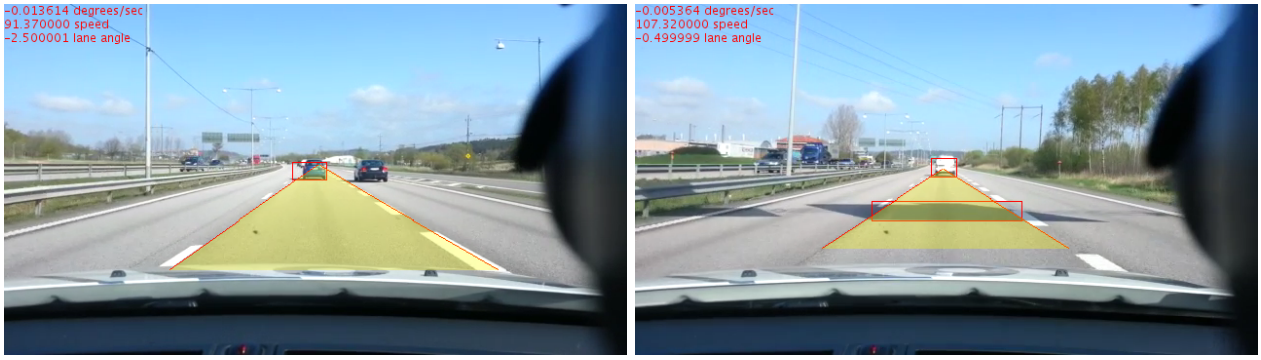


Figure 4.9: *Examples where shadows spoil the result of the object detection algorithm. To the left, the shadow of a car is included as part of the object. To the right, the shadow from a road sign hanging over the road is detected as an object.*

The other image in Figure 4.9 illustrates how shadows can be detected by themselves. This is a big problem for the developed algorithm as it will always lead to a false positive, and many times triggers the obstacle warning. It is however a hard task to get rid of these false positives without any kind of depth. Any radar, sonar or stereo camera setup could clearly detect that the shadow is flat on the ground and so is not a real obstacle. Our setup however has no means to do so.

A similar problem arise when driving over road markers on the road, as in Figure 4.10. The line classifiers can sometimes remove these objects because they have similar attributes in the color spectra as the lane markers on the side of the road. When they do not match it is usually because the asphalt is significantly lighter where the wheels of cars usually travel, compared to close to the side line markers where the asphalt is darker. If the classifier fails the lines will simply be detected, due once again to the lack of dept in the single image. Generally the line classifiers are observed to work worse on rainy days than clear days. Just like the curved line fitting, the line classifiers have less contrast to work with.

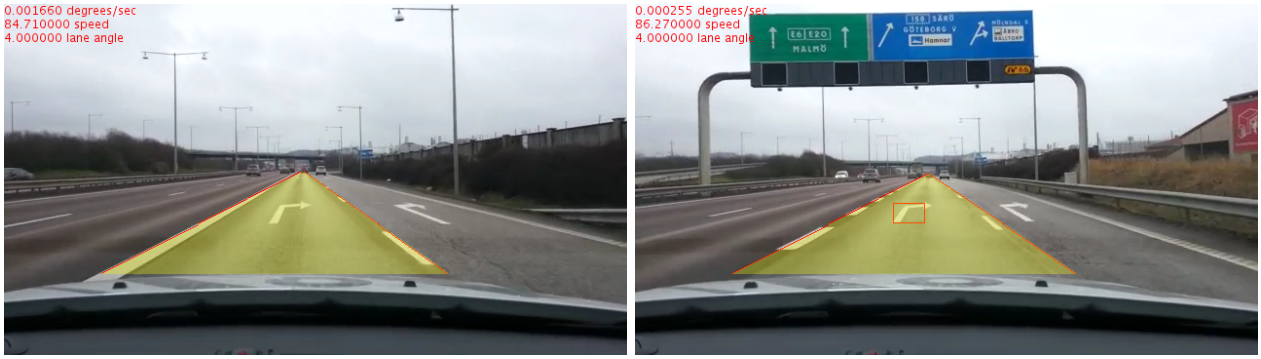


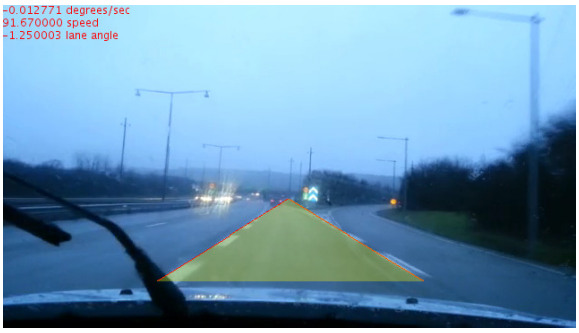
Figure 4.10: *Examples of arrows on the lane that are not detected and detected respectively.*

Permanent Objects

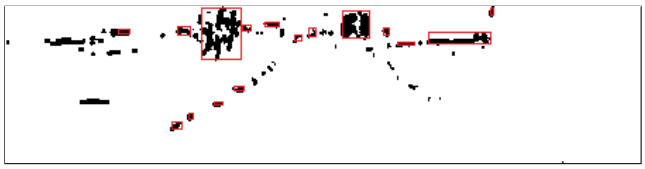
In most of the videos, and almost no matter how the phone is mounted, some part of the car hood is visible. This is preferred by the model, as the startup function looks for the hood when determining where in the image the lane zone is most likely placed. However, in some videos there are other objects present, which is not desired. For instance, in Figure 4.9, the mount holding the phone is clearly visible on the right side of the image. Because of the placement of the mount, it never triggers an obstacle warning, since it will never be inside the lane. In the same image there is also a black dot close to the left lane line because the window was not clean. This dot is in fact found by the algorithm each frame, but then ruled out because it is too small. The only way it could impact the result is if there is a real object close to that dot. Then the merge object part of the algorithm will merge them. In this case no false obstacle warning will be triggered, meaning that in worst case the object will be deemed larger than expected.

The wipers in figure 4.11 are not triggering any object either. This is perhaps a bit more surprising. First of all, the wipers are close to the camera, which means that they are not in focus. This combined with their movement results in blurry representations in the still images treated by the algorithm, as can be seen in Figure 4.11a. Although true in most cases, in Figure 4.11b the wiper is in its end position, where it is standing still for a moment resulting in less blurry edges. Looking at Figure 4.11c we can see that the wiper is not present, meaning the filters are actively removing it. Specifically, the filter specifying what direction the gradient should have for the algorithm to keep the edge as foreground is removing the wiper.

(a) *Example of the wiper in the middle of a rotation. The wiper is blurred because of its motion and therefore no edges are detected.*



(b) *Example of the wiper in its stop position where it is less blurry.*



(c) *Filtered Sobel video corresponding to* b. *As can be seen, the wiper is removed by the filtering process.*

Figure 4.11: *Evaluation of why the wiper blades are not detected by the algorithm.*

4.4 Lane Departure Warning

The Lane Departure Warning system constructed works fairly good. After watching videos with 30 real lane changes, the algorithm detects 19 (63 %) of those, while triggering 21 false positives. While this does not sound great it is actually not that bad compared to other sources. In [17] for instance Lee claim to be approximately 96 % accurate in detecting lane departures. To get this number however the Lee uses the number of frames as a measure, which will give completely different data to ours. In fact, Lee has more frames with a falsely triggered LDW than he has with a correctly triggered one, 42 false and 29 correct. This is comparable to our data, meaning that this model does in fact perform at least on par with Lee's.

Upon closer investigation of why the LDW is not perfect, a few things that would improve the performance is obvious. The algorithm is based on the angle of the lines, which is closely related to the position of the cellphone. If the phone is not pointing straight ahead, the normal straight case will have an angle difference between left and right line. If one knew the angle of the phone, the angle difference could easily be biased to neutralize this offset. Unfortunately the angle of the phone is not available to the model. In reality this generates a few false positives when driving closer to one side of the lane and having the camera positioned off and can lead to a few missed warnings where the LDW should have triggered.

The biggest issue with the LDW as it is designed here however is that the lane detection in close field is designed to find lane markers when the car is driving in the lane. As soon as the car starts to drift over a line, that algorithm is usually not as stable. This means that every now and then, a lane change is made but the angle never increase enough to trigger the LDW, because the lane markers start to drift over the new lane instead, illustrated in Figure 4.12.

Figure 4.12: *A frame where the lane markers have drifted to the right as the car is moving from the left to the right lane. The drift occurs because no new lines are detected and therefore the old ones are kept.*

4.5 Evaluation of Algorithm Speed

Since the algorithm is required to run in real time, the running speed is essential. The evaluation speed has been tested by timing the model when it runs a given number of frames, from which the fps was calculated. This test has been done for a full model with all safety functions included and one model where only the near field detection and LDW is included. The test has been done on both a computer and a Raspberry Pi 2 to compare the difference in performance. On the computer the test has also been done when the video input is in lower resolution from the beginning and no resize of the video input is done in the model. This is done because the image processing part of the algorithm is the most time consuming part. In Table 4.1 the runtime fps and the time to process a frame for the different tested models are presented.

Hardware	**Video Resolution**	**fps**		**seconds per frame**	
		Full Model	*Only LDW*	*Full Model*	*Only LDW*
Computer	720x1280	3.79	5.87	0.26	0.17
Computer	360x640	5.80	14.96	0.17	0.07
Raspberry Pi 2	480x640	4.16	8.66	0.24	0.12

Table 4.1: Table of the FPS and process time for a frame for the different models.

The results in Table 4.1 shows that the fps is influenced quite a lot by the resize function that convert the video from 720x1280 px to 360x640 px and reduce the fps. The fps alone may not explain very well the effect of the different versions of the model tested. But by instead studying the time it takes to process each frame, one get a better picture. To obtain the time each frame takes to process the fps is inverted. By comparing the rows in Table 4.1, we can see that reading the larger video file and then resizing it to the desired dimensions takes roughly 0.1 s. Comparing the LDW model to the full model, the same amount of time, 0.1 s, is saved for each frame when run on the computer. On the Raspberry Pi 2 however the difference is slightly larger, 0.12 s. This is most likely due to the different type of processor and memory used on the single board computer, but could also be affected by the inaccuracy in the benchmarking used.

The time measures of the Pi should not be compared directly with the times for the computer models. When run on the Pi a live camera is used as an input whilst the computer model reads the video from file. The latter requires time to get the data from the file, while the foremost needs timing with the camera and could also lead to buffering or other factors that might affect the processing speeds. In general it does appear to be slightly slower processing speeds on the Raspberry Pi 2.

In section 2.4.1 the needed frame rate is introduced to be around 6 fps to get a working lane departure warning in speeds up to 120 km/h. The working speed for full model working at the larger video input and the Raspberry Pi 2 is therefore to slow to be safe. On the other hand if the model with only LDW is used the Raspberry Pi 2 will be able to process the data in a safe way. When using a smaller video input the full model can be processed in 5.80 fps that is enough and the model with only LDW is even faster.

The Raspberry Pi 2 is too slow to run the full model at max vehicle speeds, but for lower speeds the frame rate will satisfy the requirements. Therefore the Raspberry Pi 2 could be used in applications for bikes or any slower moving vehicles, unless the model optimized further.

5 Conclusions and Future Work

This chapter contains conclusions drawn from this thesis work along with suggestions for future work.

5.1 Algorithms

For lane detection in near field the Hough lines are performing very accurately. Compared to the gold standard, the model is for most frames within the standard deviation of the gold standard values and comparable to a human. It is also fast enough to be deployed to a real time target though with a lower frame rate required.

The extension with curved lines, where a parabolic curve is fitted to maximize the number of covered foreground pixels, is not very accurate. It can sometimes find the correct line, but most of the time it does not. As for speed it is one of the slower parts of the model, and therefore for future work another method to detect curved lines should be implemented.

The connected-component labeling algorithm is theoretically perfect in accuracy, as it labels all objects. However, since it iterates over the entire image and sometimes manipulates large areas of the corresponding matrix multiple times each execution it is relatively slow. In the developed model roughly 30 % of the computational time is spent on this task alone. Since it labels everything, it is not a matter of missing objects but rather how one can filter out objects that are not interesting. For future improvements of this functionality investigating other methods of ignoring the lines and other objects on the ground plane are proposed. Alternatively using a stereo-camera or a radar system to create a real 3D image of the environment could be a solution.

The algorithm is robust to noise that is permanently present in the image as long as it is small. Further, wiper blades are not detected as obstacles by the algorithm in the general case. A large object on either side of the frame will not trigger the system either, as it is not inside the lane.

The object tracking used to trigger the obstacle warning does a decent job for its purpose. For a future model, a more complicated method where all objects are tracked should probably be implemented. Only keeping track of the object closest to the hood can lead to missing objects, e.g. objects that are closing in fast.

5.2 Model Extensions

Since the data can be gathered by a cellphone alone, and the power of a Raspberry Pi 2 is enough to run the model with only LDW, it should be possible to do that in an app for a phone. Such an app would have a great value for demonstrating some functions that can be developed in the field of active safety, especially for people who are not familiar with the field. It could also be used in literary any car, assuming one owns a holder for the phone. The only drawback might be the warning signals, which could be hard to interpret on a small screen. Perhaps sounds could be used as warnings instead.

The addition of a cheap radar or sonar sensor to the system could potentially contribute greatly. Pointing it to the sides one could easily find the edge of the road, greatly assisting the lane detection systems. Having access to steering wheel angle and turn signal would also be beneficial. The steering wheel angle could provide info for the curved road detection. Turn indicator signals could prevent unwanted LDWs, as no warning need to be issued in a planned lane change.

Another proposed extension of the model would be to also trace the lane next to the driving lane. That would require additional work, but could enable detecting cars turning into the current lane, bikes in the bike lane etc.

During the lane change the obstacle warning is turned off completely in this model, because the detected lane is not robust leading to a lot of false obstacle warnings. In future work a special method developed to handle the lane detection during lane change to ensure robust safety zone definitions should be developed.

5.3 Hardware Setup

The used setup in the thesis is working well, but to improve the model a few minor changes could be done to the hardware. Filming in too high quality is not necessary because it will not improve the result but are more time consuming. The processing power required to resize the video to a smaller format is extensive resulting in a time-consuming process. This could easily be avoided by setting the camera to the correct size when filming. To make the model run faster on a Raspberry Pi 2 it should also be developed for the Raspberry Camera's 4x3 resolution from the beginning, eliminating yet another image conversion.

If the model could be based on a known fixed camera position some of the inaccuracies could be removed. The angle difference in the LDW would never need a bias for instance, and the safety zone's y height in the video frames could be fixed as well.

In this work's setup the camera uses an autofocus, which means that sometimes the focus is not good. A camera with a fixed lens and focus point could probably generate sharper video in better quality using the same frame rate and resolution as the cellphone.

5.4 Raspberry Pi 2

The slightly modified model developed in the thesis can be run on the Raspberry Pi 2 without any complications. The computational power of the Pi does however limit the usability as the working fps is very low. From an active safety standpoint it is therefore concluded that the Raspberry Pi 2 along with this model does not meet the requirements. The Pi can be used to show simple functions that will work for easy demonstrations though. For instance, an LDW based on only Hough lines runs smoothly with more than 8 fps.

A model that works on the Raspberry Pi 2 could be developed by using the speed of the vehicle as an input, and then use the full model for low speeds and the only LDW model for high speeds. To get the speed from the car, a way to read the OBD-II signals to the Raspberry Pi 2 has to be developed. If such OBD-II reader is developed, it will also be possible to read more signals from the car that could help make the model more robust.

References

[1] AUTOSAR. *Welcome to the AUTOSAR development partnership*. 2015. URL: http://www.autosar.org/ (visited on 05/05/2015).

[2] R. P. FOUNDATION. *What is a Raspberry Pi*. 2015. URL: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/ (visited on 05/05/2015).

[3] Volkswagen. *Proximity Sensing*. 2015. URL: http://www.volkswagen.co.uk/technology/proximity-sensing (visited on 05/08/2015).

[4] V. C. Corporation. *IntelliSafe*. 2015. URL: http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe (visited on 05/08/2015).

[5] R. Gonzalez and R. Woods. *Digital Image Processing*. third. International edition. Pearson Education, 2010. ISBN: 9780133002324.

[6] N. Otsu. A Threshold Selection Method from Gray-Level Histograms. English. *IEEE transactions on systems, man, and cybernetics* **9**.1 (1979), 62–66.

[7] M. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. English. *Journal of the ACM (JACM)* **39**.2 (1992), 253–280.

[8] G. Stockman and L. Shapiro. *Computer vision*. first. Prentice Hall, 2001. Chap. 3, pp. 69–75. ISBN: 0130307963.

[9] A. Rosenfeld and J. Pfaltz. Sequential Operations in Digital Picture Processing. English. *Journal of the ACM (JACM)* **13**.4 (1966), 471–494.

[10] A. Broggi. A massively parallel approach to real-time vision-based road markings detection. English (1995), 84–89.

[11] D. J. Kang, J. W. Choi, and I. S. Kweon. "Finding and tracking road lanes using "line-snakes"". English. 1996, pp. 189–194. ISBN: 9780780336520; 0780336526.

[12] Y. Wang, D. Shen, and E. K. Teoh. Lane detection using spline model. English. *Pattern Recognition Letters* **21**.8 (2000), 677–689.

[13] Y. Wang, E. K. Teoh, and D. Shen. Lane detection and tracking using B-Snake. English. *Image and Vision Computing* **22**.4 (2004), 269–280.

[14] J. W. Park, J. W. Lee, and K. Y. Jhang. A lane-curve detection based on an LCF. English. *Pattern Recognition Letters* **24**.14 (2003), 2301–2313.

[15] C. R. Jung and C. R. Kelber. Lane following and lane departure using a linear-parabolic model. English. *Image and Vision Computing* **23**.13 (2005), 1192–1202.

[16] K.-Y. Chiu and S.-F. Lin. "Lane detection using color-based segmentation". English. Vol. 2005. 2005, pp. 706–711. ISBN: 0780389611; 9780780389618.

[17] J. W. Lee. A Machine Vision System for Lane-Departure Detection. English. *Computer Vision and Image Understanding* **86**.1 (2002), 52–78.

[18] Z. Kim. Robust Lane Detection and Tracking in Challenging Scenarios. *Intelligent Transportation Systems, IEEE Transactions on* **9**.1 (Mar. 2008), 16–26. ISSN: 1524-9050. DOI: 10.1109/TITS.2007.908582.

[19] H. Jung, J. Min, and J. Kim. "An efficient lane detection algorithm for lane departure detection". English. IEEE, 2013, pp. 976–981. ISBN: 1931-0587.

[20] M. Nishigaki et al. "Moving obstacle detection using cameras for driver assistance system". English. IEEE, 2010, pp. 805–812. ISBN: 1931-0587.

[21] M. Bertozzi et al. "Obstacle detection and classification fusing radar and vision". English. IEEE, 2008, pp. 608–613. ISBN: 1931-0587.

[22] Z. Yankun, C. Hong, and N. Weyrich. "A single camera based rear obstacle detection system". English. IEEE, 2011, pp. 485–490. ISBN: 1931-0587.

[23] J. Lalonde, R. Laganiere, and L. Martel. "Single-view obstacle detection for smart back-up camera systems". English. IEEE, 2012, pp. 1–8. ISBN: 2160-7508.

[24] S. Carlsson. "Object detection using model based prediction and motion parallax". English. *Computer Vision — ECCV 90*. Ed. by O. Faugeras. Vol. 427. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1990, pp. 297–306. ISBN: 978-3-540-52522-6. DOI: 10.1007/BFb0014876. URL: http://dx.doi.org/10.1007/BFb0014876.

[25] C. Braillon et al. “Real-time moving obstacle detection using optical flow models”. English. IEEE, 2006, pp. 466–471. ISBN: 9784901122863; 490112286X.

[26] G. Ma et al. “A real-time rear view camera based obstacle detection”. English. 2009, pp. 1–6. ISBN: 1424455197; 9781424455195.

[27] H. Media. *Körning på landsväg, Hålla rätt avstånd*. 2015. URL: `https://korkortonline.se/teori/landsvag/` (visited on 04/13/2015).

[28] O. U. Press. *Definition of gold standard in English from the Oxford dictionary*. 2015. URL: `http://www.oxforddictionaries.com/definition/english/gold-standard` (visited on 04/28/2015).

A Hardware Specifications

A.1 Raspberry Pi 2 specifications

The Raspberry Pi 2 model B+ used in this thesis has the following components:

- A 900 MHz quad-core ARM Cortex-A7 CPU
- 1 Gb of RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Combined 3.4 mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

A.2 Computer

The computer used for performance measurements had the following specifications.

- A 2.53 GHz 64 bit Intel core i5 processor (M540)
- 4 gb of RAM
- An Intel X-25M SSD hard disk drive

B Thresholds

In this short chapter, all thresholds used together with their value and a short explanation of the are presented.

Threshold	**Value**	**Explanation**
Min hough line	30	Minimum length of a detected Hough line
T_{curve}	0.01	The angular speed needed to start looking for curved lane markers
T_{LDW}	8	Angle difference of lane markers to trigger the LDW
Min size of objects	25 px	Smallest pixel value of area of a detected object for it to be saved
Max distance for merging	20 px	The maximum distance between two objects for them to be merged as 1 object
$T_{obsClose}$	60 px	Trigger obstacle warning if any object is closer than this threshold
$T_{obsFast}$	10 px	Trigger obstacle warning if any object moves more than this threshold close to the car during 1 frame
T_{Slow}	2 px	Trigger obstacle warning if any object moves more than this threshold for 3 out of 5 of the last frames