UNIVERSITY OF PATRAS, GREECE
POLYTECHNIC SCHOOL

Department of Electrical and Computer Engineering
Department of Computer Engineering and Informatics

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Master of Science Program

Integrated Hardware and Software Systems

Master of Science Thesis

Distance Estimation between Vehicles Based on Fixed Dimensions Licence Plates

Author: **Vasileios Karagiannis**

Registration Number: 1043895

Thesis Supervisor: **Theodore Antonakopoulos**
Professor
Department of Electrical and Computer Engineering

Thesis Advisor: **Evangelos Dermatas**
Associate Professor
Department of Electrical and Computer Engineering

Grading Committee: **Theodore Antonakopoulos**
Evangelos Dermatas
Emmanouil Psarakis

Patras, February 2017

Abstract

Intelligent Transportations Systems (ITS) have become an integral part of every modern car since they provide advanced assistance in driving conditions. According to a study [1], 90% of rear-end collisions can be avoided if the driver is notified one second earlier. For this reason, in this thesis we focus on the problem of tracking the front vehicle through an on-board camera module and estimating its distance. We include a literature review on technologies that are currently being employed for distance estimation and we point out advantages and disadvantages. Additionally, we design a novel algorithm for vehicle tracking and distance estimation based on the licence plates that are, according to the law, in a visible place at the rear of each vehicle. The algorithm detects the plates based on their rectangular shape and colour variation and calculates the distance based on the standardized dimensions enforced by each country. Moreover, we use a low power ARM processor and a low cost 640x480 webcam to test its performance. Tracking the front vehicle works for distances of up to 9.6 meters from the camera, has an error of approximately 5% of the estimated distance and the ARM processor can process at least 20 frames per second in a video sequence. The results make it ideal for real time applications but also allow the possibility of increasing the image resolution in order to achieve longer distances during tracking and still being able to perform in real time.

Keywords: Intelligent Transportation Systems, Advanced Driving Assistance Systems, Vehicle Tracking, Distance Estimation, License Plates Detection, Real Time Application.

Table of Contents

List of Figures

Chapter 1

Introduction

Traffic accidents can have tragic outcomes such as injuries, permanent damage or even death. Most of them happen because of sudden distractions, inattention to the road and inability to sustain consciousness due to fatigue or influence of certain substances. To prevent accidents from happening as much as possible every effort is being made and many automotive companies and research groups turn their attention to developing intelligent transportation systems and advanced driving assistance systems. This thesis' target is the development of a system that can be used for tracking the leading vehicle and measuring its distance from the host vehicle by processing the input of an on-board camera module. Such implementation can be used as an integrated solution for vehicles that provide applications as automatic stopping, warning systems and even autonomous driving, all based on front vehicle proximity.

There are two major categories we can divide systems that use computer vision for vehicle safety:

- Stereo vision based systems.
- Monocular vision based systems.

Stereo vision based systems use epipolar geometry principles to combine two views, but because of the complexity of the algorithms, these systems usually require large storage space, high computational ability and are sensitive to noise and light [2]. On the contrary, monocular vision based systems do not need the recourses to combine views which is why they are generally more convenient and low cost and are more suited for the field of vehicle safety. Monocular vision based algorithms usually aim in detecting certain features (e.g. edges, shadows, symmetry) and translate that into conditions that occur while driving. For example, when a camera detects a symmetric shadow ahead, it could mean that there is a leading vehicle on the road. Other monocular vision based algorithms rely on the use of machine learning. These algorithms, that usually employ neural networks,

require more computational ability and therefore, it is more difficult for them to meet the time requirements of real time applications [2].

1.1 Problem Statement

According to the traffic law, depending on the speed of the vehicle, there is a minimum distance that every driver should keep from the leading vehicle. If for any reason a vehicle suddenly stops, following drivers should have enough space to slow down and immobilize their vehicles without endangering their lives. However, this safety rule is often not followed either because drivers are overconfident or ignorant. This thesis focuses on developing a system that detects the front vehicle, estimates its distance and can potentially initiate an event based on the value of the estimated distance (e.g. warning sound if the distance is dangerously short). Current implementations [3], [4], [5], [6], [7] rely on characteristics that can be influenced by weather and illumination conditions (e.g. vehicle shadow) and their suitability for real time applications can be argued since results related to time efficiency are either poor or absent. Meeting the requirements of real time applications can be very hard for embedded systems integrated in vehicles because the capacity of the hardware resources is usually limited.

1.2 Thesis Overview

This thesis is organized in 6 chapters:

Chapter 2 includes a literature review on related work. We describe ways to detect the leading vehicle and measure its distance according to already implemented systems. In addition, we comment on their efficiency by pointing out advantages and disadvantages.

Chapter 3 is about the implementation of a novel algorithm for vehicle tracking and distance estimation. We describe its function by providing figures, code samples and clarifications for each stage of the algorithm and we also include general observations and comments on its efficiency.

Chapter 4 presents a detailed analysis of the algorithm's possible error when estimating the distance from the leading vehicle. We take into account all factors that affect the distance measurement and we calculate the range of possible error. Moreover, chapter 4 includes information on the applicable distance which is the

maximum distance for which the algorithm is able to detect and follow a leading vehicle.

Chapter 5 contains the experimental results of the algorithm being tested on a low power ARM processor using 640x480 video from various driving conditions. The featured results are outcome of the evaluation of the algorithm based on three aspects (time efficiency, reliability and accuracy).

Finally, **Chapter 6** summarizes the thesis by providing overall conclusions for the implemented algorithm and by stating lines for further research in the area of tracking and estimating the distance of the leading vehicle.

Chapter 2

Literature Review

This chapter presents ways to track the front vehicle and measure its distance according to the literature. Each subsection describes a robust method that has been tested and works and can actually be used to provide assistance to drivers. All the methods along with their results come from the literature, we did not implement the algorithms. We describe their function and comment on their strengths and weaknesses.

2.1 Based on Vehicle Shadow and Symmetry

In [3] the authors introduce an algorithm for tracking a leading vehicle and estimating its distance based on multiple image features. They use corner detection, edge detection, vehicle symmetry and image matching to identify a car figure and based on its width and height they estimate its distance.

2.1.1 Algorithm Outline

This algorithm relies on the fact that the bottom area of a car (shadow) as shown in figure 1, displays very strong characteristics and is a very distinct region compared to the rest of the road. It is also assumed that there are not any other obstacles in front of the car other than the leading vehicle. Such assumption is very reasonable though, especially during normal driving conditions.

Figure 1: Typical front car image (shadow inside rectangle) [3]

To detect this region the algorithm starts with Gaussian blurring in order to smoothen the image and lessen possible noise. After that, it tries to detect the bottom left side of the car that shapes like an “L” by using corner and edge detection as shown in figure 2.

Figure 2: Corner detection (left) and edge detection (right) [3]

To identify the bottom left side of the car they align an “L” shape template for every corner point and find its cross correlation with the edge image. A region of interest is identified as the corner point for which the correlation has the maximum value. Then, to ensure that this region is actually part of a car they use a threshold saying that all enclosed points must have a lower value. This basically means that the shadow of the car must be darker that the rest of the road and that there is road only below the vehicle. Figure 3 shows the tracking of the bottom left side of a vehicle. To identify the right side of the car a column direction projection is used in the region of interest.

Figure 3: Tracking bottom left corner [3]

At this point, left and right side of the car are known and thus, the width of the car is known as well. To figure out the height of the car, they take advantage of the fact that for each car vehicle category (e.g. sedan, SUV, truck) there is a fixed ratio of width/height. They use a matching template technique to decide the category of the vehicle and after that they define both vehicle type and height. This way, the car is detected more accurately as shown in figure 4.

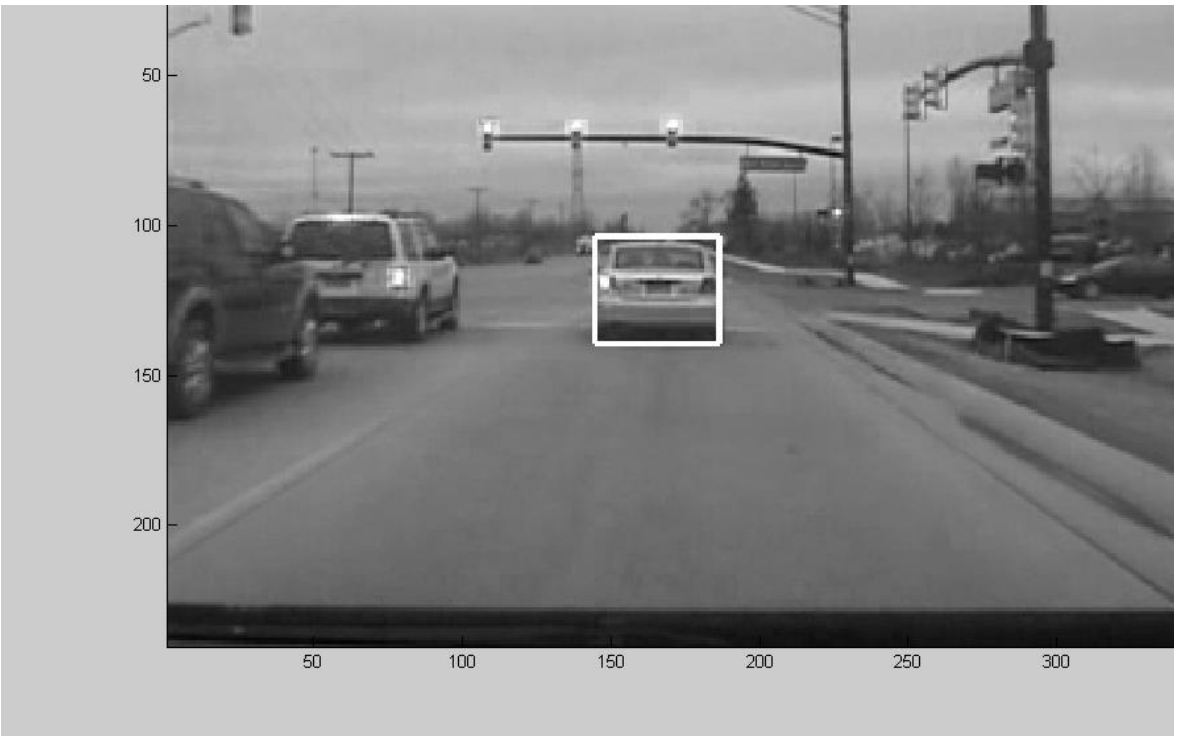


Figure 4: Vehicle tracking [3]

2.1.2 Comments and Results

According to the authors, the algorithm was simulated using Matlab and its performance was found satisfactory. Figure 5 shows the results of the algorithm tracking the front vehicle in real driving conditions.

Figure 5: Tracking for different vehicles [3]

However, there is no analysis on the limits of the applicable distance or the range of possible error. It is mentioned that it worked for distances from 10 to 20 meters and that the Kalman filter was used to estimate the distance but there is no further information.

This algorithm uses several features in order to identify a car, which minimizes the possibility of error. However, the more features are considered, the more time is needed to process each frame. One of the initial steps of the algorithm is to find all corners and for each corner to calculate a value that comes from the cross correlation with the edge image. This process can take a lot of time depending on the number of corners in the image. A typical image of a car can result in a significant number of corners and taking into account any object inside the point of view of the camera, the amount of corners can create considerable delay. Additionally, different illumination conditions can weaken the vehicle's shadow which might result in detection failure. To conclude, the algorithm seems robust due to the many conditions that need to be satisfied in order to identify a car but further testing regarding time efficiency and illumination conditions is necessary. More information on the algorithm along with its references can be found in [3].

2.2 Based on Vehicle Contour and Symmetry

In [4], the authors describe a feature based algorithm that tries to find the silhouette of front vehicles. To avoid false detection of stationary objects inside the point of view of the camera, they use two consecutive frames and combine their results. They verify the vehicle's symmetry by matching candidate silhouettes with fixed templates of different types of vehicles (e.g. sedan, SUV, track, bus). Upon success, the distance is calculated by using a scaling factor of known templates based on the type of the vehicle. Additionally, they include the case of difference illumination conditions and describe ways to overcome difficulties that might occur during night time.

2.2.1 Algorithm Outline

At first the algorithm uses active contours to detect vehicle silhouettes and after that it matches geographic differences of two consecutive sample frames in order to identify object boundaries (figure 6).

Figure 6: Contours of non-stationary objects [4]

This step might also result in false detection of non-stationary objects as shown in figure 7.

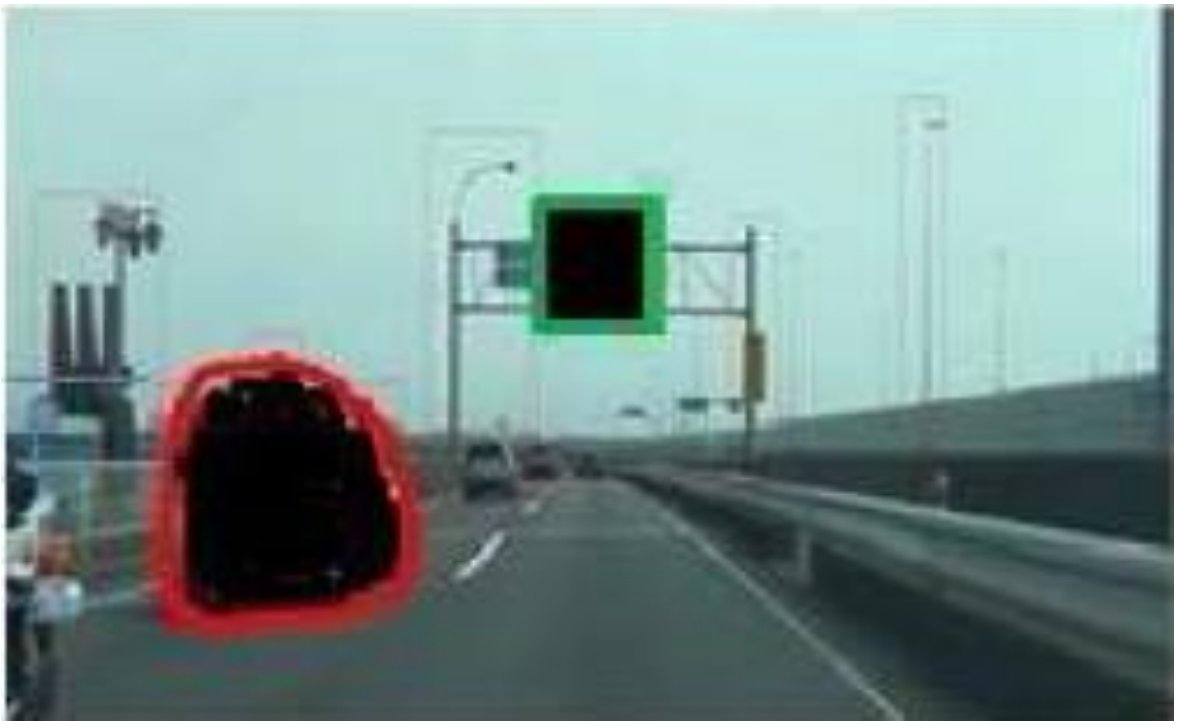


Figure 7: False detection [4]

For every frame that arrives, information from the previous frame is used so that the moving objects can be tracked and stationary objects can be eliminated. To accomplice that, each frame is masked with the stencil of stationary objects. This way, only moving objects are followed in a sequence of frames.

Continuously, the candidate silhouette is compared (width to height ratio) to known templates of vehicle categories (e.g. car, track, VAN) and the type of vehicle is recognized. The symmetric property is used as verification and finally, the distance is determined by using a scaling factor. This means that for every type of vehicle there is a known relation between size and distance.

During night time however, vehicle features are not very distinct. According to the authors the only feature that can be used is the light from the headlights. For this reason, it is proposed to use either known templates of headlights or Haar classifiers in order to detect the vehicle's lights. An example of a template and a classifier is shown in figure 8.

Figure 8: Use of template (left) and Haar classifier (right) [4]

The Haar classifiers for front and rear view are included as well (figure 9).

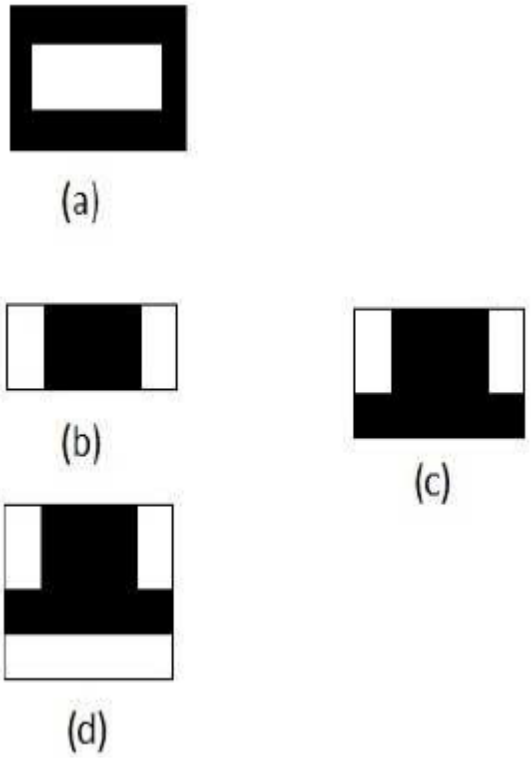


Figure 9: Set of Haar classifiers [4]

Another feature that they use to identify rear view is the red colour of the rear lights of the vehicle. An example is shown in figure 10.

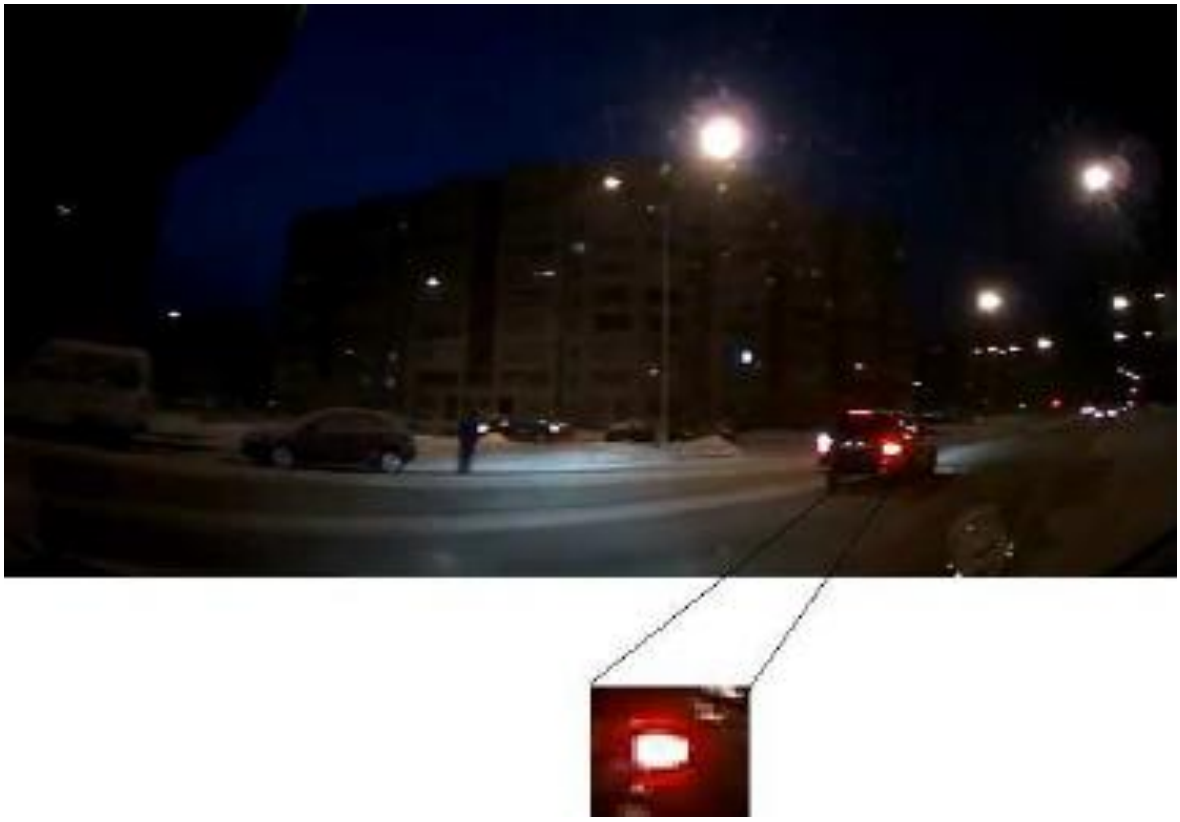


Figure 10: Red colour in rear lights [4]

2.2.2 Comments and Results

Numerical results such as applicable distance, possible error range and average process time for each frame are not included. However, calculations are kept to minimum necessary for the detection and tracking of the front vehicle and the featured images show satisfactory performance, which implies suitability for real time implementation. This is not certain though, since there are no measurements of processing time. A major concern regarding real time performance is that all calculations are performed on RGB images which increases process time significantly for each frame compared to converting the image to grayscale before processing. It is not known if RGB images are necessary for all stages of the algorithm though. Grayscale frames can increase time efficiency but could possibly reduce reliability.

Providing an alternative for night time conditions, where the main algorithm is most likely to fail, is definitely an advantage of this implementation. There is no information on how day time and night time algorithms are combined though. It is not mentioned if the time of the day is an input or if it is part of the algorithm to determine illumination conditions. Latter case means that the duration needed to detect current illumination creates additional overhead which needs to be taken into account when considering time efficiency.

2.3 Based on Road Lane and Vehicle Shadow

In [5], the proposed algorithm relies on lane detection and detection of the front vehicle's shadow and after that, it estimates the distance. The lane is recognized as two straight lines in front of the camera using the Hough transform. Then, the algorithm utilizes the Sobel operator to find a horizontal edge line inside the lane that could indicate the existence of a vehicle shadow. The distance is then estimated based on the position of the shadow on the lane.

2.3.1 Algorithm Outline

This method begins with the Cunny operator in order to find all the edges in a frame. After that, all vertical and horizontal lines are deducted because they cannot be part of the lane. This way the image becomes clearer as shown in figure 11 and is then ready for the Hough transform. This step intends to reduce the calculation time of the transform.

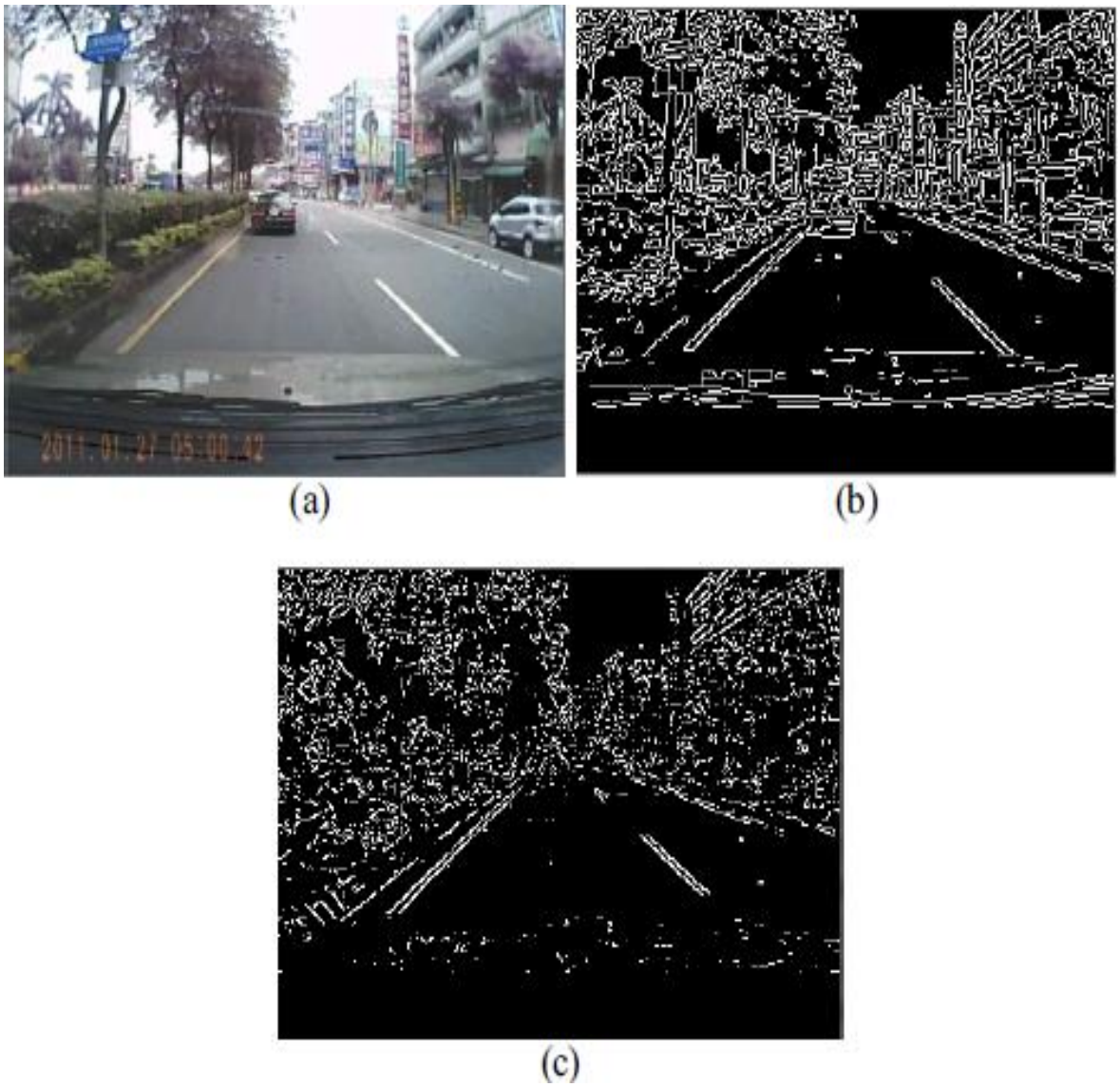


Figure 11: Initial frame (a), Cunny operator (b), after reducing noise (c) [5]

To acquire the correct lines from the Hough transform there are two filtering rules. The lane lines are found in the most intensive intersection of the transform and these lines are required to be accumulated for a period of time. The end of the road, which is the point where the two lines of the lane collide is referred to as the vanishing point. To acquire the correct lane and vanishing point, multiple frames are used. Results of using the Hough transform over a period of time to identify the lane are shown in figure 12.

Figure 12: Segmentation of the lane until the vanishing point [5]

After that, a vehicle's shadow is recognized by using the Sobel operator and by searching for a short horizontal line inside the lane.

Based on the position of the shadow, the vanishing point and the road scale, distance is calculated according to the equalities in figure 13.

$$DD = rate*(H - Car\_y),$$
$$rate = Set\_d / (H - Vanishing\_y)$$

Figure 13: Distance estimation equalities [5]

Set_d is the distance between the host car and the vanishing point.
Vanishing_y is the value of the y coordinate of the vanishing point.
Car_y is the value of the y coordinate of the car
H is the width of the frame.
DD is the distance.

2.3.2 Comments and Results

The authors include in their study a table of experimental results shown in figure 14.

The results are outcome of using three videos of urban and suburban environment

with various backgrounds. DN is the number of correct vehicle detections, FN stands for the number of false vehicle detections and MN is the number of vehicles. Rates are also included, DR indicates the rate of correct detections and FR is the rate of false detections.

Test videos	*MN*	*DN*	*FN*	*DR*	*FR*
Video-1	1058	962	64	91%	6%
Video-2	756	589	127	78%	17%
Video-3	1195	956	227	80%	19%

Figure 14: Evaluation of the algorithm based on three videos [5]

This algorithm keeps the calculations to minimum and seems to be able to cope with real time requirements, even though results on processing time or average frames per seconds are not included. Based on the three videos, success rate is at least 78% and false detections can be up to 19%.

Other than time efficiency comments that are missing, another concern regarding this algorithm is that one of its inputs is the distance between the host vehicle and the vanishing point, which is referred to as Set_d in figure 13. This distance cannot be regarded as a fixed value since it changes based on the road and the driving conditions. This parameter affects the value of the final distance estimation and when it deviates from its exact value, it can create additional error.

2.4 Based on Vehicle Shadow and Licence Plates

In [6], the authors propose an algorithm that performs detection based on the shadow underneath the vehicle and the horizontal edges caused by its silhouette. Then for verification of the existence of a vehicle and for estimating its distance they make use of the licence plates whose dimensions and shape are standardized for each country. They provide a deep analysis of their approach along with experimental results. Finally, they conclude their study by saying that tests using actual vehicles in real urban traffic conditions were made and the algorithm showed excellent robustness and reliability both in vehicle and licence plate detection and very good accuracy in distance estimation.

2.4.1 Algorithm Outline

At first, a static region of interest is obtained from the frame. The goal of this region is to acquire only the part of the frame that corresponds to the front part of the car. This way, objects inside the point of view of the camera that are not in front of the car are eliminated (figure 15 a).

The vehicle detection is based on two features, the vehicle shadow and the horizontal edge caused by the shadow. The shadow intensity depends on the illumination conditions but it is always more intense and dense compared to the rest of the road. To identify the shadow, they use an adaptive threshold based on the histogram of the grayscale region of interest. The histogram displays two peaks. Since the shadow of the car is most likely to be the darkest area inside the region of interest, the lower peak of the histogram corresponds to the shadow underneath the vehicle. The higher peak corresponds to the road. Apart from the two peaks, there might be higher values that correspond to the lines of the road since they are usually white or yellow. After the shadow threshold, they use edge detection to find horizontal lines that are considered vehicle candidates. The length of the horizontal line is identified as the vehicle's width. The height is considered to be 130% of the width in order to include all types of vehicles (e.g. car, van, SUV). The above steps are shown in figure 15.

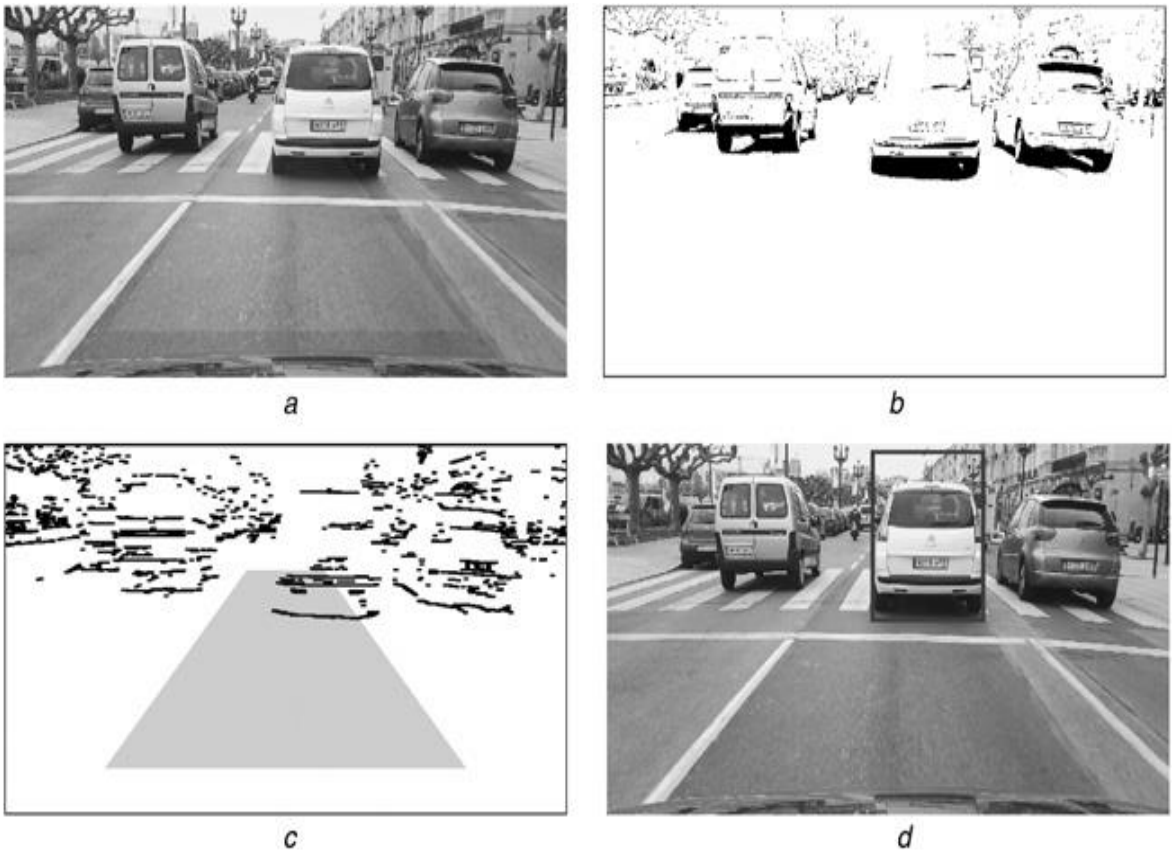


Figure 15: Region of interest (a), shadow threshold (b), horizontal lines (c), candidate vehicle (d) [6]

At this point, a first estimation of the distance is obtained based on the location of the lower edge of the vehicle's bounding box. To acquire this distance, they use a template of preconfigured values that relates the position of the front vehicle in the image with distances (in meters) from the host vehicle. These values have been established through experimental tests.

To detect the licence plates, initially they use morphological closing to reduce noise and then the Top Hat operator. The Top Hat operator is usually used to extract certain features form an image. The returned image includes objects that are relatively small and brighter than their surroundings. After Top Hat, the image is turned to binary using the Otsu method. The Otsu method calculates a threshold based on the intensity of the image and it uses it to convert the image to binary. The steps so far are shown in figure 16.

Figure 16: Candidate vehicle (a), morphological closing (b), Top Hat method (c), Otsu method (d) [6]

Continuously, they make use of information that has already been determined. Since the width and height of the vehicle has been calculated, there is already a rough estimation on the location and the dimensions of the licence plates. On the current image (figure 16 d), all elements whose horizontal separation is less than a threshold are joined together and from the result if an element is taller than the expected height of the plates or wider than the expected width, it is eliminated. To verify the existence of plates in the remaining candidate areas they use features of the characters that are standardized in each country (e.g. number of characters, distance between two consecutive characters, separation gap between numbers

and letters). The remaining candidate contained in rectangular areas that correspond to the location of the initial estimation based on the vehicle detection are the plates. The steps of this stage are shown in figure 17.

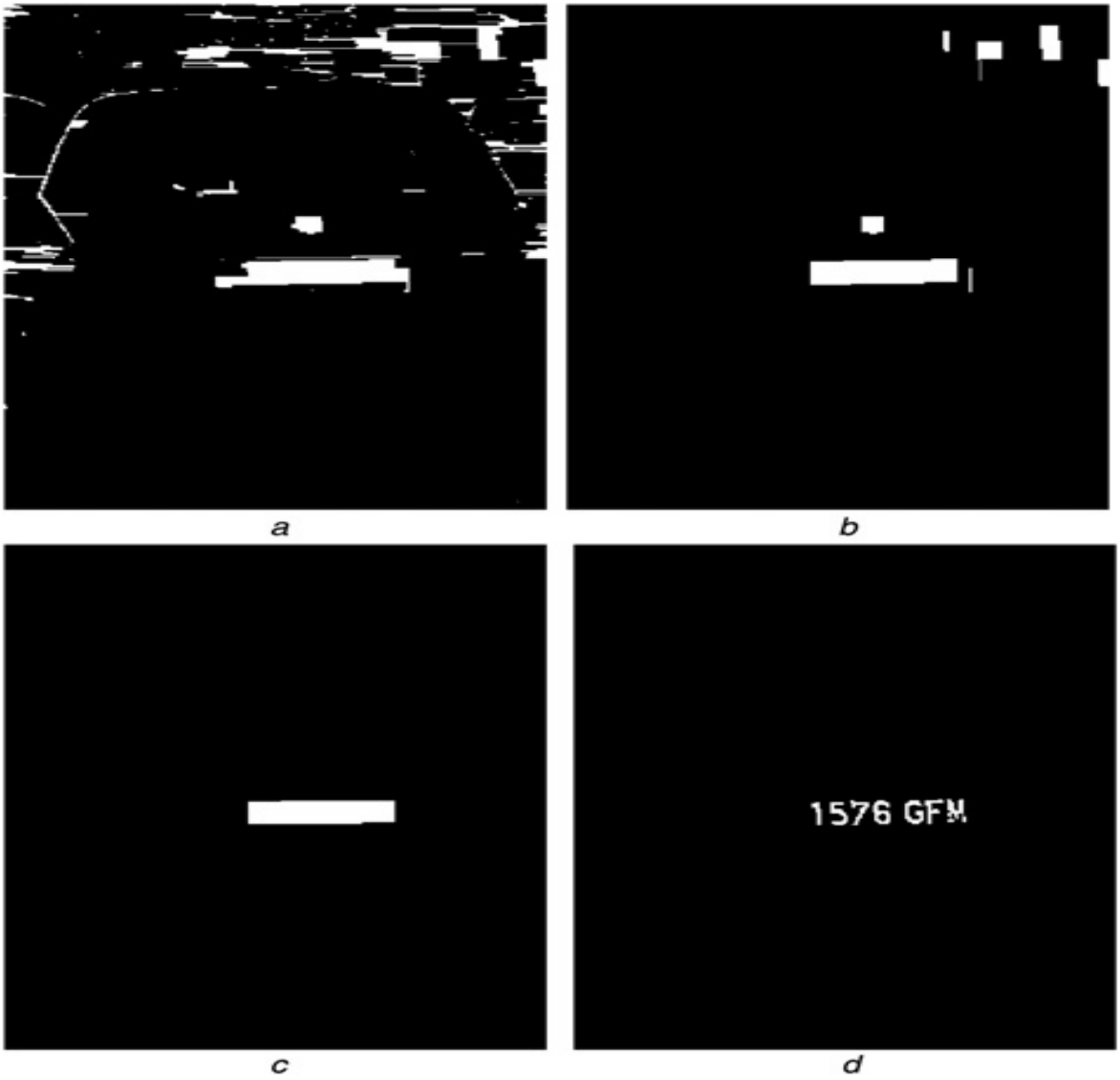


Figure 17: Elements horizontally joined (a), width/height elimination (b), plate candidates (c), inside rectangular candidates (d) [6]

After the licence plates have been detected, the distance is estimated based on a template of preconfigured values. These values have been extracted from experimental tests and relate the width of the licence plate number and the height of the characters to the distance of the front vehicle. All the features needed from the licence plates are shown in figure 18.

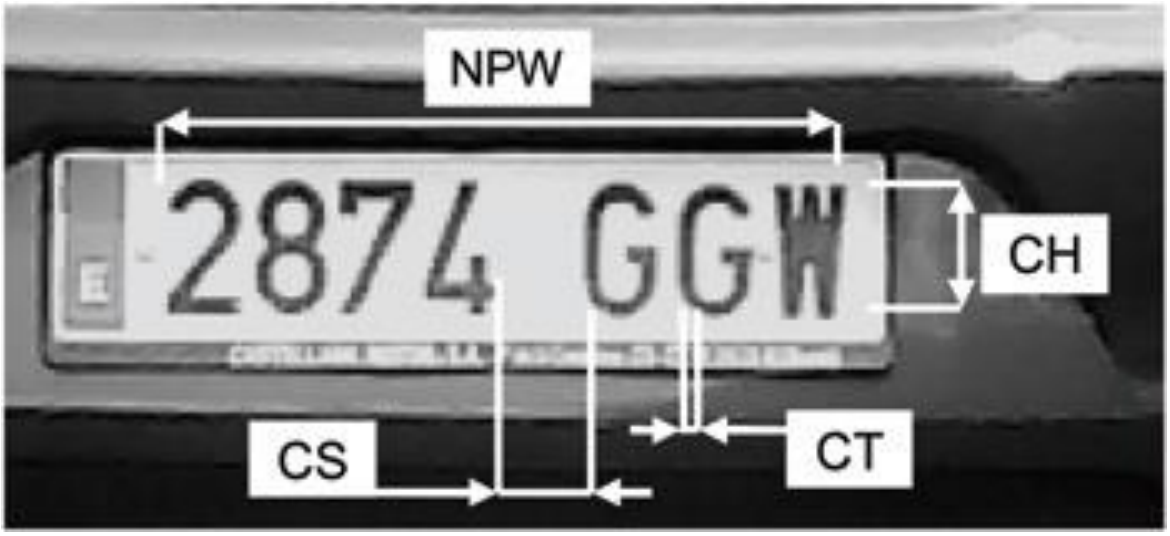


Figure 18: Plate number width (NPW), height of character (CH), separation gap (CS), character thickness (CT) [6]

2.4.2 Comments and Results

According to the authors, the system is tested in real driving conditions of various locations and weather conditions. They use a camera with resolution of 1280x720 to record sample videos and the processing takes place on an Intel Pentium 4 3.06GHz processor with 480 MB RAM using Matlab. To obtain the real distance so it can later be compared to the system's output values, a LASER distance meter is used with functional distance of up to 100 meters. Figure 19 shows the results of vehicle detection.

	Total number of test frames	NFV in the safety area	NFNV in the safety area	NVD	NFD	VDR, %	FVDR, %
cloudy	1000	486	514	478	8	98.35	1.55
sunny	1000	502	498	498	4	99.20	0.80
rainy	1000	515	485	508	7	98.64	1.44
tunnel	400	344	56	335	2	97.38	3.57

VDR = vehicle detection rate, FVDR = false VDR, NFV = number of frames with vehicle in the safety area, NFNV = number of frames with no vehicle in the safety area, NVD = number of vehicles detected and NFD = number of false detections. $\text{VDR}(\%) = \frac{\text{NVD}}{\text{NFV}} \times 100$ and $\text{FVDR}(\%) = \frac{\text{NFD}}{\text{NFNV}} \times 100$.

Figure 19: Analysis of vehicle detection [6]

The algorithm presents impressive results with successful detection rate of at least 97.3% and false detection rate of up to 3.5%. Figure 20 shows the results of licence plate detection.

	Total number of test frames	NFV in the safety area	NVD	NLPL	NFLPL	LPLR, %	FLPLR, %
cloudy	1000	486	478	475	0	99.37	0
sunny	1000	502	498	494	0	99.19	0
rainy	1000	515	508	504	0	99.21	0
tunnel	400	344	335	334	0	99.40	0

NFV = number of frames with vehicle in the safety area, NVD = number of vehicles detected, NLPL = number of license plate localisations, NFLPL = number of false license plate localisations, LPLR = license plate localisation rate and FLPLR = false license plate localisation rate. $LPLR(\%) = \frac{NLPL}{NVD} \times 100$ and $FLPLR(\%) = \frac{NFLPL}{NFD} \times 100$.

Figure 20: Analysis of licence plate detection [6]

Licence plates are almost always found successfully because licence plate detection is only performed on identified vehicles. Figure 21 show the results of distance estimation.

Laser dist	Cloudy		Sunny		Rainy	
	NPW Dist	CH Dist	NPW Dist	CH Dist	NPW Dist	CH Dist
0.5	0.500	0.51	0.494	0.51	0.506	0.51
1	1.020	0.97	1.033	0.97	1.007	0.97
1.5	1.484	1.46	1.503	1.46	1.464	1.46
2	1.967	2.09	1.995	2.09	1.941	2.09
2.5	2.466	2.63	2.502	2.63	2.431	2.63
3	2.994	3.07	3.040	3.07	2.948	3.07
3.5	3.504	3.32	3.561	3.32	3.448	3.32
4	3.999	3.89	3.999	3.89	3.932	3.89
4.5	4.515	4.58	4.515	4.58	4.436	4.58
5	5.032	4.98	5.032	4.98	4.940	4.98
5.5	5.527	5.43	5.527	5.43	5.423	5.43
6	5.974	5.93	5.974	5.93	5.857	5.93
6.5	6.471	6.51	6.471	6.51	6.471	6.51
7	7.027	7.17	7.027	7.17	7.027	7.17
7.5	7.488	7.17	7.488	7.17	7.488	7.17
8	7.990	7.96	7.990	7.96	7.990	7.96
8.5	8.540	7.96	8.540	7.96	8.540	7.96
9	8.935	8.89	8.935	8.89	8.935	8.89
9.5	9.574	8.89	9.574	8.89	9.574	8.89
10	10.035	10.03	10.035	10.03	10.035	10.03

Laser Dist = distance provided by the laser, NPW Dist = distance provided by the width of the number plate and CH Dist = distance provided by the character's height.

Figure 21: Analysis of distance estimation [6]

The results of distance estimation are rather impressive as well. When using the width of the licence plate number, the distance has an error of up to 14 centimetres which actually happens during raining conditions.

The algorithm presents outstanding results regarding reliability and accuracy. The processing time for each frame varies and depends on the number of possible candidates. The average time of each frame is 248 milliseconds which implies approximately 4 frames per second on a 3.06 GHz Intel processor using Matlab. However, according to the authors, running on a more suitable processing unit, the algorithm could achieve a higher rate of frames per second.

Apart from the suitability of the algorithm for real time application which can be argued due to high processing duration needed for each frame, there is another concern. The algorithm uses pre-configured values that have been established for the specific vehicle using the specific camera resolution. If a different vehicle model is used these values will change. Same thing will happen if the camera resolution is changed. The pre-configured valued will no longer correspond to the actual measurements. This means that the algorithm has to be pre-configured for each vehicle model that is being used, otherwise it will create an additional error to the distance estimation.

Another possible weakness of the algorithm is that to verify the licence plates, it makes use of licence plate number features. The features of the licence number are no longer distinct after a few meters of distance. However this does not show in the results which means that it does not happen probably because of the high resolution of the camera.

2.5 Based on Road Lane, Vehicle shadow and a Neurofuzzy Network

In [7], the authors create an algorithm for avoiding lane departure and front vehicle collision. The system detects the road lane that the host car follows and keeps track of the position of the car. If the car starts moving away from the lane it alarms the driver. Additionally, the algorithm uses the area inside the lane as a region of interest and it searches for vehicle shadows that could mean the existence of another vehicle in front of the host car. Upon successful recognition of a front vehicle, the number of pixels in the frame between the two vehicles is given to a neurofuzzy networks that estimates the distance.

2.5.1 Algorithm Outline

The algorithm starts with lane detection. The host vehicle is assumed to be driving in the middle of the lane so the right line is in the right side of the image and the left line is in the left side of the image. To identify the lane lines, the grayscale image is divided in two parts (left and right) and each part is used for finding the corresponding line. To identify the lines, they define two edge detection masks as shown in figure 22.

Left mask

1	0	0	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	-1	0
0	0	0	0	-1

Right mask

0	0	0	0	-1
0	0	0	-1	0
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0

Figure 22: Edge detection masks [7]

If the value of an edge is greater than 30, its colour is set to green (positive edge) and if its value is less than -30, its colour is set to red (negative edge). Background colour is set to black. If the distance between two colours (red and green) is less than five pixels, then white colour is used to fill in the pixels in between. The result of the edge detection is shown in figure 23.

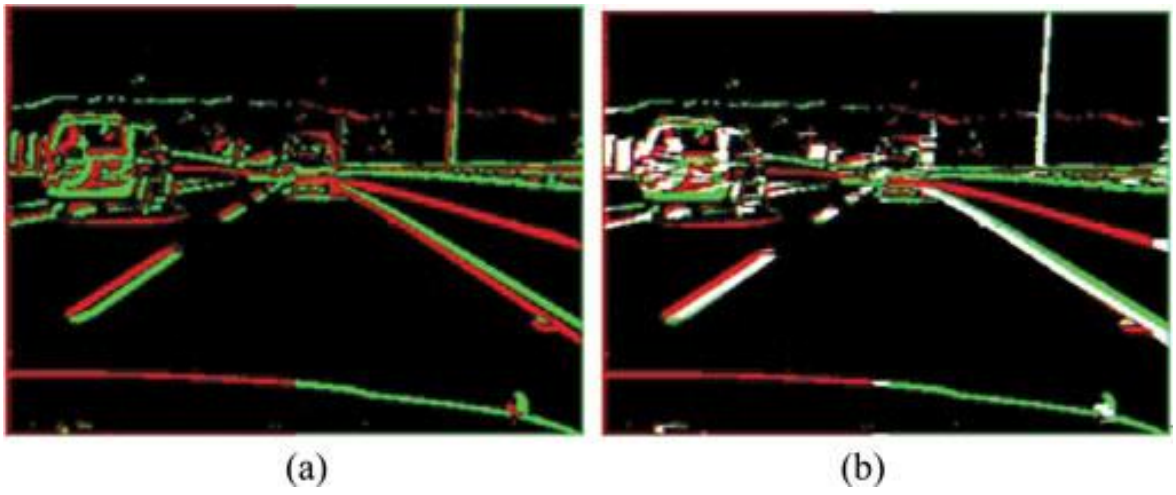


Figure 23: Red and green edges (a), red-green distance filled with white [7]

After that, the image is converted to binary by turning all pixels apart from the white ones to black and then, they keep only the edges of the white pixels. Results are shown in figure 24.

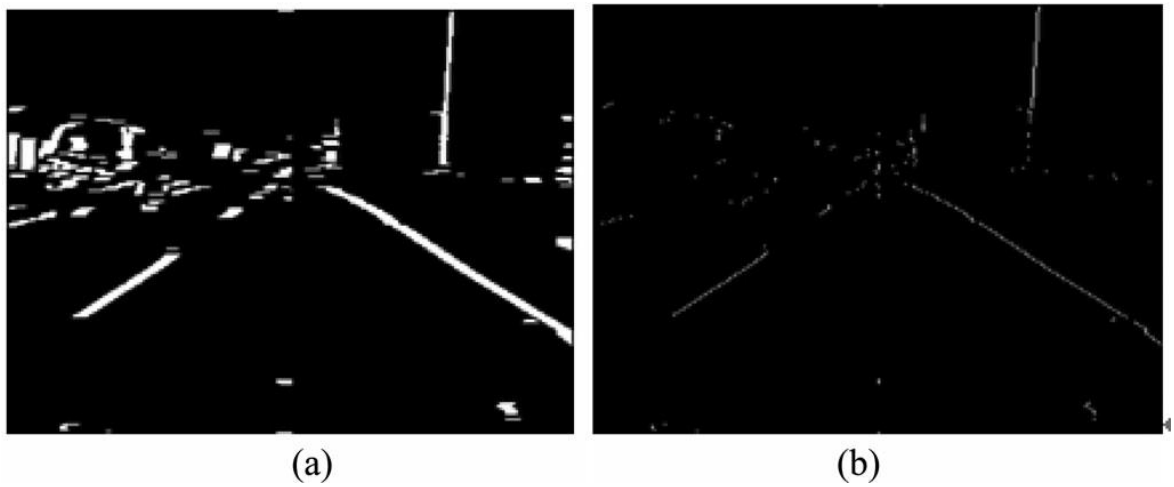


Figure 24: White lines (a), edges of white lines (b) [7]

Continuously, the goal is to identify the lanes. The edges are grouped into line segments and then the line segments are combined to form longer lines. At first, the segments are sorted to left or right-side segment. Then the biggest segment is used as a base and the other segments are added to the base segment if they belong to the same line. To calculate if two segments belong to the same line they use the straight line equation $ax + bx + c = 0$ which checks if the points are part of the same straight line. If the segments are part of the same straight line they are merged to one segment. Eventually the biggest segment for each side corresponds to the lane line. To avoid processing the background, all segments over the vanishing point are excluded.

To prevent lane departure, they use the position of the car on its current lane and check the corners that are formed from the lane on the road. The movement of the car on the lane has an impulse on these corners. Judging from the change of both left and right corner the direction of the car's movement can be established. Figure 25 shows the lane boundaries and the corners that are formed.

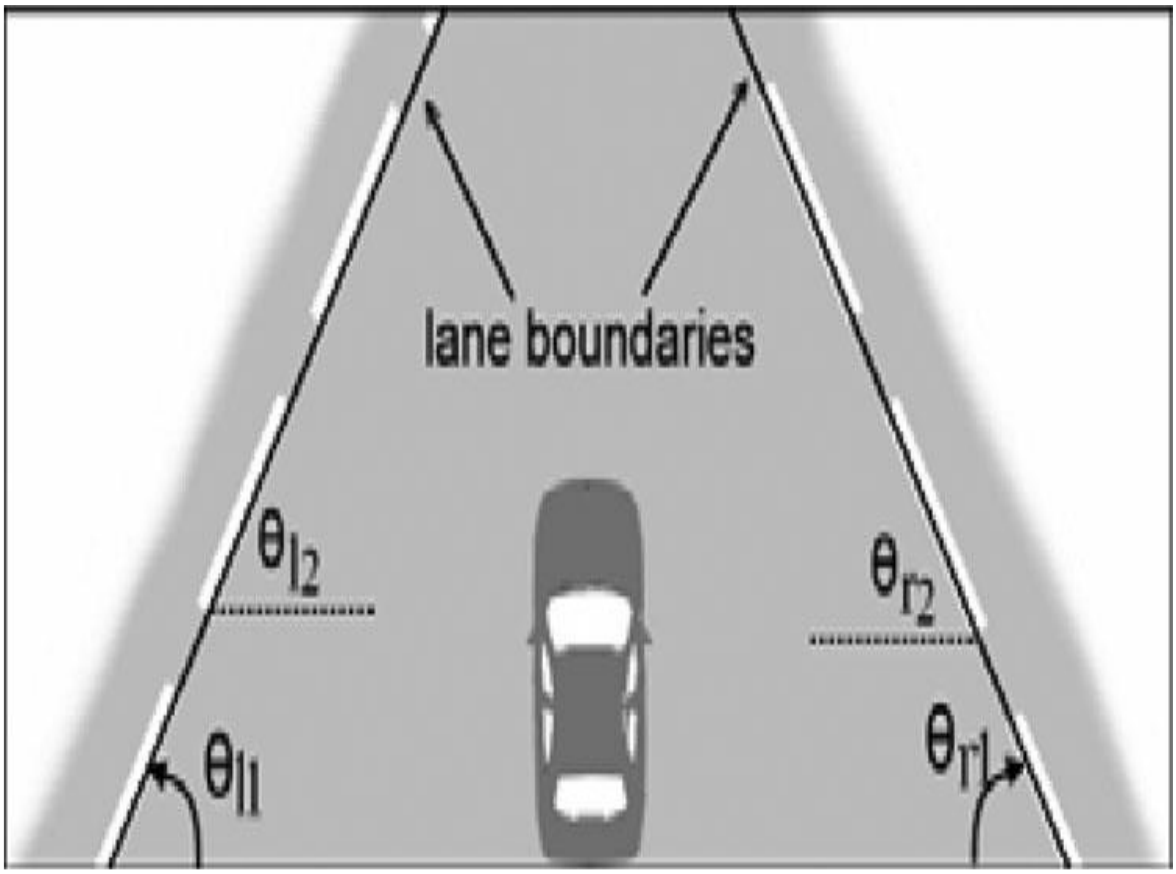


Figure 25: Lane boundaries and lane corners [7]

The algorithm proceeds with finding the front vehicle. The area inside the lane is used as a region of interest so only vehicles in the same lane are considered for detection. Initially, the image is converted to grayscale and after that it is turned to binary. The threshold for the conversion to binary is the average value of all the pixels in the image. The goal of this process is to identify the vehicle's shadow as one of the darkest areas in the image. Then, morphological closing is used to fill in small holes in the background. Continuously, only the edge of each white area is kept since the size of the shadow is not necessary. The width of each shadow is compared to the width of the lane as a way to confirm a vehicle's shadow. The steps of this stage are shown in figure 26.

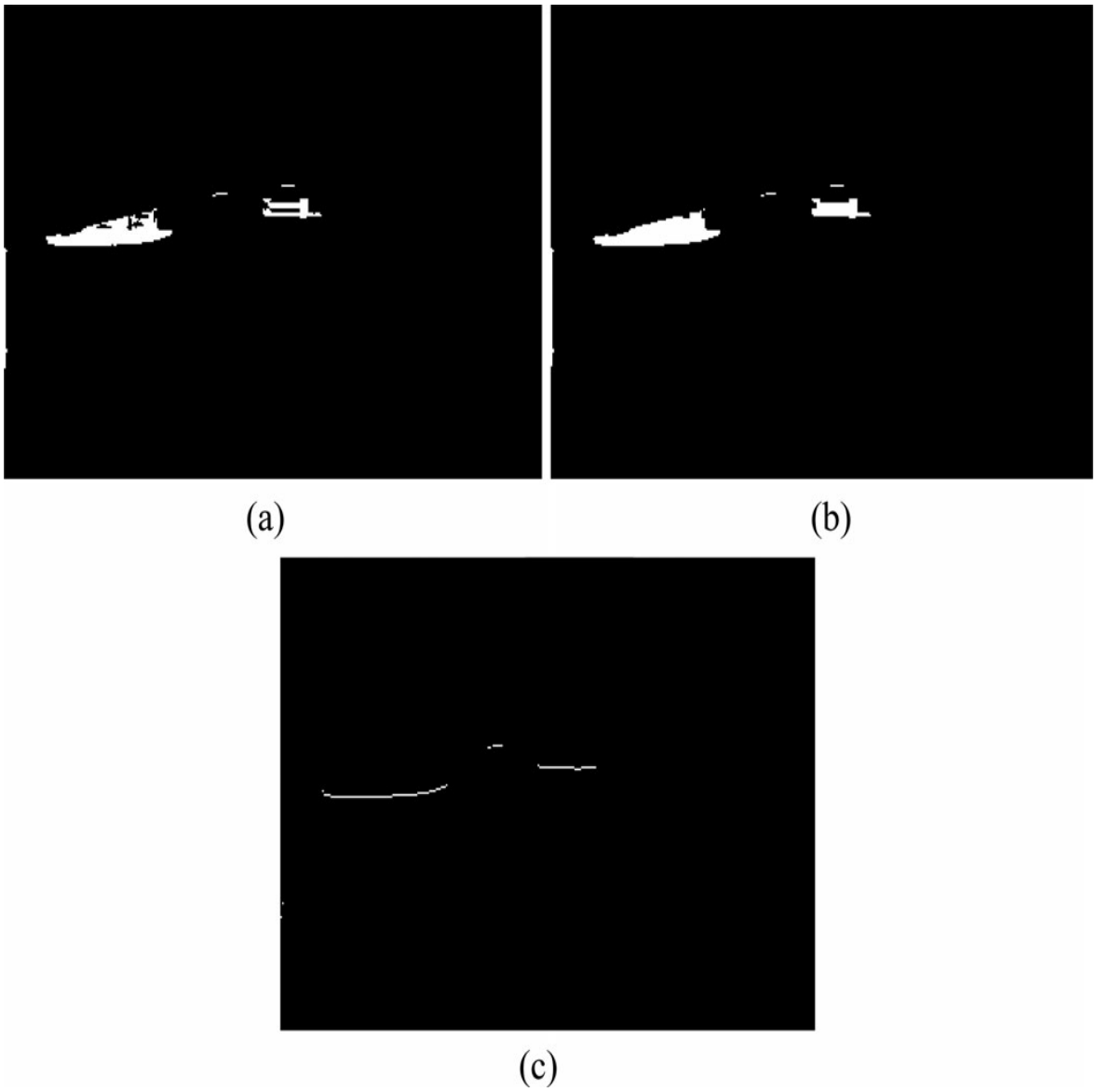


Figure 26: Binary image (a), morphological closing (b), edges of shadows (c) [7]

Since the exact position of the vehicle's shadow is now known, the horizontal and vertical Sobel operator is used in order to find the edges of the vehicle so that the contour can be recognized and the vehicle can be detected. To make the algorithm perform well during night time, instead of the shadow, the taillights are used. Results of the detection are shown in figure 27.

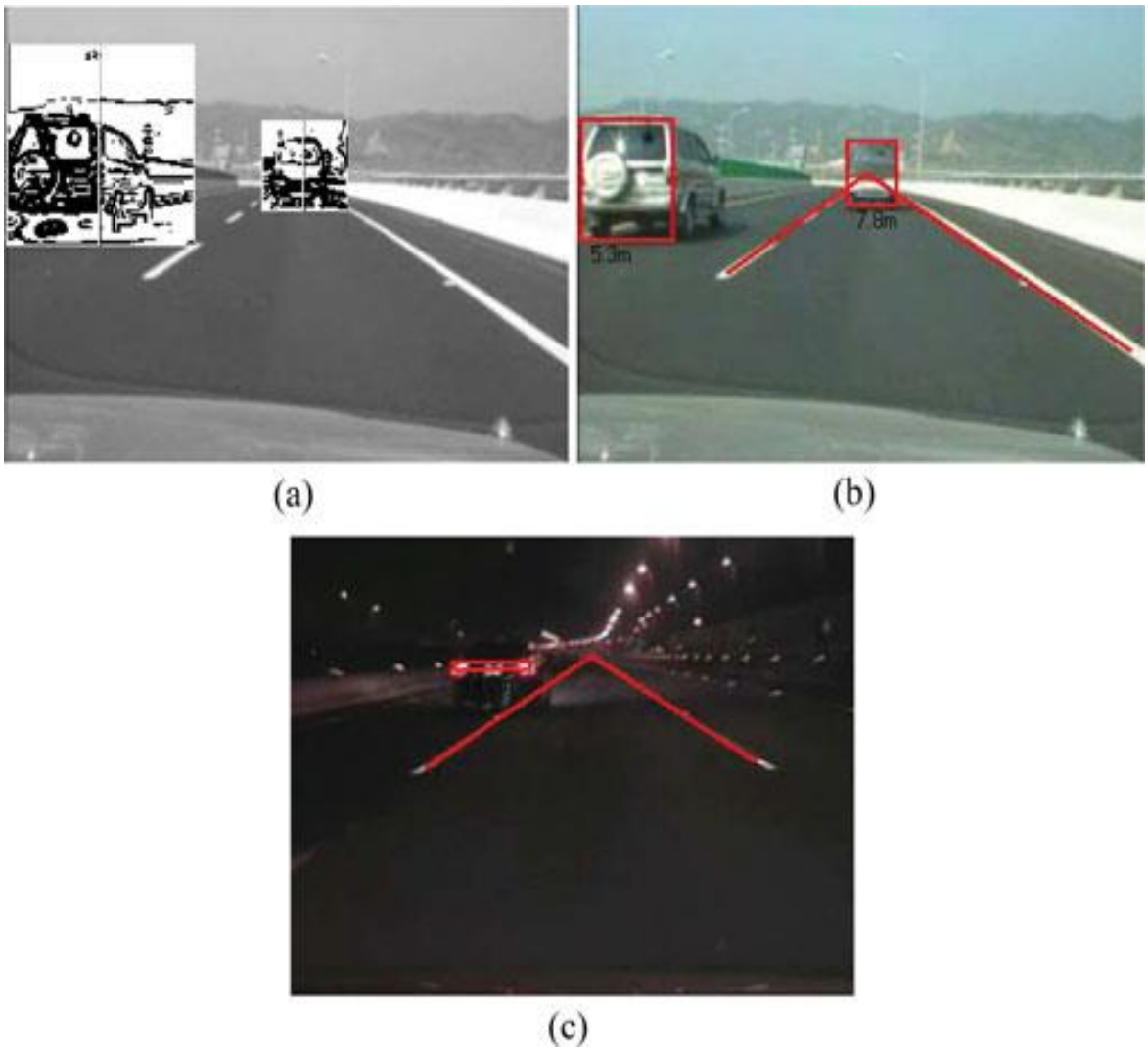


Figure 27: Sobel operator (a), vehicle detection (b), taillight detection (c) [7]

Finally, after the vehicle has been detected, the distance (number of pixels) between the two vehicles is used as input for the neurofuzzy network. The structure of the proposed neurofuzzy network is shown in figure 28.

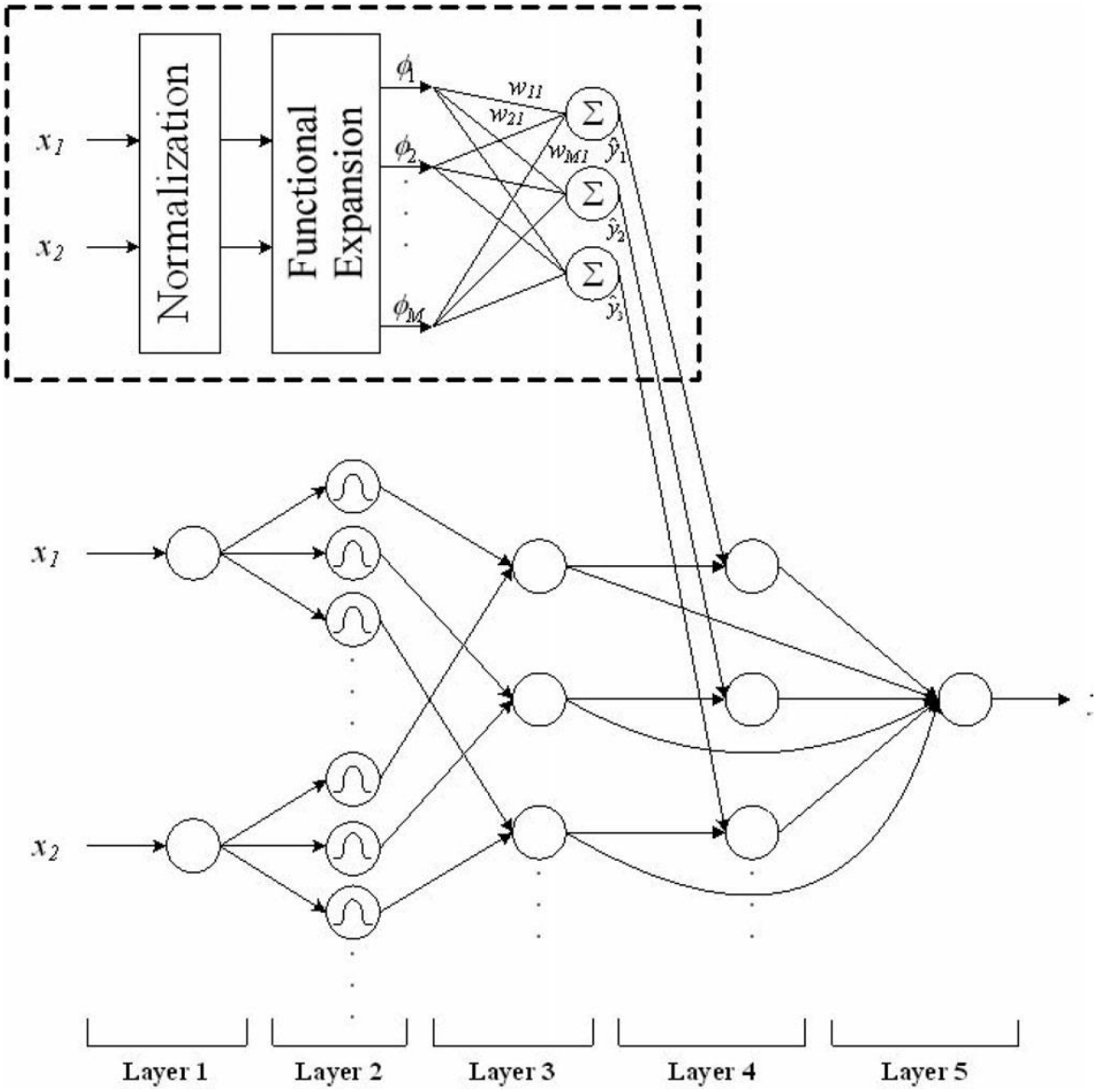


Figure 28: Structure of the neurofuzzy network [7]

The flow diagram of the learning process is shown in figure 29.

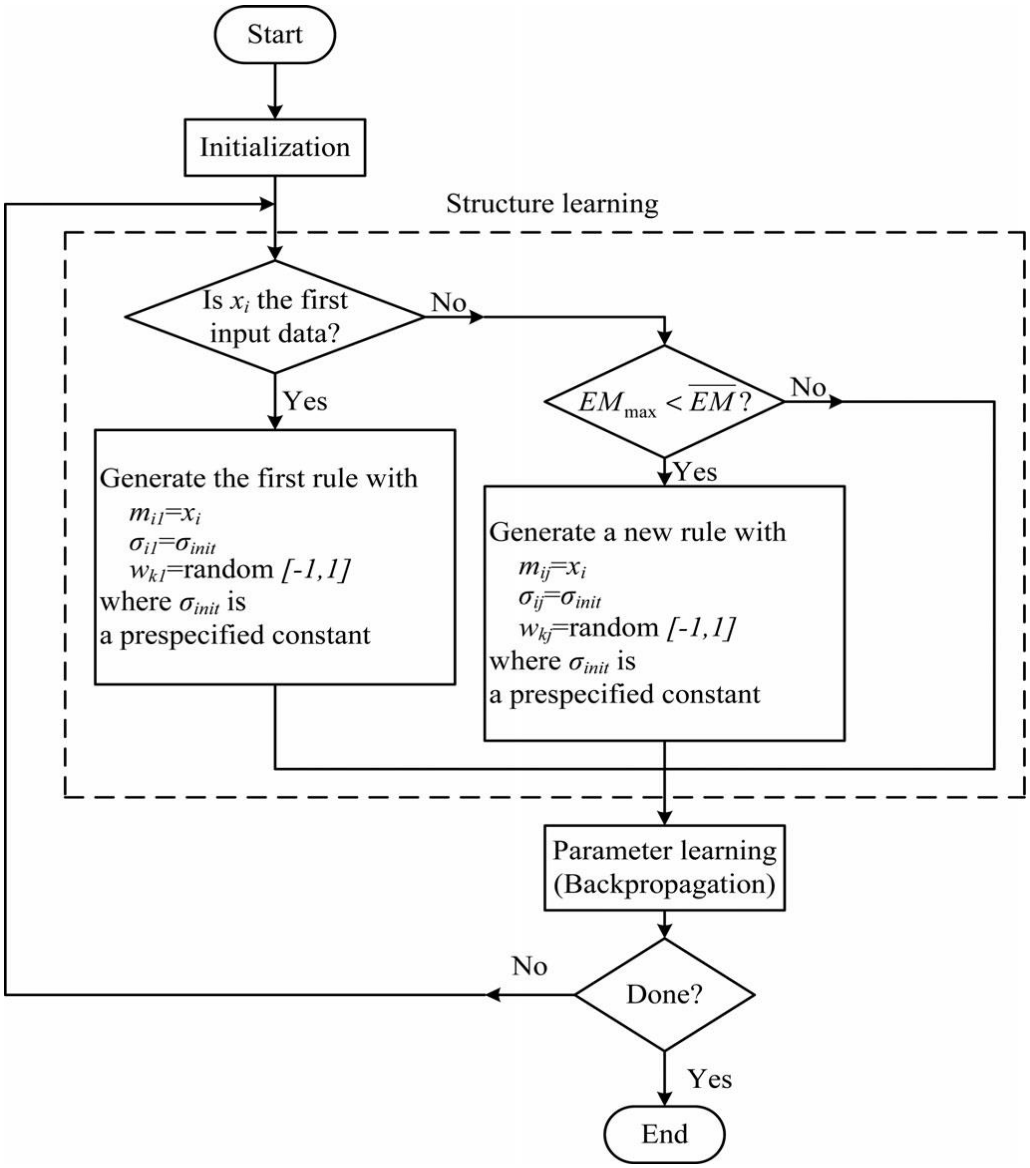


Figure 29: Flow diagram of the learning process [7]

2.5.2 Comments and Results

The authors used real measurements to train their network. The training dataset is shown in figure 30.

Pixel (bottom)	33	54	65	73	78	81	84	89
Pixel (taillight)	77	84	90	97	98	99	100	103
Distance	4	6	8	10	12	14	16	18
Pixel (bottom)	92	94	95	96	97	98	99	100
Pixel (taillight)	105	105	105	106	106	106	106	107
Distance	20	22	24	26	28	30	32	34
Pixel (bottom)	101	102	103	104	105	106	107	
Pixel (taillight)	107	107	108	108	108	109	109	
Distance	36	38	40	42	44	46	48	

Figure 30: Real distance for different pixels [7]

The measurements were taken from real driving conditions. The frames are shown in figure 31.

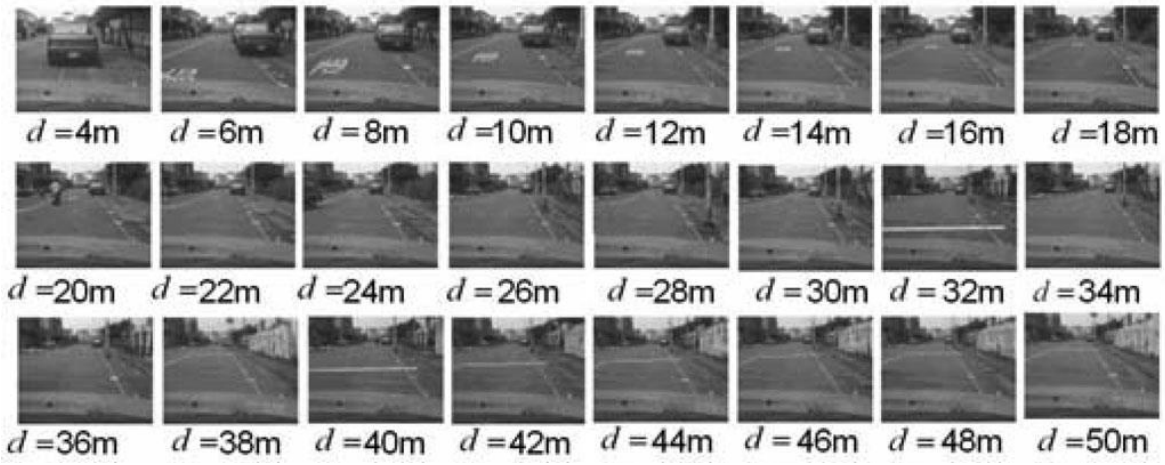


Figure 31: Images of measuring real distance [7]

The algorithm was tested using video from real driving conditions. Results of the algorithm are shown in figure 32.

Pixel	33	54	65	73	78	81
Desired output	4	6	8	10	12	14
FNFN output	4.0	6.1	7.9	10.0	12.1	13.9
Pixel	84	89	92	94	95	96
Desired output	16	18	20	22	24	26
FNFN output	16.0	18.0	19.7	22.3	24.0	25.9
Pixel	97	98	99	100	101	102
Desired output	28	30	32	34	36	38
FNFN output	27.9	30.0	32.0	34.1	36.0	38.0
Pixel	103	104	105	106	107	
Desired output	40	42	44	46	48	
FNFN output	40.0	42.0	44.0	46.0	48.0	

Figure 32: Training and testing dataset of the neurofuzzy network [7]

Accuracy rates for lane detection and front vehicle identification are shown in figure 33.

ACCURACY RATES OF THE LANE-DETECTION METHOD

Video	The total number of frames	The total number of frame of detection failure	Accuracy rate	The frequency of changing lane
1	1106	111	89.96%	4
2	869	26	97.01%	0
3	5000	269	94.62%	2

ACCURACY RATES OF THE FRONT VEHICLE IDENTIFICATION

Video	The total number of frames	The total number of frames of front vehicle at the same lane and neighboring lanes	The total number of frames of detection failures	Accuracy rate
1	1106	0	-	-
2	869	336	90	73.21%
3	5000	5000	502	89.96%

Figure 33: Accuracy rates for lane detection and for vehicle detection [7]

The overall results of the implementation are impressive, especially their tracking method that, as can be seen in their results (figure 32) can detect a front vehicle in 48 meters and estimate its distance with zero error.

An obvious weakness is that the vehicle detection depends on the lane of the road which means that if there are no lanes or if they are not distinct due to weather or corrosion, no vehicle will be detected. Another issue would be that the training dataset has values that have been determined from a specific vehicle. For different vehicle models these values might require adjustments or they could create additional error in the distance measurements. However, the advantage of the neural approach is that the network can go through a training process again for different models which means that it can adapt to different vehicles.

Regarding time requirements, the only clue mentioned is that the lane detection process was tested in a highway with image processing rate of 20 frames per second. 20 frames per second is a satisfactory rate for the specific application. However, it does not include the time needed for processing the image until the distance estimation is calculated. This means that the processing rate for the

distance estimation process is actually less than 20 frames per second, but we cannot assume that it is prohibitively less.

2.6 Overall Conclusions from the Literature

Conclusions from the literature include the following.

- A major observation is that time is not widely taken into account. The fact that we are processing a video stream means that we can take advantage of time. The leading vehicle cannot suddenly disappear from one place and appear somewhere else. We can use multiple consecutive frames in a pipelined manner to identify a car in order to increase reliability. This way even if the algorithm makes a false detection in a frame, it won't be taken into account since it would require multiple frames to verify a vehicle.
- There are many different models of vehicles with different dimensions and features. Even if testing an algorithm for a specific vehicle model indicates no error in estimating distance, it cannot be assumed that there is no error for any kind of vehicle. Installing the system on a different vehicle model might create additional error. It cannot be assumed that a system that works well installed on a short vehicle will have the exact same response when installed on a very tall vehicle. Every algorithm should be accompanied by an analysis of its possible error and its applicable distance.
- Advanced driving assistance systems have to be able to provide information in real time. If the algorithm takes 3 seconds to process each frame it means that in a dangerous situation it is not going to alert the driver on time. To be considered real time it must be able to produce multiple outputs every second. Each system should include an analysis of its processing rate (frames/second) along with the algorithm description.
- Most algorithms that intend to calculate the distance of the leading vehicle depend on the vehicle's shadow which is recognized as a very dark area on the road. The road on the other hand, is assumed to be of the same colour and intensity excluding the white/yellow lines that indicate lanes. The assumption of a well-conditioned road is not without reason. Distance estimation is mainly intended for highways and roads that support high speeds that are always in good condition. However, driving on bad roads without lanes is not rare.

- Another feature that is assumed in the image received from the camera, is that the vehicle's contour is a very distinct area compared to the background. Again, that is not without reason. However, the colour of the vehicle and its surroundings can create various conditions.

To conclude, the features that are considered for distance estimation are based on driving conditions and can be regarded as almost certain. However, almost certain means that there is always doubt. What happens if a car is almost as black as its dark shadow or if the colour of the background resembles the colour of the leading car? These conditions might create false measurements, or even complete failure in detecting the leading vehicle.

Continuously, we design an algorithm that relies on vehicle features that have been standardized (licence plates) in order to optimize the detection process. Moreover, we provide an extend analysis of the proposed algorithm's performance including time efficiency, error range and applicable distance.

Chapter 3

Novel Algorithm for Vehicle Tracking and Distance Estimation

This chapter describes the implementation of the algorithm developed for the needs of this thesis. The algorithm is written using the C++ programming language and the Eclipse development platform. The C++ language is selected because the final application is intended to run on embedded systems that are commonly programmed with C/C++. Additionally, the openCV library is used for capturing frames from the camera and for its data types that support image variables.

The proposed algorithm, that is described as follows, uses an on-board camera module to detect the leading vehicle based on the existence of licence plates. According to the law, there are licence plates of fixed dimensions in a visible place at the rear of each vehicle and it is not realistic to assume that there could be licence plates on the road without a vehicle. That is why the plates are an ideal feature to detect the front vehicle. Even if it's a bus, a truck or a car the plates are always there and the dimensions are always the same.

Our approach on the distance estimation problem includes:

- Detection of licence plates that verify a vehicle.
- Tracking of the plates with a dynamic region of interest.
- Estimating the distance based on the width of the plates.

3.1 Algorithm Overview

The algorithm is designed considering time efficiency and reliability. It makes use of multiple frames to identify a vehicle so it can't be deceived by coincidental features that resemble a car. Moreover, it uses information from previous frames in order to reduce processing and produce outputs as soon as possible. In addition, it utilizes a dynamic region of interest that focuses on the processing of the licence plates only, in order to keep the calculations to minimum.

There are three stages in the process of estimating the distance of the front vehicle (figure 34):

- Detection of candidate.
- Verification of vehicle.
- Tracking and estimating distance.

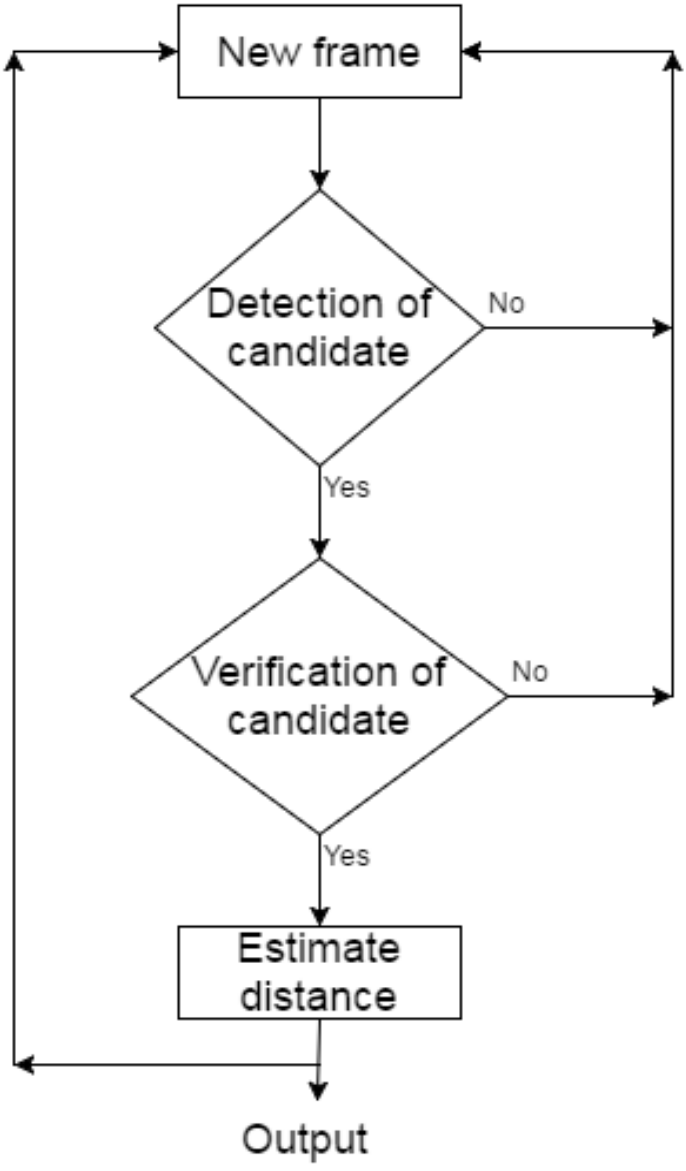


Figure 34: Brief flow diagram of the algorithm

Each frame that arrives from the camera is used for one the above stages. Initially, all frames are used for detecting a candidate. When a candidate is identified, the following frames are aimed for verification. Upon successful verification of a candidate, all the frames are used for tracking and distance estimation until the front

vehicle moves away from the camera. The functions used within the three stages of the algorithm are the following:

- Corner detection.
- Rectangle detection.
- Contour detection.
- Closest candidate.
- Dynamic region of interest for next frame.
- Distance estimation.

As follows, the above functions are analysed with code samples and figures and after that, the three stages of the algorithm are presented along with their function in order to finalize the description of the algorithm.

3.2 The Functions of the Algorithm

In this section, we describe the process that every frame from the camera is going through until the final distance estimation has been calculated.

3.2.1 Initial Editing

The initial image is received from the camera in RGB level. Before processing each frame, we crop a region of interest. This region of interest is the part of the image that corresponds to the front part of the vehicle. This way, objects that are not in front of the host vehicle are eliminated and also, the rest of the processing will be performed on a smaller image so it will take a shorter amount of time. To reduce the calculations even more the region of interest is converted to grayscale. Grayscale images require significantly less time for processing comparing to RGB images. Notice that cropping precedes the grayscale conversion. That is because grayscale conversion requires many multiplications for each pixel. We only need to grayscale the region of interest, the rest of the image would not only be pointless but it would slow down this step as well. The steps of the initial editing are shown in figure 35.

Figure 35: Initial image (a), cropped image (b), grayscale image (c)

3.2.2 Corner Detector

When the region of interest is in grayscale level, it is passed to the corner detector. The corner detector we use is a custom design that only looks for 90 degree corners or similar. This detector aims in finding the four corners of the licence plates. Other type of corner detectors that detect corners generally are not suited for this application because a common image of a car with an urban background could have thousands of corners that would require too much time to process.

To detect the four corners of the plates we rely on the properties of a 90-degree corner. As shown in figure 36, in a 90 degree corner the exterior colour is darker than the colour inside the corner. In other words, the pixels on the sides have lower values than the pixels in the middle. That is a fundamental property and is always applicable to all the corners of licence plates.

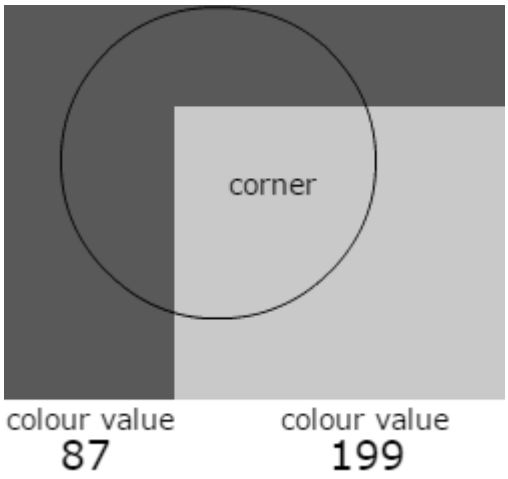


Figure 36: Common 90-degree corner

To identify these corners, we use a mask that checks the values of the pixels inside the image and if the exterior pixels are darker than the interior pixels, the corner is stored along with its features. The mask is 10x10 and checks 4 pixels for each side of a possible corner. The mask is shown in figure 37.

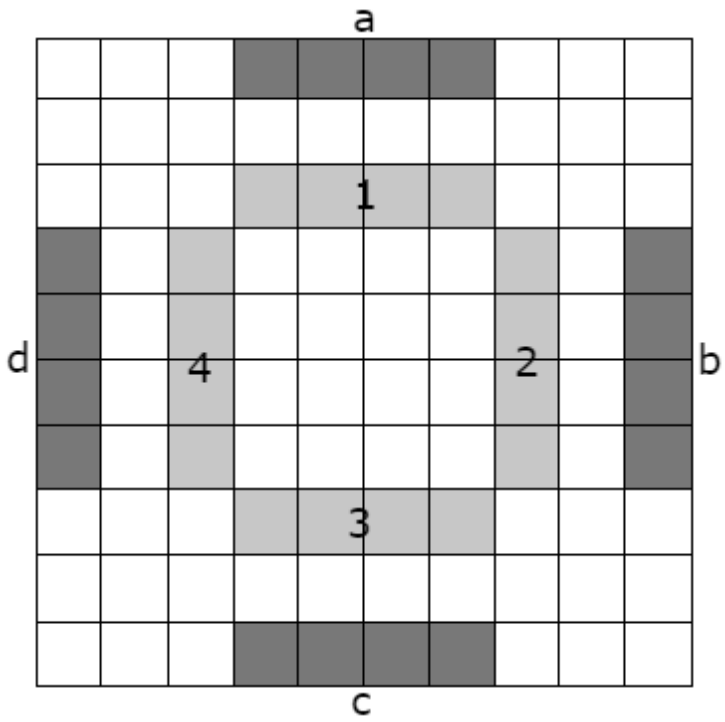


Figure 37: Corner detection mask

The mask checks if the values of all upper pixels (“a” in figure 37) are lower than the values of the pixels below them (“1” in figure 37). If the condition is satisfied, then it checks for the values of the pixels d and 4. If the second condition is satisfied as well, then a corner of type 1 is found. There are 4 types of corners in a rectangle as shown in figure 38. Respectively all four corners are checked. To improve time efficiency, if one corner is found, no more corners are checked at the same place.

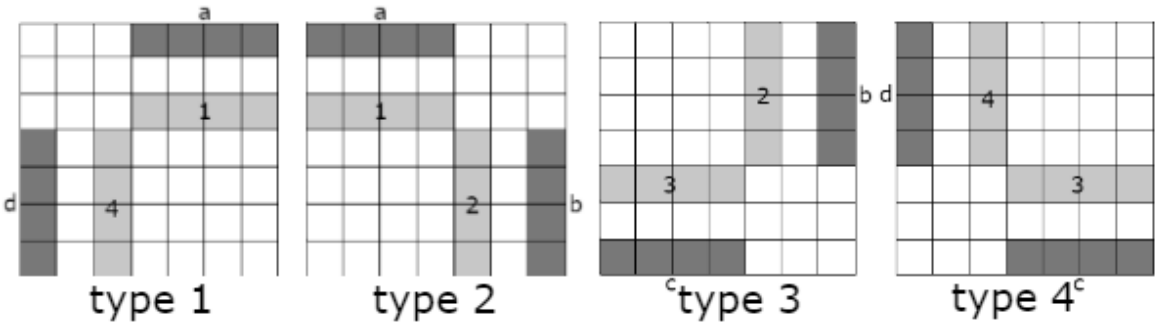


Figure 38: 4 types of corners

Note that there is a blank line between dark and light pixels. This line of pixels is not checked during the execution of the corner detector. The reason is that when there is no blank line, the detector might fail to detect slightly rotated 90 degree corners. However, using one blank line helps detecting these corners as well. Figure 39 shows an example of slightly rotated plates that have been successfully detected.

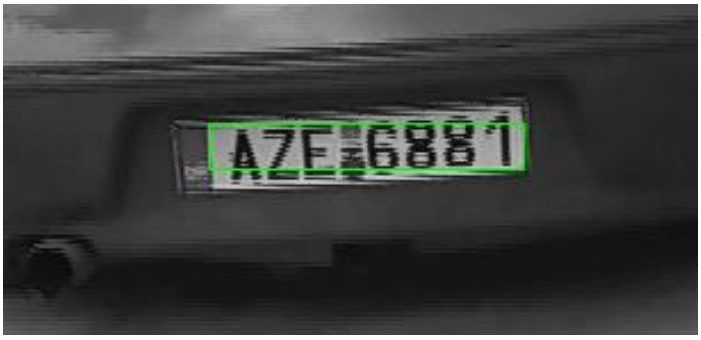


Figure 39: Corner detection on slightly rotated plates

A code sample of the corner detector including the detection of type 1 corners is shown in figure 40.

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <cstdint>
using namespace cv; using namespace std;

Mat find_corners8(Mat image, uint8_t con_frames, uint8_t darkest_white){
    Mat corner_table(250,4, CV_8U); //table to store the corners
    uint8_t corner_row=0; //keep row of corner table
    uint8_t threshold_line=8; //difference between light/dark part of corner
    uint8_t up_flag; //flag that indicates upper part of corner found
    uint8_t stop_flag; //flag that stops the corner searching
    uint8_t half=image.cols>>1;// search left corner at left side, right corner at right side _
    if(con_frames==0){threshold_line=30;} // detection of candidate stage
//start searching for corners
    for (uint8_t i=image.rows-9-3-1;i>=3;i--){//start searching from bottom
        for(uint8_t j=3;j<image.cols-9-3;j++){
        if(corner_row<corner_table.rows-1){
            up_flag=0;
            stop_flag=0;
//upper line
            if(image.at<uchar>(i+2,j+3)-image.at<uchar>(i,j+3)>threshold_line &&
            image.at<uchar>(i+2,j+4)-image.at<uchar>(i,j+4)>threshold_line &&
            image.at<uchar>(i+2,j+5)-image.at<uchar>(i,j+5)>threshold_line &&
            image.at<uchar>(i+2,j+6)-image.at<uchar>(i,j+6)>threshold_line ){
                up_flag=1;
            }
//left line
            if(up_flag==1 && image.at<uchar>(i+2,j+2)>=darkest_white &&
            (con_frames==0 || j<half) &&
            image.at<uchar>(i+3,j+2)-image.at<uchar>(i+3,j)>threshold_line &&
            image.at<uchar>(i+4,j+2)-image.at<uchar>(i+4,j)>threshold_line &&
            image.at<uchar>(i+5,j+2)-image.at<uchar>(i+5,j)>threshold_line &&
            image.at<uchar>(i+6,j+2)-image.at<uchar>(i+6,j)>threshold_line ){

                corner_table.at<uchar>(corner_row,0)=1; //store type 1 corner
                corner_table.at<uchar>(corner_row,1)=i;
                corner_table.at<uchar>(corner_row,2)=j;
                corner_table.at<uchar>(corner_row,3)=image.at<uchar>(i+2,j+2);
                corner_row++;   stop_flag=1;
            }
```

Figure 40: Code sample of the corner detector

The variable *threshold_line* keeps the value of the lowest difference between the dark and the light pixels of the corner. This variable, changes value during the execution of the algorithm. If the vehicle is a candidate, then the corner detector is strict and *threshold_line* has a high value. If plates have been identified from previous frames, the detector is more tolerant and *threshold_line* has a lower value. The values of thresholds will be discussed in the section “3.3 the three stages of the algorithm” in more detail (same for all the parameters that change value during the execution time).

It is not impossible for the detector, even if it looks only for 90 degree corners, to find so many corners that could potentially slow down the whole process. For this

reason, we put a limit to the number of corners that can be found in one image. This way, we ensure that the detector will not take too much time even for very complicated images. To make sure that the corners of the plates are always among the corners that are found, we start the mask from the bottom of the image moving up. The bottom of the image is the road and even in its worse condition, it cannot result in too many 90 degree corners. Thus, starting from the bottom means that the first corners that will be found are the road's (if any), then the front vehicle's and then the background's that will be ignored if too many. An example is shown in figure 41. The maximum of 250 corners is used, even though it is always more than enough.

Figure 41: Corners ignored at the background

Finally, the output which is what is returned from the processing of the region of interest from the corner detector is a 250x4 table where each line is a corner with the following properties:

- *Corner type*: is a value from 1 to 4 that indicates the type of the 90-degree corner. Consider that a rectangle has four different 90 degree corners. 1 is the upper left type, 2 the upper right, 3 the down right and 4 the down left corner (figure 43).
- *Row*: the number of the row of the corner.
- *Column*: the number of the column of the corner.
- *Colour*: is the value of the white pixel inside the corner. This value is used later when finding rectangles because all four corners of the plates have approximately the same value of white.

The corner detector does not need to output an image with the corners, figure 41 shows the output of the detector which was produced only for demonstration.

Figure 42: Corners found with the custom corner detector

3.2.3 Rectangle Detector

The rectangle detector is a function that has as input a table of corners. It processes all the fields of the corner table and finds which of the corners can form rectangles. To form a rectangle it requires two corners, an upper left and a down right or an upper right and a down left. Since the type of the corner is the first field of the corner table, it means that corners of types 1 and 3 or 2 and 4 can possibly form rectangles. Figure 43 shows the types of corners.

Figure 43: Types of corners

The conditions that need to be satisfied for a pair of corners to form a rectangle are the following:

- It must be physically possible. Meaning that to form a rectangle with type 1 and 3 corners, the row of the type 1 must be a lower value than the row of the type 3 and column of type 1 must be a lower value than the column of type 3. Respectively, it must be physically possible for corners of type 2 and 4 to form rectangles as well.
- There is a maximum and minimum value for the width of the rectangle. That is because there is a range in the size of licence plate rectangles in the images from the camera. Not all sizes can be considered as candidates for plates.
- The width to height ratio is fixed. We only accept rectangles that could be plates. The fixed width to height ratio for plates is between 3 and 6. Rectangles with different ratio are ignored.
- The light colours of the two corners that form the rectangle need to be approximately of the same value. The fourth field of the corner table keeps the value of the colour of the light colour for each corner. All four corners of the licence plates have the same colour (white) inside. To form a rectangle

from two corners, these two corners must have approximately the same colour inside as well, otherwise they are ignored.

- If a rectangle has been identified as plates and is currently being tracked, it is not allowed to change its width violently. That is because a car that is currently being followed cannot change its distance instantly. The size of the plates of the front vehicle is changed gradually over time and the plates' width is expected to have low deviation between consecutive frames.

A code sample of the rectangle detector is shown in figure 44. The sample is from the detection of rectangles from type 1 and 3 corners.

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <cstdint>
using namespace cv;  using namespace std;

Mat find_rectangles2( Mat corner_table, uint8_t plates_width, uint8_t frames, uint8_t con_frames ){
    Mat rectangle_table(100,5, CV_8U);//table to store the rectangles
    const uint8_t shape1=2;//width/height low limit (>2)
    const uint8_t shape2=6; //width/height high limit (<6)
    const uint8_t max_plate_width=170;//max plate width
    const uint8_t min_plate_width=35; //minimum plate width
    const uint8_t min_height=7; //minimum plate height
    uint8_t rectangle_row=0; //keep the row of the rectangle_table
    const uint8_t max_white_difference=5;//difference in white of two corners of the same rectangle
    uint8_t change=6;//allowed change in width for 2 consecutive frames
    if(con_frames!=frames){change=3;} //verification stage, strict rules

    for(uint8_t i=0;corner_table.at<uchar>(i,0)!=0;i++){
//up left to down right corners. type 1 and 3
        if(corner_table.at<uchar>(i,0)==1){
            for(uint8_t j=0;corner_table.at<uchar>(j,0)!=0;j++){
                if(corner_table.at<uchar>(j,0)==3){
                if(rectangle_row<rectangle_table.rows-1){
                    if(corner_table.at<uchar>(j,1)>corner_table.at<uchar>(i,1) &&
                    corner_table.at<uchar>(j,2)>corner_table.at<uchar>(i,2)){
                        const uint8_t width=corner_table.at<uchar>(j,2)-corner_table.at<uchar>(i,2);
                        const uint8_t height=corner_table.at<uchar>(j,1)-corner_table.at<uchar>(i,1);
                        const uint8_t shape=width/height;
                        if(width>min_plate_width && width<max_plate_width && height>min_height &&
                        shape<shape2 && shape>shape1 &&
                        ((width<plates_width+change && width>plates_width-change) || (plates_width==0 &&
                        abs(corner_table.at<uchar>(i,3)-corner_table.at<uchar>(j,3))<max_white_difference))){
                            rectangle_table.at<uchar>(rectangle_row,0)=corner_table.at<uchar>(i,1); //store
                            rectangle_table.at<uchar>(rectangle_row,1)=corner_table.at<uchar>(i,2);//rectangle
                            rectangle_table.at<uchar>(rectangle_row,2)=corner_table.at<uchar>(j,1);
                            rectangle_table.at<uchar>(rectangle_row,3)=corner_table.at<uchar>(j,2);
                            if(corner_table.at<uchar>(i,3)>corner_table.at<uchar>(j,3)){
                                rectangle_table.at<uchar>(rectangle_row,4)=corner_table.at<uchar>(j,3);}
                            else{rectangle_table.at<uchar>(rectangle_row,4)=corner_table.at<uchar>(i,3);}
                            rectangle_row++;
```

Figure 44: Code sample of the rectangle detector

The output of the detector is a table of rectangles that are candidates for being licence plates. The fields of the rectangle table are:

- First 4 fields are the coordinates of the type 1 and 3 corners that define the rectangle (column and row of each corner).

- Fifth field is the lowest value between the light pixels of the two corners that formed the rectangle. This value is used later for conversion to binary (explained in the next section).

The rectangle detector does not need to output an image with the rectangles, figure 45 shows the output of the detector which was produced only for demonstration.

Figure 45: Rectangles found using the detector

3.2.4 Contour Detector

The contour detector takes as input a table of rectangles and the grayscale region of interest. Its purpose is to check if there is a dark contour around each rectangle that could be identified as the black contour of the licence plates. As shown in figure 46, all licence plates have a black contour around them no matter the colour or the type of the vehicle.

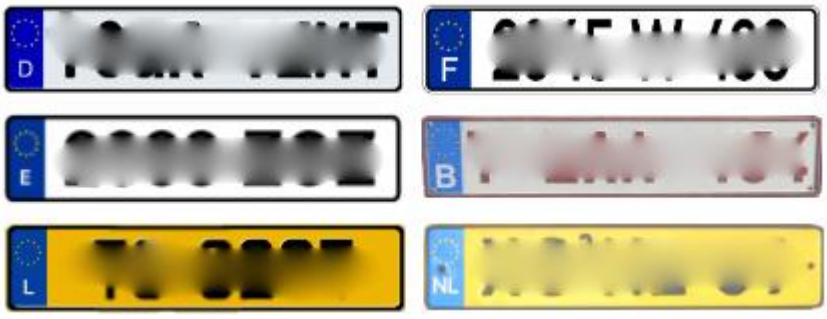


Figure 46: Contour of licence plates.

To check if there is a black contour in the rectangles that have been found from the rectangle detector we convert the image to binary. The conversion to binary however, needs a threshold value which is not easy to find. One option is to set a middle value. This means that all pixels with value lower than 255 / 2 are turned to black and all pixels with value higher than 255 / 2 are turned to white. The goal of the binary image is to separate the white part of the plates from the black part of the plates. On cloudy days however, a fixed threshold could turn the white part of the plates to black and during shiny days the black part of the plates to white. To be able to distinguish the contour under any illumination conditions we cannot use a fixed threshold. Figure 47 shows an example of using a fixed threshold for conversion to binary.

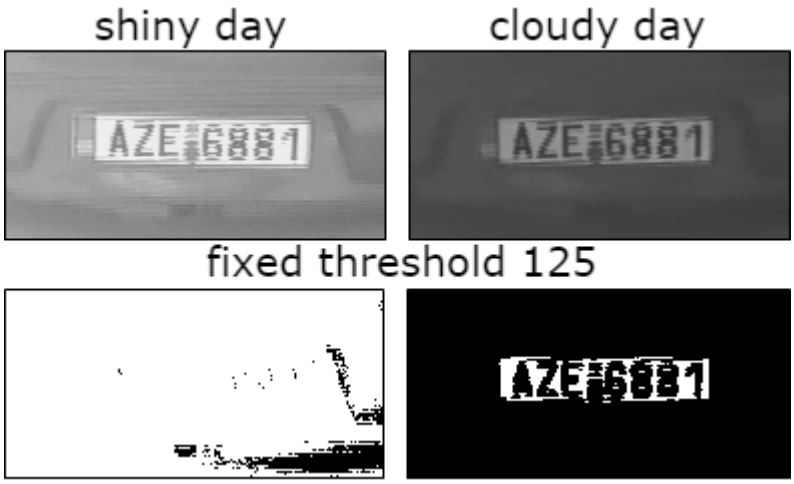


Figure 47: Binary conversion with fixed threshold

Another option would be to calculate the average value of all the pixels from the image around each rectangle and use it as threshold. This method would probably work sufficiently, but there is no proof that the average value will always be a suitable threshold. Even if the average was always a suitable threshold, the processing for calculating an average for every rectangle would create additional overhead and would eventually slow down the processing.

To overcome this problem, we use a custom method with adaptive threshold. In the fifth field of the rectangle table we have kept the value of the light pixels inside the corners. This value corresponds to the white part of the plates and is the ideal value for threshold that works under any illumination. No matter how dark the plates are, if we know the exact value of the supposedly white part, we can perform the conversion to binary successfully. Thus, to check for a contour around each

rectangle we use each rectangle's fifth field that contains the colour of the light part of its corners. Figure 48 shows an example of our custom method for binary conversion with adaptive threshold.

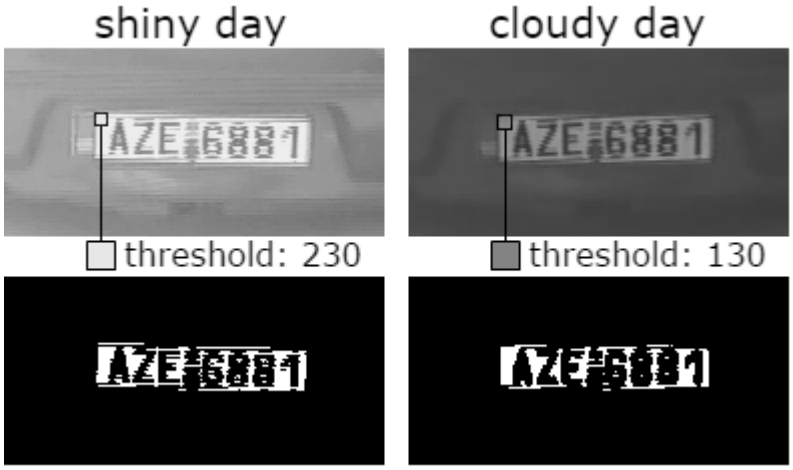


Figure 48: Adaptive threshold

After the conversion to binary all pixels are black or white with values 0 or 1 respectively. We look for continuous black pixels around the rectangle in order to check if there is a black contour. Rectangles without contours are eliminated. Another feature that is checked from the contour detector is the colour of the four corners. In licence plates, the inside of the four corners has approximately the same value. These values are checked here as well.

The output of the contour detector is a table of rectangles exactly like the output of the rectangle detector but without the rectangles that lack contours.

More examples of the adaptive threshold are shown in figure 49.

Figure 49: Various adaptive threshold examples

3.2.5 Closest Candidate

This function takes as input a table of rectangles with verified contours. Its purpose is very simple. In case there are many rectangles with contours (candidates) pick the one with the longest width. The reason this function exists is to pick a candidate if two candidates arrive at the same time. Meaning that if two cars appear in the region of interest at the same time, the one closest to the host car (longer width) will be selected for tracking.

3.2.6 Dynamic Region of Interest for Next Frame

To reach this point, it means that there has been exactly one rectangle that is a possible candidate for being licence plates. This candidate has been through the following tests:

- Has at least two corners that formed a rectangle.
- Rectangle is of the correct shape regarding width to height ratio.
- Rectangle has a dark colour contour around it.
- It is the one closest to the host car.

To verify this rectangle as plates, we track it through time. We use multiple consecutive frames in a pipelined manner in order to make sure that the same rectangle is not a coincidental shape formed from the background but a robust rectangular shape of correct width to height ratio that actually travels along with the host vehicle. We use a dynamic region of interest for the next frames that follows the candidate and makes sure that its movement is steady over time. At this point, we assume that a plates-like shape with steady movement over time that is in front of the host vehicle, undoubtedly indicates licence plates and therefore a leading vehicle.

To follow the movement of the front vehicle we use a dynamic region of interest which is actually a moving window that follows the plates inside the point of view of the camera. During the candidate verification process, we use a small window (8 pixels around the rectangle) that follows the candidate, because we want the verification process to be strict. Upon successful verification, we use a bigger window (18 pixels around the rectangle) because the plates have been verified. In addition, a bigger window can ensure the tracking of the plates even when the front vehicle manoeuvres. Note that the size of the window is directly connected to the

performance of the algorithm. For each frame, we find the window which is the place that we are going to look for the plates in the next frame. If it takes two seconds to process the frame then the front car might have moved away from the window (during processing of the frame). The faster the processing of each frame, the smaller the size of the window can be.

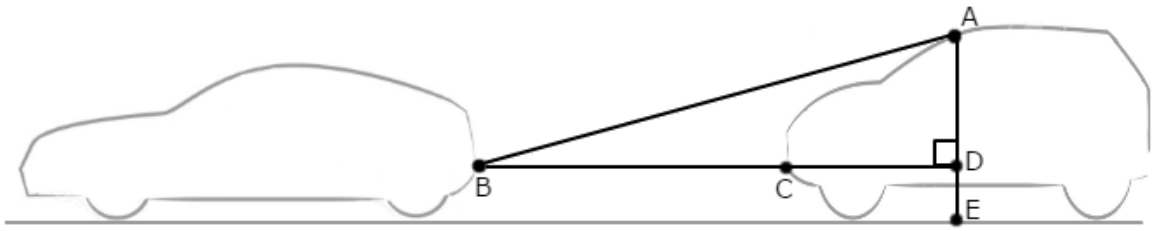
This dynamic region of interest is the only part of the image that is being processed in each frame. Apart from eliminating the unwanted background it also allows to the whole algorithm to run significantly faster since instead of the whole image only a small part is processed. Figure 50 shows the dynamic region of interest during candidate detection, candidate verification and upon successful verification, respectively.

Figure 50: Dynamic region of interest for the three stages of the algorithm

The number of frames that is used for the verification process is discussed later on in section 3.3.2 “Verification of vehicle”.

3.2.7 Distance Estimation

To reach this stage it means that the plates of the leading vehicle have been successfully identified and their coordinates and dimensions are known. To estimate the distance, we rely on the fixed dimensions of the plates. Figure 51 shows a typical situation of a car preceding another car.



A: Location of the camera
B: Location of license plates
C: Front part of host vehicle
D: Vertical projection of B on AE
E: Ground

AB: Distance from plates to camera
AE: Distnace from camera to the ground
DE: Distnce from plates to the ground

Figure 51: Distances during driving conditions

To estimate the requested distance BC we need to calculate some other lengths. At first, we find the length AB. This length is a straight line from the camera to the plates. To find this distance, we conduct an experiment. We create a replica of licence plates and we put it at known distances from the camera. Then, we measure the lengths in pixels at the pictures. Figure 52 shows the pictures taken from the camera along with their length in pixels.

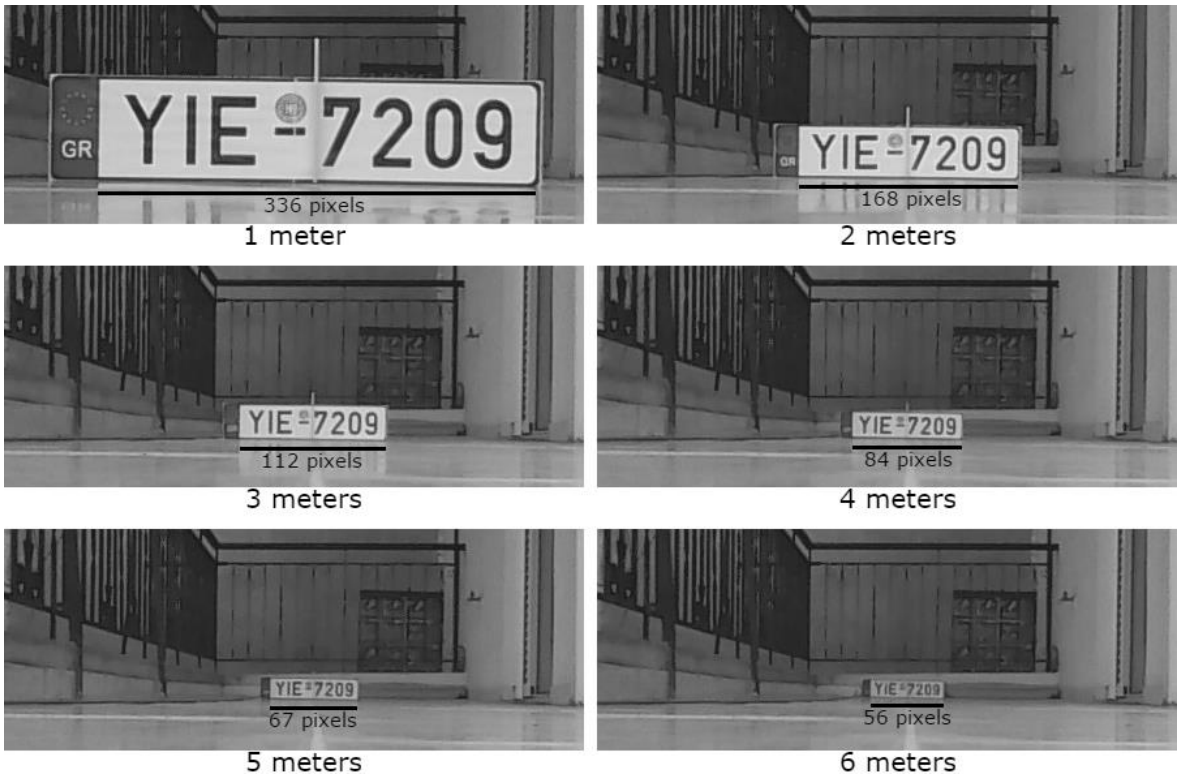


Figure 52: Fixed size plates at known distances and their size in pixels

This experiment proves that the distance of an object from the camera is inversely proportional to the number of pixels that correspond to the object width in the picture. For instance, as we can see in figure 52 when the meters double, the pixels are reduced by half. One meter distance corresponds to 336 pixels, two meters correspond to 168 pixels and 4 meters correspond to 84 pixels. This means that the function that translates distance to pixels is linear and thus, we can find the exact distance of an object by counting its pixels in the picture. For this example (figure 52) dividing 336 (1 meter) by any number of pixels will result in its distance from the camera (in meters). The only information required for this method to work is a known distance along with its corresponding pixels of the object in the picture (e.g. 1 meter = 336 pixels). The length in pixels from the picture might have an error since the plates could be slightly rotated or bent. Error in estimating distance is discussed later on, in section 4. To conclude, since the plates' dimensions are fixed, by counting the pixels in the image we can calculate the distance and thus, AB can be found. Next, we try to find DE.

Length DE is not fixed for any vehicle. Actually, this length depends more on the front car than on the host car. DE is the vertical length from the plates to the ground.

Figure 53 shows this length and how it changes according to the location of the plates of the front car.

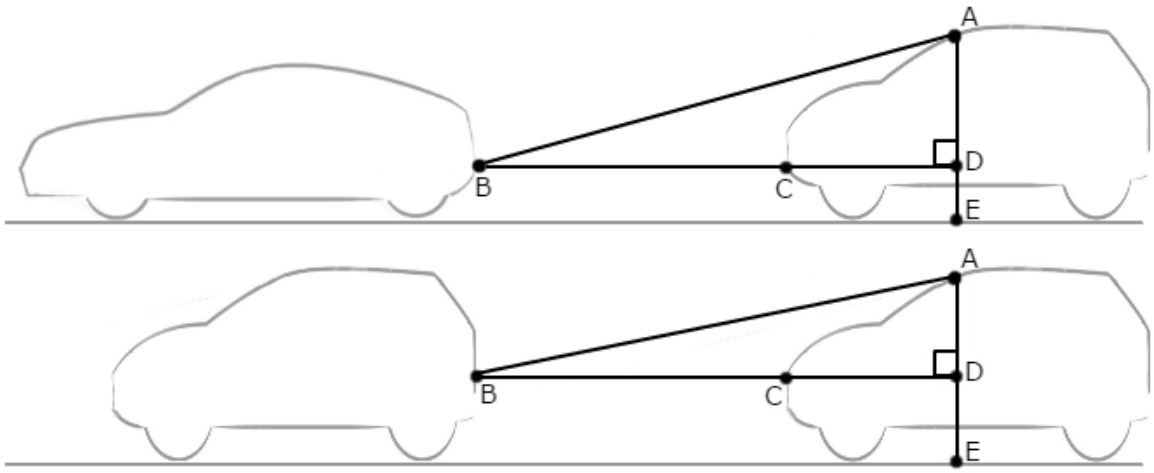


Figure 53: Change in DE according to the leading vehicle

This is not a fixed value since front vehicles can vary. However, we can estimate a range of its possible values. Its lowest value is acquired when a vehicle with low placed plates is followed and its highest value is acquired when a vehicle with high placed plates is followed. Figure 54 shows both these cases. According to this range we use a middle value for DE and later on, we calculate the limits of the error that might occur when DE is different. The error is discussed in section 4.

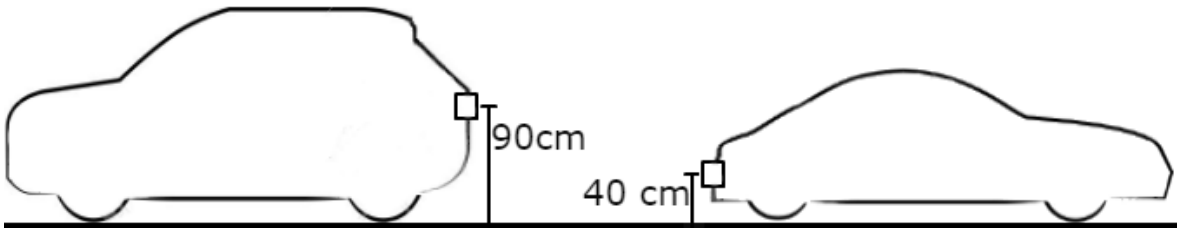


Figure 54: Lowest and highest value of DE

The values in figure 54 are actual measurements from real vehicles (an SUV is used for high placed plates and a low car is used for low placed plates). DE is then determined as 65 cm.

So far, we know the lengths AB and DE (figure 51). The length AE is the height of the camera from the ground and even if it varies for each vehicle, once the system has been installed, AE remains a constant. For this length, we can either use an average value (and estimate possible error when the actual value is different) or we can measure AE manually. This value would only be needed to be measured once, because it is not expected to change after installation. For this reason, we measure the length AE and we input it on the algorithm where it is stored and used ever after. This is part of the calibration process that only needs to be performed once so that the algorithm can adapt on the specific host vehicle and camera (explained in section “3.4 calibration process”).

Since AE is now considered known, AD can be calculated by subtracting DE from AE. At this point, we already know AB and AD so we can use the Pythagorean Theorem to calculate BD. Next, we need to find the length CD. CD same as AE are properties of the host car. We can either measure CD manually one time during the calibration process, or we can use an average value and estimate the error that might occur when the actual value is different. The proposed method for every car is to complete the calibration process so that the algorithm can adapt on the specific host vehicle model and on the camera module. This way, the distance measurement will be much more accurate. Finally, CD is considered known from the calibration process (section 3.4) and thus, the requested length BC = BD – CD.

3.3 The Three Stages of the Algorithm

In this section, we explain how all the algorithm’s functions are used during execution time. The algorithm has three stages (figure 34):

- Detection of candidate
- Verification of vehicle
- Tracking and estimating distance

Each stage uses the functions in a slightly different manner in order to perform optimally.

3.3.1 Detection of Candidate

This is the initial and default stage of the algorithm. During this stage, there is a fixed region of interest. This area corresponds to the front of the car. It reduces

calculations by processing a smaller part of the image as well as eliminates automatically side objects.

The size of the region is set to 250x250 which was found satisfactory after testing it with 640x480 images. Note that we intend to detect licence plates only, thus the whole car does not need to be present, only the plates. Figure 55 shows the static region of interest during the detection of candidate stage.

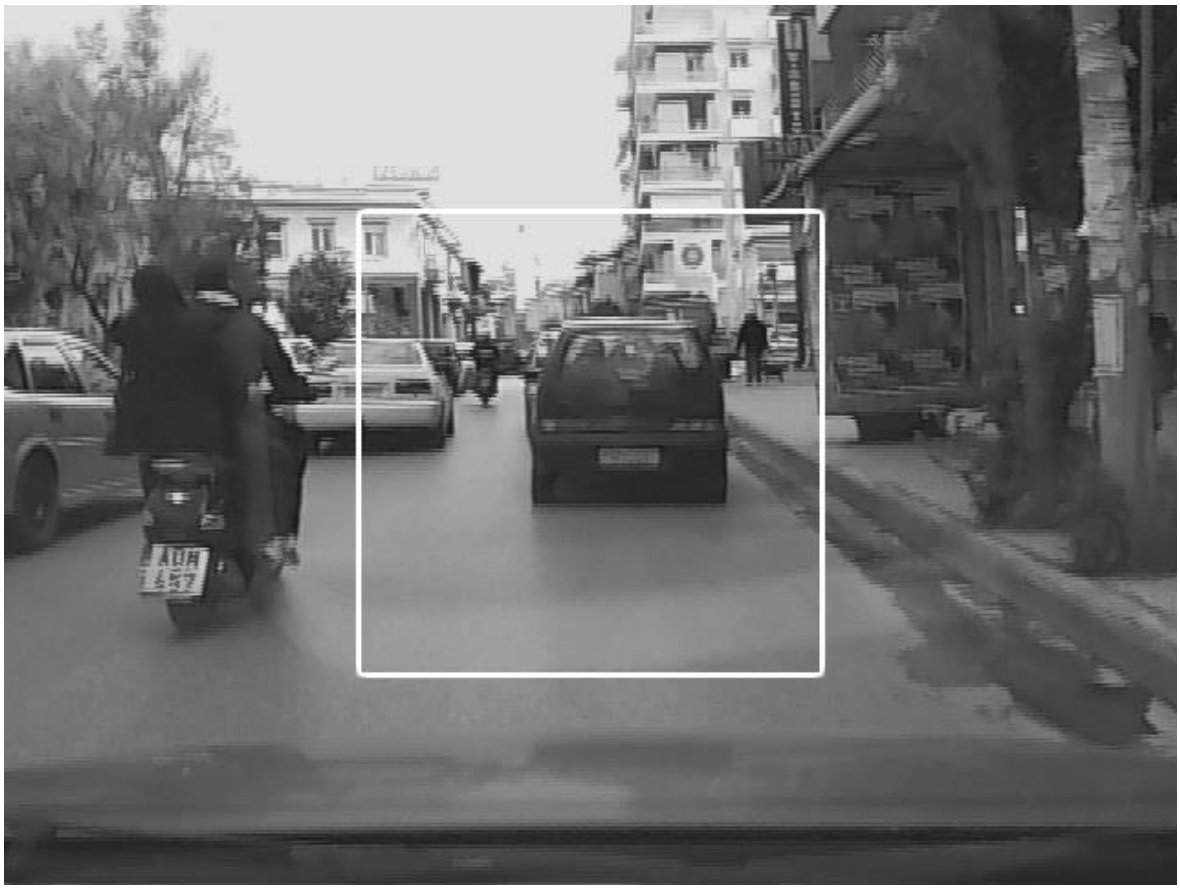


Figure 55: Static region of interest for 640x480 image

This region which is in grayscale after initial editing (explained in section 3.2.1) is passed to the corner detector. At this stage, the corner detector uses the value 30 as threshold. This means that in order to detect a corner, its dark pixels and its light pixels must have a difference of 30, the least. The reason we use a high threshold for this stage is because at this point we look for solid corners. We don't want the algorithm to be concerned with coincidental shapes or weak corners. Thus, the corner detector at this step produces just few well-shaped corners. These corners are then checked for rectangles and contours by the two detectors respectively.

This stage runs on default on consecutive frames until one frame provides a candidate. When this happens, the algorithm uses the coordinates of the candidate to define a new dynamic region of interest and moves to the verification stage. Figure 54 shows a possible candidate. Note how the region of interest changes on two consecutive frames upon detection of a candidate.

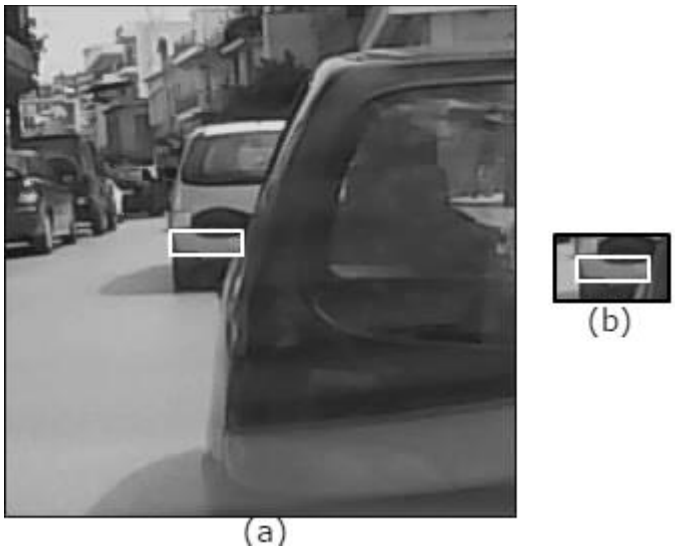


Figure 56: Detection of candidate (a), verification of candidate (b)

3.3.2 Verification of Vehicle

At this stage, there is already an available candidate and the purpose now is to identify this candidate as plates or to confirm that it is not plates and go back to the previous stage in order to find another candidate. To do so, we take advantage of time and features from previous frames. The region of interest now is 8 pixels around the candidate (figure 56 b).

The corner detector during this stage is a bit different. If the candidate that was found in the previous frame during the detection of candidate stage is indeed plates, then not only its corners will be inside the region of interest in the current frame but the corners in the current frame will have approximately the same colour as the corners in the previous frame. At this stage, the corner detector uses a colour threshold for the light part of the corners and thus, it only finds corners that resemble the corners from the detection of candidate stage. This step eliminates the possibility of changing candidates inside the region of interest during the verification process. Moreover, the corner detector at this stage has a significantly lower threshold of value 8 (previously 30 at the detection of candidate stage). The new lower threshold

guarantees that all corners will be found even the weak ones or the slightly distorted ones. So, the corner detector at this stage looks for corners that resemble in colour with the initial candidate even if they are weak. Figure 57 shows the difference of the corners found in the two stages. Note the difference in the amount of corners found by the detector due to lowering the threshold.

Figure 57: Corners from detection of candidate (a), corners from verification of vehicle (b)

The rectangle detector works the same way as in the previous stage but in this stage, it has one more added condition. Rectangles in this stage are not allowed to change size violently. Along two consecutive frames the size of the candidate is not allowed to change more than 3 pixels. This stage lasts less than a second (approximately 0.5 seconds discussed later on “5.1 Time efficiency”) and multiple frames are used. If the candidate is indeed a vehicle, during two consecutive frames it cannot move so much that the plates’ size will be altered considerably.

The contour Detector works the same as before. Uses adaptive threshold for conversion to binary, checks for contours and colour resemblance of the four corners.

The number of frames that is used for the verification process is directly connected to the reliability of the algorithm. The more consecutive frames are used for verification the more certain it is that a rectangle is actually plates. However, the rules during verification are stricter and more processing happens during verification than after it. This means that the verification process is time consuming so we don’t want to check too many frames. Generally, one second is enough time to verify a

vehicle. But if the algorithm is slow and can process for instance 3 frames per second then it might need more than a second. Results from our algorithm will be presented later on in section 5 but we used 10 consecutive frames for verification which actually takes a lot less than a second.

3.3.3 Tracking and Estimating Distance

This is the final stage of the algorithm. To be in this stage, it means that plates have been identified and confirmed through time. This stage aims in tracking the front vehicle and calculating its distance. If the tracking fails it means that the vehicle moved away and the first stage of this algorithm will be executed in order to find another candidate. If the tracking succeeds, then the distance will be calculated and a new region of interest will be acquired from the next frame.

During this stage, the corner detector works the same as in the previous stage with a very low threshold so the front vehicle won't be falsely missed. The rectangle detector works the same as before as well as, but now that the vehicle has been verified the size of the plates is allowed to change more during two consecutive frames (previously 3, now 6). This way the front vehicle is tracked even if it makes sudden moves like sudden stop or accelerate. The size of the region of interest is bigger now for the same reason (previously 8, now 18).

The contour detector at this stage is a bit different. Its initial goal is to check the contour of the plates and to check the colour of the four corners. Now that the plates are identified we no longer need to check for the contour. We find the corners that form a rectangle at (approximately) the same size as before, at (approximately) the same place as before. The fact that the four corners are of the same colour is enough to ensure that it is the same rectangle thus, during tracking we only check rectangle, size, location and colour. This simplifies a lot the processing and the algorithm can run faster.

3.4 Calibration Process

This process in intended to run one time during the installation of the system on a vehicle and its purpose is for the algorithm to get familiar with the camera module that is used, the fixed dimensions of the licence plates and the host vehicle. During the implementation, we used the standardized European plates and a specific car model. However, there is no reason for the algorithm not to work for any fixed

dimensions plates or any vehicle model as long as the calibration process has been completed.

During this process, we place the desired plates at a known distance from the camera and we let the algorithm count the width in pixels as shown in figure 58.

Figure 58: Width of plates at exactly one meter

The algorithm stores this value in a file and retrieves it every time the system is powered on. Then, to calculate the distance from the camera to the plates, it finds the number of pixels that correspond to the width of the plates at unknown distance and it divides the original value by the newly found value. The outcome is the distance in meters. For instance, if the plates are 336 pixels long at one meter, and the current size is 56 pixels it means that the distance is 336 / 56 = 6 meters long. This step does not require the system to be installed on a car.

Other measurements that need to be obtained manually one time when installing the system on a car are:

- The vertical distance from the camera to the ground (AE in figure 59).
- The horizontal distance from the camera to the front part of the car (CD in figure 59).

These values are inserted to the algorithm through a user interface with an intuitive menu.

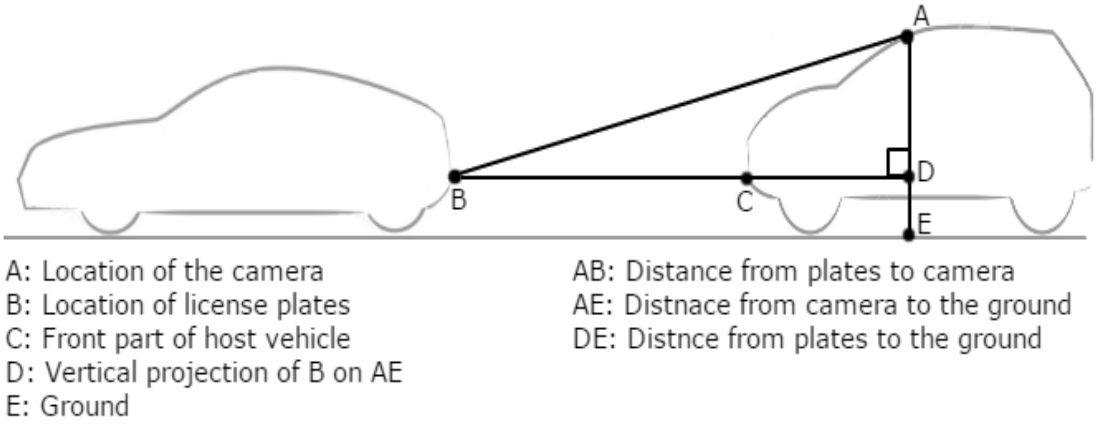


Figure 59: Distances AE and CD (needed for calibration)

3.5 General Observations

This section presents some observations for the whole implementation of the algorithm. There are also comments and notes on the algorithm's behaviour during various conditions.

3.5.1 Illumination Conditions

A successful algorithm for licence plate detection has to be able to identify plates under any illumination conditions that might occur while driving. Figure 60 shows successful detections under different illumination conditions.

Figure 60: Successful detections during various illumination conditions

As long as there is a distinct difference between the black contour of the plates and the white colour inside, the algorithm will most likely find the plates.

During night time however, the conditions are a bit different because of darkness and the lights of the road. Figure 61 shows successful detections during night time.

Figure 61: Successful detections during night time

3.5.2 Colour Variation

The colour of the vehicle should not affect the efficiency of the algorithm as long as the black contour of the plates is visible. However, the plates' contour might vanish from the image if the distance is too long. Fortunately, in this case the colour of the vehicle will take its place as long as it is darker than the white part of the plates. If the vehicle is as white as the plates then it will be recognized only if the plates' contour is visible. Figure 62 shows the detection of a white vehicle and how it fails when the distance becomes longer and the contour is no longer distinct enough.

Figure 62: Detection of a white car (left) and failure (right)

Generally, the algorithm will fail if there is no contour. The distance that can cause this failure depends on the camera. If the camera can pick up the black contour, the algorithm will not fail. Using a low cost 640x480 webcam the detection of a white car failed after 5 – 6 meters from the camera.

3.5.3 Different kinds of Plates

Other than the traditional licence plates, there are plates with different colour that indicate a property of the vehicle (e.g. taxis or government vehicles). These plates are usually yellow with black letters or white with red letters etc. There is no reason for the algorithm to be confused by this matter. As long as there is a distinct contour of darker colour than the interior of the plates, the algorithm will not fail. Figure 63 shows an example of the detection of yellow plates.

Figure 63: Detection of yellow plates

3.5.4 Absence of Edge detection

Edge detection is widely used for licence plate detection since the plates have many edges because of the black characters on white background. In our implementation, we completely ignore features from the characters. We do that for two reasons:

- The reflective material that is used on the plates can sometimes shine so much, especially due to night lights, that the characters become invisible.
- Long distances can cause the characters to almost vanish and become one with the background.

Even if the characters might become unnoticeable, the plates' contour is still very distinct. For this reason, we decide to rely on the plates' contour and shape and not on the characters. Figure 64 shows an example for each case (distance and shinning) where the detection is successful, even without visible characters.

Figure 64: Plates detection without distinct characters

3.5.5 Weaknesses

We have identified two weaknesses in the implementation of this algorithm. The first is the white colour of the front car which was explained in detail in section 3.5.2. The second is a feature that we use for tracking.

As mentioned before (section 3.3.3) when tracking a vehicle, we use the colour of the corners from the detection of candidate stage as a threshold and we proceed with finding corners in the following frames with approximately the same colour. This might create an error when the illumination conditions change instantly. For example, if a car that is being tracked drives through a tunnel during daytime, the colour of the corners inside the tunnel will not resemble the initial corners. In this case the tracking fails, the algorithm looks for a new candidate with the new illumination conditions and the new candidate is verified and being tracked.

The algorithm is not flexible enough to keep the tracking going during such sudden changes, but it is very adaptable and it picks up on the new conditions almost instantly. The whole process of losing the front car and finding it again with the new illumination takes as much time as it takes to verify a new candidate which is approximately 0.5 seconds (time efficiency is discussed later on in section 5.1).

Chapter 4

Error Analysis and Applicable Distance

This chapter presents an analysis of the possible error that might occur in the measurement of the distance. At first, we mention all the reasons that the distance measurement may be skewed, then we estimate the possible error for each occasion and finally, we calculate the overall range of the error. After that, we discuss the applicable distance of the algorithm which is the maximum distance for which the algorithm is able to track the leading vehicle.

4.1 Error Analysis

There are four factors that might affect the distance measurements of the front vehicle:

- The height of the plates from the ground.
- The gradient of the road.
- Slightly rotated or bent plates
- Error from not completing the calibration process

4.1.1 The Height of the Plates from the Ground

The height of the plates from the ground is a distance that depends completely on the front vehicle and it cannot be measured because it is different for each vehicle model. For this distance, we have no other choice but to use an average value and calculate the range of possible error when the value is different. As shown in figure 65, we measure a vehicle with very high placed plates (SUV) and a vehicle with very low placed plates (low car) and we determine an average value of 65 cm.

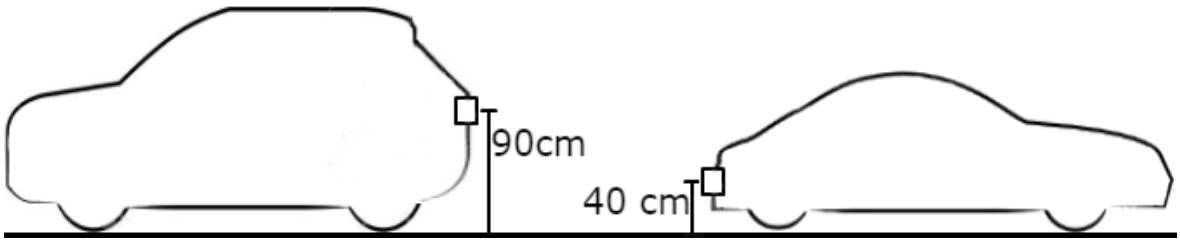


Figure 65: High placed plates (left), low placed plates (right)

To calculate the error of using an average value, we place two vehicles at known locations and we measure all distances manually. Then, instead of using the actual (plates – ground) distance, we use the average (65cm) and we calculate the error. The experiment is performed at two meters for high placed (90cm) and low placed (40cm) plates. As it can be seen in figure 66, when the average DE is used at 2 meters distance, it creates an error of 6 cm for low placed plates and 4cm for high placed plates. We have determined that plates can be placed at 40 – 90 cm from the ground which means that all other possible values are actually closer to the average and thus, their error is lower. This means that the values 40 and 90 create the maximum error.

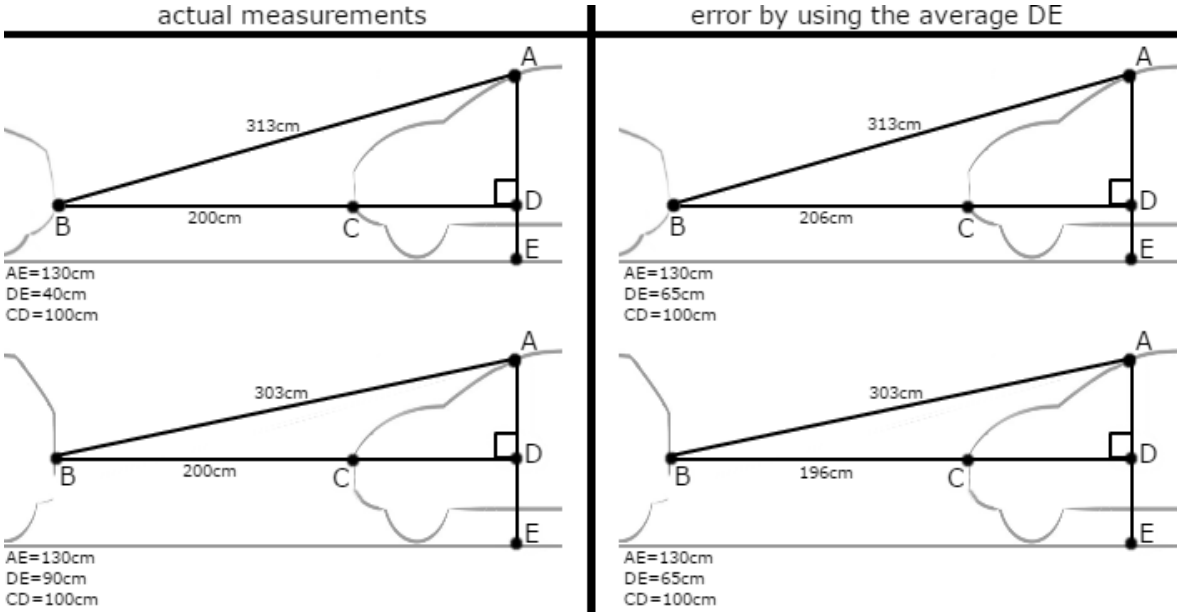


Figure 66: Error measurement at two meters due to average DE

After that, we recreate the experiment but for distances of one to eight meters. The maximum error for each distance is shown in figure 67.

Distance between vehicles	Maximum error
100cm	9.4cm
200cm	6.3cm
300cm	4.8cm
400cm	3.8cm
500cm	3.2cm
600cm	2.7cm
700cm	2.4cm
800cm	2.1cm

Figure 67: Max error in different distances due to average height

4.1.2 Roads with gradient

Our approach to the distance estimation problem is based on the Pythagorean Theorem. This means that the two vehicles are considered to be on the same level. If the road has a gradient (uphill or downhill) the Pythagorean Theorem still applies as shown in figure 68.

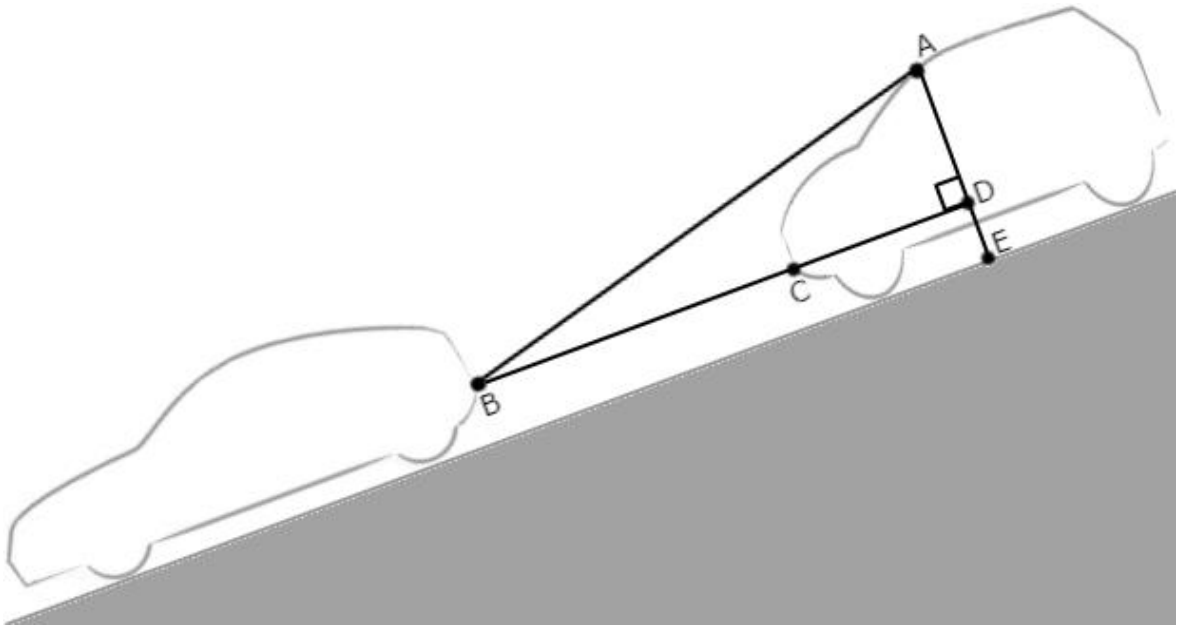


Figure 68: Pythagorean on roads with gradient

The only case that the Pythagorean Theorem does not apply is if the two vehicles are on different gradient. That could happen only in the beginning or the end of a change in gradient of the road as shown in figure 69.

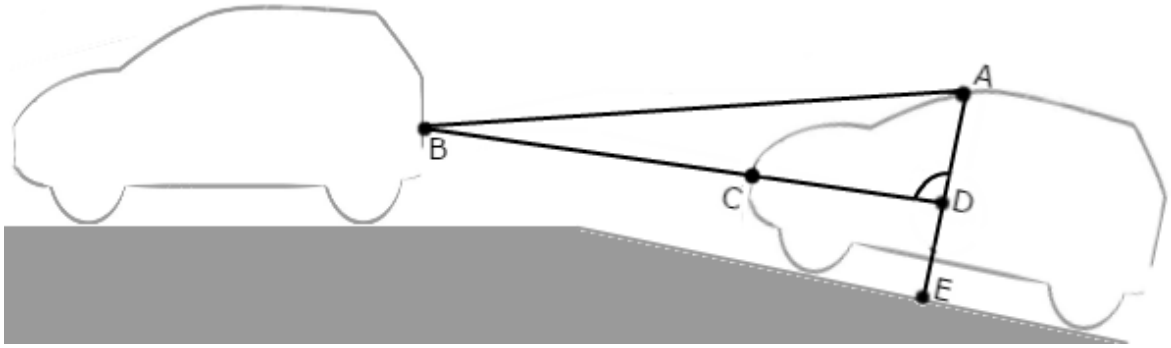


Figure 69: Vehicles on a road with different gradient

However, during driving conditions, this situation rarely happens and when it does, it lasts only for a split of a second. Thus, the difference it makes on the distance estimation is considered trivial.

4.1.3 Slightly Rotated or Bent Plates

It is highly likely that the plates of the front vehicle are not in perfect condition and shape. In each frame, the conditions of the plates can be different. For instance, during a turn the plates are slightly rotated. The length of the plates in pixels when they are slightly rotated or bent could be ± 1 pixel different than if it was level. In this section, we calculate possible error that might occur when, for any reason, the detection of the plates misses a few pixels. Figure 70 shows an example.

Figure 70: Example of extra pixels in detection

To calculate this error, we assume a known distance paired with the respective length in pixels and then we calculate the distance that would have resulted if the length had been one or two pixels longer or shorter. Figure 71 shows the results.

Length in pixels	distance from camera	error for ±1 pixel	error for ±2 pixels
336	100cm	0.3cm	0.6cm
168	200cm	1.2cm	2.4cm
112	300cm	2.7cm	5.4cm
84	400cm	4.8cm	9.7cm
67	500cm	9.0cm	16.9cm
56	600cm	10.9cm	22.2cm
48	700cm	14.8cm	30.4cm
42	800cm	19.5cm	40.0cm

Figure 71: Max error from missed pixels in different distances from camera

Note that the above values represent the error of the distance from the plates to the camera. However, we are interested in the error of the distance between the two vehicles. In other words, figure 71 shows the error in the hypotenuse but we are interested in the error of the requested distance BC (figure 72). Based on the error of the hypotenuse, we can calculate the error in BC which corresponds to the error of the distance between the two vehicles.

To do that we assume all values known as shown in figure 72 and then we replace the hypotenuse with the faulty value in order to check the error in BC.

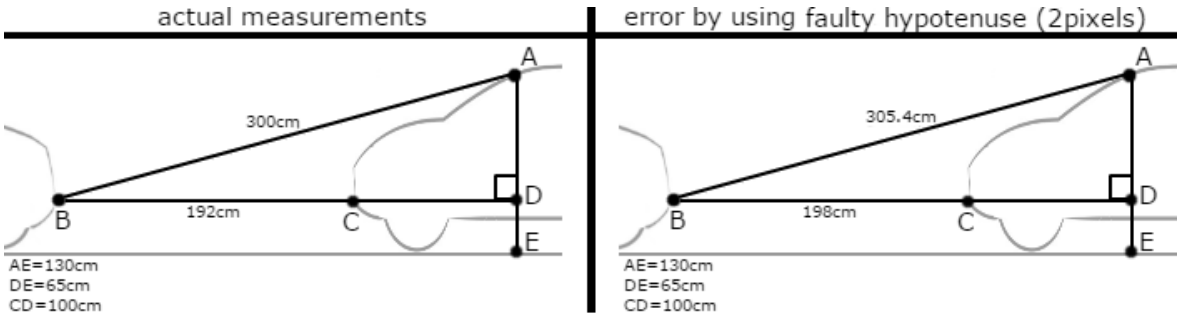


Figure 72: Error from missed pixels in BC

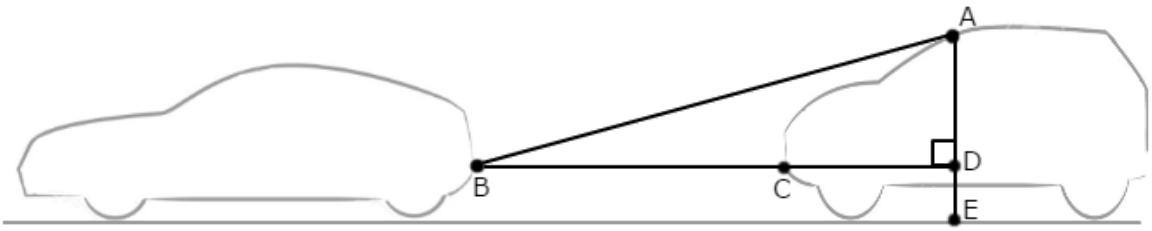
Continuously, we repeat the experiment to determine the maximum error for one and two-pixel deviation in the distance between the two vehicles.

distance between vehicles	error for ±1 pixel	error for ±2 pixels
75.9cm	0.4cm	0.9cm
189.1cm	1.3cm	2.5cm
292.8cm	2.8cm	5.6cm
394.6cm	4.9cm	9.9cm
495.7cm	9.1cm	17.1cm
596.4cm	11cm	22.4cm
696.9cm	14.9cm	30.6cm
797.3cm	19.6cm	40.1cm

Figure 73: Max error from missed pixels in the distance between the two vehicles

4.1.4 Uncompleted Calibration Process

Assuming that for any reason the calibration process has not been completed and the distances CD and AE (figure 74) are not known. In this case, we have to use average values that will result in error when the actual values decline from average.



A: Location of the camera
B: Location of license plates
C: Front part of host vehicle
D: Vertical projection of B on AE
E: Ground

AB: Distance from plates to camera
AE: Distnace from camera to the ground
DE: Distnce from plates to the ground

Figure 74: Distances during driving conditions

To find the requested distance BC we subtract CD from BD. We can use the average value of 1.5 meters as a fixed value but that would mean that if for instance, CD is 1 meter long, then the error would be 0.5 meters. Any error in CD will result in the same error in BC. The length AE is the distance from the camera to the ground. This

distance can vary depending on the type of the vehicle and the position of the camera. During implementation, we placed the camera on the rear-view mirror as shown in figure 74. However, there are more suitable cameras that can be attached on the windscreen. The algorithm does not require a specific place for the camera as long as the calibration process has been completed and the height of the camera from the ground has been given as input during installation.

These values are critical to the algorithm. Our approach to finding distance is based on the Pythagorean Theorem. If all required values are given as input, there is no reason for the algorithm to produce additional error. For this reason, it is highly recommended that the calibration process is completed during installation. Manual measurements might be slightly skewed but that creates trivial error.

4.1.5 Overall Error

So far, we have estimated the maximum error from:

- Missed pixels during licence plate detection.
- Using average height between the licence plates and the ground.

We can now sum the error based on the distance in order to acquire the maximum overall error. We use the maximum error from detecting the plates with 2 missed pixels (figure 73) and the maximum error from using the average plates' height (figure 67). Figure 75 shows the maximum error based on the distance between the two vehicles.

Distance between vehicles	Maximum error
75.9cm	10.3cm
189.1cm	8.8cm
292.8cm	10.4cm
394.6cm	13.7cm
495.7cm	20.3cm
596.4cm	25.1cm
696.9cm	33cm
797.3cm	42.2cm

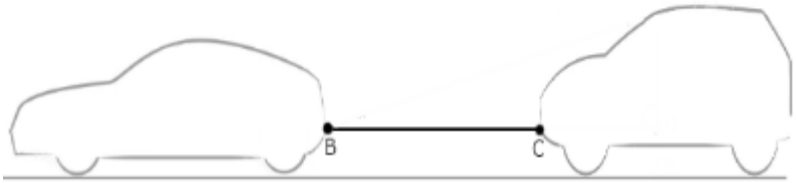


Figure 75: Max error based on distance

Note that to form the overall error we used the values from figure 67 which actually correspond to slightly longer distances than the ones used in figure 75. This means that the actual maximum error is slightly smaller. We do that to compensate for possible error that might have been caused due to measuring distances manually during the calibration process.

4.2 Applicable Distance

As applicable distance, we define the maximum distance for which the algorithm is able to track the front vehicle. For longer distances, the features of the plates are no longer distinct and thus, the tracking fails. After experimenting with various videos from driving conditions, we concluded that when the length of the plates is shorter than 30 – 35 pixels, the plates are no longer recognizable from the algorithm.

During the implementation of the algorithm, we define a threshold for the shortest identifiable rectangle width as 35 pixels, which means that any plates of length shorter than the threshold are rejected. Now, the applicable distance of the algorithm depends on the camera resolution. For the resolution of 640x480 pixels, the maximum applicable distance from the camera to the front car is the length of the plates in pixels at one meter (336 pixels) divided by 35 which is the lowest possible

value. This means 336 / 35 = 9.6 meters (distance estimation is explained in 3.2.7). For greater image resolutions, the applicable distance grows too.

It has to be noted that the maximum applicable distance applies to tracking. Remember that during the detection of candidate stage, the rules for finding plates are stricter in order to increase reliability. This means that the front vehicle will be tracked for up to 9.6 meters if it has been previously detected. For the detection to take place the vehicle has to be closer. After experimenting with videos from driving conditions we concluded that for the 640x480 resolution, the front vehicle is detected for distances of up to 6 – 7 meters from the camera (hypotenuse), and tracked up to 8 – 9.5 meters from the camera (hypotenuse).

Chapter 5

Experimental Results

To measure the efficiency of the algorithm we set up an experimentation environment. We use the Zedboard platform [10] which integrates an ARM Cortex A9 processor (up to 667 MHz) which is selected due to the long-established use of ARM processors in embedded systems. On the Zedboard we set up Linaro [11], which is a Linux based operating system for ARM processors. The C++ programming language is used for implementing the proposed algorithm and the GCC compiler [12] is used for compiling the source code and creating executable files. Additionally, the OpenCV [13] library is utilized in order to acquire the frames from the camera as well as for manipulating images. Finally, attached to the Zedboard there is a Logitech C170 640x480 webcam [14] which is used for receiving video from the front part of the host vehicle.

In order to test the algorithm, we record video (with the 640x480 Logitech camera) of real driving conditions under various illuminations and during driving in various locations with different degree of surrounding traffic. Then, we run the recorded video through the implemented algorithm and we evaluate its performance based on three aspects:

- Time efficiency
- Reliability
- Accuracy

5.1 Time efficiency

Time efficiency is considered very important since distance estimation can potentially prevent road accidents. Driving assistance systems are supposed to be able to provide feedback according to the driving conditions in real time. A system might be very reliable and accurate but if it cannot keep up with time it is unhelpful. During driving, the conditions of the road and traffic change continuously. The output represents the current state of the road based on the frame from the camera. If processing one frame takes two seconds, then each output corresponds to the state of the road two seconds ago. To be able to prevent accidents, driving assistance

systems have to be able to process multiple frames per second and identify potential dangers as soon as possible.

To evaluate the time efficiency of the algorithm developed within the context of this thesis, we record video from real driving conditions and we perform the processing of the video on the experimentation platform that we have set up (ARM A9 processor). The algorithm is implemented using the C++ programming language. In order to count the amount of frames that are processed each second we use the <ctime> library [15] and more specifically the function clock() and the macro CLOCKS_PER_SEC.

The function clock() returns the amount of clock ticks elapsed since a specific moment and the CLOCKS_PER_SEC is the amount of clock ticks that correspond to one second. By combining these two, we can use clock() as a timer and when clock() == CLOCKS_PER_SEC it means that one second has passed. This way we can use a counter to find out how many outputs have been produced during that time.

Using the <ctime> library we measure the time efficiency of the algorithm. Figure 76 shows the results of performing the processing of videos from real driving conditions on the ARM processor.

	Min frames/sec detection of candidate	Max frames/sec detection of candidate	Min frames/sec distance estimation	Max frames/sec distance estimation
VIDEO1 1:17	20	21	22	23
VIDEO2 4:54	19	21	20	23
VIDEO3 1:48	19	21	21	22
VIDEO4 4:30	19	21	21	24
VIDEO5 6:40	20	21	20	23

Figure 76: Results of the algorithm regarding time efficiency

As it can be seen from the results (figure 76) the algorithm can process at least 19 frames per second during the detection of candidate stage and 20 frames per second during the tracking and distance estimation stage. The verification of

candidate stage which only needs 10 frames is performed in approximately 0.5 seconds.

Based on the implementation, it is during the tracking stage that the algorithm calculates the distance and can potentially alert the driver when the front vehicle is dangerously close. For this reason, the time efficiency of the algorithm depends on the tracking stage. During the tracking and distance estimation stage the processing rate is between 20 – 24 frames per second which means that the algorithm can indeed perform in real time.

5.2 Reliability

To measure reliability, we test the algorithm using videos from real driving conditions and we check how often the algorithm fails. There are two scenarios that we consider failure:

- There is a vehicle in front of the host car that is not being detected.
- There is no front vehicle, but one is being detected (faulty detection).

Based on the implementation of the algorithm, there is a trade-off between the two scenarios of failure. We can use very strict rules for the verification of candidate stage so that the possibility of producing faulty detections would be minimum. That however, would cause the algorithm to take longer time to detect a vehicle and it would raise the possibility to not detect a front vehicle. On the other hand, if the rules for verification are very loose, the front vehicles will be detected and verified faster but that would increase the possibility of producing faulty detections.

The main variables that determine this trade-off are:

- Consecutive frames. The number of consecutive frames of detecting the same vehicle before verifying its existence.
- Threshold_line. The difference that is needed between dark and light pixels when identifying a corner (explained in section 3.2.2).

For instance, if we use three consecutive frames for verification, the process of verifying a candidate will be almost instant, but the outcome will not be as reliable as by using 20 consecutive frames. The latter case would require more time though.

Based on experimentation with multiple values and videos we ended up using 10 consecutive frames for verification. Since the average processing rate is 20 frames per second, it means that the whole verification stage requires less than a second to either verify a front vehicle or reject it and look for another candidate. For the corner threshold, we use the value 30 for initial detection which is a relatively strict threshold that eliminates weak candidates. After the initial detection, since a robust shape has been detected we use the value 8 for threshold which is a very loose threshold and guarantees that the initial shape will be followed over time even if it gets unclear.

Figure 77 shows the results of processing videos from real driving conditions on the ARM processor regarding reliability.

	Visible plates (seconds)	Plates detected (seconds)	Plates missed (seconds)	Faulty detections	Success rate
VIDEO1 1:17	53	50	3	0	97%
VIDEO2 4:54	100	80	20	0	94%
VIDEO3 1:48	19	16	3	0	98%
VIDEO4 4:30	195	156	39	0	86%
VIDEO5 6:40	340	312	28	0	93%

Figure 77: Results of the algorithm regarding reliability

To calculate the error rate, we divide the number of seconds of the video by the sum of seconds of plates missed plus seconds of faulty detections. For instance, for VIDEO1 (3 + 0) / 77 = 0.03.

5.3 Accuracy

Accuracy refers purely to the distance estimation. A robust way to measure accuracy would be to install the system on a car and at the same time use a different proven accurate tool to measure the distance. That was not an option during implementation so Instead, we utilized video recording of real cars. We completed the calibration process for the test host vehicle and made it move back and forth at known distances from a stationary front vehicle. We used marked locations for fixed distances and we compared the output of the algorithm to the manually measured

distances. The comparison is performed for distances between the two vehicles as shown in figure 78.

Actual distance	Output from algorithm
100cm	105cm
200cm	208cm
300cm	309cm
400cm	403cm
500cm	495cm
600cm	575cm
700cm	677cm
760cm	739cm

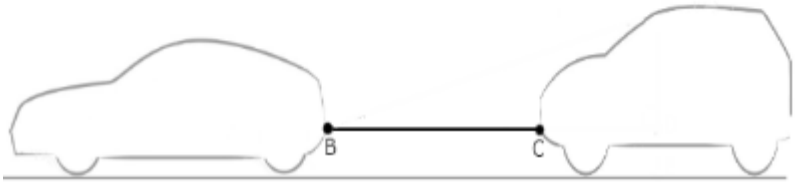


Figure 78: Results of the algorithm regarding accuracy

As it can be seen from figure 78 the error from the measurements is within range according to the overall error of the algorithm shown in figure 75.

Chapter 6

Conclusion – Future Work

Driving assistance systems can possibly aid in preventing accidents and potentially save human lives. For this reason, they are widely developed and implemented, and are considered an essential part of modern vehicles. In this thesis, we focus on the problem of detecting front vehicles and estimating their distance from the host car. With knowledge of this distance an electronic system can potentially alert the driver using sound signals when a front vehicle is dangerously close. After providing a literature review of relevant studies and systems that have already been implemented, we proceed with developing our own algorithm for vehicle tracking and distance estimation.

Compared to other implementations, our algorithm takes advantage of time. We make use of multiple consecutive frames to confirm a front vehicle in order to increase reliability. Additionally, to estimate the distance between the two vehicles we use a standardized feature which is the same for all vehicles, that being the licence plates. Moreover, we include a calibration process so that the algorithm can be adapted to different vehicle models and camera modules. This way, the results apply to all vehicles and the implementation does not depend on a specific camera module. In fact, higher resolution cameras can be used for tracking in longer distances. Along with the algorithm, a full error analysis and a performance analysis is included. The algorithm is evaluated based on time efficiency, reliability and accuracy. Special emphasis is given to time efficiency since the implementation is intended to run in real time, for which the minimum processing rate is 20 frames per second. This was achieved by using a dynamic region of interest which allows us to process only a small part of the initial frame during the tracking and distance estimation stage. The algorithm's performance was satisfactory based on all three aspects (time efficiency, reliability and accuracy).

The high processing rate of the algorithm regarding time efficiency creates additional opportunities for further research. The algorithm was tested using a 640x480 camera resolution for which it achieved a maximum applicable distance of

9.6 meters from the camera. The fact that it can perform so well for this resolution means that we can use greater camera resolutions in order to increase the applicable distance. A higher frame resolution would decrease the processing frame rate. However, the algorithm can perform so fast that it can afford to lower its processing rate in exchange for a longer applicable distance.

Additional research on the topic includes the integration of vehicles on the Internet of Things [8], [16]. Distance values and speed values can be encapsulated within UDP or TCP messages (UDP being more suitable for real time applications [9]) and sent and stored to an online internet cloud for further processing. Results from additional processing on the cloud can be an analysis of the driver's habits regarding average distance from front vehicles or even helpful insight for investigating traffic accidents.

References

[1] Kim, G. and Cho, J.S., 2012. Vision-based vehicle detection and inter-vehicle distance estimation for driver alarm system. Optical review, 19(6), pp.388-393.

[2] Wang, W., Yang, S., Li, Y. and Ding, W., 2015, June. A rough vehicle distance measurement method using monocular vision and license plate. In Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on (pp. 426-430). IEEE.

[3] Chen, Y., Das, M. and Bajpai, D., 2007, May. Vehicle tracking and distance estimation based on multiple image features. In Computer and Robot Vision, 2007. CRV'07. Fourth Canadian Conference on (pp. 371-378). IEEE.

[4] Jayalath, A.N. and Wang, Z., 2013, November. Vision based inter-vehicle distance estimation with extended outlier correspondence. In 2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013) (pp. 323-327). IEEE.

[5] Chen, C.H., Chen, T.Y., Huang, D.Y. and Feng, K.W., 2015, November. Front Vehicle Detection and Distance Estimation Using Single-Lens Video Camera. In 2015 Third International Conference on Robot, Vision and Signal Processing (RVSP) (pp. 14-17). IEEE.

[6] Arenado, M.I., Oria, J.M.P., Torre-Ferrero, C. and Rentería, L.A., 2014. Monovision-based vehicle detection, distance and relative speed measurement in urban traffic. IET Intelligent Transport Systems, 8(8), pp.655-664.

[7] Wu, C.F., Lin, C.J. and Lee, C.Y., 2012. Applying a functional neurofuzzy network to real-time lane detection and front-vehicle distance measurement. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(4), pp.577-589.

[8] Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F. and Alonso-Zarate, J., 2015. A survey on application layer protocols for the internet of things. Transaction on IoT and Cloud Computing, 3(1), pp.11-17.

[9] Karagiannis, V., 2016. Area Limitations on Smart Grid Computer Networks. International Journal of Wireless and Microwave Technologies (IJWMT), 6(3), p.71.

[10] http://zedboard.org/product/zedboard, cited 24/8/2016

[11] http://www.linaro.org/about/, cited 24/8/2016

[12] https://gcc.gnu.org/, cited 24/8/2016

[13] http://opencv.org/about.html, cited 24/8/2016

[14] http://www.logitech.com/product/webcam-c170, cited 24/8/2016

[15] http://www.cplusplus.com/reference/ctime/, cited 24/8/2016

[16] Karagiannis, Vasileios. Building a Testbed for the Internet of Things. Diss. BSc Thesis. TEITHE, Greece, 2014.

View publication stats