5-2017

Real-time Vision-Based Lane Detection with 1D Haar Wavelet Transform on Raspberry Pi

Vikas Reddy Sudini
Utah State University

REAL-TIME VISION-BASED LANE DETECTION WITH 1D HAAR WAVELET TRANSFORM ON RASPBERRY PI

by

Vikas Reddy Sudini

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Vladimir Kulyukin, Ph.D. Major Professor	Dan Watson, Ph.D. Committee Member
_____	_____
Nicholas Flann, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2017

ABSTRACT

Real-time Vision-Based Lane Detection with 1D Haar Wavelet Transform on Raspberry Pi

by

Vikas Reddy Sudini, Master of Science

Utah State University, 2017

Major Professor: Vladimir Kulyukin, Ph.D.
Department: Computer Science

An algorithm is presented for real-time vision-based lane detection on a Raspberry Pi computer coupled to a Raspberry Pi Camera board. The computer-camera unit is placed inside a car, next to the windshield, and is powered by a regular 12V-to-5V car charger. The algorithm is based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows. The algorithm is called GreedyHaarSpiker and is implemented in Python 2.7.9 with OpenCV 3.0. The performance of the algorithm was tested in situ on a Raspberry Pi 3 Model B ARMv8 1GB RAM computer on four image samples, three of which consisted of one thousand, and one consisted of 775 360 x 240 PNG images. The images were captured by a Raspberry Pi Camera Board v2 placed inside a Jeep Wrangler driven on two different days at a speed of 60 miles per hour on a Northern Utah highway. On the first sample, the accuracies of detecting both lanes and at least one lane were 61.90% and 91.20%, respectively; on the second sample, the accuracies of detecting both lanes and at least one lane were 34.10% and 77.4%, respectively; on the third sample, the accuracies of detecting both lanes and at least one lane were 16.90% and 64.10% respectively; on the fourth sample, the accuracies of detecting both lanes and at

least one lane were 15.74% and 57.03%, respectively. The current implementation processes 20 frames per second.

(42 pages)

PUBLIC ABSTRACT

Real-time Vision-Based Lane Detection with 1D Haar Wavelet Transform on Raspberry Pi

Vikas Reddy Sudini

Rapid progress is being made towards the realization of autonomous cars. Since the technology is in its early stages, human intervention is still necessary in order to ensure hazard-free operation of autonomous driving systems. Substantial research efforts are underway to enhance driver and passenger safety in autonomous cars. Toward that end GreedyHaarSpiker, a real-time vision-based lane detection algorithm is proposed for road lane detection in different weather conditions. The algorithm has been implemented in Python 2.7 with OpenCV 3.0 and tested on a Raspberry Pi 3 Model B ARMv8 1GB RAM coupled to a Raspberry Pi camera board v2. To test the algorithm's performance, the Raspberry Pi and the camera board were mounted inside a Jeep Wrangler. The algorithm performed better in sunny weather with no snow on the road. The algorithm's performance deteriorated at night time or when the road surface was covered with snow.

ACKNOWLEDGMENTS

I would like to use this opportunity to express my deepest gratitude to my advisor, Dr. Vladimir Kulyukin, who has supported me throughout my research, encouraged me to think of possible solutions, supported and guided me during hard times. His experience and knowledge are secret ingredients to the success of the project.

I acknowledge my gratitude to my committee members, Dr. Dan Watson and Dr. Nicholas Flann for continuous support, reading my reports, commenting on my views and helping me understand difficult concepts.

Finally, I would like to thank my beloved family and friends for cheering me up and encouraging me to cross the hurdles and succeed in my academic pursuit.

Vikas Reddy Sudini

CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Autonomous cars, i.e., cars capable of navigating various environments without human input, have featured prominently in many research and commercial projects for several decades. The Carnegie Mellon University Navigation Laboratory (Navlab) has built a series of robot cars, sport utility vehicles (SUVs), and buses since 1984. The latest robotic car, Navlab 11, is a robot Jeep Wrangler equipped with a range of sensors for obstacle avoidance, path planning and following, and pedestrian detection [1]. The European Technology Platform on Smart Systems Integration project has reported significant contributions to collision avoidance, fleet management, autonomous cruise control, and cooperative driving [2]. Over the past several years the Google and Tesla corporations have been aggressively commercializing their self-driving platforms [3, 4].

Proponents of driverless cars argue that the major benefits of driverless cars include, but are not limited to, less traffic congestion, enhanced mobility of the elderly and the disabled, significant increases in roadway capacity, and reduction in traffic accidents [5, 6]. Opponents of driverless cars point out that the widespread adoption of autonomous vehicles will result not only in major job losses in driving jobs but also will likely lead to loss of privacy and increased risks of hacking attacks and terrorism [7]. Some researchers argue that lack of stress during driving and more productive time on the road may create additional incentives to live further from cities, which will increase the carbon footprint of motor transportation systems [8].

While we believe that completely autonomous cars may become a reality in the long term, provided that not only technical failures [9, 10] but also social and legal

implications [11] of autonomous car adoption are properly addressed, human drivers are, and will remain indispensable in the short and medium terms. Consequently, it is important to seek solutions that enhance their safety. Robust vision-based lane detection is one such enhancement. Specifically, vision-based lane detection modules will gradually become an integral part of autopilots in semi-trucks to improve the drivers' safety on long, monotonous highway stretches with low or no traffic. Such autopilots will be similar to the ones already in existence in aircraft and ships and will keep the human in the loop in that the decision to engage and disengage the autopilot will be under the sole control of the driver.

In this thesis, an algorithm, called GreedyHaarSpiker, is presented for in situ real-time vision-based lane detection on a Raspberry Pi computer coupled to a Raspberry Pi Camera board, the computer-camera unit is placed inside a car, next to the windshield, and is powered by a regular 12V-to-5V car charger. The algorithm is based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows and polynomial line fitting. The algorithm is implemented in Python 2.7.9 with OpenCV 3.0 [12].

The remainder of this thesis is organized as follows. In Chapter 2, related work is reviewed. In Chapter 3, the concept of a 1D Haar Wavelet Spike (1D HWS) is formally developed. Chapter 4 describes our in-situ algorithm and gives its pseudocode. In Chapter 5, the highway experiments are described and analyzed. In Chapter 6, the findings are summarized and conclusions are presented.

CHAPTER 2

RELATED WORK

Vision-based lane detection has been the focus of many research and development projects in the past two decades. Wang et al. [13] establishes a novel B-Snake based lane model, which describes the perspective effect of parallel lines for generic lane boundaries. This approach enabled the detection and tracking model for a range of lane structures. Also, instead of detecting left side and right side lane markings they detect a single middle lane of the road. An algorithm, called CHEVP, is developed for providing initial positions for the B-Snake model. A minimum error method is proposed to determine the control points of the B-Snake model by the image forces on both sides of a lane. Experimental results presented in the paper suggest that the algorithm is robust against noise, shadows, and illumination variations in captured images of marked and unmarked roads.

Kim [14] presents a lane-detection-and-tracking algorithm to detect lane curvatures, lane changes, and splitting lanes. The algorithm is a RANdom SAmple Consensus (RANSAC) combined with a particle-filtering-based tracking algorithm by a probabilistic grouping framework. The detected lane markings are grouped into separate left and right lane boundary hypotheses to handle merging and splitting lanes. The hypotheses are evaluated and grouped with a probabilistic, Markov-style process framework.

Hsiao et al. [15] propose an embedded real-time lane departure warning system (LDWS) for daytime and nighttime driving. The LDWS features a lane detection algorithm based on peak finding for feature extraction to detect lane boundaries. 1D Gaussian

smoothing and global edge detection are applied to reduce noise in images, the reported lane detection rates were 99.57% during the day and 98.88% at night on a sample of highway images.

Erickson and Landberg [16] proposed a lane detection algorithm that uses Hough lines combined with a parabolic second degree fitting for curvature detection. They also proposed connected-component labeling algorithm for object detection. As the connected-component labeling algorithm iterates over the entire image and manipulates large areas in each image, the execution is relatively slow. On the Raspberry Pi 2 model the algorithm's performance was found to be inadequate for high-speed driving. However, when the object detection is removed from the algorithm the Raspberry Pi 2 model meets the real-time performance requirements.

Mandlik and Deshmukh [17] have developed a lane departure detection system that uses the OpenCV library [18] to detect vehicle lane departure on the Raspberry Pi hardware. The algorithm uses the OpenCV implementations of the Canny Edge Detector [19] and the Hough Transform [20] to detect straight and curved lanes. The experiments are conducted using a toy vehicle with a USB camera mounted on top of it that sends images of white paper lanes on a black floor surface to a Raspberry Pi powered by a laptop.

The algorithm presented in this thesis shares the position advocated in [16] and [17] that, to be economically viable and broadly shareable, vision-based lane detection algorithms must be implemented and tested in situ on off-the-shelf hardware platforms such as the Raspberry Pi. The creation of replicable hardware and open source software

solutions will enable citizen science drivers to build, test, and broadly share driver's safety enhancements.

CHAPTER 3

1D HAAR WAVELET SPIKES

The GreedyHaarSpiker algorithm described in Chapter 4 depends on the concept of the 1D Haar Wavelet Spike, which we develop in this Chapter. In the 1D Haar Wavelet Transform (1D HWT), a signal is a vector in $R^n, n = 2^k, k \in N$. Following the formalization in [21], let $W_a^{(k)}$ be a $2^k \times 2^k$ matrix for computing k scales of the 1D HWT. This matrix can be effectively computed from the n canonical base vectors of R^n. If $x = (x_0, \dots, x_{2^k-1})$ is a signal in R^n, then y is the k-scale 1D HWT of x defined in (1).

$$W_a^{(k)}.x^T = y \tag{1}$$

Then,

$$y^T = \left(a_0^{(0)}, c_0^{(0)}, c_0^{(1)}, c_1^{(1)}, \dots, c_0^{(k-1)}, \dots, c_{2^{k-1}-1}^{(k-1)}\right) \tag{2}$$

In (2), $a_0^{(0)} = \mu(y)$ and $c_i^{(j)}$ is the coefficient of the i^{th} basic Haar Wavelet at scale j [22]. For example, (3) defines the matrix for computing the 1D HWT in R^2.

$$W_a^{(2)} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & -0.25 & -0.25 \\ 0.50 & -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.50 & -0.50 \end{bmatrix} \tag{3}$$

If the input signal $x = (0, 1, 1, 0)$, then (4) gives the 1D HWT of x computed as $W_a^{(2)} x^T = y$. The actual values of the transform are $y^T = (0.5, 0, -0.5, 0.5)$.

$$\begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & -0.25 & -0.25 \\ 0.50 & -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.50 & -0.50 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.50 \\ 0.00 \\ -0.50 \\ 0.50 \end{bmatrix} \tag{4}$$

HWTs are used to detect significant changes in signal values [23]. In this thesis, we claim that some changes can be characterized as signal spikes. Specifically, four types of spikes are proposed: up-down triangle, up-down trapezoid, down-up triangle, and down-up trapezoid. The difference between up-down and down-up spikes is the relative positions of the climb and decline segments. In trapezoid spikes, flat segments are always in between the climb and decline segments, regardless of their relative positions.

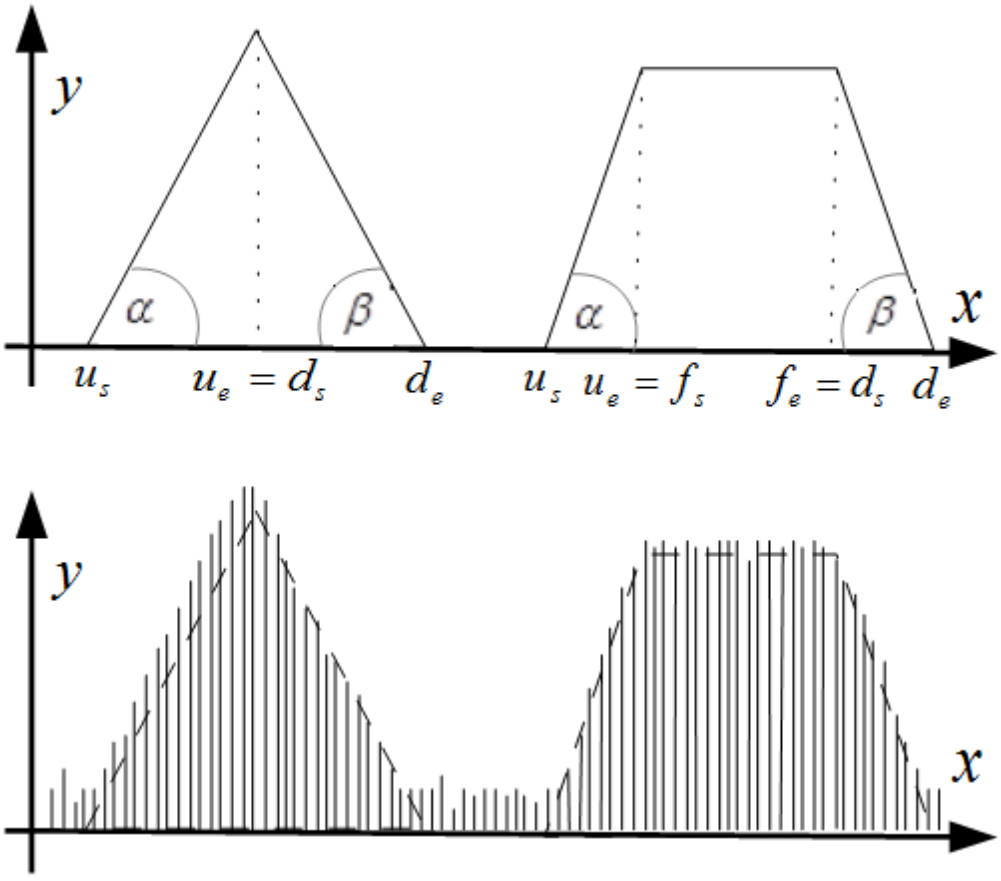

Fig. 3.1 Up-Down Spikes

Fig. 3.1 shows up-down triangle and trapezoid spikes; Fig. 3.2 shows down-up triangle and down-up trapezoid spikes. In both figures, the lower graphs represent the possible values of the corresponding Haar wavelets at a chosen scale k. Up-down spikes describe signals that first increase and then, after an optional flat segment, decrease. Down-up spikes describe signals that first decrease and then, after an optional flat segment, increase. Formally, a spike is a nine-element tuple whose elements are real numbers given in (5).

$$(u_s, u_e, \alpha, f_s, f_e, \gamma, d_s, d_e, \beta) \tag{5}$$

Finally, for a trapezoid up-down or down-up spike, the flat segment is characterized by $f_s, f_e,$ and γ, where f_s and f_e are the abscissae of the beginning and end of the spike's flat segment, respectively, over which the wavelet coefficients either remain at the same ordinate or have minor ordinate fluctuations. If $cf_s^{(k)}$ and $cd_e^{(k)}$ are the k-th scale wavelet coefficients corresponding to f_s and f_e, respectively, the spike's flatness angles is $\gamma = \tan^{-1}(f_e - f_s, cf_e^{(k)} - cf_s^{(k)})$. The absolute values of γ are close to 0.

The first two elements, u_s and u_e, are the abscissae of the beginning and end of the spike's climb segment, respectively, when the wavelet coefficients of the 1D HWT increase. If $cu_s^{(k)}$ and $cu_e^{(k)}$ are the k-th scale wavelet coefficient ordinates at u_s and u_e, respectively, then the climb segment of the spike is measured by the angle $\alpha = \tan^{-1}(u_e - u_s + 1, cu_e^{(k)} - cu_s^{(k)})$.

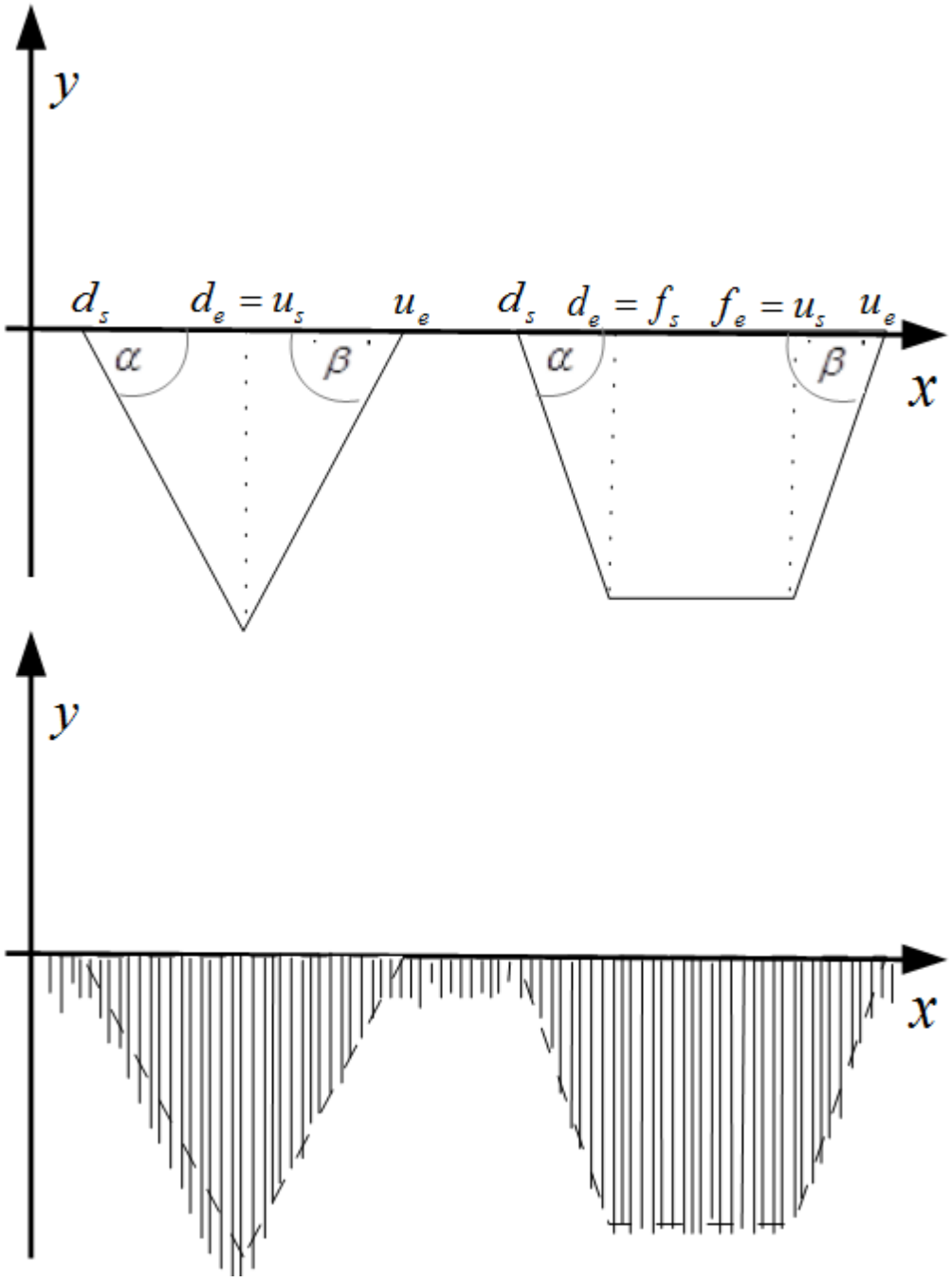

Fig. 3.2 Down-Up Spikes

Similarly, the decline segment of the spike is characterized by d_s, d_e, and β, where d_s and d_e are the abscissae of the beginning and end of the spike's decline segment, respectively, when the wavelet coefficient decrease. If $cd_s^{(k)}$ and $cd_e^{(k)}$ are the k-th scale wavelet coefficient ordinates at d_s and d_e, respectively, then the decline segment of the spike is measured by the angle $\beta = \tan^{-1}(d_e - d_s + 1, cd_e^{(k)} - cd_s^{(k)})$.

CHAPTER 4

GREEDYHAARSPIKER: A LANE DETECTION ALGORITHM

4.1 Overview

In this chapter, we describe our real-time vision-based lane detection algorithm. The algorithm comprises of three stages: pre-processing, spike detection and line fitting. During the pre-processing stage, the region of interest (ROI) of the image, where the lanes are likely to appear is cropped, rendered on grayscale, the edges in the ROI are smoothened using a Gaussian Blur algorithm, and finally converted to a binary image using the OTSU threshold operator. We run the GreedyHaarSpiker on each row of the ROI to extract spikes and further calculate the exact position of each spike in the Spike Detection stage. This algorithm is applied twice: once to detect the left lane and once to detect the right lane. The line fitting stage uses the 2D spike positions returned by GreedyHaarSpiker and fits a straight line through them using standard one-dimensional polynomial line fitting algorithm.

4.2 Hardware

Fig. 4.1, 4.2 and 4.3 shows the hardware on which our lane detection algorithm currently runs. In Fig. 4.1, a seven-inch Raspberry Pi (RPi) touchscreen display is shown. The monitor is attached to a Raspberry Pi 3 model B ARMv8 1GB RAM identified with a green arrow in Fig. 4.2. The RPi computer is attached to the back of the monitor and coupled to an RPi Camera Board v2. The camera identified with a red arrow in Fig. 4.2 is attached to a small cardboard box and taped with a small piece of tape to the windshield for balance. In the future, more stable structures will be designed and deployed.

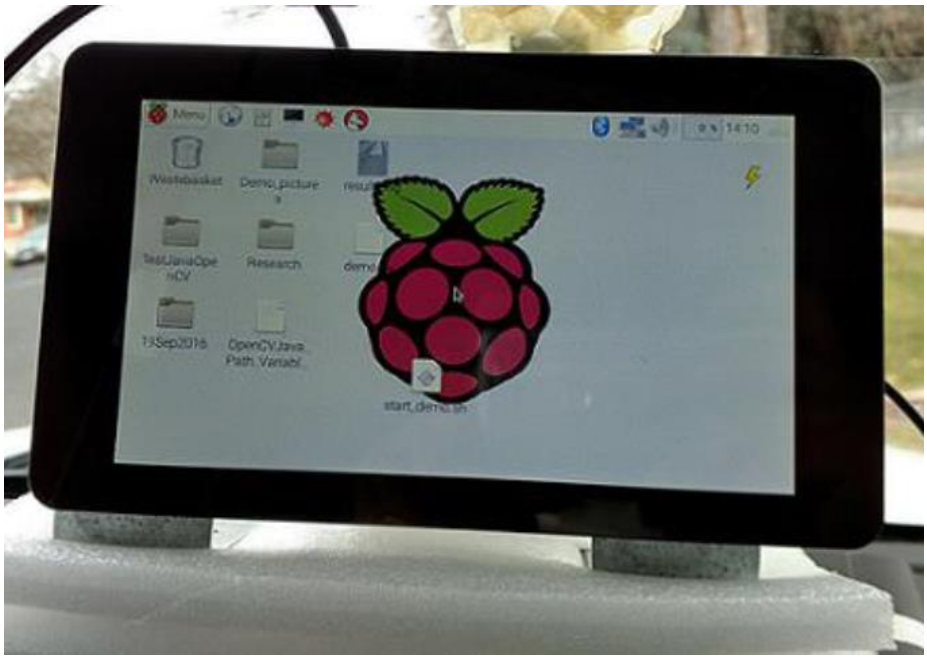

Fig. 4.1 Raspberry Pi Touchscreen Display: An RPi touchscreen display is attached to an RPi (behind it) and mounted next to the windshield of a Jeep Wrangler

Fig. 4.2 Raspberry Pi Camera Board, and Raspberry Pi Computer Board: An RPi camera board v2 (red arrow) attached to small cardboard box for balance and taped to windshield; RPi computer (green arrow); RPi monitor (blue arrow)

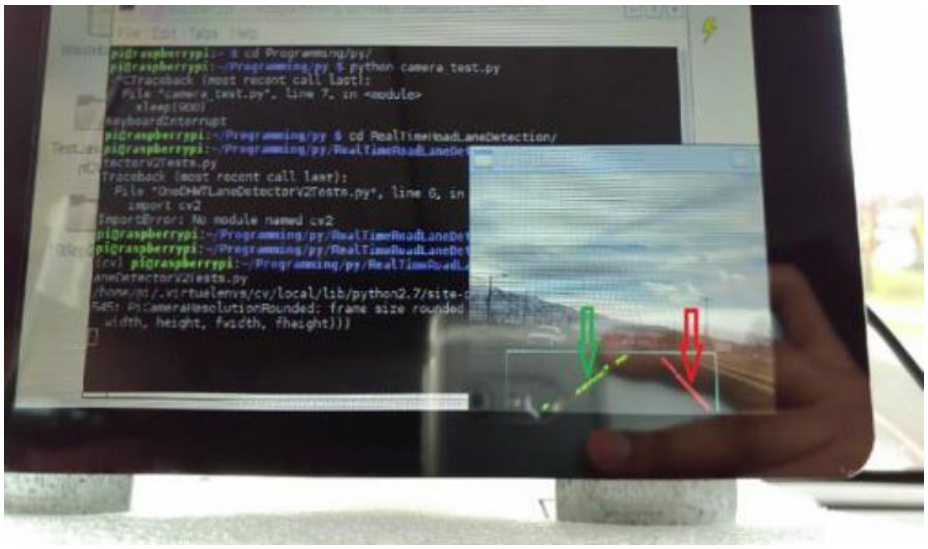

Fig. 4.3 GreedyHaarSpiker Output on RPi Touchscreen Display: As GreedyHaarSpiker runs, lane detection results are graphically displayed in bottom right corner of RPi monitor; green arrow points to detected left lane; red arrow points to detected right lane

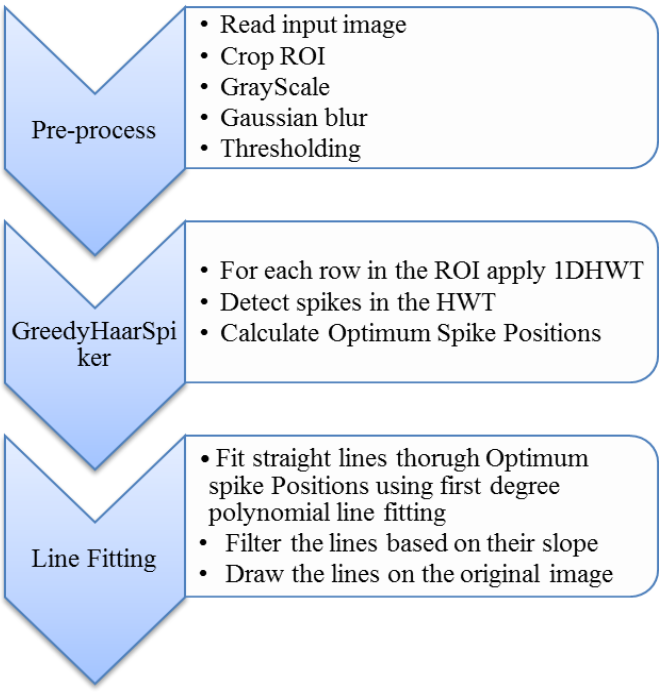

Fig. 4.4 Flow Chart of GreedyHaarSpiker Algorithm

In Fig. 4.3, the RPi monitor displays the left and right lanes as they are being detected by the algorithm in real time as the vehicle is driven. The system requires 10 Watts power (5V and 2A), and is powered using a standard 12V-to-5V car charger where the USB power line for the RPi is plugged in.

4.3 Pre-Processing Stage

The pre-processing stage can be divided into four steps. During the first step, the area of the image where the road lanes are likely to appear, called the region of interest (ROI), is cropped. The second step involves the rendering of the ROI to grayscale. Gaussian smoothing is applied in the third step. Finally, the ROI is converted into a binary image using OTSU thresholding operator. All the steps are shown in Fig. 4.6 – 4.9. We have used various algorithms implemented as functions in OpenCV 3.0 image processing library for all the pre-processing steps.

4.3.1 Cropping Image

As the average size of the image being processed is 200 KB with an average resolution of 360 x 240 pixels, it is time-consuming to perform image operations on the whole image in situ. Hence, we conducted several in-situ camera calibration experiments to find the most probable region of the image where the road lanes are likely to present. The coordinates of the region are set in a configuration file and used in the algorithm to crop the region of interest. Fig. 4.5 and Fig. 4.6 show the output of this stage. We can use fixed ROI safely because the camera position also fixed and the perspective view of camera will not change.

Fig. 4.5 Sample Input Image

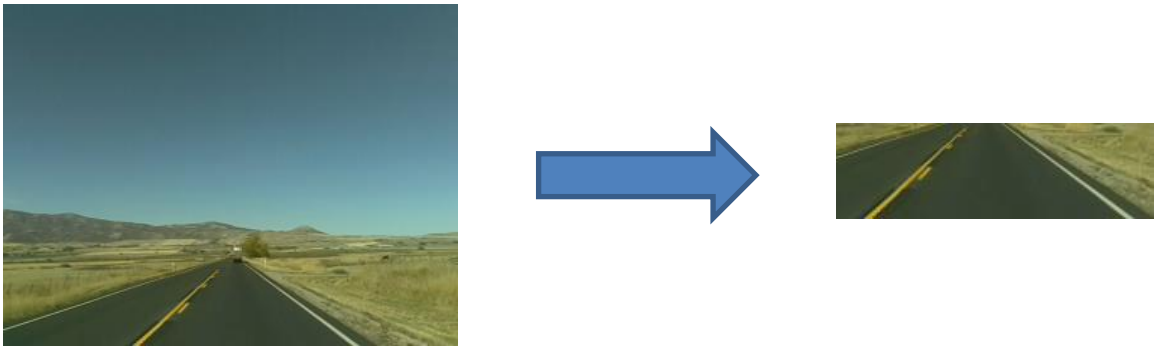

Fig. 4.6 Cropping Region of Interest from Sample Image

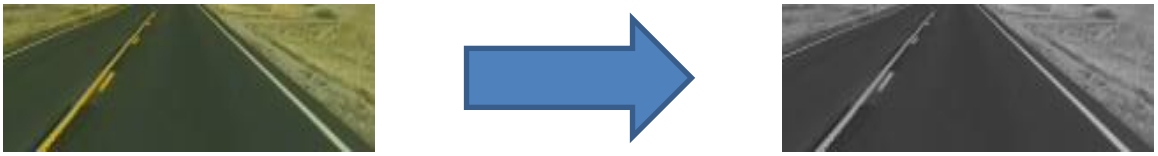

Fig. 4.7 Grayscaling

4.3.2 Grayscaling

The ROI extracted in the above step is rendered to grayscale image as shown in Fig. 4.7. This was done using cvtColor method in OpenCV. The method converts each RGB pixel to equivalent grayscale using the equation 6.

$$\gamma = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{6}$$

In equation 6, R, G, and B are red, green and blue components of a pixel respectively. This will convert the image from three channel image into a single channel image which can be used to convert into a binary image.

4.3.3 Gaussian Smoothing

The ROI returned from above step is smoothened using gaussianBlur method in OpenCV. A Gaussian kernel of size 7x7 is used to that method. This ensures reduction in noise, edges as shown in Fig. 4.8.

The Gaussian function [24] used in gaussianBlur method is defined in equation 7.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{7}$$

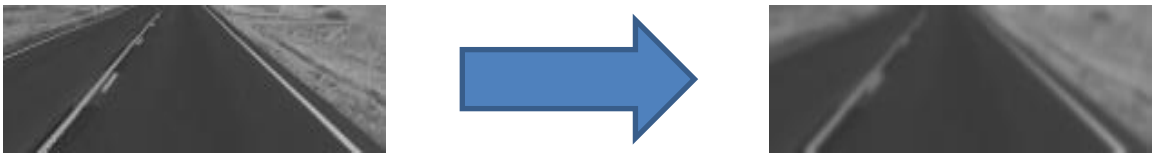

Fig. 4.8 Gaussian Smoothing

4.3.4 Thresholding

At last, the ROI returned from the previous step is converted to a binary image i.e. black and white using OTSU thresholding algorithm in OpenCV which finds a threshold

value which lies in between two peaks such that variances to both classes are minimum. This is shown in Fig. 4.9.

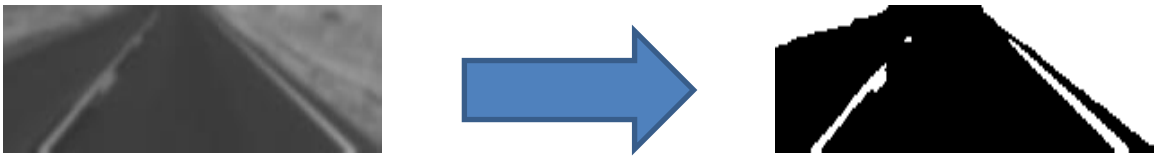

Fig. 4.9 Thresholding

4.4 Detecting road lanes: GreedyHaarSpiker

The algorithm takes the ROI image from the pre-processing stage as an input along with three integer parameters. The first two parameters, sr and er, specify the start row and end row respectively in the ROI where the spikes will be detected. The third integer parameter, $delta$, specifies a step value for generating the exact row numbers for spike detection.

The algorithm iterates through each row (step) in the ROI from bottom to top. In other words, lane detection starts from the rows closest to the moving vehicle.

Fig. 4.10 Scanlines (white lines) and Spikes (red and blue lines) Used to Detect Lanes

Two scan lines of 64 pixels are chosen on the left and the right of the ROI in row r. The left scan line starts from column 0 to 64 and the right scan line starts from $w - 64$ to w, where w is the width of the ROI which is 200 in our case. Then 1D HWT is applied

on both left and right scan lines resulting in two HWTs: the left HWT ($LHWT$) and the right HWT ($RHWT$). The spike detection algorithm is applied on both the $LHWT$ and $RHWT$ to detect left spikes ($LSpikes$) and right spikes ($RSpikes$) respectively.

If the value of $LSpikes$ is not null i.e., at least one spike has been detected, the optimum spike position is calculated. A spike position is defined as the center of a spike's flat segment $(f_s + f_e)/2$, and the optimum spike position is defined as the average of all the spike positions detected in the row. Thus, if multiple spikes are detected, their positions are reduced to one optimal position. The next left scan line is centered on the optimum spike position so that the scan line will follow the road lane. The right scan line is also processed in the same way as $LSpikes$ except that the spikes detected, if any are saved in $RSpikes$.

If no spikes are detected on the scan line of seven consecutive rows, then the scan line is shifted by 7 pixels to the right on the left scan line and by 7 pixels to the left on the right scan line.

When the GreedyHaarSpiker algorithm finishes running, the lists $LPoints$ and $RPoints$ contain (x, y) tuples representing the mid points of the flat segments of spikes detected in the left and right scan lines in each of the selected rows. In this way, all the spike positions are saved for both left and right lanes for each row in the ROI. Then the coordinates of the spikes' positions are scaled back to the original image.

4.5 Line Fitting

First degree polynomial line fitting is applied on the spike positions obtained from above stage for both left lane and right lane. This was done using methods poly1d, and

polyfit available in SciPy python package [25]. The resulting straight lines are filtered based on their slope to reduce false positives. The left lane inclination thresholds are from -60 to -30, and right lane inclination thresholds are from 30 to 60.

Fig. 4.11 Two Lanes Drew in the Original Image Inside the Rectangular ROI

The straight lines which pass the stated filter are then drawn on the original image and shown on the Raspberry Pi touchscreen display. The sample image in which the detected lanes are drawn is shown in Fig. 4.11. The pseudocode for the implementation of the proposed algorithm can be found in Appendix Pseudocodes.

CHAPTER 5

EXPERIMENTS

5.1 Overview

In this chapter, we discuss the experiments that we conducted to evaluate the accuracy of the proposed lane detection algorithm. The images for the experiments were captured by the hardware shown in Fig. 4.1, 4.2 and 4.4 installed inside a Jeep Wrangler. The car was driven on two different days in September (Sample 1) and November 2016 (Sample 2), once during the night on January 6 (Sample 3), and also during the day on January 7, 2017 (Sample 4) under snowy road conditions at a speed of 60 miles per hour on Utah State Route 30, a two-lane highway in Northern Utah. Each driving session was approximately 35 miles long. A sample of around one thousand 360 x 240 PNG images were selected from the captured video.

5.2 Experimental Procedure

The evaluation of the algorithm was done manually by two human evaluators. The evaluators visually compared the lanes drawn in the image by the algorithm and the actual lanes. The images were placed into one of the three categories: both lanes detected, at least one lane detected, and no lanes detected. An actual lane was considered detected if the lane line drawn by the algorithm was exactly aligned with it.

Table 5.1 shows the accuracy of the algorithm for all samples. As shown in Table 5.1, in sample 1, recorded on a sunny day, both lanes were accurately detected in 61.90% and at least one lane was detected in 91.20% of the images. The accuracy of the algorithm

Table 5.1 Lane Detection Accuracy

Sample	Num. of Images	Both Lanes (%)	At least 1 Lane (%)	False positives (%)
1	1,000	61.90%	91.20%	1.60%
2	1,000	34.10%	77.40%	2.70%
3	1,000	16.90%	64.10%	8.30%
4	775	15.74%	57.03%	11.48%

on the second sample, taken on a cloudy day, were lower. Both lanes were detected in 34.10% and at least one lane was detected in 77.40% of the images. The third sample's accuracy further decreased. Both lanes were detected in 16.90% of the images and at least one lane was detected in 64.10%. The reduction in accuracy can be explained by the negative impact of bad weather on the algorithm's accuracy These images were taken at night time using a night vision camera. These results were influenced mainly because parts of the road covered in snow. Finally, in the fourth sample of 775 images, both lanes were detected in 15.74% of the images and at least one lane in 57.30% of the images. These images were taken when the road conditions were snowy and only parts of the road lanes were visible.

CHAPTER 6

RESULTS

Here we briefly discuss the existing algorithm implementations in terms of hardware used, performance and cost.

Wang et al. [13] tested B-snake lane detection on a computer with Pentium 3 (1.4 GHz) processor, 128 MB of RAM. There was no mention of how they captured the image data. The links in the paper for image data were found to be broken. They also mentioned that the algoirhm runs only if the pictures were taken in ideal conditions such as roads with well painted lanes on a sunny day. Further, they did not discuss either the cost or the power requirements of the equipment.

The algorithm proposed by Kim [14] was tested on a computer with Pentium 4 (3GHz) processor. The link that was provided for the data was found to be broken. There was no access to the source code or the pseudocode. There were no specific details about the equipment used to capture the pictures, cost or the power requirements of the equipment.

The embedded solution proposed by Hsiao [15] was run on a embedded processor module – ARM7 32bit 66MHz, a reconfigurable field programmable gate array (FPGA) with 138,000 logic cells. The total power consumption for the system was mentioned to be 1W. They mentioned that they achieved 25 frames per second but they did not provide the image data that they used to test their approach.

The solution proposed by Eriksson and Landberg [16] for lane departure warning and object detection was tested on a Raspberry Pi 2 (900 MHz quad-core ARM Cortex-A7

CPU and 1 GB RAM). They have captured the image data, GPS data using an android mobile phone (phone model was not mentioned). The phone was also connected to car's on-board diagnostics port (OBD-II) via bluetooth to collect data from car's built in systems. Here they did not test their algorithm in real-time instead they have captured different types of data and then conducted their experiments on Raspberry Pi 2 and a computer (2.53 GHz 64 bit Intel core i5 processor and 4 GB RAM). Then compared the performance in terms of frames per second between Raspberry Pi 2 and computer. As their system was not tested in real-time there were no specific details on the power requirements, and cost to implement this solution in real-time. They did not publish their image data, and though they have provided pseudo-code of some parts of their solution which uses GPS data to calculate the anglar velocity, we could not compare their solution with ours.

Mandlik and Deshmukh [17] proposed an algorithm for lane departure detection system. They have used a Raspberry Pi 2 (900 MHz quad-core ARM Cortex-A7 CPU and 1 GB RAM) connected to an USB camera (Intex IT-305WC webcam) which was mounted on a toy car. However the results shown in the paper were computed using a computer (1.80 GHz Intel Core i3 CPU). Also, they have mentioned that the system consumes low power but they did not mention specific details on both power requirements and cost of their equipment.

It was difficult to make a valid comparison between the proposed algorithm and existing algorithms since there was no availability of either source code or data used in other publications. The proposed algorithm was tested in real-time on a Raspberry Pi 3 Model B ARMv8 1GB RAM computer board that was connected to a Raspberry Pi camera

board v2, and to a seven inch Raspberry Pi touch screen display. The whole system was mounted inside our test vehicle Jeep Wrangler. The system consumes 10W of power and is powered using a standard 12V-to-5V car charger, and costs less than 150 USD.

Fig. 6.1 shows the ROI for success and failure cases under different conditions for the proposed lane detection algorithm. The first sample image set which was captured on a sunny day resulted in an accuracy where in at least one lane was detected in 91.90% of the sample and with the very low false positive rate of 1.60%.

The second sample image set was captured on a cloudy day. The accuracy was found to be significantly lower than first image set, mainly due to poor lighting conditions which caused the road lanes to appear faded as shown in Fig. 6.1 (b) resulting in an increased number of true negatives i.e. the algorithm couldn't recognize the lanes though they were present in the image.

The third sample image set was captured on a snowy night with a night vision camera. In this scenario, we faced many challenges like poor picture quality due to lack of light, parts of the roads being covered by snow, and glare of the head lights of vehicles coming in the opposite direction causing a decrease in the accuracy.

The fourth sample image set was captured on a snowy day. Poor road conditions due to partially snow covered roads resulted in the increase of false positives, and an increase in the true negatives had a negative impact on the accuracy of the algorithm.

The above results indicate that the algorithm is sensitive to various conditions such as shadows, road textures, and weather.

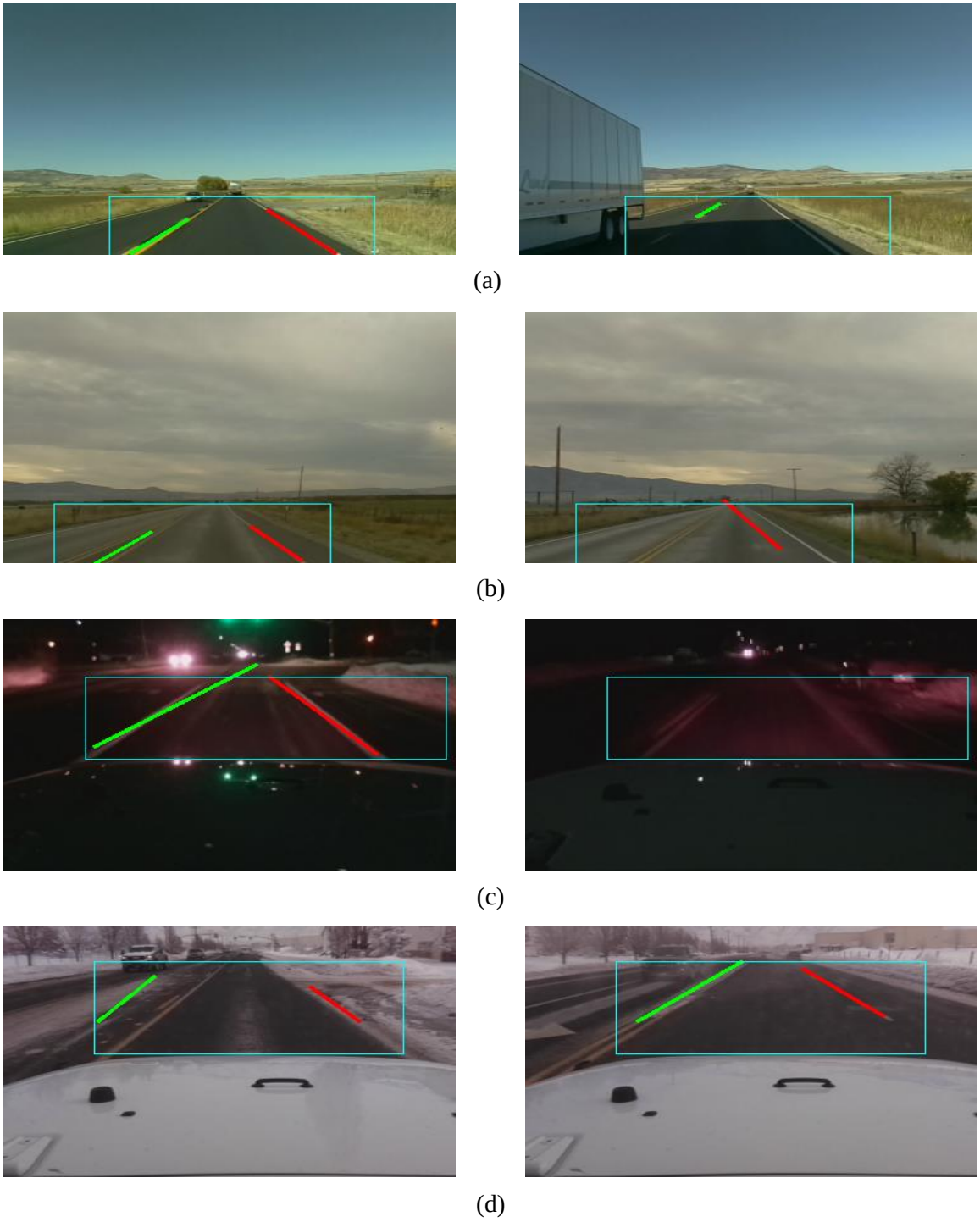

Fig. 6.1 Algorithm's Success and Failure in Various Cases: During (a) Sunny Day (b) Cloudy Day (c) Snowy Night (d) Snowy Day conditions

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

An algorithm called GreedyHaarSpiker was presented for in situ real-time vision-based lane detection on a cost-effective hardware platform – an RPi computer coupled to an RPi Camera board and connected to an RPi touchscreen display. The system's hardware can be placed inside a car, next to the windshield, and can be powered through a regular 12V-to-5V car charger i.e. the power requirement is 10W. Since the algorithm can operate on low voltage devices with smaller RAMs, it is more suitable for ecologically sustainable computing. The hardware and software components of the presented algorithm can be replicated with off-the-shelf hardware components and open source software. The image data that was used for the experiments in the current thesis can be accessed at https://www.dropbox.com/sh/yqpq0adt42n54dt/AAA6m5OF4s_C2KYYDLI4kc2ra?dl=0 and the source code of the algorithm, which includes code for capturing the images using RPi, can be accessed at https://github.com/VKEDCO/PYPL/tree/master/haar_spiker.

The algorithm is based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transform of image rows. The algorithm is currently implemented in Python 2.7.9 with OpenCV 3.0. The performance of the algorithm was tested in situ on a Raspberry Pi 3 Model B ARMv8 1GB RAM computer on two image samples, each of which consisted of one thousand 360 x 240 PNG images. The images were captured by a Raspberry Pi Camera Board v2 placed inside a Jeep Wrangler driven by the first author on four different days with varying weather conditions at a speed of 60 miles per hour on a Northern Utah highway. On the first sample, the accuracies of detecting both lanes and at least one lane

were 61.90% and 91.20%, respectively; on the second sample, the accuracies of detecting both lanes and at least one lane were 34.10% and 77.40%, respectively; on the third sample, the accuracies of detecting both lanes and at least one lane were 16.90% and 64.10%, respectively; on the fourth sample, the accuracies of detecting both lanes and at least one lane were 15.74% and 57.03%, respectively The current implementation processes 20 frames per second.

As discussed in Chapter 6 Results, the performance of the current implementation of the algorithm is affected by factors like weather conditions (sunny, cloudy, night, snow, etc.), shadows and road surface textures (asphalt, concrete, etc.). This can be mainly attributed to the Pre-processing stage during the conversion of the ROI to Binary Image. Configuring OTSU to account for the texture of the road surface may provide a better binary image thus increasing the accuracy of the algorithm. Further, using second degree polynomial line fitting could enable the algorithm to draw curvatures on lanes as opposed to straight line fitting that is being performed in the current implementation.

REFERENCES

[1] S. Thrun. “Toward robotic cars.” Communications of the ACM, vol. 53, no. 4, pp. 99–106, 2010, doi:10.1145/1721654.1721679.

[2] J. Dokic, B. Müller, G. Meyer. European roadmap smart systems for automated driving. Berlin, Germany: European Technology Platform on Smart Systems Integration, 2015.

[3] T. Simonite. “Data shows Google's robot cars are smoother, safer drivers than you or I.” MIT Technology Review, Oct. 2013.

[4] G. Nelson. “Tesla beams down 'autopilot' mode to Model S.” Automotive News. Oct. 14, 2015.

[5] C. Mui. “Will the google car force a choice between lives and jobs?” Forbes, Dec. 2013.

[6] T. Lassa. “The beginning of the end of driving.” Motor Trend, Jan. 2013.

[7] O. Miller. “Robotic cars and their new crime paradigms.” LinkedIn Pulse, Sept. 3, 2014.

[8] M. Ufberg. “Whoops: The self-driving tesla may make us love urban sprawl again.” Wired, Oct. 10, 2015.

[9] D. Yadron, D. Tynan. (2016-07-01). “Tesla driver dies in first fatal crash while using autopilot mode.” The Guardian, Jul. 1, 2016.

[10] V. Mathur. “Google autonomous car experiences another crash.” Government Technology. 17 Jul. 2015.

[11] J. Boeglin. “The costs of self-driving cars: reconciling freedom and privacy with tort liability in autonomous vehicle regulation.” Yale Journal of Law and Technology, vol. 17, iss. 1, article 4, 2015.

[12] “OpenCV” [Online]. Available: http://opencv.org/

[13] Y. Wang, E. Teoha, D. Shen. “Lane detection and tracking using B-Snake.” Image and Vision Computing, vol. 22, pp. 269–280, 2008.

[14] Z. Kim. “Robust lane detection and tracking in challenging scenarios.” IEEE Trans. on Intelligent Transportation Systems, vol. 9, no. 1, pp. 16 – 26, Mar. 2008.

[15] P. Hsiao, C. Yeh, S. Huang, L. Fu. “A portable vision-based real-time lane departure warning system: day and night.” IEEE Trans. on Vehicular Technology, vol. 58, no. 4, pp. 2089 – 2094, May 2009.

[16] J. Eriksson, J. Landberg. Lane departure warning and object detection through sensor fusion of cellphone data. Master's thesis in Applied Physics and Complex Adaptive Systems. Department of Applied Mechanics, Chalmers University of Technology. Göteborg, Sweden 2015.

[17] P. Mandlik, A. Deshmukh. “Raspberry-pi based real time lane departure warning system using image processing.” International Journal of Engineering Research and Technology, vol. 5, issue 06, June-2016, pp. 755 – 762.

[18] R. Laganiere. OpenCV 2 Computer Vision Application Programming Cookbook. Packt Publishing LTD, 2011.

[19] J.F. Canny. “A Computational approach to edge detection.” IEEE Trans. on Pat. Anal. And Mach. Intel., vol 8, pp. 679-698, 1986.

[20] R. O. Duda, P. E. Hart. “Use of the Hough transformation to detect lines and curves in pictures.” Comm. ACM, vol. 15, pp. 11–15, Jan. 1972.

[21] A. Jensen, A. Cour-Harbo. Ripples in mathematics: the discrete wavelet transform. New York: Springer, 2001.

[22] Y. Nievergelt. Wavelets made easy. Boston: Birkhäser, 2001.

[23] S. Mallat, W. Hwang. “Singularity detection and processing with wavelets.” IEEE Trans. on Information Theory, vol. 38, no. 2, Mar. 1992, pp. 617-643.

[24] Gaussian function [online]. Available: https://en.wikipedia.org/wiki/Gaussian_blur

[25] “SciPy” [Online]. Available: https://www.scipy.org/

APPENDICES

Appendix - Pseudocodes

```
1.  Procedure DetectLanes(Img);
2.      ROI = cropROI(Img);
3.      convertToGrayScale(ROI);
4.      gaussianBlur(ROI);
5.      thresholdOTSU(ROI);
6.      LPoints, RPoints = GreedyHaarSpiker(ROI);
7.      fitLine(Img, LPoints);
8.      fitLine(Img, RPoints);
```

Pseudocode of DetectLanes procedure

The above pseudocode gives an overview of the algorithm. The ROI of the input image Img is cropped, converted the ROI into grayscale image, applied Gaussian blur, and applied OTSU thresholding in lines 2,3,4, and 5 respectively. In line 6 GreedyHaarSpiker was applied on the pre-processed image and the results LPoints, RPoints (containing the road lane positions for both left, right lanes) returned. The returned positions for both left lane, and right lane are fitted on to the original image in lines 7, and 8.

```
1.  LPoints = [];
2.  RPoints = [];
3.  LSpikes = NULL
4.  RSpikes = NULL
5.  Procedure GreedyHaarSpiker(ROI, sr, er, delta)
6.      For (r = er, r <= sr, r+=delta)
7.          LLine = getLeftScanLine(ROI, r, LSpike)
8.          RLine = getRightScanLine(ROI, r, RSpike)
9.          LHWT = ordered1DHWT(LLine)
10.         RHWT = ordered1DHWT(RLine)
11.         LSpikes = DetectSpikes(LHWT);
12.         RSpikes = DetectSpikes(RHWT);
13.         IF LSpikes != NULL
14.             THEN add mid-point of optimum LSpikes' flat segment to LPoints;
15.         IF RSpikes != NULL
16.             THEN add mid-point of optimum RSpikes' flat segment to RPoints
17.     ENDFOR
18. RETURN LPoints, RPoints
```

Pseudocode of GreedyHaarSpiker

In the above pseudocode, LPoints, RPoints are two lists initialized to hold detected lane positions for both left, right lanes in lines 1, and 2. In line 5 the procedure GreedyHaarSpiker is defined, which takes ROI, starting row, ending row, and delta as arguments. In line 6 the for loop iterates from starting row to ending row in the steps of delta. In line 7, 8 both left, right scan lines are retrieved for each row from the ROI. Ordered 1DHWT is applied on both the scan lines and the wavelet transformations are stored in LHWT, and RHWT in lines 9, 10. Spike detection algorithm is applied on the LHWT, and RHWT and the returned spikes are stored in LSpikes, and RSpikes respectively in lines 11, 12. If there is at least one spike is detected in that row then optimum spike position is calculated, and stored in the corresponding Spikes' list in lines 13 to 16. After finishing the for loop the lists LPoints, and RPoints are returned in line 18.