

Real Time Vision-Based Lane Detection on Raspberry Pi with 1D Haar Wavelet Spikes

Vladimir A. Kulyukin

Vikas Reddy Sudini

Abstract—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 through 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. 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 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 67% and 89.70%, respectively; on the second sample, the accuracies of detecting both lanes and at least one lane were 47.30% and 77.40%, respectively. The current implementation processes 20 frames per second.

Index Terms—computer vision, lane detection, discrete wavelet transform, haar wavelets, autonomous cars

I. 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 CMU Navigation Laboratory (Navlab) has built a series of robot cars, 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 even 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 paper, 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 through 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 implemented in Python 2.7.9 with OpenCV 3.

The remainder of this paper is organized as follows. In Section II, related work is reviewed. In Section III, the concept of a 1D Haar Wavelet Spike (1D HWS) is formally developed. Section IV describes our in situ algorithm and gives its pseudocode. In Section V, the highway experiments are described and analyzed. In Section VI, conclusions are drawn.

II. RELATED WORK

Vision-based lane detection has been the focus of many R&D projects in the past two decades. Wang et al. [12] propose a B-Snake based lane detection and tracking model for a range of lane structures. 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

Manuscript received December 5, 2016; revised December 24, 2016.

Vladimir A. Kulyukin is with the Department of Computer Science of Utah State University, Logan, UT 84322 USA (phone: 434-797-2451; fax: 435-791-3265; e-mail: vladimir.kulyukin@usu.edu).

Vikas Reddy Sudini is with the Department of Computer Science of Utah State University, Logan, UT 84322 USA (phone: 434-797-2451; fax: 435-791-3265; e-mail: s.vikasreddy2009@gmail.com).

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 [13] presents a lane-detection-and-tracking algorithm to detect lane curvatures, lane changes, and splitting lanes. 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. [14] 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 [15] proposed a lane detection algorithm that uses Hough lines combined with a parabolic second degree fitting for curvature detection. On the Raspberry Pi (RPI) 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 RPI 2 meets the real time performance requirements.

Mandlik and Deshmukh [16] have developed a lane departure detection system that uses the OpenCV library [17] to detect vehicle lane departures on the RPI hardware. The algorithm uses the OpenCV implementations of the Canny Edge detector [18] and the Hough Transform [19] to detect straight and curved lanes. The experiments are conducted on 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 an RPI powered by a laptop.

The algorithm presented in this paper shares the position advocated in [15] and [16] 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 RPI. The creation of replicable hardware and software solutions will enable citizen science drivers to build, test, and broadly share driver's safety enhancements.

III. 1D HAAR WAVELET SPIKES

The GreedyHaarSpiker algorithm described in Section IV depends on the concept of the 1D Haar Wavelet Spike developed in this section. 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 [20], 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).

$$W_a^{(k)} \cdot x^T = y \quad (1)$$

Then

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

In (2), $a_0^{(0)} = \mu(y)$ and $c_i^{(j)}$ is the coefficient of the i^{th} basic Haar wavelet at scale j [21]. For example, (3) defines the matrix for computing the 1D HWT in R^2 .

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

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 $y^T = (0.5, 0, -0.5, 0.5)$.

$$\begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & -0.25 & -0.25 \\ 0.50 & -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.50 & -0.50 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.50 \\ 0.00 \\ -0.50 \\ 0.50 \end{bmatrix} \quad (4)$$

HWTs are used to detect significant changes in signal values [22]. In this paper, 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.

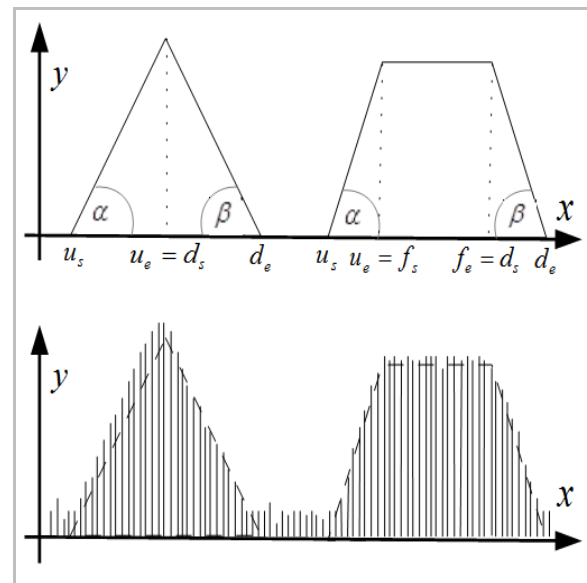


Fig. 1. Up-down spikes.

Fig. 1 shows up-down triangle and trapezoid spikes; Fig. 2 shows down-up triangle and 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).

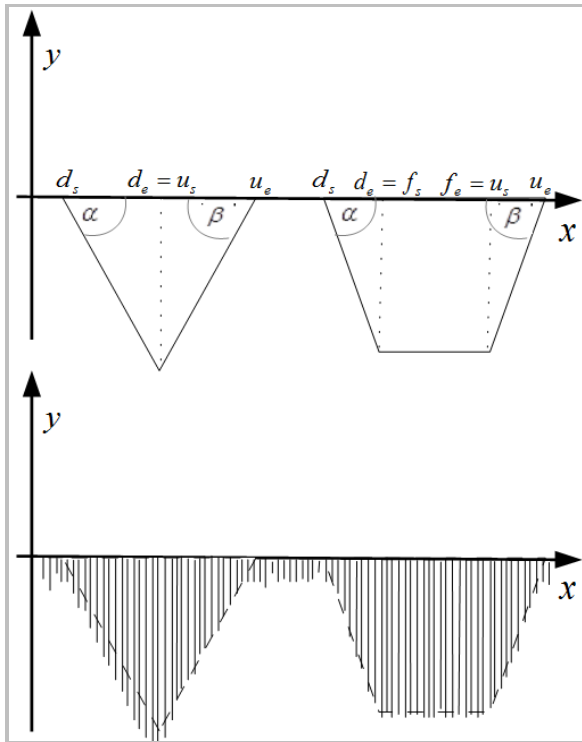


Fig. 2. Down-up spikes.

$$(u_s, u_e, \alpha, f_s, f_e, \gamma, d_s, d_e, \beta) \quad (5)$$

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 $\alpha = \tan^{-1}(u_e - u_s + 1, cu_e^{(k)} - cu_s^{(k)})$.

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 coefficients 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 $\beta = \tan^{-1}(d_e - d_s + 1, cd_e^{(k)} - cd_s^{(k)})$.

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 $cf_e^{(k)}$ are the k -th scale

wavelet coefficients corresponding to f_s and f_e , respectively, the spike's flatness angle is $\gamma = \tan^{-1}(f_e - f_s, cf_e^{(k)} - cf_s^{(k)})$. The absolute values of γ are close to 0.

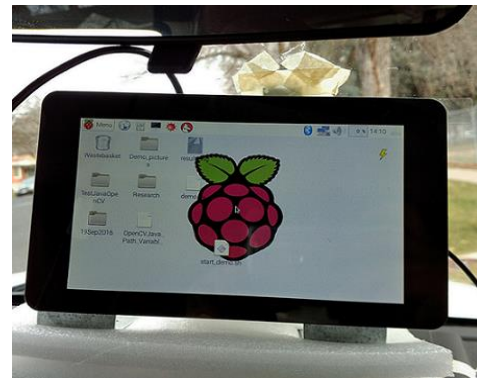


Fig. 3. An RPi monitor attached to an RPi (behind it) and mounted next to the windshield of a Jeep Wrangler.



Fig. 4. 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. 5. 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.

IV. GREEDYHAARSPIKER: A LANE DETECTION ALGORITHM

Figures 3 – 5 show the hardware on which GreedyHaarSpiker, our lane detection algorithm, currently runs. In Fig. 3, a seven-inch RPi touchscreen display is shown. The monitor is attached to a RPi 3 Model B ARMv8 1GB RAM identified with a green arrow in Fig. 4. 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, 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. In Fig. 5, 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 is

powered with a standard 12V-to-5V car charger where the USB power line for the RPi is plugged in.

Figures 6 and 7 give the pseudocode of the DetectLanes procedure and the GreedyHaarSpiker procedure called by DetectLanes. The procedure DetectLanes takes as input a 360 x 240 PNG image. Fig. 8 shows a sample input image. In line 2 of Fig. 6, a 56 x 200 region of interest (ROI) is cropped in the bottom center portion of the image in Fig. 8. The cropped ROI is shown in Fig. 9a.



Fig. 8. A sample input image.

```

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

```

Fig. 6. Pseudocode of DetectLanes procedure.

```

1. LPoints = [];
2. RPoints = [];
3. LSpike = NULL;
4. RSpike = 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.    LSpike = DetectSpike(LHWT);
12.    RSpike = DetectSpike(RHWT);
13.    IF LSpike != NULL
14.      THEN add mid point of LSpike's climb to LPoints;
15.    IF RSpike != NULL
16.      THEN add mid point of RSpike's climb to RPoints;
17.   ENDFOR

```

Fig. 7. Pseudocode of GreedyHaarSpiker.

Lines 3 – 5 in Fig. 6 specify the image preprocessing steps applied to the ROI. The ROI is grayscaled (Fig. 9b), blurred with the Gaussian 7 x 7 kernel (Fig. 9c), and thresholded with the Otsu thresholding operator (Fig. 9d). In line 6 in Fig. 6 the GreedyHaarSpiker algorithm, outlined in Fig. 7, is applied to the thresholded image. In Fig. 7, two empty lists, LPoints and RPoints, are initialized. LPoints is a list of (x, y) tuples used to find the left lane in the image by the fitLine procedure in line 6 of Fig. 6. RPoints is a list of (x, y) tuples used to find the right lane in the image by the fitLine procedure in line 7 of Fig. 6. This procedure is described at the end of this section.

In lines 3 and 4 of Fig. 7, two variables, LSpike and RSpike, are defined where detected spikes are temporarily saved to guide the selection of the scanline in the next row by the procedures getLeftScanLine and getRightScanLine in lines 7 – 8 of Fig. 7.

The procedure GreedyHaarSpiker takes a preprocessed ROI and three integer parameters. The parameters *sr* and *er* specify the start and end rows, respectively, in ROI where the spikes are detected. The third integer parameter, *delta*, specifies a step value for generating the exact row numbers for spike detection.

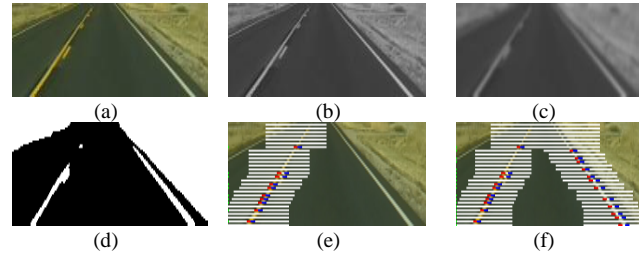


Fig. 9. (a) ROI from input image; (b) grayscaled ROI; (c) blurred ROI; (d) ROI thresholded with OTSU; (e) scanlines (white lines) and spikes (red and blue lines) used to detect left lanes; (f) scanlines and spikes used to detect both lanes.

As indicated in Line 6 of Fig. 7 where the for-loop starts, the processing of rows goes from bottom to top. In other words, the lane detection starts from the rows closest to the moving vehicle.

In lines 7 and 8 of Fig. 7, two scanlines of 64 pixels are chosen on the left and right parts of ROI in row *r*. If the value of LSpike is NULL, the left scanline starts at column 0; similarly, if the value of RSpike is NULL, the right scanline starts at column *w*-1, where *w* is the width of the ROI, which is equal to 200 in our case. If the value of LSpike is NULL, which means that a spike has been detected in a row below, the left scanline, saved in the LLine variable, is centered on the middle of the two ordinates of the detected spike's climb segment, i.e., the ordinates of u_s and u_e defined in Section III. The flat and down segments are currently not taken into account in the algorithm. The right scanline is detected and saved in the RLine variable in the same way except that the spike saved in RSpike is used. In Fig. 9e and Fig. 9f, the scanlines are shown as white lines.

In lines 9 and 10 of Fig. 7, the ordered 1D HWT is applied to the left and right scanlines, respectively. In lines 11 and 12, up-down spikes are shown in Figures 9e and 9f as short red-blue lines on white lines. The red segments show the climb segments; the blue segments indicate the decline segments.

The procedure DetectSpike uses the present thresholds for the angles of climb, flat, and decline spike segments, i.e., α , γ , β and returns the leftmost spike that clears the thresholds. In the current implementation, $\alpha = \beta = 60^\circ$ and $\gamma = 5^\circ$. In other words, in the current implementation of the algorithm, the up-down spikes whose climb or decline angles are less than 60° are filtered out, and flat segments are detected so long as consecutive wave coefficients fluctuate within $\pm 5^\circ$ of 0° .

The algorithm is greedy in that it always returns exactly

one leftmost spike in each left scanline and exactly one leftmost spike in each right scanline. All other spikes are ignored. If no spikes clear the angle thresholds, the value of NULL is returned.

When the GreedyHaarSpiker procedure finishes running, the lists LPoints and RPoints contain (x, y) tuples representing the mid points of the climb segments of spikes detected in the left and right scanlines in each of the selected rows. The procedure fitLine called in lines 7 and 8 takes these points and fits lines through them. Currently, no standard polynomial fitting procedure, like linear regression, is used. Instead, to make the line fitting faster, the procedure connects two points, a lower point and an upper point, by considering them the end points of a triangle's hypotenuse and connecting them so long as the absolute value of the inclination angle of the hypotenuse is within 15° of 35°. Fig. 10a shows the two lines fit through the points in Llanes and Rlanes detected in the original image in Fig. 8. In Fig. 10b, the ROI, the scanlines, and the detected spikes are explicitly shown.

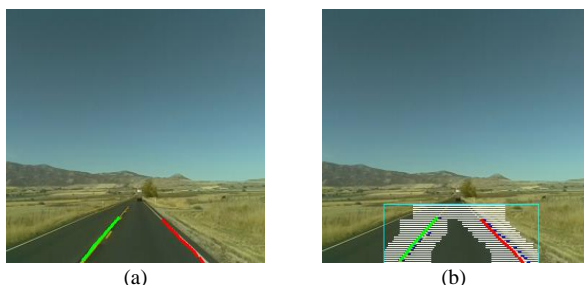


Fig. 10. (a) Two lanes drawn in the original image in Fig. 8; (b) the white rectangular ROI, scanlines, spikes, and lanes drawn on the original image.

V. EXPERIMENTS

The images for the experiments were captured by the hardware shown in Figures 3 – 5 installed inside a Jeep Wrangler. The car was driven on two different days in September and November 2016 at a speed of 60 miles per hour on a two-lane Northern Utah highway. Each drive was approximately 35 miles long. A sample of one thousand 360 x 240 PNG images was selected from the captured video.

The evaluation of the algorithm was done manually by the authors. The authors visually compared the lane lines drawn in the image by the algorithm and the actual lines. The images were placed into one of three accuracy 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 I
Lane detection accuracy.

Sample	Num. Images	Both Lanes (%)	At least 1 Lane (%)
1	1,000	67.00%	89.70%
2	1,000	47.30%	77.40%

Table I shows the accuracy results for both samples of images. As the results in Table I indicate, in sample 1, both lanes were accurately detected in 67% of the frames and at least one lane was detected in 89.70% of the images. The detection results on the second sample of 1,000 images were

lower. Both lanes were detected in 47.30% of the images and at least one lane was detected in 77.40% of the images, which indicates bad weather had a negative impact on the algorithm's accuracy.



Fig. 11. Two false positives.

Fig. 11 illustrates two common problems that had a negative impact on the algorithm's accuracy. Both the left and the right lanes are detected inaccurately. The left lane, a small bright green line in Fig. 11, is accurately aligned with a real road lane. The problem is that it is a wrong lane. This misalignment shows a problem with the greedy approach in that the algorithm always chooses the leftmost spike in each scanline. The red line, drawn above and almost perpendicular to the bright green line, is a false positive.

We are currently implementing improvements to overcome both problems. The first improvement is to use not just the climb segment of each detected spike but also the flat and decline segments when computing the 2D points of a potential line either on the left or on the right. The second improvement is to use a standard polynomial curve fitting algorithm, such as linear regression, to find the best line that fits a set of points. A potential drop in the number of frames processed per second may be compensated by more accurate line detection. The third improvement is to use simple geometrical constraints to filter out invalid matches like the right lane false positive in Fig. 11. For example, one possible constraint can be that the left and right lane lines may not intersect within the region of interest.

While algorithms' accuracy is important, it should be pointed out that the algorithm was implemented in Python 2.7.9 with OpenCV 3.0 on a RPi 3 Model B with an ARMv8 processor and 1GB of RAM. The current implementation of the algorithm processes 20 frames per second. Thus, there is sufficient potential to add more sophisticated algorithms, e.g., polynomial curve fitting, without sacrificing the real time performance of the algorithm.

VI. CONCLUSION

An algorithm, called GreedyHaarSpiker, was presented for in situ real time vision-based lane detection on a RPi computer coupled to a Raspberry Pi Camera board. The system's hardware can be placed inside a car, next to the windshield, and be powered through a regular 12V-to-5V car charger. 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 algorithm is based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows. The algorithm is currently implemented in Python 2.7.9 with OpenCV 3. 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 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 67% and 89.70%, respectively; on the second sample, the accuracies of detecting both lanes and at least one lane were 47.30% and 77.40%, respectively. The current implementation processes 20 frames per second.

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

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] Y. Wang, E. Teoha, D. Shen. "Lane detection and tracking using B-Snake." *Image and Vision Computing*, vol. 22, pp. 269–280, 2008.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] R. Laganier. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing LTD, 2011.
- [18] J.F. Canny. "A Computational approach to edge detection." *IEEE Trans. on Pat. Anal. And Mach. Intel.*, vol. 8, pp. 679-698, 1986.
- [19] 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.
- [20] A. Jensen, A. Cour-Harbo. *Ripples in mathematics: the discrete wavelet transform*. New York: Springer, 2001.
- [21] Y. Nievergelt. *Wavelets made easy*. Boston: Birkhäuser, 2001.