Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems

Wael Farag[1,2,4], Zakaria Saleh[3,5]

[1]College of Eng. and Technology, American University of the Middle East, Kuwait.
[2]Electrical Eng. Dept., Cairo University, Egypt
[3]University of Bahrain, Bahrain.
[4]wael.farag{at}aum.edu.kw, [5]zsaleh{at}ubo.edu.bh

***Abstract*—In this paper, a fast and reliable lane-lines detection and tracking technique is proposed. The proposed technique is well suited to be used in Advanced Driving Assistance Systems (ADAS) or self-driving cars. The main emphasis of the proposed technique in on simplicity and fast computation capability so that it can be embedded in affordable CPUs that are employed by ADAS systems. The proposed technique is mainly a pipeline of computer vision algorithms that augment each other and take in raw RGB images to produce the required lane-line segments that represent the boundary of the road for the car. Each used algorithm is described in details, implemented and its performance is evaluated using actual road images and videos captured by the front mounted camera of the car. The whole pipeline performance is also tested and evaluated on real videos. The evaluation of the proposed technique shows that it reliably detects and tracks road boundaries under various conditions. The usefulness and the shortcomings of the proposed technique are also discussed in details.**

Keywords—Computer Vision, Lane Detection, Self-Driving Car, Autonomous Driving, ADAS.

I. INTRODUCTION

Increasing safety, reducing road accidents and enhancing comfort and driving experience are the major motivations behind equipping modern cars with Advanced Driving Assistance Systems (ADAS) [1-4]. In the past couple of decades, major car manufacturers introduce many sophisticated ADAS functions like Lane Departure Warning (LDW), Lane Keep Assist (LKA), Electronic Stability Control (ESC), Anti-lock Brake System (ABS) … etc. Both LDW and LKA functions are examples of how important for the car to detect and track the road lane lines or the road boundaries accurately and on time. Future ADAS functions like Collision Avoidance, Automated Highway Driving (Autopilot), Automated Parking and Cooperative Maneuvering requires more and more fast and reliable road boundaries detection, which is among the most complex and challenging tasks. The road boundaries detection functionality requires localization of the road, the determination of the relative position between vehicle and road, and the analysis of the vehicle's heading direction.

Computer vision techniques are the main tools that provide the capabilities of sensing the surrounding environment for the detection, identification, and tracking of road lane-lines. The detection of lanes consists mainly of the finding of specific patterns/features such as the lane markings (colored segments) on painted roads surfaces. Such kind of specification streamlines or guides the process of lane detection. However, there are some situations, when it happened, can obstruct the lane detection. As an example, the existence of other cars on the same lane that hides out, fully or partially, the lane markings ahead of the ego car. Another example is the existence of scattered shadow regions caused by highway walls, buildings, trees, etc. In this paper, a vision-based approach capable of reaching a real-time performance in the detection and tracking of structured road boundaries (painted or unpainted lane markings) with a slight curvature, and is robust enough in the presence of shadow conditions, is proposed.

There are currently several vision-based road lane detection algorithms proposed in the literature to avoid fatal driving accidents [5-14]. One of these early endeavors is what is called the GOLD system, which is developed by Brogg [4]. In this system, the lane detection is done based on edge detection. The captured image is transformed to a new mapping based bird's eye view of the road. In this view, the lane boundaries or dashed lines appear very close to vertical lines with contrast color on a black background. An adaptive filtering method is employed to detect and isolate vertical line segments that can be interpolated to construct longer lane lines.

Moreover, during the same time, the LOIS algorithm is proposed by Kreucher et. al. [6] which is based on a deformable template approach. All possible forms that lane markings can take place in an image are parametrized is a collection of shapes. Then, an evaluation function is constructed to give a numerical value to how well a particular lane shape/marking is matching the pre-specified parametrized lane forms. Then, the maximum value of this function, at a particular position in the image, is used to highlight that a lane is detected.

An early endeavor is carried out by Carnegie Mellon University by developing a system called AURORA [7]. This system uses a color camera mounted on the side of the car and pointed downwards the road. The camera is used to track the lane markings exist on a structured road surface. AURORA uses only a single scan line and applied it to the image to detect the lane markings.

Ran et al in [8] proposed an algorithm that can deal with painted and unpainted roads. The algorithms use some color cues to perform image segmentation and remove shadows.

978-1-5386-9207-3/18/$31.00 ©2018 IEEE

Specifically, Hough transformation [10] is used and applied to edge images to detect the lane boundaries with the assumption that the lane lines are long enough with smooth curving.

Hough transform is used again by *Assidiq et al* [8] in his proposed vision-based lane detection algorithm. *Assidiq et al* tried to reach real-time performance with adequate robustness for lighting change and performs well in under shadow areas. The algorithm used a pair of hyperbolas that are fitted to the lane edge positions.

Approaches based on neural networks and deep learning [11-14], and specifically convolutional neural networks (CNN) stimulate a promising research direction despite its overwhelming computational overhead. In other words, using deep learning is a promising approach regards performance, however, it is very slow in terms of detection and tracking speeds (4 to 5 frames per second on a very high profile hardware). It requires expensive computational resources (GPUs are a must) that make these approaches not suitable for ADAS features in the near future. Considering that the lane detection runs on vehicle-based systems, where computation resources are severely limited, the computational cost of a lane detection method should also be considered as a key indicator of the overall performance.

In this paper, to strive for a generalized, low computational cost, and real-time vehicle-based solution, a lane detection method called *LaneRTD* (Lane Real-Time Detection) is proposed as a further step towards the prospects of autonomous driving. The main innovation of the proposed solution is the delicate balance between the speediness, the small footprint (low requirements of computing resources in terms of memory and CPU), and the reliability and robustness that suit at least the ADAS applications requirements.

II. Overview of the LaneRTD Algorithm

One of the advantages of the *LaneRTD* algorithm is that it is only using a single CCD camera. This camera should be mounted on the car's front-windshield mirror to capture the front view of the road. By assuming that the baseline is adequately horizontal, then the horizon is assured to be in the image parallel to the X-axis. However, the calibration data of the camera can be used to fix the orientation if the horizon is not adequately parallel to the X-axis. For the matter of clarification, the lane boundaries are arranged in pairs of markings, which usually have rectangular forms (or approximate).

In this paper, the captured RGB color image of the camera has a size of 960x540 pixels. This image is then converted to grayscale, as the first step of the *LaneRTD*, in order to lower the processing (execution) time. The next step the algorithm does is lowering the noise in the grayscale image, to avoid incorrect edge detection, by applying the Gaussian Blur (smoothing) algorithm [15]. The Canny algorithm [16] with automatic thresholding, is then used for edge detection and produces what is called an "edged image", which is a much-simplified image. To improve the focus and accuracy of finding the lane lines, The Region of Interest (ROI) is then extracted from the edged image. As the next step, the edged image with ROI is then sent to the line detector function, which will identify the right and left lane boundary segments. Actually, the horizon is determined by finding the projected intersection of these two line segments.

The Hough transformation [10] is then used to detect potential straight line segments, in the edged image, that can be later part of the lane boundary. The search for these line segments is only done within the ROI. The detected line segments are then grouped together in a way to construct both the left and right lanes. Two straight lines will be fitted to the grouped line-segments and point to represent the lane boundaries. For visualization purposes, the *LaneRTD* algorithm is illustrated by the flowchart in Fig. 1, and the resultant straight lane lines are displayed on the original color image as shown by the example in Fig. 2.

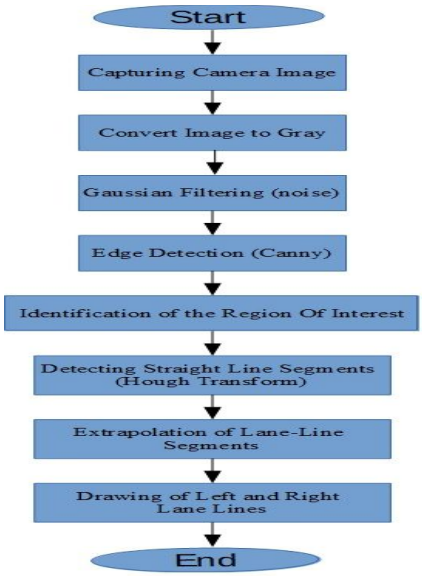

Fig. 1. The flowchart of the *LaneRTD* algorithm.

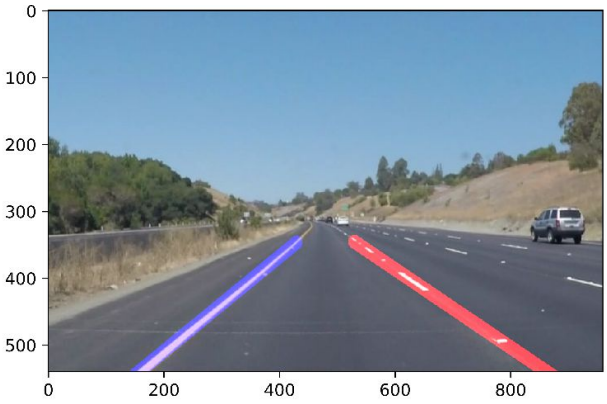

Fig. 2. Detected lane boundaries by the LaneRTD algorithm.

III. Canny Edge Detector

The Canny edge detection algorithm is mainly implementing the following 5 steps in order or execution [17]:

1) Applying a Gaussian filter for noise removal and image smoothening.
2) Computing the intensity gradients for all the pixels in the image.
3) Applying a process called "non-maximum suppression" to avoid unauthentic response to edge detection.

4) Applying a double-threshold categorization to evaluate edges, and determine the potential ones.

5) Evaluating edges by categorization: completing the detection of edges by removing all the other edges that are in the low category or are weak but not associated (close to or connected) to edges in the high category.

The convolution between a carefully designed Gaussian filter and the designated image produces a smoothed image. This smoothing step reduces the effects of the recognizable noise on the edge detector. The equation for a Gaussian filter kernel of size $(2k+1)\times(2k+1)$ is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right);$$
$$1 \le i,j \le (2k+1) \quad (1)$$

The educated choice of the Gaussian kernel size is crucial to the overall performance of the edge detector. Larger kernel sizes result in lower sensitivity of the detector toward apparent noise. Moreover, the larger the Gaussian filter kernel size, the larger the localization error for edge detection. Some recommendations [17] show that a 5×5 is an appropriate size for most use cases. However, it may differ depends on the specifics of application, so smaller or larger sizes are not totally out of the question.

An edge in an image may point in a variety of directions, so the Canny algorithm uses four filters to detect horizontal, vertical and diagonal edges in the blurred image [17]. The edge detection operator returns a value for the first derivative in the horizontal direction (G_x) and the vertical direction (G_y). From this the edge gradient and direction can be determined as:

$$G = \sqrt{G_x^2 + G_y^2} \quad (2)$$

$$\theta = \tan^{-1}\frac{G_y}{G_x} \quad (3)$$

Non-maximum suppression is an edge thinning technique that is applied to "thin" the edge. After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. Thus it will nullify all the pixels' gradient values (by assigning them to zeros) except the ones with locally maximum values. Those ones indicate locations with the sharpest change of intensity value. The technique used for each pixel in the image with gradients is:

1) For the current pixel, its edge strength is compared with the other pixels' strengths in both negative and positive gradient directions.

2) If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (i.e., the pixel that is pointing in the x-direction, it will be compared to the pixel to the right and to the left of it in the horizontal axis), the value will be preserved. Otherwise, the value will be suppressed.

After the non-maximum suppression process is being applied, significantly more accurate and real edges will be demonstrated in the image. Nevertheless, still some false edges and their associated pixels, which is considered noise, need to be removed or in other words, filtered out. This can be accomplished by extracting only edges with high gradient values and eliminate the weak ones. This can be achieved by thresholding, using two threshold values, high and low. If the edge gradient is higher than the high threshold value, the edge pixels will be preserved and highlighted as a strong edge. Otherwise, if the edge gradient is lower than the low threshold value, it is considered as a noise and removed. Moreover, if the pixel gradient value is in between the low and high threshold, it will be kept but marked as a weak edge.

It is that, of course, the strong edge pixels should contribute to the final edged image, as they represent the true edges. Nevertheless, there are different opinions on the weak edges, how they should contribute to the final edged image. A very good rule of thumb is the position of these weak pixel edge values. If they are close or connected to the strong pixel value, then this association strengthen this pixel and should be preserved. On the other hand, if the weak-edge pixel is isolated, then this pixel can be removed. Blob analysis [18] is a very good tool to track such an association between the weak-edge pixel and strong edge pixel, by checking the eight surrounding neighborhood pixels. If one of them is strong, then this weak-edge pixel should be identified as a preserved one, otherwise, it is identified as noise.

IV. HOUGH TRANSFORM

Hough transform can be used to detect straight lines, circles, ellipse, and other arbitrary shapes in images. It finds the location of lines in an image as given by equations (4) → (6) [19]:

$$y = mx + c \quad (4)$$

rewriting the above line equation as:

$$c = (-x)m + y \quad (5)$$

$$c = (-x_i)m + y_i; for\ every\ point, i \quad (6)$$

The straight line is represented by equation (6) in the parameter space as a point (c, m). Edge detection is used as pre-processing before applying the Hough transform. The input to Hough transform is a threshold edge image:

1) Quantize or divide the parameter space into small cells (called also as the *accumulator*).

2) Then the process of voting is implemented. For each edge point, increment the value of the associated array element (accumulator bin) (a 2-dimensional array is constructed to represent the parameter space) as in equation (7) for all possible values of m and c:

$$P[c_j, m_i] = P[c_j, m_i] + 1 \quad (7)$$

then the array element (parameter cell) with the maximum value is taken as a solution.

3) Find the local maxima in the parameter space by finding the accumulator bins with the highest values, and then applying some form of a threshold to extract the most likely lines.

4) Perform Polar representation of lines:

$$\rho = x\ cos\theta + y\ sin\theta \quad (8)$$

where θ is calculated from the magnitude of gradients vectors (S_x, S_y) as:

$$Magnitude\ (S_x + S_y), \rho = \sqrt{S_x^2 + S_y^2} \quad (9)$$

$$Direction, \theta = \tan^{-1}\frac{S_y}{S_x} \quad (10)$$

Each Hough line (line segments) can be identified by its two endpoints [20]. Hough transform can be modified to predict the curves as well which represent lane markers in particular during turning [21].

V. Implementation of the LaneRTD Algorithm

The algorithm is implemented using Python and OpenCV computer vision library [22] and the following steps describe the implemented pipeline in order of execution:

1) Reading test images: It simply reads all the images in the designated directory "test_images" one at a time in alphabetical order. An example of these images is shown in Fig. 3.

2) Converting the color test image to grey: this is done using the OpenCV function "ctvColor". Fig. 4 shows the image in Fig. 3 in grayscale.

3) Filtering the noise: this is done using the OpenCV function "GaussianBlur" which executes the Gaussian filter algorithm. A kernel size of 5 has been selected. Fig. 5 shows the image in Fig. 5 after filtering.

4) Edges Detection and Extraction: this is done using the OpenCV function "Canny" which executes the well-known Canny algorithm [16]. Fig. 6 shows the image in with edges detected. The following table (TABLE I.) lists the several parameters for the algorithm operation that have been selected after careful tuning and many trial and errors sessions.

TABLE I. Canny algorithm parameters.

Parameter	Value
Low threshold	50
High threshold	150

5) The Identification of the region of interest: This is implemented by masking a trapezoidal area in an image with edges detected (e.g. Fig. 6) to produce an image which has only lines corresponding to road lane lines as shown in Fig. 7. The vertices of the identified region of interest is shown below in TABLE II.

TABLE II. Area of interest vertices.

Vertex	X	Y
Lower Left	0	539
Lower Right	959	539
Upper Left	450	330
Upper Right	490	330

6) Detecting Straight Lines: straight lines are identified using the Hough algorithm [10] in polar coordinates using the OpenCV HoughLinesP() function. The resultant Hough line segments are depicted in Fig. 8. Moreover, additional functionality is added to not only calculate Hough line segments but also to draw extrapolated lines on the original image as shown in Fig. 9 & Fig. 10. The left lane line is always drawn in blue and the right lane-line is drawn in red. The parameters that have been found, after careful and recursive tuning, and used in the Hough algorithm are listed in TABLE III.

TABLE III. Hough Transform Parameters.

Parameters	Value
rho (ρ)	1
theta (θ)	$1^{o} = \pi/180$
minimum_votes_threshold	15
min_line_length	7
max_line_gap	3
line_thickness	1

7) Tracking Lane Lines: After successful detection of lane lines in stationary images as per the description in steps 1 => 6, the tracking of the lane in real-time videos are just the application of these steps for each camera frame. Some smoothing techniques are applied for the transition from frame to the next as described in Section VI.

Fig. 3. The original image.

Fig. 4. The original image in grey scale.

Fig. 5. The image in grey after the Gaussian Blur Filter is applied.

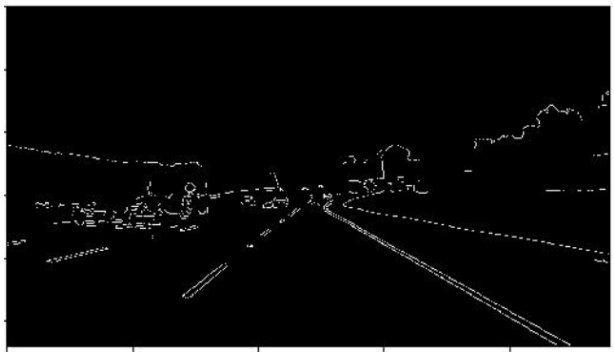

Fig. 6. The image after the Cunny algorithm is applied.

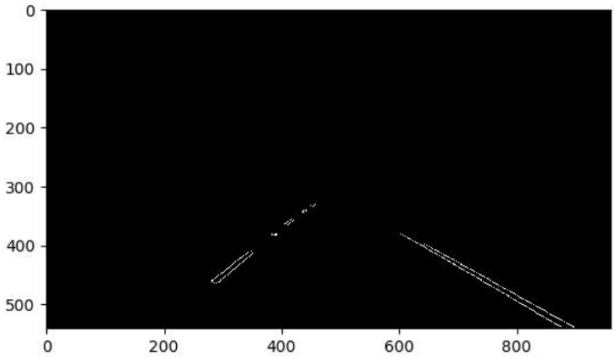

Fig. 7. The image after extracting the area of interest.

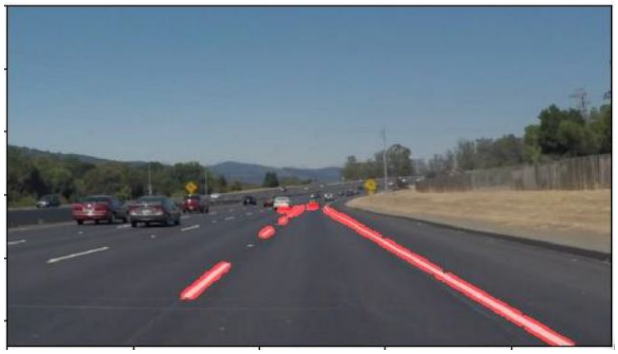

Fig. 8. The image with Hough line segments.

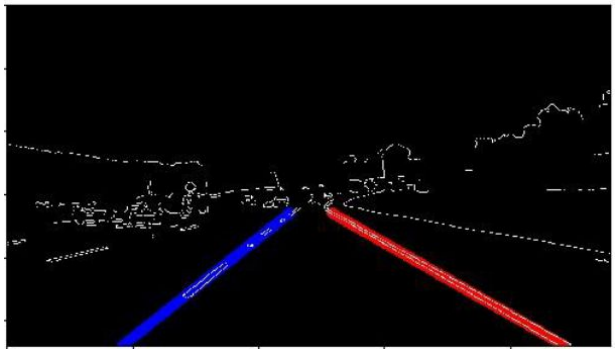

Fig. 9. The image after drawing lane lines (blue and red) by extrapolating the Hough lines segments.

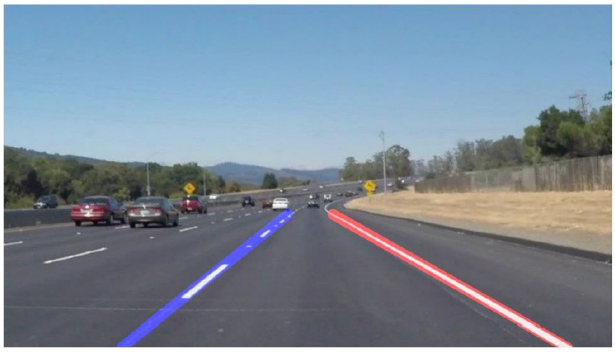

Fig. 10. The image after drawing lane lines (blue and red) by extrapolating the Hough lines segments.

VI. THE IMPLEMENTED LINE-DRAWING FUNCTION

The line drawing function "Draw_Lines()" is implemented to connect the line segments for each lane-line (left or right) to produce one solid line that tracks the actual lane-line in the images. The line segments are produced by Hough transform as shown by Fig. 8 and the "Draw_Lines()" function is connecting them to be like the ones in Fig. 9 and Fig. 10. The function is achieving that through the following steps:

1) Classification (Left or Right): Classifying all the Hough line-segments as to belong to the left or right lines. This is done based on the slope of the line segment. If the slope is positive and lies between 0.4 => 1.0 then this segment belongs to the left line class, and if it is negative and -0.4 => -1.0 then it belongs to the right line class.
2) For all the classified Left & Right line-segments, the lengths and intercepts (intersections with the x-axis) are calculated and stored along with their slopes.
3) A line fitting technique is used for each class (Left and Right) taking the slope of each line segment as an indication of a good or a bad (noise) line segment.
4) Now the Left and Right lines can be drawn, however, to reduce the jitter, the information of the previous frames are incorporated to produce an N^{th} order filter (it has been tested till the 30^{th} order).
5) The FIR filter equation considered in this implementation is given as:

$$Y_k = a_o*X_k + a_1*X_{k-1} + a_2*X_{k-2} + a_3*X_{k-3} \ldots a_n*X_{k-n} \quad (11)$$

The resultant FIR filter parameters and coefficients used in this implementation are given below in TABLE IV. :

TABLE IV. FIR FILTER PARAMETERS.

Parameters	Value
Order	7
a_o	0.075
a_1	0.125
a_2	0.175
a_3	0.250
a_4	0.175
a_5	0.125
a_6	0.075

6) The resultant lines slopes and intercepts and using the information of "the region of interest" calculated in step 5 of the pipeline (V) are used to draw the Left line in blue and the right line in red.

VII. TESTING AND VALIDATION

The developed LaneRTD algorithm is further tested on many images representing different scenarios. Samples of the results of the testing are shown in Fig. 11, Fig. 12, Fig. 13 and Fig. 14. The presented results show that the algorithm performs very well under different conditions. Furthermore, for robustness testing and validation of the developed pipeline, the algorithm is applied to several real-time video samples representing different driving conditions. The LaneRTD proved to be very robust except on one condition, where there are various scattered areas of shadows that misleads the algorithm to produce inaccurate lane line detections as shown in Fig. 15. Therefore, this problem needs to be addressed in future work.

The pipeline proved to be acceptably fast in execution to be used in real time. Using an Intel Core i5 with 1.6 GHz and 8 GB RAM which very moderate computational platform, the following measurements are collected for three sample testing video streams:

TABLE V. COMPUTATION SPEED FOR THE LANERTD ALGORITHM.

Sample Name	No. of Frames	Total Time Sec	Frame per Sec
solidYellowLeft Video	682	48.0	18.76
solidWhiteRight Video	222	8.0	25.6
Challenge Video	251	23	10.9

The lowest measured processing speed is 11 frames per second, which is more than enough for the proper execution for accurate lane-line detection. Actually, a rate of 10 frames per second is believed to be adequate for this application.

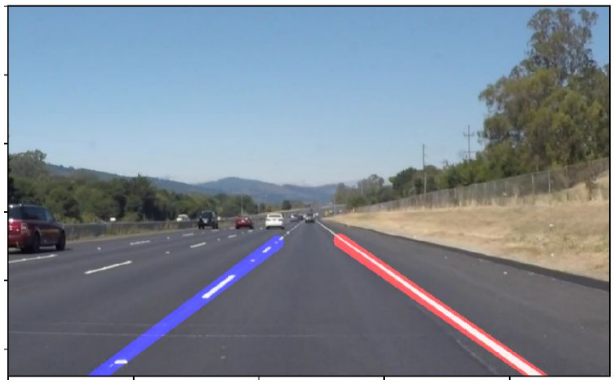

Fig. 11. Test image with solid and dotted white lane lines segments (right lane with cars).

Fig. 12. Test image with solid yellow and dotted white lane lines segments (left turned lane with no cars).

To compare these results with the state-of-the-art in lane detection. The work done by *Lee et al* in [13] used successfully an end-to-end fully convolutional neural network for lane detection and tracking, however, their model is tested on NVIDIA GTX Titan X hardware [23] to reach 20 frames per second performance. However, the used hardware is extremely expensive (~2000 USD) and it is at least 100 times faster than the hardware used in this paper. Accordingly, this method is not applicable in current *ADAS* systems or even in the near future.

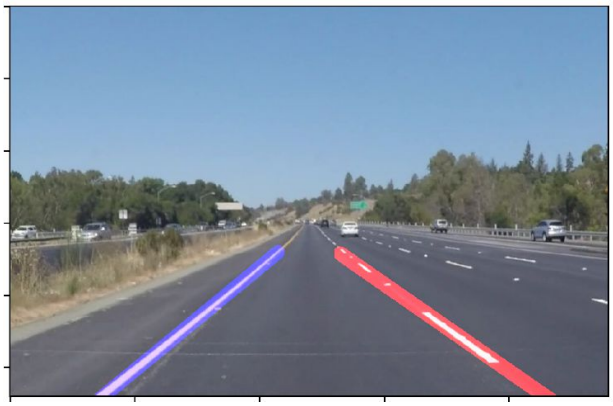

Fig. 13. Test image with solid yellow and dotted white lane lines segments (left straight lane with cars).

Fig. 14. Test image with solid yellow and dotted white lane lines segments (left lane with no cars).

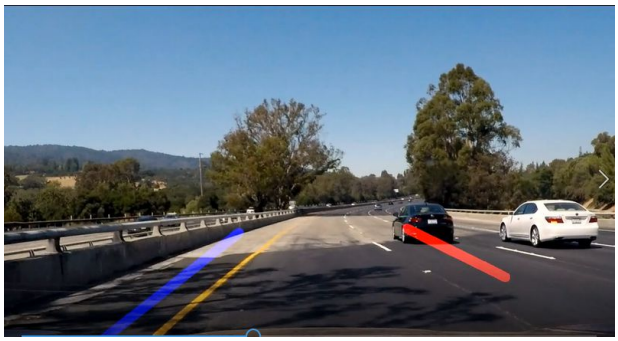

Fig. 15. Inaccurate lane lines detection due to shadow patterns.

VIII. A SHORTCOMING OF THE IMPLEMENTED APPROACHES

The following list summaries the identified shortcomings:

1) Even though a high order FIR filter is used to reduce the jitter of the left and right lane-lines, however, the results still need improvement especially at light color roads and complex shade patterns.

2) The developed algorithm only detects the straight lane lines. It still needs more work to handle curved lanes (or the curvature of lanes). The investigation of using perspective transformation should be carried out as well as poly fitting lane lines rather than fitting to straight lines.

3) How the current approach can handle up-hill or down-hill roads, is still a big question? How to determine the region of interest in this case. Needs further study.

4) Testing with having a very close car in front of the ego car need to carried-out. Also, how this approach can handle this case needs more investigation.

IX. SUGGESTED IMPROVEMENTS

The following list summaries the suggested improvements:

1) Using line-segment length as a differentiator between strong line segments and weak line segments while applying the line–fitting technique.

2) More investigation needs to be done in the design of the FIR filter: trying higher orders, trying different low pass polynomials like Butterworth, Chebychev, Elliptical … etc.

3) Identification for the type of the lane (dashed or solid) is important in driving regulations, so it needs to be added.

X. CONCLUSION

In this paper, a fast and reliable lane-lines detection and tracking technique is developed, presented thoroughly and given the name "*LaneRTD*". *LaneRTD* uses a pipeline of well-known algorithms like Canny edge detection and Hough transform. Moreover, the pipeline uses a comprehensive lane line detection and drawing technique to produce the final output. The proposed technique needs only raw RGB images from a single CCD camera mounted behind the front windshield of the vehicle. The performance of the LaneRTD is tested and evaluated using tons of stationary images and many real-time videos. The validation results show fairly accurate and robust detection except in one scenario where complex shadow patterns exist. The measured throughput (execution time) using an affordable CPU proved that the LaneRTD is very suitable for real-time lane detection without much overhead. Therefore, the proposed technique is well suited to be used in Advanced Driving Assistance Systems (ADAS) or self-driving cars.

A comprehensive discussion and analysis regarding the usefulness and the shortcomings of the proposed technique as well as suggestions for improvements and future work are presented.

REFERENCES

[1] Karim Mansour, Wael Farag, "AiroDiag: A Sophisticated Tool that Diagnoses and Updates Vehicles Software Over Air", *IEEE Intern. Electric Vehicle Conference (IEVC)*, TD Convention Center Greenville, SC, USA, March 4, 2012, ISBN: 978-1-4673-1562-3.

[2] Wael Farag, Zakaria Saleh, "Traffic Signs Identification by Deep Learning for Autonomous Driving", *Smart Cities Symposium (SCS'18)*, Bahrain, 22-23 April 2018.

[3] Wael Farag, "Recognition of traffic signs by convolutional neural nets for self-driving vehicles", *International Journal of Knowledge-based and Intelligent Engineering Systems*, IOS Press, Vol: 22, No: 3, pp. 205 – 214, 2018.

[4] Wael Farag, "CANTrack: Enhancing automotive CAN bus security using intuitive encryption algorithms", *7th Inter. Conf. on Modeling, Simulation, and Applied Optimization (ICMSAO)*, UAE, March 2017.

[5] B. M, Broggi, "GOLD: A parallel real-time stereo Vision system for generic obstacle and lane detection", *IEEE Transactions on Image Processing*, 1998, pp. 4-6.

[6] C. Kreucher, S. K. Lakshmanan, "A Driver warning System based on the LOIS Lane detection Algorithm", in the *IEEE Intern. Conf. On Intelligent Vehicles*, Stuttgart, Germany, 1998, pp. 17 -22.

[7] M. Chen., T. Jochem D. T. Pomerleau, "AURORA: A Vision-Based Roadway Departure Warning System", in the *IEEE Conference on Intelligent Robots and Systems*, 1995.

[8] B. Ran and H. Xianghong, "Development of A Vision-based Real Time lane Detection and Tracking System for Intelligent Vehicles", *In 79th Annual Meeting of Transportation Research Board,* Washington DC, 2000.

[9] A. Assidiq, O. Khalifa, M. Islam, S. Khan, "Real time lane detection for autonomous vehicles", *Intern. Conf. on Computer and Communication Eng.*, May 13-15, 2008 Kuala Lumpur, Malaysia

[10] R.O. Duda, and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures", *Comm. ACM*, Vol. 15, pp. 11–15 (January 1972).

[11] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, and R. Chengyue, "An empirical evaluation of deep learning on highway driving", *Computer Science*, 2015.

[12] Y. Jiang, F. Gao, and G. Xu, "Computer vision-based multiple-lane detection on straight road and in a curve", *In Intern. Conf. on Image Analysis and Signal Processing*, pages 114–117, 2010.

[13] S. Lee, I. Kweon, J. Kim, J. Yoon, S. Shin, O. Bailo, N. Kim, T.-H. Lee, H. Hong, and S.-H. Han, "Vpgnet: Vanishing point guided network for lane and road marking detection and recognition", *In 2017 IEEE Intern. Conf. on Computer Vision (ICCV)*, pp. 1965–73, 2017.

[14] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep neural network for structural prediction and lane detection in traffic scene", *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):690–703, 2016.

[15] Mark S. Nixon and Alberto S. Aguado, "Feature Extraction and Image Processing", *Academic Press*, 2008, p. 88.

[16] J. Canny, "*A Computational Approach To Edge Detection*", IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[17] Wikipedia, https://en.wikipedia.org/wiki/Canny_edge_detector, (retrieved 8/23/18)

[18] Michael B. Dillencourt, Hannan Samet, Markku Tamminen, "A general approach to connected-component labeling for arbitrary image representations", *Journal of the ACM*. 39 (2): 253. (1992).

[19] Jasmine Wadhwa, 1G.S. Kalra and 2B.V. Kranthi, "Real Time Lane Detection in Autonomous Vehicles Using Image Processing",

Research Journal of Applied Sciences, Engineering and Technology 11(4): 429-433, 2015. DOI: 10.19026/rjaset.11.1798.

[20] V. Stefan, S. Constantin and D. Rudiger, "Roadmarking analysis for autonomous vehicle guidance", *Proceeding of the 3rd European Conference on Mobile Robots (EMCR)*, 2007.

[21] S. Yuan and T. Zheng, "Vision based lane detection in autonomous vehicle", *Proceeding of the 5th World Conference on Intelligent Control and Automation*, 6: 5258-5260, 2004.

[22] OpenCV Python Library, https://opencv.org/.

[23] NVIDIA GTX Titan X, https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/, (retrieved 11/05/18)