Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 133 (2018) 594–603

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Robotics and Smart Manufacturing (RoSMa2018)

Fast and accurate on-road vehicle detection based on color intensity segregation

Manne Sai Sravan[a], Sudha Natarajan[b,*], Eswar Sai Krishna[c], Binsu J Kailath[c]

[a]*Arizona State University, Tempe-85281, USA*
[b]*Tata Elxsi Limited, Chennai-600113, India*
[c]*IIIT DM Kancheepuram, Chennai-600127, India*

Abstract

A new method for detection of on-road vehicles based on color intensity segregation is proposed. This method has two steps. Firstly, details such as pavements or lanes in the image frame are utilized to extract the region of interest. Secondly, a new filter is proposed that utilizes the intensity information to filter the illumination variations, shadows and cluttered backgrounds from the extracted region of interest and detect the vehicles subsequently. The proposed method is evaluated on videos from KITTI vision benchmark suite and on our videos recorded during cloudy and rainy days with variable resolution. The experimental results demonstrate the effectiveness of the proposed vehicle detection method by achieving 90% detection rate and also reduced computational load on hardware, making it suitable for real-time applications.

Keywords: Vehicle detection; Color intensity segregation; Autonomous driving

1. Introduction

In the foreseeable future, self-driving vehicles would emerge as the mode of transportation around the world, and the only question is how long it would take? Currently, the major challenge being confronted by this technology is commercial viability. In order to make this technology commercially viable, numerous new methods and algorithms are being developed to achieve better performance and simplify the hardware. In particular, the vision-based approach

* Corresponding author.
E-mail address: sudhanatraj@yahoo.com

10.1016/j.procs.2018.07.090

has gained prominence in the past lustrum because of its cost effectiveness and the hardware involved.

The vision-based approach is open to several challenges such as illumination variations, shadows, cluttered background, varying viewpoints and partial observation due to camera's field of view. Detection of other on-road vehicles is an important task in autonomous driving. To date, many works are published that discuss various vehicle detection methods using a monocular camera, while addressing the above-mentioned issues. The various methods reported until now can be broadly classified into knowledge-based, feature-based, learning-based, stereo-based, and motion-based [1]. In knowledge-based approach, predetermined data about vehicle characteristics such as color, size, and shape are utilized to detect the vehicle from the background [2]. In feature-based approach, the first stage involves feature extraction and region proposals are made based on the extracted features. In the second stage, a classifier is used to identify the vehicles out of the region proposals made [3-5]. In learning-based approach, multi-layered artificial neural networks such as Convolutional Neural Networks [6] are trained to detect the on-road vehicles. This approach is computationally intensive and usage of GPU is suggested by authors to improve the processing speeds.

In stereo-based approach, the depth information is used to distinguish the vehicle from the background. Further, disparity mapping [7] and inverse perspective mapping [8] are two widely used stereo matching methods. In motion-based approach, the velocity data of the ego vehicle is utilized to separate the moving objects (vehicles) from the stationary ones (background) [9-11]. This method is indifferent to vehicles which are moving at zero velocity with respect to the ego vehicle. No matter what the algorithm is, the output desired is the same, but, in order for a method to be employed in Advanced Driver Assistance System (ADAS), it should operate reliably in real world scenarios, and meet the real-time processing speeds required for ADAS systems while utilizing minimal hardware.

Motivated by the preceding discussions, we proposed a novel method for on-road vehicle detection in this paper. In this method, we designed a new algorithm that can efficiently separate the vehicles from the selected ROI. The proposed method is robust to variable illumination conditions such as sunny, cloudy, rainy and also, to shadows and cluttered backgrounds. The effectiveness of the proposed method has been demonstrated on KITTI datasets [12]. The paper is organized as follows: Section 2 discusses the complete vehicle detection method. Details of the proposed algorithm for separation of vehicles from the selected ROI are discussed in Section 3. The method is implemented in both MATLAB and Python. In Section 4, the results obtained in each stage of the implemented method are given. Section 5 concludes the paper.

2. System Overview

Fig. 1 presents the outline of the proposed method. A brief description on the various steps in the proposed method is as follows:

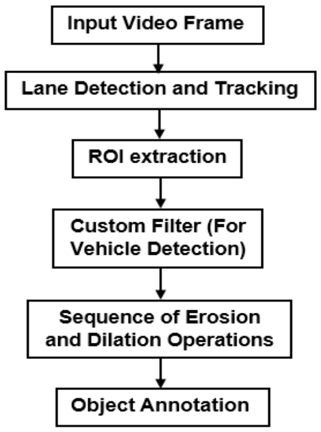
Input Video Frame
Lane Detection and Tracking
ROI extraction
Custom Filter (For Vehicle Detection)
Sequence of Erosion and Dilation Operations
Object Annotation

Fig. 1: Flow chart of the proposed method

2.1 Lane Detection and Tracking

The input video frame (grayscale image) is initially truncated by a factor (F_w) times the height to remove the upper region containing sky, buildings and other unnecessary data. By observing the KITTI dataset it is safe to assume that half of the image frame can be truncated. A convolution operation using the kernel $[-1, 0, 1]$, followed by a threshold operation is done to convert the gray scale image to binary form. Hough Transform is then applied to find out the dominant lines in the binary image. Further, the lane tracking algorithm as given in [13] is finally employed.

2.2 ROI Extraction

The lanes are detected in the previous step and the area in between the lanes is the region of interest. This region can be extracted by simple masking operation. This step is crucial because it will reduce the overall computations that have to be performed in further steps.

2.3 Custom Filter for Vehicle Detection

This is the key step in vehicle detection. A detailed description of the algorithm involved in this step is discussed in Section 3.

2.4 Erosion and Dilation

The goal of this operation is to connect all edges belonging to a particular vehicle and to convert the whole object into a white blob in order to generate the contour or bounding box in the next stage. Erosion followed by a series of five dilation operations (five iterations) is performed on the output (binary image) from the previous stage. The structuring element used here is of circular shape of radius 12 units. The size of the structuring element (S_r) and the number of erosion and dilation operations (N_e, N_d) that ought to be performed are purely determined by trial and error method.

3. Proposed Vehicle Detection Algorithm

Fig. 2 presents a brief outline of the various stages of the proposed algorithm.

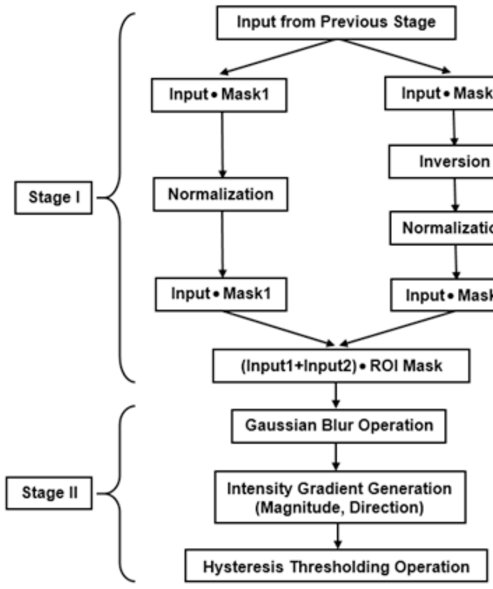
Input from Previous Stage
Input • Mask1
Input • Mask2
Inversion
Stage I
Normalization
Normalization
Input • Mask1
Input • Mask2
(Input1+Input2) • ROI Mask
Gaussian Blur Operation
Stage II
Intensity Gradient Generation (Magnitude, Direction)
Hysteresis Thresholding Operation

Fig. 2: Flow chart of the Custom Filter

A detailed description of the various processes involved in the individual stages is as follows:

3.1 Stage 1

The goal of this stage is to suppress the high-intensity regions in the image frame, while, improving the low intensity regions. This entire operation can be termed as non-linear normalization. This process is crucial in order to remove the noise due to illumination variations, and shadow cluttered background.

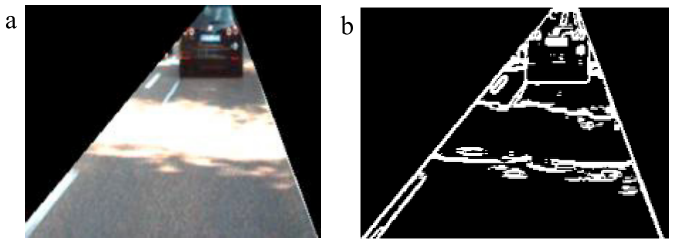

Fig. 3: (a) Extracted ROI. (b) Noise caused due to illumination variation.

The operation performed in Stage 1 is expressed in Eq. (1).

$$I_N[i,j] = \begin{cases} \left(\dfrac{0.25-0.3}{1-0.3}\right)\left(I[i,j]-0.3\right)+0.3, & \text{for } 0.3 \le I[i,j] \le 1; \\ \left(\dfrac{0.3-0.2}{0.3-0}\right)I[i,j]+0.2, & \text{for } 0 < I[i,j] < 0.3 \end{cases} \tag{1}$$

where, $I[i,j]$ is the intensity value of image pixel at position *i,j*, $I_N[i,j]$ is the newly computed intensity value. Here, the intensity values within 0.3 to 1 are normalized between 0.25 and 0.3, and the values upto 0.3 are normalized between 0.2 and 0.3.

However, in order to implement this expression, we need to run through individual pixel values followed by an if-else conditional statement. This process will certainly constrain the processing speed. Instead, the masking operations presented in Stage 1 are employed. Initially, two masks are applied. Mask 1 separates pixel intensities between 0 and 0.3 (Eq. 3) while Mask 2 separates the ones between 0.3 and 1 (Eq. 2). Image processing frameworks such as OpenCV and MATLAB have built-in functions to create masking matrices. These functions tend to be faster as they make use of Basic Linear Algebra Subprograms (BLAS) at the core level for execution. Then the inversion operation that is performed after masking with Mask2 is represented in Eq. (4).

$$I_{N2} = I \bullet Mask2 \tag{2}$$

$$I_{N1} = I \bullet Mask1 \tag{3}$$

$$I_{N2} = 1 - I_{N2} \tag{4}$$

where I_{N2}, I_{N1} are the outputs matrices after masking operation, I is the extracted ROI matrix or input to the Stage 1. Thus, by separating the elements of the matrix we were able to process the conditional statements in Eq. (1). Next, the normalization process can be performed on the entire matrices (I_{N1}, I_{N2}) simultaneously. The input and the output frames of Stage 1 are shown in Fig 4.

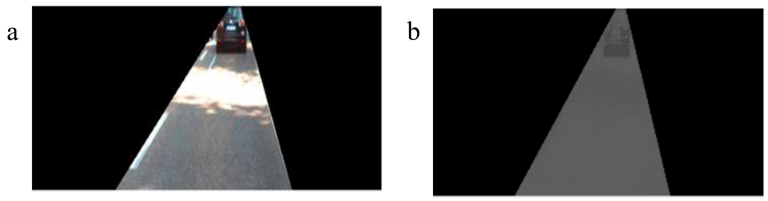

Fig. 4: (a) ROI input to Stage 1. (b) Output of Stage 1

3.2 Stage 2

After performing the series of masking and normalization operations as stated in Stage 1, a smoothening operation is performed on the output matrix to suppress the noise and the texture variations. The Gaussian filter is used for this purpose. The two-dimension form of Gaussian filter response, with standard deviation (σ) as parameter is given by Eq. (5). The distribution generated by this function is discretized to form a kernel of dimension (5×5) for a standard deviation of 0.743. The implementation of this discretization is available in [14], [15].

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5}$$

where x and y are distances from the origin in horizontal and vertical directions respectively. σ is the standard deviation of the Gaussian distribution.

The following is the kernel matrix.

$$\begin{pmatrix} 0.000457 & 0.004895 & 0.01068 & 0.004895 & 0.000457 \\ 0.004895 & 0.052402 & 0.11432 & 0.052402 & 0.004895 \\ 0.01068 & 0.11432 & 0.249402 & 0.11432 & 0.01068 \\ 0.004895 & 0.052402 & 0.11432 & 0.052402 & 0.004895 \\ 0.000457 & 0.004895 & 0.01068 & 0.004895 & 0.000457 \end{pmatrix}$$

Since the Gaussian filter is a separable filter, the same filter operation involving a two-dimensional kernel can be achieved by convolving the input matrix with two one-dimension kernels in horizontal and vertical direction respectively. This operation would require fewer computations when compared to convolution operation using a two-dimensional kernel. The below matrix represents the one-dimensional kernel.

$$\begin{pmatrix} 0.021385 & 0.228914 & 0.499402 & 0.228914 & 0.021385 \end{pmatrix}$$

Sobel operator is then used to compute the intensity gradient of the filtered image frame [15]. The gradient magnitude (G_M) and direction matrices (G_P) are utilized to identify the local maxima and minima in the intensity gradient matrix. This process is executed as follows:

- Each element in the gradient direction matrix is transformed into one of the four directions: $0^\circ, 45^\circ, 90^\circ, 135^\circ$
- A new matrix (N) with the same dimension as the gradient magnitude matrix is formed and if

  $G_M(p_a)$, $G_M(p_b) < G_M(p)$, then $N(p) = G_M(p)$, otherwise $N(p) = 0$.

  where G_M is the gradient magnitude matrix computed using Sobel-operator, p is the current pixel, p_a and p_b are two neighbouring pixels in the direction of gradient.

Here, $p = (i, j)$ and for a gradient direction of

$$\begin{aligned}
0^\circ &: p_a = (i, j-1), p_b = (i, j+1), \\
45^\circ &: p_a = (i+1, j-1), p_b = (i-1, j+1), \\
90^\circ &: p_a = (i+1, j), p_b = (i-1, j), \\
135^\circ &: p_a = (i-1, j-1), p_b = (i+1, j+1).
\end{aligned}$$

This process suppresses the majority of the noise level in the input image frame and clearly pictures the edge patterns in the input image frame.

The hysteresis threshold operation [13] is performed next. This is a key operation that involves manual calibration to obtain the desired output. Here, the lower (T_l) and upper (T_h) threshold values to filter the edges are found out by trial and error method. Based on the threshold levels the edges in the image frame can be divided into three categories: edges whose value is greater than the upper threshold (Red), in between the upper and lower threshold (Yellow), and below the lower threshold (Blue). The edges whose value is below the lower threshold (Blue) can be completely discarded, and the edges whose value is above the upper threshold can be retained (Red). The edges marked in yellow are retained only if they are connected to those marked in red. Fig. 5 shows the hysteresis threshold operation. The output of Stage 2 in comparison to the noise present in the image is shown in Fig. 6.

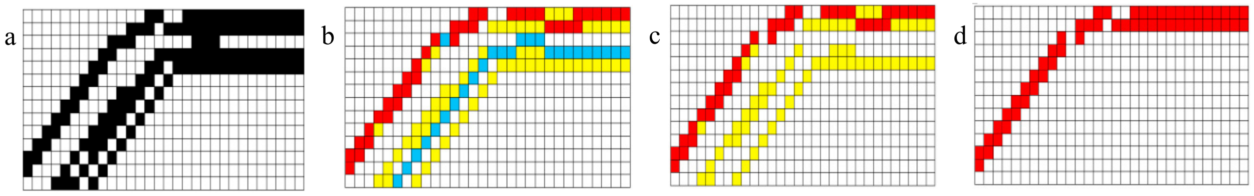

Fig. 5: (a) Edges in an image. (b) Edges categorized based on thresholds. (c) Image after discarding edges below the lower threshold (T_l). (d) Final image after hysteresis threhold operation

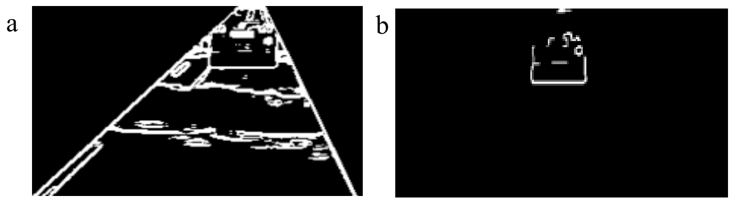

Fig. 6: (a) Noise present in the extracted ROI. (b) Output of Stage 2.

4. Experimental Results

The outputs of the intermediate stages of the algorithm are shown in the Fig. 7

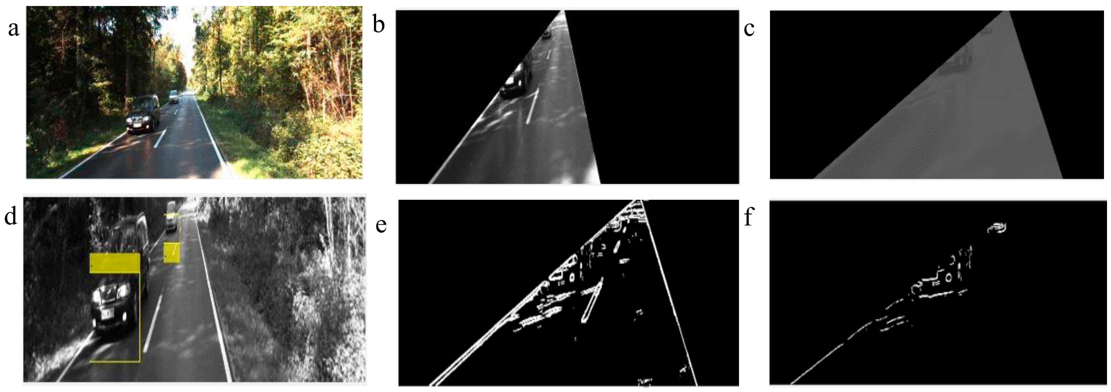

Fig. 7: (a) Input video frame. (b) Extracted ROI. (c) Output of Stage 1. (d) Final output of the algorithm (e), (f) Comparison between the noise in the ROI and the output of Stage 2.

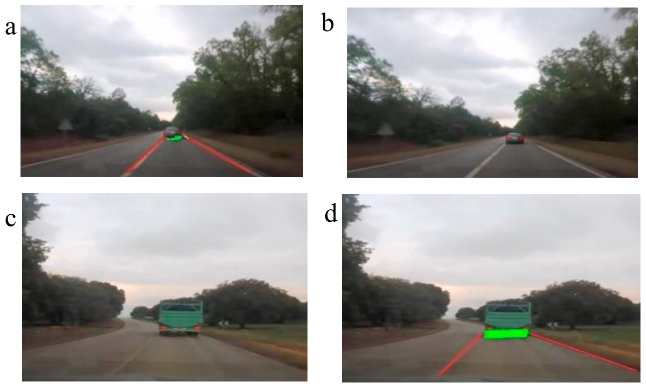

Fig. 8: (a), (c) Video taken on a cloudy day. (b), (d) Output of the algorithm

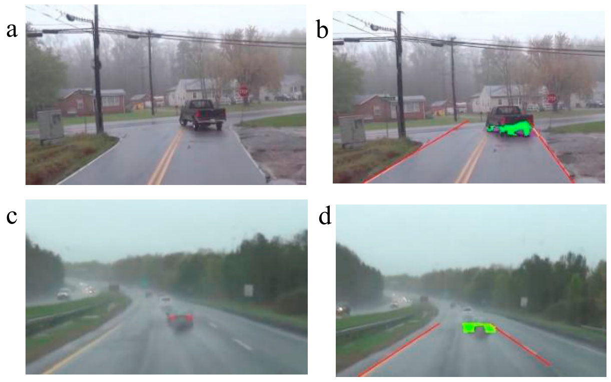

Fig. 9: (a), (c) Video taken on a rainy day. (b), (d) Output of the algorithm

In Figs. 8 and 9 the red lines indicate the ROI, and the green markings indicate the portion of the vehicle that falls within the ROI. It can be observed that the vehicles are properly detected even in cloudy and rainy weather conditions, however, in Fig. 9 (d) due to severe rain the noise level in the image has significantly risen, leading to additional area around the car being detected as vehicle. Further, in case of rainy weather conditions it is observed that there is a slight increase in the number of false-negative detections, while the number of false-positive detections remains the same, when compared to sunny and cloudy weather conditions. However, the overall detection rate is observed to be greater than 90%. Moreover, in an actual real-time system implementation a detection algorithm would be followed by a tracking algorithm that would filter the outliers to produce a consistent object detection in each frame, essentially producing an output independent of the weather conditions.

The algorithm was tested on video sequences that were captured at various times of the day. The video sequences depict challenges such as shadow cluttered backgrounds, markings and symbols on road, pavements, and broken lane marking. Also, the resolution of the video sequences varies from (240×320) to (512×1392). Sample sequences were initially used to determine all the thresholds (S_r, N_e, N_d, T_l, T_h) except (F_w). This is because the factor by which the video frame has to be truncated depends on the mounting height of the camera. This factor essentially remains constant for all the video sequences in a dataset, but when a video sequence from a different dataset is tested this factor is adjusted manually. For the actual system this parameter is included in the auto calibration system of the vehicle. The same set of thresholds (S_r, N_e, N_d, T_l, T_h) is used to test all video sequences stated in Table 1. It is observed that the selected thresholds worked well on all the video sequences tested, this can be justified by the fact that the binary image resulting from the hysteresis threshold operation is filtered in the initial stages to rectify illumination variations, showdown effects etc., and therefore, the calibrations made based on the binary image essentially remain constant when tested for various video sequences.

4.1 Detection Rate

The various challenges contained in the KITTI video sequences are listed in Table 1. They include shadows on road due to trees and buildings, broken white line lane markings, symbols and text written on road, pavements and lane change due to the front or the ego vehicle.

Table 1: Challenges addressed in the video sequence

Sequence	Shadows on Road	Broken white lane markings	Lane Change	Symbols on Road	Pavement	Broken Road
A	✓	✓				
B	✓	✓				✓
C	✓	✓		✓	✓	✓
D		✓	✓	✓		
E		✓	✓	✓		
F		✓	✓	✓		✓

Table 2: Detection Summary

Sequence	Resolution	Total Frames	True-Positive	False-Positive	False-Negative	Detection rate %
A	374*1238	301	284	0	17	94.3
B	375*1242	52	50	0	2	96.1
C	512*1392	281	274	2	7	97.5
D	240*320	591	583	9	8	98.6
E	240*320	957	935	12	22	97.7
F	376*1241	837	761	6	76	90.9

As per the performance metrics for evaluating object detection systems stated in [16], detection rate and computation time per frame are chosen, as they are widely employed by authors in several works. In Table 2, the results are provided in terms of true-positive, false-positive and false-negative detection. A frame is considered as true positive only if all the objects in the frame are detected, and in case of even a single unsuccessful object detection the entire frame is considered to be false-negative, thereby making the evaluation highly rigorous. It can be observed that the detection rate has reached a maximum of 98%. Comparing the achieved detection rate with that of other approaches [2-6] performed at different test conditions, we can conclude that the performance of the proposed algorithm is on par with them.

It can also be observed that the video sequences D and E, which have the least resolution input video frames, have the highest detection rates and also a higher number of false detections. This can be explained by the fact that low-resolution images have faint details of shadows and uniform background illumination. Due to lack of sharp variation between background and the object, there is a slight increase in the chance of false-positives. This shows that the algorithm is robust to various video qualities.

4.2 Computation Time Analysis

Commercial feasibility is one of the key goals in the development of this algorithm. Therefore, the computational load on the hardware is analyzed and the results are presented in Table. 3. The algorithm was developed in Python and was implemented using a Windows workstation powered by Intel i5 with 8GB RAM, 3.2 GHz clock frequency.

Table 3: Average processing time

Sequence	I(ms)	II(ms)	III(ms)	Cycle Time	Cycle Time(ms)
	Finding ROI	Detection	File I/O	I+II	I+II+III
A	15.2	8.3	12.0	23.5	35.5
B	17.0	9.0	12.8	26	38.8
C	35	31.0	11.7	43.7	55.4
D	6.1	2.2	8.0	8.3	16.3
E	6.2	2.2	8.1	8.4	16.5
F	15.9	7.6	12.4	23.5	35.9

The algorithm can be divided into two phases, finding the ROI and detecting the on-road objects. The average time for finding the ROI and detecting the on-road objects is recorded for the sequences A to F. It can be observed that the average time for finding the ROI is comparatively greater than the detection time. This is because, in order to compute the ROI, all the initial calculations have to be performed on the entire area of the input video sequence, whereas, once the ROI is found, the area on which the detection operation has to be performed is reduced. Thus, the detection phase though being complex has lesser execution time.

The cycle time (I + II) for low-resolution video sequences D and E is less than 10 ms, whereas for higher resolution video sequences it varies from 20 to 45 ms depending on the resolution. The time required to load the data from hard disk to the RAM (File I/O) is also mentioned in the Table 3. This operation would be performed only once in the beginning of execution of the sequence. However, this can be ignored when the algorithm is implemented in real time. Overall, the low cycle time (I+II) demonstrates the suitability of the proposed method for real-time applications.

The graphs presented in Fig. 10 depict the run-time analysis of the algorithm for the ROI extraction phase, object detection phase and combination of both. The run-time analysis of the algorithm (ROI extraction and object detection combined) deviates from linear behaviour for input image sizes higher than 450000 as seen in the Fig. 10 (c). This is due to the fundamental limitations set by the hardware on which the algorithm is tested. Overall the system time complexity is of $O(n)$, where n is the number of pixels in the image. Nevertheless, the algorithm has a processing rate greater than 38 frames per second for input resolutions less than [375*1242] without utilizing any GPU capabilities, which is highly recommended for real time ADAS.

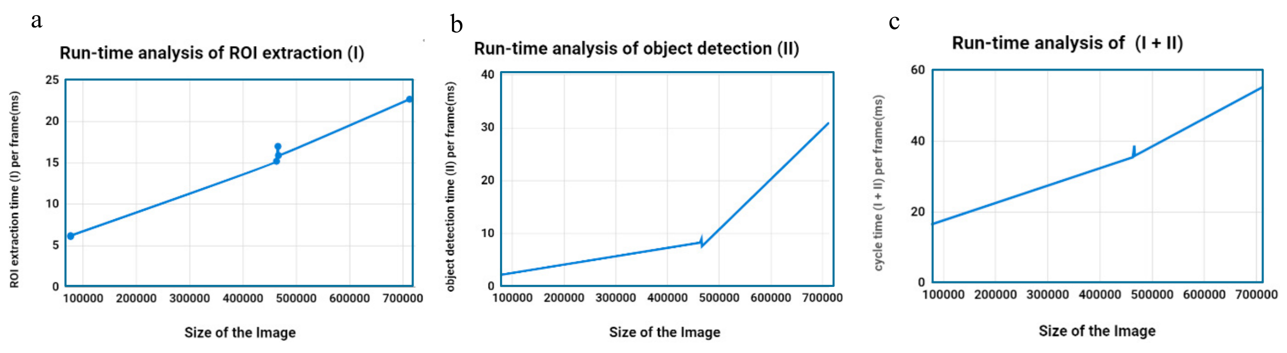

Fig. 10: (a) ROI extraction time (b) Object detection time (c) Total processing time

5. Conclusion

In this paper a new filter for vehicle detection is proposed. It has a better detection rate and reduced computational load on hardware in comparison to existing methods. The least recorded computational time is 8.3 ms for an input resolution of [240,320]. Since this method relies only on the color intensity difference between the vehicle and the surroundings, it can output the desired results for various weather conditions and screen resolution. This method is validated by testing on videos from KITTI dataset and also, on random videos captured during cloudy and rainy days with variable resolution.

Tracking of detected vehicles can be an appealing extension to this work, this method can further improve the detection performance. Further, this algorithm can be implemented using C++ and on NVIDIA GPU (used in real-time ADAS). It should be noted that, for the proper extraction of ROI there should be sufficient details in the video frame i.e. proper lane marking, pavements, color variation between road and surrounding areas.

6. References

[1] Zi Yang, Lilian S.C., Pun-Cheng (2018), “Vehicle detection in intelligent transportation systems and its applications under varying environments: A review”, *Image and Vision Computing* **69**: 143-154.

[2] C. Wang, Y. Fang, H. Zhao, C. Guo, S. Mita and H. Zha, (2016), "Probabilistic Inference for Occluded and Multiview On-road Vehicle Detection," *IEEE Transactions on Intelligent Transportation Systems* **17(1)**: 215-229.

[3] Yun Wei, Wing Tian, Jianhua Guo, Wei Huang, Jinde Cao (2018), “Multi-vehicle detection algorithm through combining Harr and HOG features”, *Mathematics and Computers in Simulation,* In press.

[4] J. Kim, J. Baek and E. Kim, Dec. (2015), "A Novel On-Road Vehicle Detection Method Using πHOG," *IEEE Transactions on Intelligent Transportation Systems* **16(6):** 3414-3429.

[5] Zhipeng Di and Dongzhi He, (2016), "Forward Collision Warning system based on vehicle detection and tracking," *International Conference on Optoelectronics and Image Processing (ICOIP)*, Warsaw, pp. 10

[6] Kai Kang, Hongsheng Li, (2016) , “T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos,” *arXiv:1604.02532v3 [cs.CV].*

[7] C. H. Lee, Y. C. Lim, S. Kwon, and J. H. Lee, (2011), “Stereo vision-based vehicle detection using a road feature and disparity histogram,” *Optical Engineering* **50(2)**.

[8] M. Bertozzi and A. Broggi, (1997), “Vision-based vehicle guidance,” *Computer* **30(7)**: 49–55.

[9] A. Giachetti, M. Campani, and V. Torre (1998), "The use of optical flow for road navigation," *IEEE Trans. on Robotics and Automation* **14:** 34-48.

[10] J. Klappstein, F. Stein, and U. Franke (2006), "Monocular Motion Detection Using Spatial Constraints in a Unified Manner," *IEEE Intelligent Vehicle Symposium*.

[11] Idan Nadav, Eyal Katz (2016), "Off-road path and obstacle detection using monocular camera", *IEEE International Conference on Science and Electrical Engineering (ICSEE)*.

[12] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun (2013), “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research (IJRR).*

[13] Lane Departure Warning System documentation, https://www.mathworks.com/help/vision/examples/lane-departure-warning-system-1.html, (accessed on January 2017).

[14] Gaussian Kernel documentation, http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.html, (accessed on January 2017).

[15] Branislav Kisačanin, S S Bhattacharyya, Sek Chai (2009), “Benchmarks of Low-Level Vision Algorithms,” in *Embedded Computer Vision,* Springer.

[16] Mark Nixon (2002), “Low-level Feature Extraction”, in *Feature Extraction and Image Processing*, Newnes.

[17] Afzal Godil (2014), “Performance Metrics for Evaluating Object and Human Detection and Tracking Systems”, *NISTIR 7972 technical report.*