International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 9, No.4 (July-2020)

http://dx.doi.org/10.12785/ijcds/090405

A Comprehensive Vehicle-Detection-and-Tracking Technique for Autonomous Driving

Wael Farag [1, 2]

[1]*College of Eng. & Tech., American University of the Middle East, Kuwait.*
[2]*Electrical Eng. Dept., Cairo University, Egypt.*

Received 20 Sep. 2019, Revised 11 Nov. 2019, Accepted 1 May 2020, Published 1 Jul. 2020

Abstract: In this paper, an advanced-and-reliable vehicle detection-and-tracking technique is proposed and implemented. The Real-Time Vehicle Detection-and-Tracking (*RT_VDT*) technique is well suited for Advanced Driving Assistance Systems (ADAS) applications or Self-Driving Cars (SDC). The *RT_VDT* is mainly a pipeline of reliable computer vision and machine learning algorithms that augment each other and take in raw RGB images to produce the required boundary boxes of the vehicles that appear in the front driving space of the car. The main contribution of this paper is the precise fusion of the employed algorithms where some of them work in parallel to strengthen each other in order to produce a precise and sophisticated real-time output. In addition, the *RT_VDT* provides fast enough computation to be embedded in CPUs that are currently employed by ADAS systems. Each used algorithm is described in detail, implemented, its performance is evaluated using actual road images, and videos captured by the front-mounted camera of the car. The evaluation of the *RT_VDT* shows that it reliably detects and tracks vehicle boundaries under various conditions.

Keywords: Computer vision, Self-Driving Car, Autonomous Driving, ADAS, Vehicle Detection, Vehicle Tracking.

1. 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, 2]. In the past couple of decades, major car manufacturers introduce many sophisticated ADAS functions [3, 4] like Electronic Stability Control (ESC), Anti-lock Brake System (ABS), Lane Departure Warning (LDW) [5], Lane Keep Assist (LKA) [6], etc. These functions represent steady incremental steps toward a hypothetical future of safe fully autonomous vehicles [7-11].

Most recent ADAS functions like Collision Avoidance, Automated Highway Driving (Autopilot), Automated Urban Driving, Automated Parking and Cooperative Maneuvering require more and more fast and reliable detection and tracking for on-road vehicles [12], which is among the most complex and challenging tasks. In order to successfully detect the other vehicles on the road, accurate localization of potential vehicles in camera images or LiDAR data is required, the relative position of these cars with respect to the road needs to be determined, and the vehicle's movement direction should be assessed and verified as well.

Computer vision techniques are considered the main tools that provide the capabilities of sensing the surrounding environment for the detection, identification, and tracking of moving vehicles. The detection of vehicles consists mainly of the finding of specific patterns/features or cues such as edges, gradients, colored segments, and color distributions in images. Such kind of specification streamlines or guides the process of vehicle detection. This paper presents an approach based on sophisticated computer vision algorithms working together to reach a real-time robust performance in detection and tracking of moving vehicles with substantial variations in shapes.

There are currently three main approaches to tackle the problem of vehicle detection and tracking, that have been proposed in the literature [13-34]. The first approach is based on the Camera as the only sensor [18, 19, 22, 23, 28, 29, and 30]. The second one is based on LiDARs or Laser Range Finders [14, 15, 17, 26, 31, and 34]. Additionally, the third approach is based on the fusion between Camera & LiDAR outputs [13].

E-mail: wael.farag@aum.edu.kw, wael.farag@cu.edu.eg

An early endeavor for the third approach has been carried out by *Premebida et al,* who proposed a vehicle detection system that combines both the information provided by both a LiDAR and a monocular Camera [13]. The system phases work in the laser space using a Gaussian Mixture Model classifier and in the vision space using the AdaBoost classifier. The results are combined using a Bayesian sum decision rule. The preliminary experimental results show the effectiveness of 84% hit rate.

Nevertheless, the pioneering work [14, 15] of *Anna Petrovskaya and Sebastian Thrun* in the Urban Grand Challenge [16] has to be highlighted, where they used laser range finders for reliable tracking of moving vehicles from a high-speed moving platform. The used approach models both dynamic and geometric properties of the tracked vehicles and estimates them using a single Bayes filter [17] per vehicle. Experimental results have shown the true positive vehicle detection rate was 97% compared to the theoretical maximum of 98%.

Moreover, *Jazayeri et al* [18, 19] modeled the motion behavior of the vehicles and the background, captured by the front Camera, probabilistically. The targets got identified using Hidden Markov Models (HMM) [20, 21]. The results showed that the identification and tracking are robust to various illumination and environments and the processing was performed in real-time. However, the identification was only based on motion only, therefore, the results of the proposed method should be fused with the results of shape analysis methods.

In [22], *Romera et al* proposed a lightweight technique for vehicle detection and tracking that is implemented on a smartphone. The technique detects lanes first and determines the vanishing points based on previous work [23]. The main pipeline has two stages, the first is the detection stage utilizing the AdaBoost classifier, and the second is the tracking stage based on Extended Kalman Filter implementation. The technique is tested on iPhone 5 and iPhone 6 producing and execution time of 132ms and 76ms respectively.

An interesting study of using deep learning with LiDAR data is carried out by *Ivan del Pino et al* [24], who used a low resolution 3D laser sensor (Velodyne VLP-16 (PUCK) [25]) to detect and track vehicles on the road, incorporating a Convolutional Neural Network (CNN) that was constructed for this purpose and applied to the point cloud data of the PUCK. Moreover, a Multi-Hypothesis Extended Kalman Filters (MH-EKF) is utilized as well, to estimate the actual position and velocities of the detected vehicles. Comparative studies between the proposed lower resolution (VLP-16) tracking system and a high-end system, using Velodyne HDL-64 [26], showing that the proposed low-resolution VLP-16 Deep Learning architecture is able to close matching the performance of the high-end HDL-64 one in close ranges up to half the distance of the high-end sensor.

An endeavor to implement a real-time car detection and tracking algorithm on very inexpensive hardware (Raspberry Pi v3 [27]) is carried out by *M. Anandhalli et al* [28]. The proposed algorithm converts the RGB video frame to the HSV one, and filtering and noise removal, the detection is mainly based on the color features, and the tracking by using a Kalman filter with the data association. The results are then compared with that of rear-view vehicle detection and tracking method [29] and morphological operation method [30] with a higher performance of 6-8%.

To detect several objects on the road not only vehicles, *Abdul Rachman* presented an integrated framework of multi-target object detection and tracking using a 3D LiDAR geared towards the urban environment [31]. The framework combines occlusion-aware detection methods, probabilistic adaptive filtering, and computationally efficient heuristic logic-based filtering to handle uncertainties. The framework is tested using real-world pre-recorded 3D LiDAR data and shows that the proposed framework is achieving promising real-time tracking performance (accuracy of 94% and a precision of 92%) in varying urban driving scenarios.

It is clear from the previous literature that LiDARs have a major role and potential in accurate vehicle detection and tracking, however, there are several drawbacks of using LiDARs in the commercial rollout. The first one is the cost, as an example, the lower end Velodyne VLP-16 (PUCK) price is 8000 USD, while the high-end Velodyne HDL-64E is 100,000 USD. The second drawback is the lack of reliability in the installation LiDARs in vehicles for commercial use [32]. The third one is the huge amount of data resulting from LiDARs that need to be processed to execute the detection algorithms which requires powerful dedicated hardware.

Approaches based on neural networks [33] and deep learning [34, 35], and specifically Convolutional Neural Networks (CNN) stimulate a promising research direction despite its overwhelming computational overhead. However, considering that the vehicle detection runs on vehicle-based systems, where computation resources are severely limited, the computational cost of vehicle detection and tracking method should also be considered as a key indicator of the overall performance.

Therefore, in this paper, a comprehensive, streamlined, vehicle detection-and-tracking algorithm is proposed and implemented. This algorithm is given the name "Real-Time Vehicle Detection and Tracking" (*RT_VDT*). *RT_VDT* is differentiated from the previously surveyed algorithms in that it streamlines a pipeline of computer vision and machine learning algorithms beginning with a camera calibration algorithm until boxing the identified vehicle. In between, several edge detection and color identification techniques are used employing multiple color spaces. The *RT_VDT* focuses on both robustness and speed with a delicate balance. The robustness is achieved by removing distortion from images and fusing multiple methods to extract the vehicle features, working in parallel to strength each other, and the speed comes from using effective methods that do not depend on iterative searches but rather a single scan per camera frame, as well as concentrates the computation in the image sectors of higher interest.

The next sections will describe the used algorithms in more detail. *RT_VDT* represents a further step towards the prospects of autonomous driving.

2. Overview of the *RT_VDT* Algorithm

The *RT_VDT* algorithm is designed to utilize a single Charge-Coupled Device (CCD) camera. This camera should be mounted on the front-windshield mirror of the car to capture the road front view. However, stereo cameras can also be employed, but for the matter of convenience, in this paper, a single front camera is only considered. In order to simplify the detection problem, it can be assumed that the setup makes the baseline horizontal, which assures "the horizon" is in the image and it is parallel to the X-axis (i.e. the projected intersection of left and right lines of the driving lane, after finding them using one of the techniques developed in [5], is referred to as "the horizon"). Nevertheless, for the matter of precision, in the *RT_VDT*, the image orientation will be adjusted using the calibration data of the front camera in conjunction with removing the visual distortions.

In this work, it is assumed that the input to the *RT_VDT* algorithm is a 1200x720 RGB color image. Therefore, the first thing the algorithm does is to remove the distortion and adjust the orientation using a camera calibration routine and chessboard images. This camera calibration routine is only executed once at the initialization of the *RT_VDT* algorithmnot with every iteration/frame, hence, not affecting the real-time performance. Then, the image will be converted to grayscale as well as several color spaces [36] (e.g. HSL, HSV, LAB, LUV, YUV, YCrCb, etc. [37]).

After the grayscale and color space conversion, several features will be extracted from the images such as the Histogram of Oriented Gradients (HOG) [38], color spatial features [39] and color histogram features [40]. These features are combined together to produce what is called "feature vectors". These feature vectors are used by a vehicle/non-vehicle classifier built by the Support Vector Machine (SVM) algorithm [41] to detect vehicles in camera images.

After the vehicle/non-vehicle classification, the vehicles are then detected using the sliding windows technique, which uses the results produced by the SVM classifier and scans each image to detect and localize the vehicle objects. The scan is not implemented of the full image, however, a Region of Interest (ROI) is defined and then extracted from each image to perform the exhaustive search. Therefore, the undesired image details are masked to improve the focus and accuracy of detecting the vehicle boundaries. The results of this scanning process are used to build active heat-maps that produce potential car boxes. The overlapped detected true-positive car boxes are then grouped in bigger boxes and labeled accordingly. As a final step, the labeled boxes are drawn on the original test image or video frame. For the matter of illustration, working examples of the resultant road boundary are displayed on the original color image as shown in Figure 1 and Figure 2.

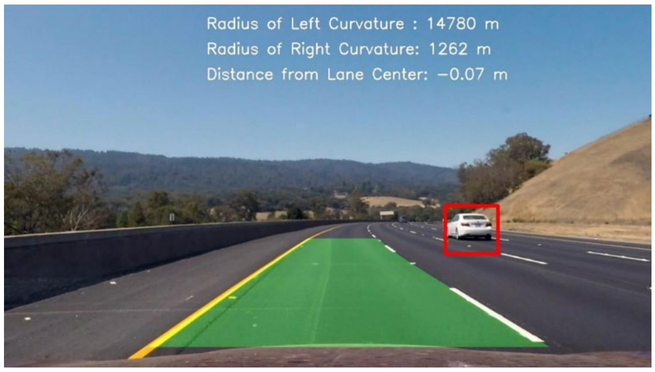
Radius of Left Curvature : 14780 m
Radius of Right Curvature: 1262 m
Distance from Lane Center: −0.07 m

Figure 1. Detected Vehicle boundaries by the RT_VDT algorithm.

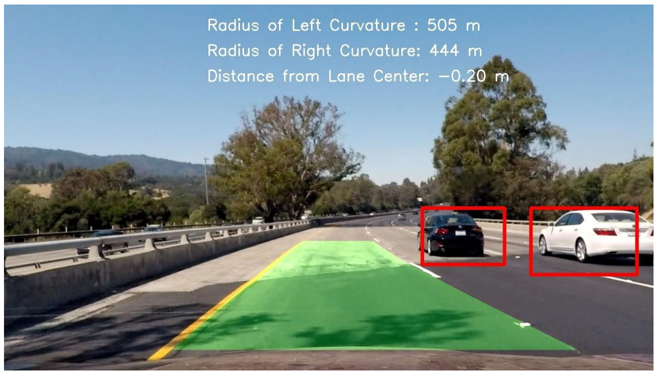
Radius of Left Curvature : 505 m
Radius of Right Curvature: 444 m
Distance from Lane Center: −0.20 m

Figure 2. Detected Vehicles' boundaries by the RT_VDT algorithm.

3. Histogram of Oriented Gradients

The motivation behind the development of the HOG algorithm is best described by the authors [38] as:

"Local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions. In practice, this is implemented by dividing the image window into small spatial regions ("cells"), for each cell accumulating a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell. The combined histogram entries form the representation. For better invariance to illumination, shadowing, etc., it is also useful to contrast-normalize the local responses before using them. This can be done by accumulating a measure of local histogram "energy" over somewhat larger spatial regions ("blocks") and using the results to normalize all of the cells in the block. We will refer to the normalized descriptor blocks as Histogram of Oriented Gradient (HOG) descriptors"

For instance, to detect a specific object 'O_{bj}' in a camera image the following steps can be followed:

1) The camera image is converted to gray.
2) Start by constructing a rectangle (or square) window that is 64 pixels tall by 64 pixels wide (the dimensions of the window are arbitrary depending on the designer choice).
3) Use it to scan the grey camera image searching for O_{bj}. The search is done by sliding the window both horizontally and vertically with a *stride* of 8 bits (as an example).
4) The object O_{bj} may have of course different sizes and occupy a bigger or small part of the image. Therefore, the analysis should be done not only on the original starting window (*64×64*) but also on a series (pyramid) of windows with an increment of 16 bits (as an example), like *80×80*, *96×96*, *112×112*, etc. This pyramid of windows corresponds to larger portions of the original camera image where O_{bj} or part of it could be inside one of them.
5) In each step of the windows slide, the HOG features are computed and get associated with the center position of the corresponding window as a matter of "feature localization".

To compute the HOG features, the input to the algorithm is expected to be a certain window 'W_I' from a gray-level image, possibly from a pyramid, and the workflow continues as follows and shown in Figure 3. :

1) Calculate the two gradient components G_x and G_y of the gradient of W_I by central differences:

$$G_x(r,c) = W_I(r,c+1) - W_I(r,c-1) \quad (1)$$

$$G_y(r,c) = W_I(r-1,c) - W_I(r+1,c) \quad (2)$$

where r and c are the corresponding row and column numbers of the pixels in window W_I.

2) The calculated gradient is then converted to polar coordinates as below, with the angle constrained to be between 0º and 180º. As a result, gradients that point in opposite directions are computed as:

$$G = \sqrt{G_x + G_y} \quad (3)$$

$$\theta = \frac{180}{\pi}\left(tan_2^{-1}\left(\frac{G_y}{G_x}\right) mod\ \pi\right) \quad (4)$$

where tan_2^{-1} is the four-quadrant inverse tangent, which yields values between -π and π.

3) Construct the cell orientation histograms by dividing the window W_I into adjacent, non-overlapping cells of size $C\times C$ pixels (could be $C = 8$). In each cell, calculate the histogram of gradient orientations that are enclosed (binned) into B bins (could be $B = 9$). If the bins are numbered *0* through *B-1* and have width $w = \frac{180}{B}$, then bin i has boundaries [wi, $w(i + 1)$] and center $c_i = w(i + \frac{1}{2})$. A pixel with magnitude G and orientation θ contributes a vote of:

$$v_j = G\frac{c_{j+1}-\theta}{w}\ to\ bin\ number\ j = \left\lfloor\frac{\theta}{w} - \frac{1}{2}\right\rfloor mod\ B \quad (5)$$

and a vote of:

$$v_{j+1} = G\frac{\theta - c_j}{w}\ to\ bin\ number\ (j+1)\ mod\ B \quad (6)$$

This scheme is called voting by bilinear interpolation and the resulting cell histogram is a vector with B positive entries.

4) The block normalization step is then carried out by grouping the cells together into overlapping blocks of *2×2* cells each. Therefore, each block has a size of *2C×2C* pixels. Accordingly, each two horizontally or vertically consecutive blocks overlap by two cells, that is, the block stride is C pixels. Consequently, each internal cell is covered by four blocks. The four-cell histograms in each block are concentred into a single block feature b and then the *block feature* 'b' get normalized by its Euclidean norm as:

$$b \leftarrow \frac{b}{\sqrt{\|b\|^2+\epsilon}} \quad (7)$$

Where ϵ is a small positive constant that prevents division by zero in gradient-less blocks.

5) The normalized block features are then concatenated into a single HOG feature vector h, which is normalized as follows:

$$h \leftarrow \frac{h}{\sqrt{\|h\|^2+\epsilon}} \quad (8)$$

$$h_n \leftarrow \min(h_n, \tau) \quad (9)$$

Here, h_n is the nth entry of h and τ is a positive threshold (τ = 0.2). Clipping the entries of h to be no greater than τ (after the first normalization) ensures that very large gradients do not have too much influence—they would end up washing out all other image detail. The final normalization makes the HOG feature independent of overall image contrast. An example of the output of the algorithm is shown in Figure 4. .

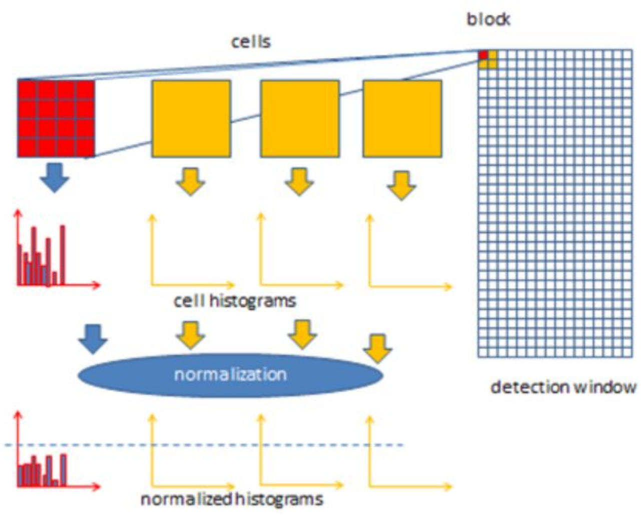

Figure 3. The Histogram of Oriented Gradiens Workflow.

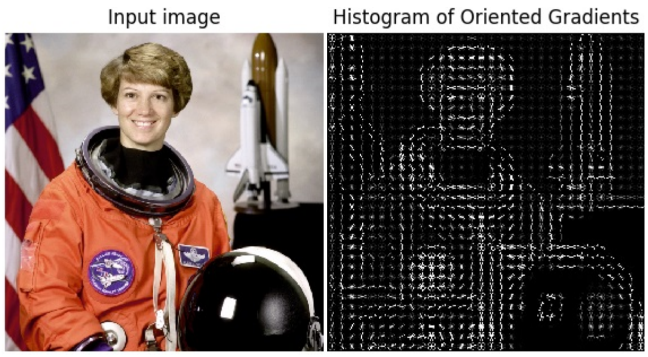

Figure 4. Results of applying HOG.

4. Support Vector Machine Classifier

Support Vector Machine (SVM) [42] is a supervised learning model with an associated learning algorithm that analyzes data used for classification and regression analysis [43, 44]. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

Given a training dataset of *n* points of the form

$$(\vec{x}_1, y_2), \dots, (\vec{x}_i, y_i), \dots, (\vec{x}_n, y_n) \tag{10}$$

where y_i are either 1 or -1, each indicating the class to which the point $\vec{x}_i$ belongs. Each is a *p*-dimensional real vector. It is required to find the "maximum-margin hyperplane" that divides the group of points $\vec{x}_i$ for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point $\vec{x}_i$ from either group is maximized.

Any hyperplane can be written as the set of points $\vec{x}$ satisfying

$$\vec{w}.\vec{x} - b = 0 \tag{11}$$

where $\vec{w}$ is the normal vector to the hyperplane. The parameter $\frac{b}{\|\vec{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\vec{w}$.

If the training data is linearly separable, the optimization problem can be written as follows:

$$\text{"}Minimize\ \|\vec{w}\| subject\ to\ y_i(\vec{w}.\vec{x}_i - b) \geq 1,$$
$$for\ i = 1,2,\dots,n\text{"} \tag{12}$$

The $\vec{w}$ and b that solve this problem determine our classifier, $\vec{x} \mapsto sgn(\vec{w}.\vec{x} - b)$.

If the training data is not linearly separable, the hinge loss function is introduced as

$$\max\big(0, 1 - y_i(\vec{w}.\vec{x}_i - b)\big) \tag{13}$$

This function is zero if the constraint $y_i(\vec{w}.\vec{x}_i - b) \geq 1$ is satisfied, in other words, if $\vec{x}_i$ lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin. Then the optimization function will be solved:

$$minimize\ \left\{\left[\frac{1}{n}\sum_{i=1}^{n} \max\big(0, 1 - y_i(\vec{w}.\vec{x}_i - b)\big)\right] + \lambda\|\vec{w}\|^2\right\} \tag{14}$$

where the parameter λ plays a role of determining the tradeoff between two opposing requirements: one is increasing the margin-size and the other is ensuring that the $\vec{x}_i$ lie on the correct side of the margin. Accordingly, for sufficiently small values of λ, the second term in the loss function will become negligible; consequently, it will perform similar to the hard-margin SVM, if the input data are linearly classifiable. However, it will still learn if a classification rule is viable or not.

If a nonlinear classification rule need to be learned, and which this non-linear rule corresponds to a linear classification rule for the transformed data points $\varphi(\vec{x}_i)$. Additionally, a kernel function k is given which satisfies $k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i).\varphi(\vec{x}_j)$. Accordingly, the classification vector $\vec{w}$ in the transformed spaces satisfies

$$\vec{w} = \sum_{i=0}^{n} c_i y_i\ \varphi(\vec{x}_i) \tag{15}$$

where the c_i's are obtained by solving the optimization problem

$$maximize\, f(c_i \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i\, k(\vec{x}_i, \vec{x}_j) y_j c_j$$

$$subject\ to \sum_{i=1}^{n} c_i y_i = 0\,, and\ 0 \le c_i \le \frac{1}{2n\lambda} for\ all\ i. \quad (16)$$

The coefficients $c_i's$ can be solved using quadratic programming [45], and then solve

$$b = \vec{\mathrm{w}}.\varphi(\vec{x}_i) - y_i = \left[\sum_{k=1}^{n} c_k y_k\, k(\vec{x}_k, \vec{x}_i)\right] - y_i \quad (17)$$

Finally, new points ($\vec{z}$) can be classified by computing

$$\vec{z} \mapsto sgn(\vec{w}.\varphi(\vec{x}_i) - b) = sgn\left(\left[\sum_{k=1}^{n} c_k y_k\, k(\vec{x}_k, \vec{x}_i)\right] - b\right) \quad (18)$$

5. Color Spaces

A color space (model) is a specific organization of colors that provides a way to categorize colors and represent them in digital images [36]. It is an abstract mathematical model describing the colors as tuples of numbers (e.g. triples in RGB or quadruples in CMYK). This representation is useful in understanding the color capabilities of a particular digital device or file (camera images). There are a variety of color spaces, such as RGB, LUV, YUV, HSV, HLS, CMY, LAB, etc. The following as some highlights on the most emphasized color spaces that have been experienced throughout this work:

1) **RGB**: is a kind of color space that uses (R=Red, G=Green, and B=Blue) to elaborate the color model [30]. Simply, it contains all possible colors, by combining the three colors with different levels. Each pixel of an image has three components R, G and B. Each component is assigned a range of 0→255 of intensity values. Obviously, it can be said, using only these three color components, there can be 16,777,216 distinct colors on the screen by different mixing ratios.

2) **HSL** and **HSV**: (Hue, Saturation, Lightness) and (Hue, Saturation, Value) color spaces are both constructed geometrically from cylindrical structures as shown in Figure 5. . They are sometimes used to define gradients for data visualization as a compromise between effectiveness for segmentation and computational complexity. Separating hue, lightness, and chroma or saturation is proven effective in some object detection applications.

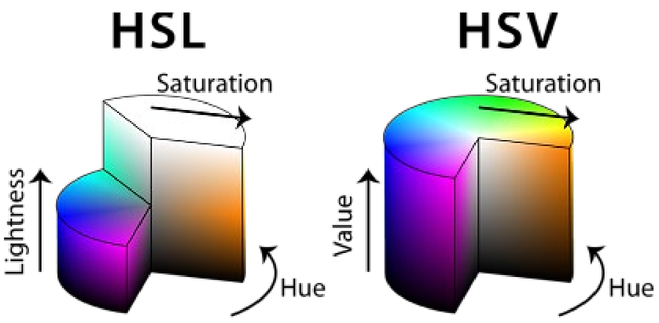

Figure 5. HSL and HSV cylindrical color spaces.

3) **LAB**: (*L*umination, '*a*' and '*b*' color channels) as shown in Figure 6. , is a color model (space) that covers the whole light spectrum, including as well spectrum outside of human vision. LAB is very powerful in identifying a *spot color*, possibly a focal "brand name" or "logo" color such as "Pepsi Blue" or "McDonald's Yellow". This specific color definition can be used to specify many items such as vehicles, traffic signs, trees, buildings, lane markings, etc.

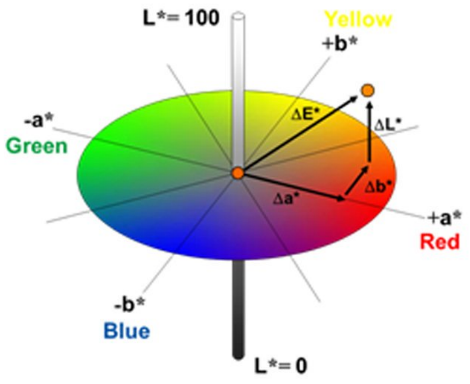

Figure 6. LAB color space structure, channels (L, 'a' and 'b').

4) **LUV**: (Lumination, 'U' and 'V' color channels) is a color model that uses U and V channels to represent the chromaticity or color values, which are completely independent of the L channel. This makes LUV color space much better suited for image difference comparisons.

5) **YUV**: (lumination 'Y', 'U' and 'V' color differences) is the principal color model used in analog color TV broadcasting. The luminance channel 'Y' can be calculated as a weighted sum of red, green and blue color components. Furthermore, the color difference, or *chrominance*, components 'U' and 'V' are formed by subtracting the channel 'Y' (luminance) from blue and from red components respectively. The principal advantage of the YUV color space in image processing is the decoupling of luminance and color information. The main advantage of this complete separation is that the luminance component of a certain image can be

dealt with without affecting these images' color components [46].

6. Camera Calibration

The conversion from three dimensional (3D) real-world scene to a two dimensional (2D) one, exhibits by a camera, results in image distortion, as the transformation from 3D→2D is not perfect. Actually, the shape and size of objects get distorted (changed) in the resulting 2D image from the original 3D appearance. Therefore, before using the resulting 2D camera images, this distortion needs to be undone so that the correct and useful information can be extracted and analyzed.

The construction of real cameras includes using a curved lens to form an image. The light rays usually bend around the edges of these lenses with low or high degrees depends on the focus and position of objects. Therefore, distortion at the images' edges happens, in a way that lines or objects appear to be more or less curved than their actual reality. This effect is called the "radial distortion", and represents the principal source of distortion.

Moreover, there is another main source of distortion that is the "tangential distortion". This distortion happens when the camera's lens is not perfectly aligned parallel to the image plane that is associated with the camera sensor. This produces a tilt effect to the image, which shows objects nearer or farther away than they actually are.

There are three needed coefficients to correct for radial distortion: k_1, k_2, and k_3. To correct the appearance of radially distorted points in an image, one can use a correction formula.

In the following equations Eq. (19), and Eq. (20), (x, y) is a point in a distorted image. To undistort these points, the first step is to use OpenCV [47] to calculate r, which is the known distance between a point in an undistorted (corrected) image ($x_{corrected}$, $y_{corrected}$) and the center of the image distortion, which is often the center of that image (x_c , y_c). This center point (x_c, y_c) is sometimes referred to as the distortion center. These points are illustrated below in Figure 7. .

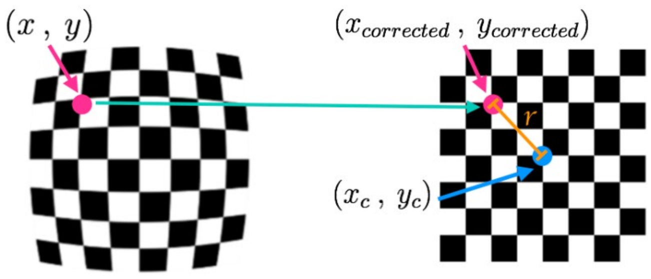

Figure 7. Points in a distorted and undistorted (corrected) images.

$$x_{distorted} = x_{ideal} + (1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (19)$$

$$y_{distorted} = y_{ideal} + (1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (20)$$

There are two more coefficients that account for tangential distortion: p_1 and p_2, and this distortion can be corrected using a different correction formula as given by Eq. (21) and (22).

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)] \quad (21)$$

$$y_{corrected} = y + [2p_1(r^2 + 2y^2) + 2p_2xy] \quad (22)$$

To correct for the mentioned distortions, images of known shapes (chessboard images) are used. Selected points in the distorted plans are then mapped to undistorted plans as shown in Figure 8. . Accordingly, the camera images will be calibrated. The following procedure is implemented to undistort the captured camera images and improve the image quality:

1) Step 1 – finding the chessboard corners: Using 20 chessboard images that have different sizes and orientations as depicted in Figure 9. , the "*cv2.findChessboardCorners()*" function from the OpenCv3 library [47] is used to locate the chessboard corners. The detected number of corners is 9x6 as shown in the 17 out of the 20 images that are depicted in Figure 9. . In the other 3 images, only 9x5 corners have been detected. The corners are drawn using the "*cv2.drawChessboardCorners()*" function of openCv3.

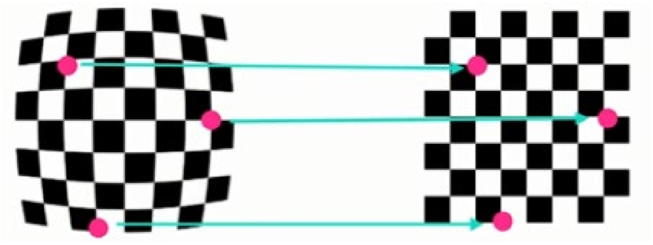

Figure 8. Mapping from a distorted chessboard image to an undistorted one.

2) Step 2 – get camera matrices: A test chessboard image that has not been used before in finding the corners; is used; after being converted to a greyscale; along with the found corners in step one; to find the camera matrices. "*cv2.CalibrateCamera()*" function is used to perform this step. To check the quality of the calibration, the gray test image together with the camera matrices to remove the distortion of this image as shown in Figure 10. .

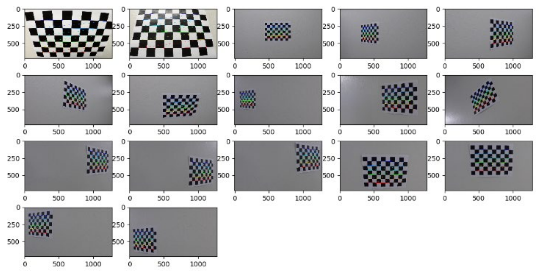

Figure 9. Chessboard images used for calibration with corners drawn.

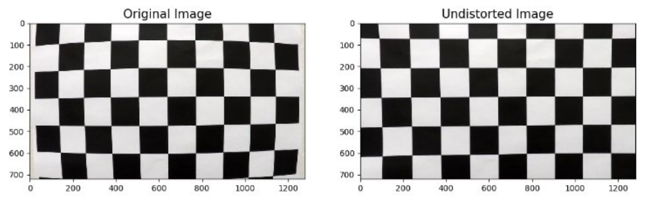

Figure 10. A test chessboard image with distortion removal.

3) Step 2 – saving camera matrices: using Pickle library [48], the camera data (the camera matrix as well as the distortion coefficients) are saved in the pickle file "*camera_calibration.p*" for easy retrieval later.

Figure 11. provides an example of applying the camera calibration procedure on one of the test images.

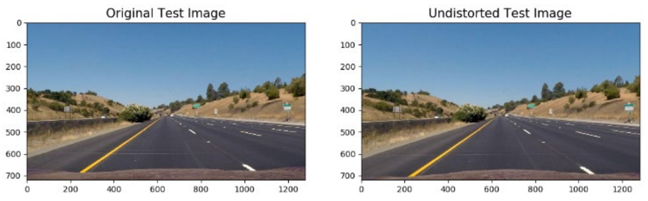

Figure 11. Camera calibration effect (undistortion of images).

7. Implementation of the Support Vector Machines Classifier

In this section, the steps to build a classifier based on the SVM algorithm described in Section 4 will be explained in detail, and it is given the abbreviation "*SVMC"*.

A. Training Data Preparation

The data preparation steps to train the *SVMC* is summarized as follows:

1) The data supplied by Udacity [49, 50]: the Udacity supplied data have been used throughout this work. The data consists of almost balanced "non-vehicles" and "vehicles" images:
   a) The "non-vehicles" collections consist of the "GTI" collection [51] and the "Extras". Both contain 8968 RGB images of size (64, 64, 3) pixels.
   b) The "vehicles" collections consist of the "GTI" collection and the "KITTI" [52]. Both contain 8792 RGB images of size (64, 64, 3) pixels.
2) These collections with an unzipped size of 149MB.
3) Data Augmentation: The data is augmented by flipping all the images around the "Y" axis. As a result, the training data become a total of 35,520 images.

B. Training Data Visualization

The following steps describe the implemented data visualization steps in order of execution:

1) Display of Vehicles Data: 50 randomly selected images of the vehicle data have been displayed as shown in Figure 12. . Each image has its order in the training data as a title.
2) Display of Non-Vehicles Data: 50 randomly selected images of the non-vehicle data have been displayed as shown in Figure 13. . Each image has its order in the training data as a title.
3) Display of HOG features of Vehicles Data: A selected image of the vehicle data has been used to extract its hog features after converting it to grayscale. Moreover, the hog features of non-vehicle examples are also extracted, and the result of both is shown in Figure 14. .

C. Training Data Visualization

The following steps describe the implemented images feature extraction functions in order of execution:

1) Color Spatial Features: a function is implemented to extract the contribution of different color channels in each image. Or in other words, to compute the binned color features. The channel of each image is resized to (32, 32) and then raveled.
2) Color Histogram Features: a function is implemented to compute the histogram of each color channel in each image with a designated number of pins, and then concatenate them.
3) HOG Features: a function is implemented to compute the histogram oriented gradients of each image channel separately and then can use them separately or append them together if this option is selected. The SciKit-Image function "hog" [53] is used in the implementation of this function.

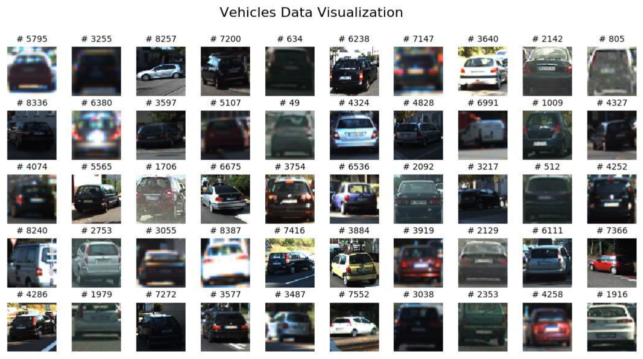

Figure 12. Visualization of 50 randomly selected vehicle images.

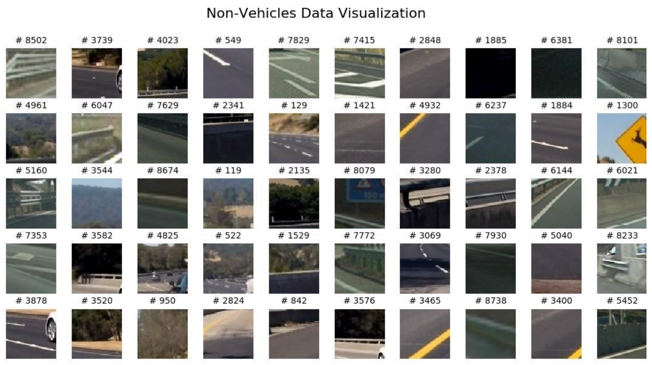

Figure 13. Visualization of 50 randomly selected non-vehicle images.

4) Combining All: The above feature extraction functions produce the following feature vectors:

   a) Using the color spatial features and 'spatial size = (32, 32)' results in a feature vector of 32×32×3 = 3072 elements.
   b) Using the color histogram features and 'histogram bins = 32' results in a feature vector of 32×3 = 96 elements.
   c) Using the HOG features and 'gradient orientations cells = 9', 'pixels per cell = 8×8', 'cells per block = 2×2', and using all hog channels results in a feature vector of 7×7×2×2×9 = 1764×3 = 5292 elements.
   d) If all the above functions are used the resulting feature vector will be of the following length: 3072+96+5292 = 8460 elements.

D. Training The Classifier

The following steps are used to build up and train the vehicle/non-vehicle *SVMC* classifier:

1) Compiling a training data set "X" of 35,520×8,460 size which includes 35,520 vehicle/nonvehicle feature vectors of length 8,460 each. This training set represents the input to the classifier.

2) The feature sets must be scaled; before combining them together; using the SciKit-Learn "StandardScaler().fit()" function [46]. Figure 15. shows the visualization of raw and normalized feature vectors for two-vehicle images.

3) Compiling an output training set "Y" of a 35,520×1 size in which each element is of a Boolean value of 1=>vehicle or 0=>non-vehicle.

4) Shuffle the training sets randomly and split them to 80% for training and 20% for testing using the SciKit-Learn "train_test_split()" function.

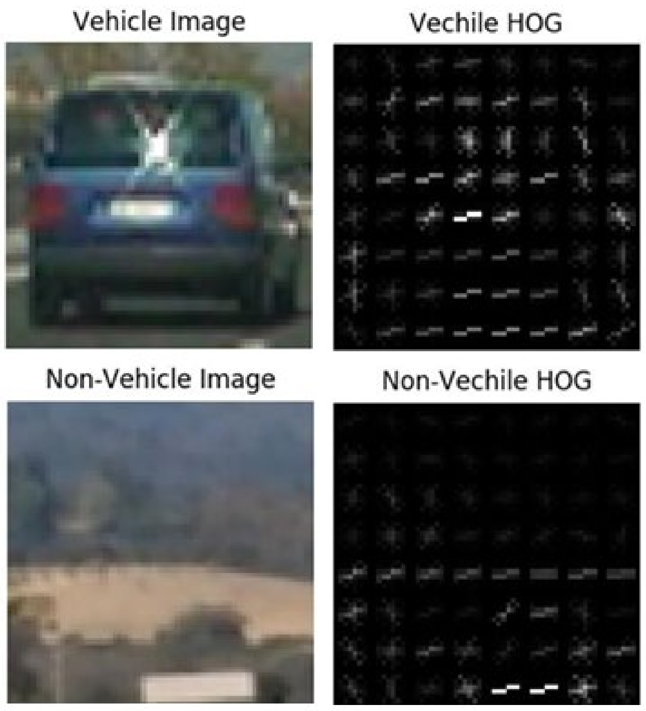

Figure 14. Visualization of HOG features for vehicles and non-vehicle images.

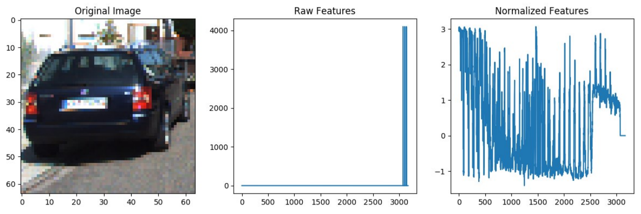

Figure 15. Visualization of feature vectors for vehicles' images.

5) Using a Linear Space Vector Machine Classifier function "LinearSVC()" of the Sci-Kit Learn library [55], the model got trained with high accuracy (above 97.7%) in almost all the selected parameters combinations. Then the trained model is tested on the prepared test images. The results were not good in several cases. Extreme experimentations have been done with many parameter combinations, however, the results still were not acceptable.

6) After several trials and errors, it is found that the color spatial features are taking a significant portion of the feature vector length (>36%) without adding a real value (sometimes even represents a confusing element) to the distinction between the vehicles / non-vehicles. Moreover, the color histogram features are of a very insignificant contribution (~ 1.1%) of the feature vector as well as to the distinction between vehicles / non-vehicles.

7) Therefore, both the color special and histogram features have been removed from the feature vector

and keeping only the HOG features. By doing that, this results in a reduction in the length of the feature vector from 8,460 to 5,292 features only. This is off course simplifies the training and the real-time application of the algorithm, and results in a huge reduction of processing and training time.

8) The new Linear SVC classifier with a training data set of size = 35,520×5,292 is constructed using several color spaces with the training results shown in Table 1.

9) Almost all the color spaces produced comparable results except the "RGB". The "LAB" color space produces the fastest performance in both training and prediction with second to highest accuracy behind the "YUV". However, while testing on test-images "YUV" produced false positives more than "LAB". Therefore, "LAB" color space is selected for the next steps.

TABLE I. LINEARSVC TRAINING RESULTS.

Colour Space	Training Time (Sec)	Prediction Time for 10 Labels (Sec)	Test Accuracy
RGB	19.5	0.01563	0.9716
HSV	8.94	0.001	0.9865
HLS	8.83	0.0015	0.9831
LUV	8.79	0.002	0.9876
YCrCb	7.76	0.002	0.9899
YUV	8.34	0.003	0.9918
LAB	5.7	0.001	0.9916

8. VEHICLE DETECTION AND TRACKING PIPELINE

The following steps constitute the pipeline used in the detection and tracking of other vehicles on the road (*RT_VDT*). These steps are presented in order of execution:

1) Finding lane lines: this function is mainly to detect the road boundaries (in other words, the lane lines in front of the car) which represent the driving space (shown in green in Figure 1.). This function is fully implemented in [2] and used here for convenience.

2) Detecting vehicles by sliding windows technique: a dedicated function is implemented and called for each camera frame and used the following parameters:

a) "orient = 9" defining the number of histogram bins per cell and it is used for the HOG feature extraction for images or video frames.

b) "pix_per_cell = 8" defining the number of HOG pixels/cell. In this case, the cell will be 8×8 pixels.

c) "cell_per_block = 2" defining the number of HOG cells/block. In this case, the cell will be 2×2 cells.

d) x_{start}, x_{stop}, y_{start}, y_{stop}: these 4 parameters define a rectangular area on the image or frame that represents the region of interest (ROI) in which the function searches for a vehicle by the sliding windows technique.

e) "step_size = 2" defining how many cells to step (or to slide) to construct a new search window that will overlap with the previous search window.

f) "Scale_Step = 0.25" defining the step at which the search window sizes increments from one search scan to the next.

g) Scale_Multiplier_Start, Scale_Multiplier_End: two parameters defining the starting and stopping of the windows sizes increment while scanning the ROI area.

The function uses the trained SVMC classifier model and applies it to each constructed search window. Sliding windows with different sizes are being constructed to cover the defined ROI as shown in Figure 16. . This function as well may be applied several times with a different set of "a→g" parameters based on if it found necessary.

3) Building active heat-maps: The goal is to construct a heat-map for each found car box during the search of a sliding-windows scan. This heat-maps is used to filter out (try to minimize) the false-positive boxes. A dedicated parameter "HEAT_THRESHOLD" is used to only pass (based on its value) the car boxes with multiple hits (true-positive boxes) as shown in Figure 17.

4) Labeling car boxes**:** the overlapped true-positive vehicle boxes are then grouped in bigger boxes and labeled using the "label()" function from the Sci-Kit Learn library.

5) Drawing the labeled car boxes: as a final step, the labeled boxes are drawn on the original test image or video frame as shown by the red boxes in Figure 1. and Figure 2. .

Figure 18. & Figure 19. show examples of the results after the execution of the above pipeline on the test images that include shadow patterns that usually confuse vision-based algorithms.

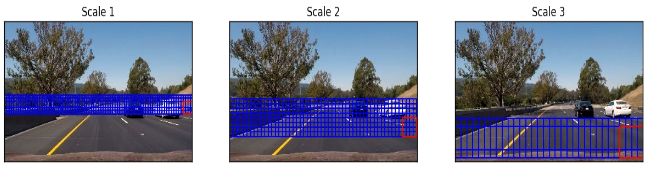

Figure 16. Sliding windows with different sizes scanning the ROI.

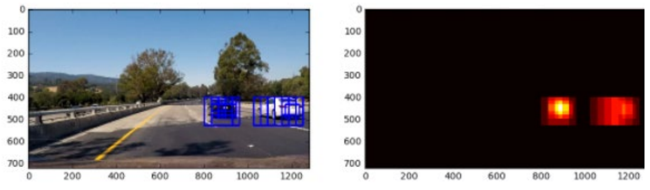

Figure 17. Detected vehicle boxes and the resulted heat-maps.

9. Testing and Validation

The developed *RT_VDT* algorithm is further tested on various images representing different scenarios. The results show that the algorithm performs very well under different conditions (at full sunrise, at sunset, with shadows, without shadows, with cars on the other lanes and without). 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 *RT_VDT* proved to be very robust in all the pre-mentioned conditions as shown in Figure 1. and Figure 2. However, the scattered areas of shadows have an effect on the precision of producing the vehicles' boundary boxes as shown in Figure 18. and Figure 19. . However, the results are still acceptable and produce functional results.

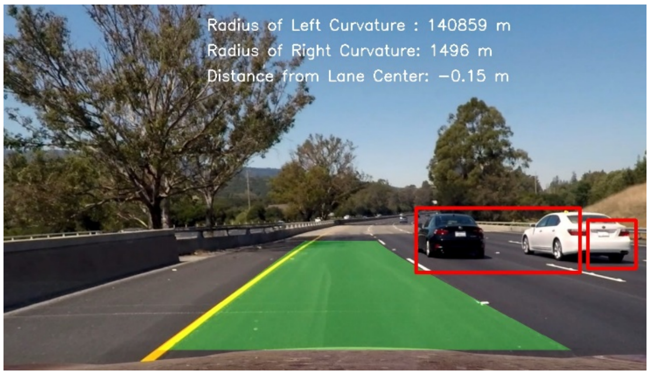
Radius of Left Curvature : 140859 m
Radius of Right Curvature: 1496 m
Distance from Lane Center: −0.15 m

Figure 18. The execution of vehicle detection and tracking pipeline.

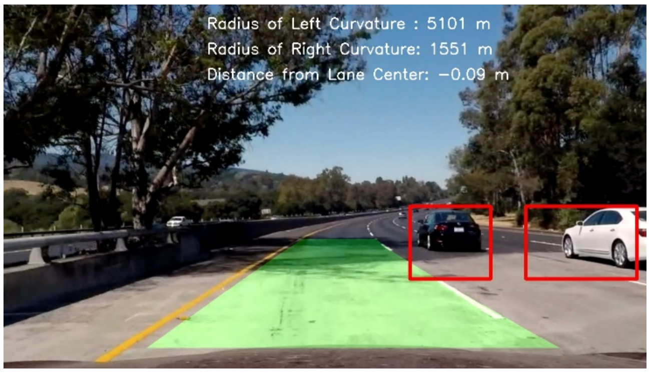
Radius of Left Curvature : 5101 m
Radius of Right Curvature: 1551 m
Distance from Lane Center: −0.09 m

Figure 19. The execution of vehicle detection and tracking pipeline.

As shown in Figure 1 , Figure 2 , Figure 18 and Figure 19 the images include as well lane detection results from the work in [6].

The pipeline proved to be acceptably fast in execution 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 two testing video streams:

TABLE II. Computation Speed for the *RT_VDT* Algorithm.

Sample Name	No. of Frames	Total Time Min:Sec	Frame per Sec
Challenge Video	485	00:39	12.52
Challenge Video + Lane Detection	485	01:24	5.77
Project Video	1261	02:06	10.01
Project Video + Lane Detection	1261	03:26	6.11

The lowest measured processing speed is 10.01 frames per second, which is considered just adequate as per the recommended performance for this application [56]. Therefore, the more powerful computational hardware if employed should significantly enhance the real-time performance of the proposed pipeline [57].

10. Discussion of The Implemented Approaches

The following points shed some light on some technical tricks and aspects that have been tried or implemented in the described pipelines:

1) Color spaces: around 7 different color spaces have been tried on both testing images and videos. Throughout the experimentation, both HSV and LAB produced the best results in both vehicle finding and lower false positives. The other color spaces like YUV, LUV, YCrCb, HLS produces comparable results. However, RGB produced the worst results among them by far. Therefore, HSV and LAB are adopted during the development and testing phases.

2) Decision function: After applying the trained SVMC model on every constructed sliding widow to search for vehicles, the decision function [54] (from the SciKit-Learn library) [52] is used instead of simple prediction function. The decision function returns the probability of the object being a vehicle or not [58]. So, positive probabilities mean that the object is at least 50% a vehicle, and accordingly, negative probabilities mean it is more than 50% non-vehicle object. By defining a new parameter "Confidence_score" which identifies the confidence for an object of being a vehicle. The higher the positive value the higher the confidence for the object of being a car. Using decision function helped reducing false positives significantly.

3) Heat-maps filtering: the calculated heat-maps on each frame are not used directly, however, they will be been filtered using an FIR filter. This FIR is designed to use the current and the previous values of the previous four frames, before applying a threshold. This technique helped to smooth out the constructed vehicle windows and helped in reducing false positives as well.

4) Vehicle box vertices filtering: Similar to the heat-maps filtering, the constructed vehicle boxes are also filtered out using FIRs. The calculated vertices are not used directly but got filtered first using the calculated values of the previous three frames. This technique helped to reduce the jitter of the position and the size of the identified final car boxes for each frame.

5) Identification of the regions of interest: the *RT_VDT* pipeline has been constructed to include the designation of several ROI search areas by both x and y-axis. This approach helped to more accurately identify search areas, reduces the search time, improve search performance and eliminates undesired false positives.

6) Frame sampling: throughout the experimentation, it is found that it is not necessary to search for vehicles every frame at the current sampling rate of the camera (25 fps), as the movement of vehicles from frame to frame is not that fast. Therefore, the active search for vehicles is restricted to every other frame, which reduces the video processing time by half and almost didn't affect the result at all.

7) Sanity checks: some sanity checks are used to improve the identified vehicle boxes like:

   a) Vehicle box size: the identified vehicle box size is being measured and checked out before it is being drawn to the image or video frame. This is done by measuring the diagonal of the identified box and compare it with certain specified constraints.

   b) Vehicle box position: some checks are added to validate the position of the identified car boxes. For example, in the test images, vehicle boxes can't be found at a position lower than "y = 400".

11. Conclusion

In this paper, reliable and sophisticated vehicle detection and tracking technique based on computer-vision algorithms are developed, presented thoroughly and given the name *RT_VDT*. *RT_VDT* uses a pipeline of well-known color spaces such as LAB, YUV, LUV, etc. Additionally, it uses computer-vision algorithms like *HOG* features, and machine learning algorithms like *Support Vector Machines*. Moreover, the pipeline uses a comprehensive image distortion suppression and camera calibration techniques to produce undistorted road images suitable for more accurate vehicle detection. In addition to that, several sanity-check tricks are exercised to improve the robustness of the techniques used. The proposed *RT_VDT* technique needs only raw RGB images from a single CCD camera mounted behind the front windshield of the vehicle. The performance of the *RT_VDT* algorithm is tested and evaluated using many stationary images and several real-time videos. The validation results show a fairly accurate and robust detection with slight insignificant deviation in one scenario where complex shadow patterns exist. The measured throughput (execution time) using an affordable CPU proved that the *RT_VDT* is very suitable for real-time vehicle detection if more processing power, like GPUs, is added. Therefore, the proposed technique is well suited to be used in Advanced Driving Assistance Systems (ADAS) or self-driving cars.

Acknowledgment

This work used the High-Performance Computing (HPC) facilities of the American University of the Middle East, Kuwait.

References

[1] Wael Farag, "Traffic signs classification by deep learning for advanced driving assistance systems", *Intelligent Decision Technologies*, IOS Press, vol. 13, no. 3, pp. 215-231, (2019).

[2] Wael Farag, Zakaria Saleh, "Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems", *Intern. Conf. on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'18)*, Bahrain, 18-20 Nov., (2018).

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

[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] Wael Farag, "A Comprehensive Real-Time Road-Lanes Tracking Technique for Autonomous Driving", International Journal of Computing and Digital Systems (IJCDS), vol. 9 (3), pp. 349-362, (2020).

[6] Wael Farag, Z. Saleh, "An Advanced Road-Lanes Finding Scheme for Self-Driving Cars", *Smart Cities Symposium (SCS'19)*, IET Digital Library, Bahrain, 24-26 March, (2019).

[7] Wael Farag, Zakaria Saleh, "Behavior Cloning for Autonomous Driving using Convolutional Neural Networks", *Intern. Conf. on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'18)*, Bahrain, 18-20 Nov., (2018).

[8] 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).

[9] Wael Farag, Zakaria Saleh, "Tuning of PID Track Followers for Autonomous Driving", *Intern. Conf. on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'18)*, Bahrain, 18-20 Nov., (2018).

[10] Wael Farag, "Safe-driving cloning by deep learning for autonomous cars", *International Journal of Advanced Mechatronic Systems,* Inderscience Publishers, vol. 7, no. 6, pp. 390-397, (2019).

[11] Wael Farag, "Cloning Safe Driving Behavior for Self-Driving Cars using Convolutional Neural Networks", *Recent Patents on Computer Science*, Bentham Science Publishers, The Netherlands, Vol. 12, No. 2, pp. 120-127(8), (2019).

[12] Wael Farag and Zakaria Saleh, "An Advanced Vehicle Detection and Tracking Scheme for Self-Driving Cars", *2nd Smart Cities Symposium (SCS'19),* IET Digital Library, Bahrain, 24-26 March, (2019).

[13] C. Premebida, G. Monteiro, U. Nunes, and P. Peixoto, "A Lidar and Vision-based Approach for Pedestrian and Vehicle Detection and

Tracking”, *2007 IEEE Intelligent Transportation Systems Conference*, 30 Sept.-3 Oct. 2007, Seattle, WA, USA.

[14] Anna Petrovskaya and Sebastian Thrun, “Model Based Vehicle Tracking for Autonomous Driving in Urban Environments”, *Robotics: Science and Systems 2008*, Zurich, CH, June 25-28, 2008.

[15] Anna Petrovskaya and Sebastian Thrun, “Model Based Vehicle Detection and Tracking for Autonomous Urban Driving”, *Autonomous Robots* (2009) 26: 123-139.

[16] DARPA Grand Challenge (2007), https://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2007), retrieved on 17th Oct. 2018.

[17] Anna Petrovskaya and Sebastian Thrun, “Efficient Techniques for Dynamic Vehicle Detection”, In: *Khatib O., Kumar V., Pappas G.J. (eds) Experimental Robotics, 2009. Springer Tracts in Advanced Robotics*, vol 54, Springer, Berlin, Heidelberg.

[18] A. Jazayeri, H. Cai, J.Y. Zheng, “Motion-Based Vehicle Identification in Car Video”, *2010 IEEE Intelligent Vehicles Symposium*, San Diego, CA, USA, 21-24 June 2010.

[19] A. Jazayeri, H. Cai, J.Y. Zheng, and M. Tuceryan, “Vehicle Detection and Tracking in Car Video Based on Motion Model”, *IEEE Trans. on Intelligent Transportation Systems*, Vol. 12(2), June 2011.

[20] X. Huang, A. Acero, and H.-W. Hon, “Spoken Language Processing”, *Prentice-Hall*, 2001, ISBN -013-022616-5.

[21] G. D. Forney, “The Viterbi algorithm”, *Proceedings of the IEEE, 61 (3): 268-278, March 1973.*

[22] E. Romera, L.M. Bergasa and R. Arroyo, “A Real-Time Multi-scale Vehicle Detection and Tracking Approach for Smartphones”, *2015 IEEE 18th Inter Conf. on Intelligent Transportation Sys.*, 15-18 Sept. 2015, Las Palmas, Spain.

[23] L. M. Bergasa, D. Almer´ıa, J. Almaz´an, J. J. Yebes, and R. Arroyo, “Drivesafe: an app for alerting inattentive drivers and scoring driving behaviors”, *IEEE Intelligent Vehicles Symp. (IV)*, 2014, pp. 240–245.

[24] I. del Pino et al., “Low Resolution Lidar-Based Multi-Object Tracking for Driving Applications”, *3rd Iberian Robotics Conference, ROBOT 2017*, pp 287-298, Springer.

[25] Velodyne VLP-16 (PUCK), https://velodynelidar.com/vlp-16.html, retrieved on 17th Oct. 2018.

[26] Velodyne HDL-64, https://velodynelidar.com/hdl-64e.html, retrieved on 17th Oct. 2018.

[27] Raspberry Pi v3, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, retrieved on 17th Oct. 2018.

[28] M. Anandhalli, V. Baligar, “A novel approach in real-time vehicle detection and tracking using Raspberry Pi”, *Alexandria Engineering Journal*, Elsevier, July 2017.

[29] B. Tian, Y. Li, B. Li, and D. Wen, “Rear-view vehicle detection and tracking by combining multiple parts for complex urban surveillance”, *IEEE Trans. on Intelligent Transportation Systems*, vol.15, no.2, pp. 597–606 (April 2014).

[30] Z. Zheng, G. Zhou, Y. Wang, Y. Liu, X. Li, X. Wang and L. Jiang, “A novel vehicle detection method with high-resolution highway aerial image”, *IEEE J. of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(6), pp. 2338–43 (Dec. 2013).

[31] A.S. Abdul Rachman, “*3D-LIDAR Multi Objet Tracking for Autonomous Driving*”, M.Sc. Thesis, Delft University of Technology, Nov. 2017.

[32] R. Vivacqua , R. Vassallo and F. Martins, “A Low-Cost Sensors Approach for Accurate Vehicle Localization and Autonomous Driving Application”, *Sensors*, 17(2359), October 2017.

[33] Wael A Farag, VH Quintana, G Lambert-Torres, “Genetic algorithms and back-propagation: a comparative study”, *IEEE Canadian Conf. on Elec. and Comp. Eng.*, vol. 1, pp. 93-96, Waterloo, Ontario, Canada, (1998).

[34] M. Siam, S. Elkerdawy, M. Jagersand, and S. Yogamani, “Deep Semantic Segmentation for Automated Driving: Taxonomy, Roadmap and Challenges”, arXiv:1707.02432v2, *IEEE 20th Intern. Conf. on Intelligent Transportation Sys. (ITSC)*, Oct. 2017.

[35] D. Feng, L. Rosenbaum, K. Dietmayer, “Towards Safe Autonomous Driving: Capture Uncertainty in the Deep Neural Network For Lidar 3D Vehicle Detection”, *21st IEEE Intern. Conf. on Intelligent Transportation Sys. (ITSC)*, Hawaii, USA, Nov. 2018.

[36] “Color Space”, https://en.wikipedia.org/wiki/Color_space, retrieved on 28th Oct., (2018).

[37] “List of color spaces and their uses”, https://en.wikipedia.org/wiki/List_of_color_spaces_and_their_uses, retrieved on 28th Oct. 2018.

[38] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection”, *IEEE Computer Society Conf. on Computer Vision and Pattern Recog. (CVPR'05)*, 20-25 June 2005, San Diego, CA, USA.

[39] MS Kankanhallia, BM Mehtreb, and HY Huang, “Color and spatial feature for content-based image retrieval”, *Pattern Recognition Letters*, Elsevier, Vol. 20, Issue 1, Jan. 1999, Pages 109-118.

[40] Szabolcs Sergyan, “Color histogram features based image classification in content-based image retrieval systems”, *6th International Symposium on Applied Machine Intelligence and Informatics*, 21-22 Jan. 2008, Herlany, Slovakia.

[41] C. Cortes, VN Vapnik, "Support-vector networks", *Machine Learning*, 20 (3): 273–297, 1995, doi:10.1007/BF00994018.

[42] A. Ben-Hur, D. Horn, H. Siegelmann, and VN Vapnik, "Support vector clustering", *Journal of Machine Learning Research*, 2: 125–137, 2001.

[43] Wael Farag, Ahmed Tawfik, “On fuzzy model identification and the gas furnace data”, *Proceedings of the IASTED International Conference Intelligent Systems and Control*, Honolulu, Hawaii, USA, August 14-16, (2000).

[44] Support vector machine, https://en.wikipedia.org/wiki/Support_vector_machine, retrieved on Nov. 1st, 2018.

[45] Jorge Nocedal, J. Stephen Wright, “Numerical Optimization”, (2nd ed.), Berlin, New York: *Springer-Verlag*, p. 449, ISBN 978-0-387-30303-1, 2006.

[46] “Developer Reference for Intel - Integrated Performance Primitives 2019”, https://software.intel.com/en-us/ipp-dev-reference-color-models, retrieved on (22 Sept. 2018).

[47] OpenCV Python Library, https://opencv.org/, retrieved on (22 Sept. 2018).

[48] “Python Pickle Module”, https://docs.python.org/3.1/library/pickle.html, retrieved on (24 Sept. 2018).

[49] Udacity vehicles data, https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip, retrieved on (24 Sept. 2018).

[50] Udacity non-vehicles data, https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip, retrieved on (24 Sept. 2018).

[51] GTI vehicle image database, http://www.gti.ssr.upm.es/data/Vehicle_database.html, retrieved on (24 Sept. 2018).

[52] KITTI vision benchmark suite, http://www.cvlibs.net/datasets/kitti/, retrieved on (24 Sept. 2018).

[53] The HOG feature descriptor, http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html, retrieved on (24 Sept. 2018).

[54] SciKit-Learn StandardScaler Function, http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html, retrieved on (24 Sept. 2018).

[55] Linear SVM Classifier Function, http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html, retrieved on (24 Sept. 2018).

[56] Jan Botsch, "Real-time lane detection and tracking on high-performance computing devices", *Bachelor's Thesis in Informatics*, Technische Universitat, Munchen, Germany, March 2015.

[57] M. Nagiub and W. Farag, "Automatic selection of compiler options using genetic techniques for embedded software design", *IEEE 14th Inter. Symposium on Comp. Intelligence and Informatics (CINTI)*, Budapest, Hungary, Nov. 19, (2013).

[58] Wael Farag, "Synthesis of intelligent hybrid systems for modeling and control", Ph.D. Thesis, *University of Waterloo*, Canada, (1998).

Wael Farag earned his Ph.D. from the University of Waterloo, Canada in 1998; M.Sc. from the University of Saskatchewan, Canada in 1994; and B.Sc. from Cairo University, Egypt in 1990. His research, teaching and industrial experience focus on embedded systems, mechatronics, autonomous vehicles, renewable energy, and control systems. He has combined 17 years of industrial and senior management experience in Automotive (Valeo), Oil & Gas (Schneider) and Construction Machines (CNH) positioned in several countries including Canada, USA & Egypt. Moreover, he has 10 Years of academic experience at Wilfrid Laurier University, Cairo University, and the American University of the Middle East. Spanning several topics of electrical and computer engineering. He is the holder of 2 US patents; ISO9000 Lead Auditor Certified and Scrum Master Certified.