IJRCS

International Journal of Robotics and Control Systems
Vol. 3, No. 4, 2023, pp. 704-727
ISSN 2775-2658
http://pubs2.ascee.org/index.php/ijrcs

ASCEE

Finding and Tracking Automobiles on Roads for Self-Driving Car Systems

Wael Farag [a,1,*], Mohamed Abouelela [a,2], Magdy Helal [a,3]

[a] College of Engineering and Technology, American University of the Middle East, Egaila 54200, Kuwait
[1] wael.farag@aum.edu.kw; [2] mohamed.abouelela@aum.edu.kw; [3] magdy.helal@aum.edu.kw
* Corresponding Author

ARTICLE INFO

Article history
Received April 30, 2023
Revised May 21, 2023
Accepted October 02, 2023

Keywords
Autonomous Driving;
Self-Driving Vehicle;
Computer Vision;
Car Detection;
Car Tracking;
ADAS;
Machine Learning

ABSTRACT

Road-object detection, recognition, and tracking are vital tasks that must be performed reliably and accurately by self-driving car systems in order to achieve the automation/autonomy goal. Other vehicles are one of the main objects that the egocar must accurately detect and track on the road. However, deep-learning approaches proved their effectiveness at the expense of very demanding computational power and low throughput. They must be deployed on expensive CPUs and GPUs. Thus, in this work, a lightweight vehicle detection and tracking technique (LWVDT) is suggested to fit low-cost CPUs without sacrificing robustness, speed, or comprehension. The LWVDT is suitable for deployment in both advanced driving assistance systems (ADAS) functions and autonomous-car subsystems. The implementation is a sequence of computer-vision techniques fused together and merged with machine-learning procedures to strengthen each other and streamline execution. The algorithm details and their execution are revealed in detail. The LWVDT processes raw RGB camera pictures to generate vehicle boundary boxes and tracks them from frame to frame. The performance of the proposed pipeline is assessed using real road camera images and video recordings under different circumstances and lighting/shading conditions. Moreover, it is also tested against the well-known KITTI database, achieving an average accuracy of 87%.

1. Introduction

The main reasons for installing ADAS subsystems in current vehicles are to increase safety, decrease traffic accidents, and improve comfort and the driving experience [1]-[3]. Major automakers have started implementing a variety of high-tech ADAS features in recent years [4][5] such as lane-departure warning (LDW), anti-lock braking system (ABS), electronic stability control (ESC) [6], lane-keep assist (LKA) [7], etc. These features signify slow but steady progress in the direction of a potential future with secure, completely autonomous cars [8]-[12].

The most recent ADAS features, such as collision warning, emergency braking, automated urban driving, autonomous highway driving (Autopilot), collaborative maneuvering, and automated parking are all examples of automated driving technologies, that call for increasingly quick and accurate finding and tracking of automobiles on roads [13], which is one of the very difficult and demanding tasks. These cars' relative positions in relation to the motorway, the assessment and verification of the

automobile's movement direction, and the correct localization of possible automobiles in camera pictures or LiDAR data are necessary for successfully detecting the other vehicles on the road.

The primary instruments that enable the ability to understand the neighboring and distant environments for the tracking, identification, and detection of moving automobiles are thought to be computer vision techniques. Finding specific patterns, characteristics, or signals in images, like color distributions, colored segments, gradients, and edges, is the main constituent of the method that speeds up or directs the vehicle-detecting procedure.

The method utilized in this publication, called LWVDT which is a lightweight automobile finding and tracking, concentrates on a thoughtful balance between the three below-listed goals:

1) Precise on-road vehicle detection from pictures captured by the car's camera mounted on the front windshield.
2) Quick enough to enable the autonomous car to make fast and precise decisions and follow up with them.
3) Avoid complicated architecture (e.g., manageable memory necessities and computing complexity) which can be implemented using inexpensive CPUs in real-time, such as those commonly used in ADAS building blocks.

Further, the methodology combines cutting-edge hand-crafted features that have been extracted from camera pictures along with a reliable classifier developed using machine-learning techniques in order to detect cars using cutting-edge hand-crafted features. This method accomplishes the coming objectives:

1) The extracted hand-crafted features can be combined and adjusted flexibly, as some of them can be integrated to create a so-called "feature vector". Its adaptability enables some sort of color channels that has been selected from various color spaces to be included in the feature vector. Additionally, you can customize the LWVDT pipeline by setting a concise number of parameters. There is no need to reconstruct or design the entire process or entirely train the employed neural networks repetitively as with the techniques that depend on deep learning. Such adaptability also assists to adjust LWVDT for multiple higher or lower camera resolutions without significant loss of precision. Also, the transparent structure of LWVDT makes it much easier to extend and improve in the future compared to deep learning-based techniques that typically have a black-box arrangement.

2) In this method, the sophisticated feature extraction stage is executed by inexpensive CPUs in a relatively short amount of time, and it does not require the involvement of GPUs in the process, which is common in approaches based on deep learning.

3) The computational resources (in the form of memory requirements and computational power) required for LWVDT are significantly less than those required for deep learning algorithms. It is therefore more appropriate for ADAS subsystems running on classical scalar processors with thirty-two-bit architecture.

When it comes to in-vehicle finding and recognition, execution time is as crucial as accuracy. Instead of sacrificing runtime for accuracy, one needs to reach the balance between runtime and accuracy. The review below shows that the methods based on deep learning have achieved substantial success in automobile detection and have improved performance over classical approaches, but these methods are computationally intensive and, in most cases, expensive. However, they failed to deliver acceptable real-time performance, even with large GPUs and multi-core processors.

According to Wei et al. [14], CNN feature maps may be enhanced with context and deeper features through the use of deconvolution and fusion techniques, improving object recognition and addressing occlusion issues. In order to evaluate the suggested CNN improvements, images with a resolution of 1280×384 were sourced from the KITTI dataset [15]. In order to conduct the evaluation, high-end hardware, such as a server with the following spec: eight CPU cores with 4.20 GHz, Intel i7-7700k, 32 GB of memory, and a GeForce GTX 1080 graphics card, has been employed. Even

though, the top estimated time of 0.24 sec is reported for the inference per frame, which corresponds to a frame rate of merely 4 frames per second.

In addition, X. Hu et al. [16] present SINet which is a convolutional neural network (CNN) that is scale-independent for quick vehicle detection with widespread alterations. Those researchers also suggest that the detection precision is enhanced by employing context-aware ROI pooling in conjunction with multi-branch decision networks. Assessment experimentation was performed on Ubuntu 14.04 employing the KITTI dataset [15] and running on a sole NVIDIA-TITAN-X GPU in addition to eight CPUs (Intel(R) Xeon(R) E5-1620 ver3 at 3.5GHz). Even though the use of very high-cost hardware, the stated inference time per frame is only 0.2 seconds, which corresponds to a speed of only five frames per second even with the use of very expensive hardware.

In his MSc thesis, Xiao uses advanced vehicle detection models to assist in the detection of vehicles [17]. A residual neural network can be used as a feature extractor in the model, and a region proposal network can be used to identify candidate ROI extractors (regions of interest). In this study, the main focus is on addressing the challenges associated with large-scale fluctuations so as to enhance automobile detectors' performance. On a GTX 1080 GPU with 11 GB of RAM, the model was able to achieve 0.269 sec/frame for the inference task (i.e., lower than 3.7 frames per second) with a GTX 1080 GPU.

As a result of this work, the following contributions have been achieved:

1) Execution in Real-time: There are several advantages to the proposed pipeline, including the fact that it focuses on enabling the deployment of ADAS object detection capabilities as soon as possible on low-cost automotive hardware. The goal is to reach 10 frames per second [18] and avoid the employment of GPUs. The throughput does not rely on iterative searches, it relies on one scan per camera image. This is a result of using an efficient method to focus computations on picture sectors of high importance.

2) Incorporation of numerous color spaces: In this study, we use compound color spaces to increase the strength of extracting features and combine them into a further all-inclusive "feature vector". The employed classifier has been trained in several color spaces.

3) Adaptability and Flexibility: The LWVDT algorithm can be customized by tuning only a small count of parameters. There is no need to reconstruct and design the entire process or entirely train the employed neural networks repetitively as is the case with techniques based on deep learning. Such manipulability also facilitates the adjustment of LWVDT for lower or higher multiple resolutions of the camera with no significant shortfall of precision. LWVDT also has a transparent structure, which makes future extensions and improvements much easier than deep learning-based techniques, which typically have a black-box arrangement.

4) Reusability: The suggested procedure has the capability to be employed multiple times with few changes to recognize more objects on the motorway, such as bikes, pedestrians, traffic lights, and road signs.

2. Method

The LWVDT pipeline is constructed to use a sole charge-coupled device video camera (CCD). Such a video camera ought to be fixed on the car's front windshield mirror to capture the front view of the motorway. Nevertheless, stereo cameras are also applicable, but for the sake of simplicity, only a front sole camera is considered in this work. To streamline the problem of detection, we knowledgeably assume that the baseline is a perfect horizontal line due to the setting. This ensures that the "horizon line" is in the picture and the X-axis is parallel to it (i.e. the intersection of the right and left lane lines is projected in a plane called "Horizon" and it is determined using one of the methods explained in [6]). However, for the sake of accuracy, LWVDT uses the front camera calibration data to orient the image and remove visual distortion. The following steps in addition to

Fig. 1 provide an overview of the overall procedure and clarify the combination and collaboration of the technologies used:

1) Calibration of the Camera: The supplied color picture to the *LWVDT* pipeline is supposed to be an RGB 1200x720 in size. Then, the algorithm initially removes the distortion and adjusts the alignment using the checkerboard image camera calibration method. This camera calibration method is performed merely once when the *LWVDT* algorithm is initialized, instead of every iteration/frame, therefore, it does not affect the real-time functionality.

2) Conversion of Color spaces: The calibrated picture is then transformed to grayscale and multiple color spaces [19] (YCrCb, HSL, LAB, HSV, YUV, LUV, etc. [20]). Each color space has its own unique characteristics that increase *LWVDT* performance. Not all of these color spaces are included in the final *LWVDT* pipeline. The final pipeline has been established and will be detailed later, and discussed in more detail in a later section of this study. Some of such color space transformations are done through research, analysis, testing, and trial-and-error to reach that goal.

3) Extraction of Features: Once the color and grayscale space transformation steps got carried out, numerous features are extracted from the calibrated camera image, such as the Histogram of Orientation Gradients (HOG) [21], color space/spatial properties [22], and color histogram properties [23]. How those features are extracted is detailed in the next section. Those properties (features) are then assimilated to create a long vector given the name "feature vector".

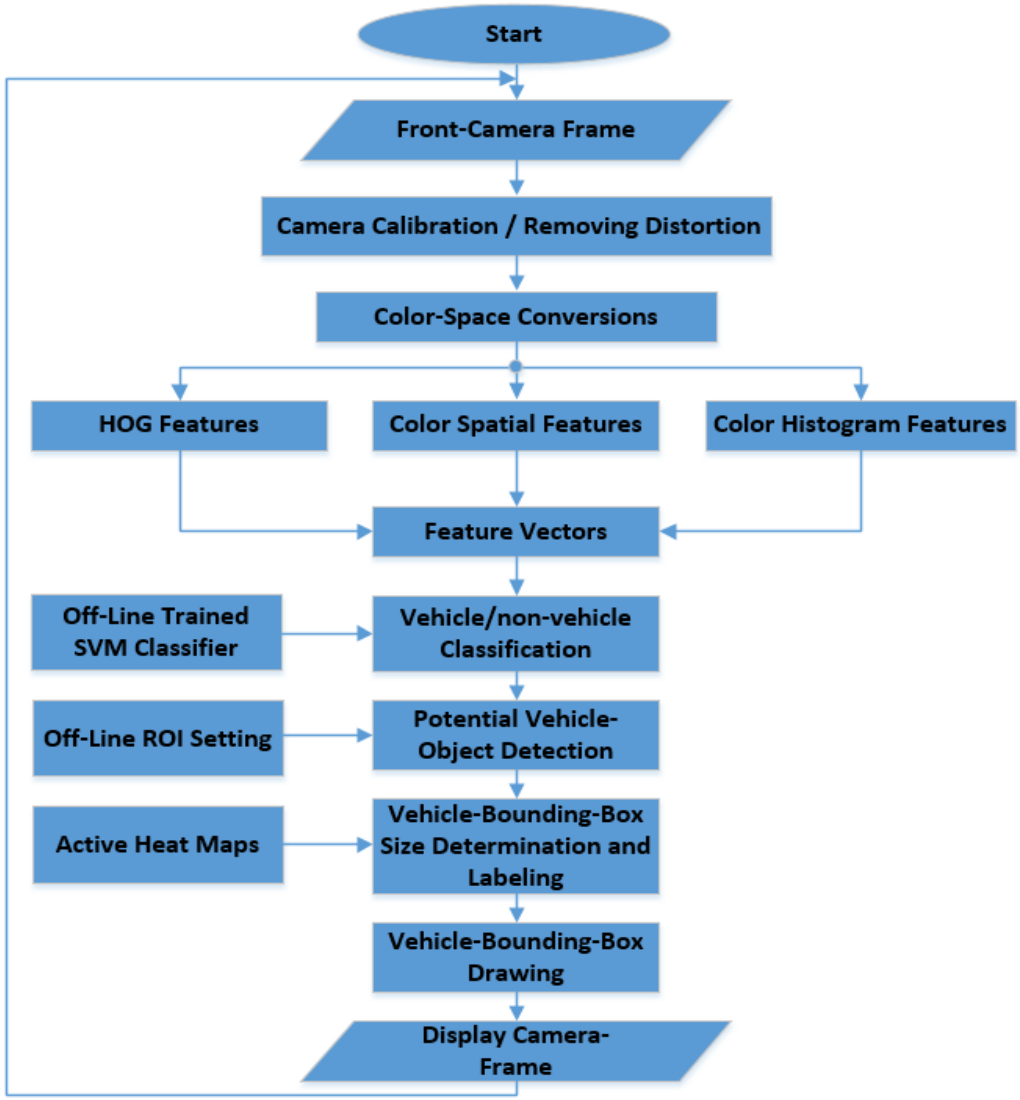

Fig. 1. The LWVDT execution process

4) Classification into vehicles or not: Those feature vectors are then supplied to a classifier that separates vehicle and non-vehicle images. The support vector machine (SVM) classifier is constructed, employed, and trained offline [24] to recognize feature vectors that represent vehicles and those that do not (the ones that represent vehicles will move on to the next step). Full creation and learning of the SVM classifier are detailed in the coming sections (no. 6).

5) Possible automobile object finding: Afterwards, the vehicle images are separated from non-vehicle images by classifying the feature vectors, prospective automobile entities in the camera pictures are found by means of a combination of SVM classifier and sliding windows, and the calibrated camera pictures are scanned to find and identify these automobile objects [25]. Scans are not performed on fully calibrated camera images, but the ROI (regions of interest) are identified and separated from each picture to execute this comprehensive search in it [26][27] with the lowest computational load. Therefore, unwanted picture details are masked to accelerate vehicle boundary finding, enrich the concentration and precision of the finding process, and produce potentially accurate car boxes.

6) Sizing and labeling of vehicle bounding boxes: The outcome of the previous scanning process is employed to create an active heatmap that gathers potential automobile boxes. The duplicated or overlapped true-positive automobile boxes are gathered into larger boxes and marked correspondingly.

7) Automobile bounding boxes drawing: In the step that terminates the process, marked boxes are sketched on the primary camera or frame image. In favor of demonstration purposes, real-life examples of the resulting street boundaries are depicted over the primary color picture, as shown in Fig. 2 and Fig. 3.

Fig. 2. Identified automobile boundary by the LWVDT technique

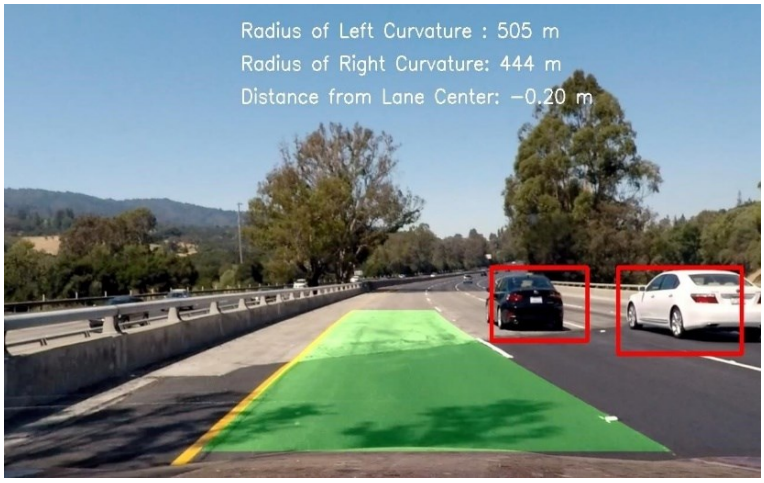

Fig. 3. Identified automobiles' boundaries by the LWVDT technique

3. Overview of Histogram of Oriented Gradients

The HOG feature descriptor is used in image processing and computer vision for the purpose of recognizing and detecting objects in images and video [21]. As an example, to find a definite object 'O_{bj}' in the camera picture, you need to follow these steps:

1) The camera picture is transformed into a grayscale.
2) Initially, create a rectangular (or squared) window with a size of 64 pixels high and 64 pixels wide (those dimensions of the created window can be selected arbitrarily, and be determined by the designer's selection).
3) Employ it to examine the gray camera picture and search for 'O_{bj}'. The examination is performed by moving the window vertically and horizontally by a stride of 8 bits (as an instance).
4) Of course, the object O_{bj} can have various sizes and can fill a larger or smaller portion of the picture. Hence, the investigation would be performed not only on the first initial window (64×64) but also on a sequence of (pyramids-like) windows, such as 80×80, 96×96, 112×112, etc., with a step size of 16 bits (as an instance). The pyramid in this window corresponds to a bigger portion of the original camera picture where O_{bj} or part of it may be within one of them.
5) With the sliding of the window, at each step, HOG features are calculated and linked to the corresponding window center location for "feature localization".

In order to calculate the HOG features, the entry point to the technique is assumed to be a specific window 'W_I' from a grayscale picture (feasibly a pyramid). As illustrated in Fig. 4, the continuation steps are as follows:

1) G_x and G_y (the two components of the gradient) are computed out of the gradient of W_I by median difference in equation (1) and (2).

$$G_x(r,c) = W_I(r,c+1) - W_I(r,c-1) \tag{1}$$

$$G_y(r,c) = W_I(r-1,c) - W_I(r+1,c) \tag{2}$$

where the row and column counts of pixels are r and c respectively in window W_I.

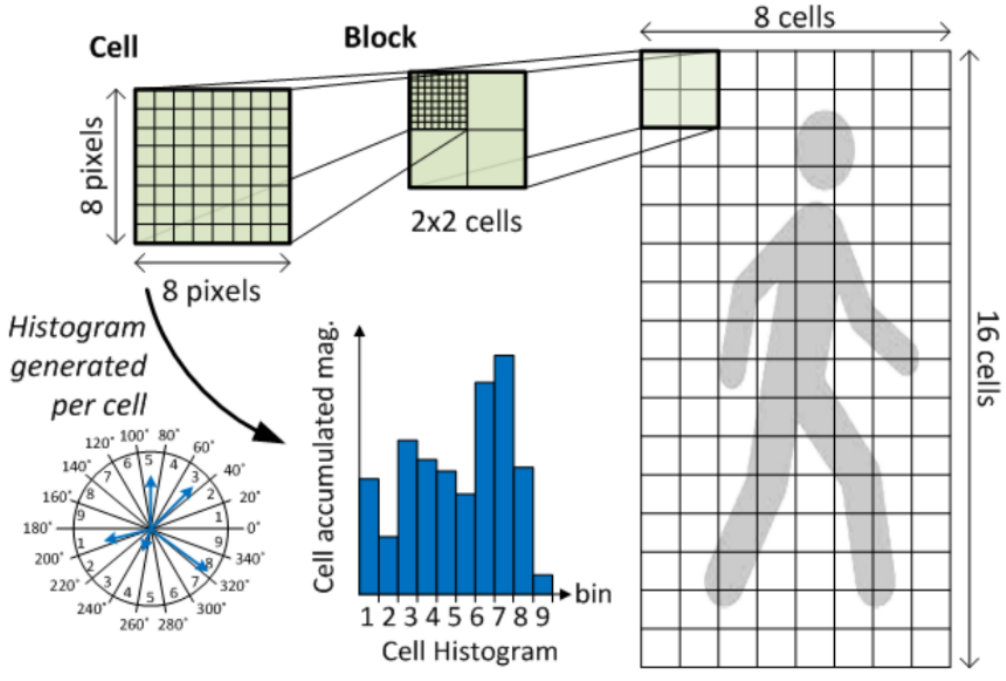

Fig. 4. Workflow of the Histogram of Oriented Gradients

2) The computed slope (e.g. gradient) is transformed to polar coordinates and the angle is restrained from 0° to 180° as equation (3) and (4): Consequently, the gradient pointing in the opposite direction is computed as (3) and (4).

$$G = \sqrt{G_x + G_y} \tag{3}$$

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

where $tan_2^{-1}$is the 4-quadrant reciprocal that produces values amongst -π → π.

3) By the division of the window W_I into non-overlapping cells and bordering each other at the same time with a size of C×C pixels (C=*8* in most cases), a cell orientation histogram can be created. For each cell, compute a histogram of gradient directions binned into B bins (possibly B=*9*). If the bins are numbered from 0 to $B-1$ and have width $w = \frac{180}{B}$, then bin i has bounds $[w_i, w(i\ +\ 1)]$ and center $c_i = w(i + \frac{1}{2})$. Thus, the contribution of a pixel of G magnitude and direction value of θ to the vote can be calculated as equation (5) and (6).

$$v_j = G\frac{c_{j+1} - \theta}{w} \quad to\ bin\ number\ j = \left[\frac{\theta}{w} - \frac{1}{2}\right] mod\ B \tag{5}$$

and the vote:

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

Such a format is named voting by bilinear interpolation. Moreover, the resultant cell histogram is a vector with B +ve numbers.

4) A block normalization phase is afterward performed by assembling the cells into overlapping blocks of *2×2* cells each. Hence, the size of each block is *2C by 2C* pixels. Therefore, two vertically or horizontally proximate blocks overlap by a couple of cells. To be exact, the block increment is C pixels. Therefore, each internal cell is masked with 4 blocks. The 4-cell histograms for each block are centered around a sole block feature b and after that *block feature* 'b' is normalized by its Euclidean norm as equation (7).

$$b \leftarrow \frac{b}{\sqrt{\|b\|^2 + \epsilon}} \tag{7}$$

where ϵ is a miniature +ve amount that does not allow dividing by zero into the blocks that may have zero gradients.

5) The block features that have been normalized are then glued together into a vector h which is a sole normalized HOG feature as in the (8) and (9).

$$h \leftarrow \frac{h}{\sqrt{\|h\|^2 + \epsilon}} \tag{8}$$

$$h_n \leftarrow \min\ (h_n, \tau) \tag{9}$$

where h_n is the n^{th} value in h and τ is the $+ve$ ceiling value ($\tau\ =\ 0.2$). Moreover, trimming the values in h below τ (after initial normalization) makes sure that very big gradients do not have an extreme effect as they eventually may make all other image details washed out. A final normalization step renders the HOG feature not relying on the overall picture contrast. For the matter of illustration, an example of technique output is illustrated in Fig. 5.

Fig. 5. HOG got applied to a car image with the results shown

4. Classification using Support Vector Machine algorithm

A supervised training model called the Support Vector Machine (SVM) [27] analyzes data used in regression and classification analysis [28][29]. An SVM training algorithm creates a model that categorizes new examples corresponding to one of two categories given a set of training examples that have each been denoted as belonging to one of the categories. By doing this, the technique can be described as a binary linear classifier which is non-probabilistic. Given n training points in a dataset of the equation (10).

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

where y_i represents the class to which the point $\vec{x}_i$ belongs and can either be 1 or -1. Each one is a real vector in p dimensions. Finding the "maximum-margin hyperplane," which is expressed in a way to maximize the separation between the hyperplane and the closest point $\vec{x}_i$ from either cluster, is necessary to separate the cluster of points for which $y_i = 1$ from the cluster of points for which $y_i = -1$. Any hyperplane can be expressed as the collection of $\vec{x}$-satisfying points as equation (11).

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

where $\vec{w}$ is the hyperplane's normal vector. The distance of the hyperplane measured from the origin along the normal vector $\vec{w}$ is determined by the parameter $\frac{b}{\|\vec{w}\|}$. The optimization objective function can be expressed as equation (12) if the training data can be separated linearly.

$$"Minimize\ \|\vec{w}\| subject\ to\ y_i(\vec{w}.\vec{x}_i - b) \geq 1,$$

$$for\ i = 1,2, \dots, n" \tag{12}$$

Our classifier, $\vec{x} \mapsto sgn(\vec{w}.\vec{x} - b)$, is determined by the $\vec{w}$ and b that solves this problem. The hinge loss function is introduced as equation (13) if the training data cannot be linearly separated.

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

If the constraint $y_i(\vec{w}.\vec{x}_i - b) \geq 1$ is met, or if $\vec{x}_i$ is on the right wing of the margin, then this function is zero. The value of the function is proportional to how far away from the margin the data are when they are on the incorrect side of the margin. It will then be possible to solve the optimization function as in the equation (14).

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

where the parameter λ determines how to balance two competing demands: one is to increase the margin size, and the other is to make sure that the $\vec{x}_i$ positioned in the proper wing of the margin. If the input data can be classified linearly, the loss function's second term will become negligible for

adequately low values of λ, performing similarly to the hard-margin SVM. It will nevertheless discover whether a classification law is workable or not.

If a classification law that is nonlinear must be obtained, and if the converted data points $\varphi(\vec{x}_i)$ correspond to a linear classification rule, then in addition, a kernel function k is provided that fulfills the condition $k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i).\varphi(\vec{x}_j)$. Therefore, the transformed spaces' classification vector $\vec{w}$ satisfies as in the equation (15).

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

where solving the equation (16) optimization problem yields the c_i's:

$$maximize\ f(c_i \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i \, k(\vec{x}_i, \vec{x}_j) y_j c_j \tag{16}$$

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

Quadratic programming can be used to solve the coefficients c_i's [28], and then solve in equation (17).

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

Last but not least, new points ($\vec{z}$) can be categorized or classified by calculating in equation (18).

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

5. Camera Calibration

A camera's display of a 3-D actual scene converts it to a 2-D one, however, the translation from three-dimensional to two-dimensional is not flawless, leading to image distortion. Actually, compared to their 3D appearance, objects' sizes and shapes are deformed (altered) in the final 2D image. Thus, this distortion needs to be corrected before employing the resulting 2D camera pictures in order to extract and evaluate the accurate and usable information.

Real cameras are built with curved lenses that produce a picture. Depending on the focus and positioning of the objects, the light normally bends to a low or high degree along the borders of these lenses. As a result, there is distortion along the margins of the images, making straight lines or bodies appear to be more or less rounded than they actually are. The main source of deformation is this phenomenon, which is known as "radial distortion".

Furthermore, "tangential distortion" is a significant source of distortion. When the lens of the camera is not completely aligned with the picture plane that is related to the camera sensor and parallel to it, distortion occurs. This causes the image to tilt, making objects appear closer or farther away than they are.

To correct for radial distortion, three coefficients are required: k_1, k_2, and k_3. A correction formula can be used to adjust the look of radially deformed aspects in a picture.

In equation (19) and (20), (x, y) represents an aspect in a deformed picture. To remove the distortion from those aspects, OpenCV [30] should be used to compute r, which is the identified offset between a point in an undeformed (adjusted) picture $(x_{corrected}, y_{corrected})$ and the center of the picture deformation, which is often the image's center (x_c, y_c). This center point (x_c, y_c) is also known as the distortion center. These points are depicted in Fig. 6.

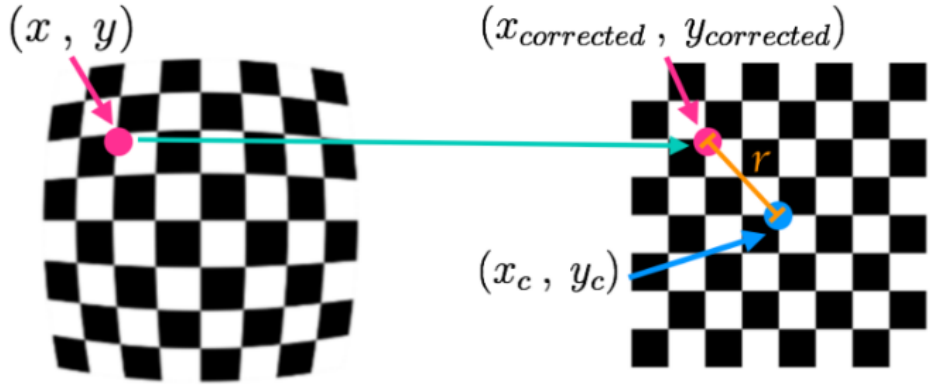

Fig. 6. Pictures with distorted and undistorted (corrected) points

$$x_{distorted} = x_{ideal} + (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (19)$$

$$y_{distorted} = y_{ideal} + (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (20)$$

Tangential distortion is accounted for by two additional coefficients: p_1 and p_2, and such a deformation can be corrected using a different rectification equation, as given by equation (21) and (22).

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \quad (21)$$

$$y_{corrected} = y + [2p_1(r^2 + 2y^2) + 2p_2 xy] \quad (22)$$

Pictures of known shapes (chessboard pictures) are employed to correct for the aforementioned distortions. As shown in Fig. 7, designated pixels in the deformed plans are afterward mapped to undeformed plans. Consequently, the images from the camera will be calibrated. To improve image quality and undistort captured camera images, the following procedure is used:

1) Step 1: Locate the chessboard corners: The "*cv2.findChessboardCorners()*" procedure from the OpenCv3 package [30] is employed to find the chessboard corners by means of 20 chessboard pictures of varying sizes and orientations, as shown in Fig. 8. The detected number of corners is 9×6, as shown in 17 of the 20 pictures shown in Fig. 8. Only 9×5 corners were detected in the other three images. The corners are sketched with openCv3's "*cv2.drawChessboardCorners()*" function.

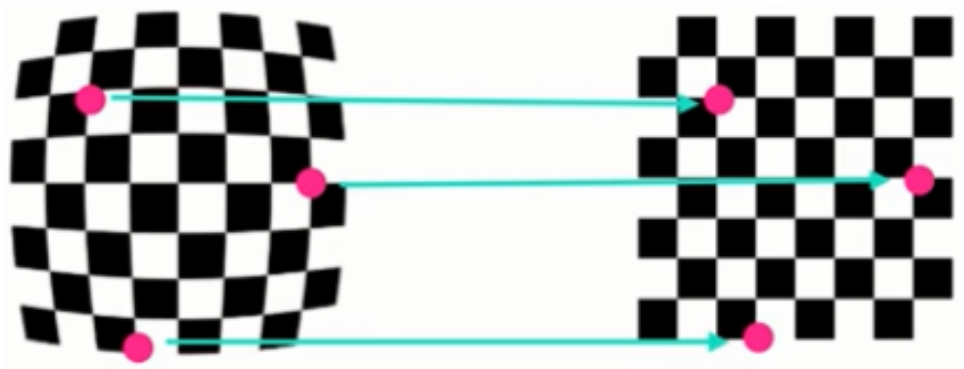

Fig. 7. Mapping from a distorted chessboard image to an undistorted one

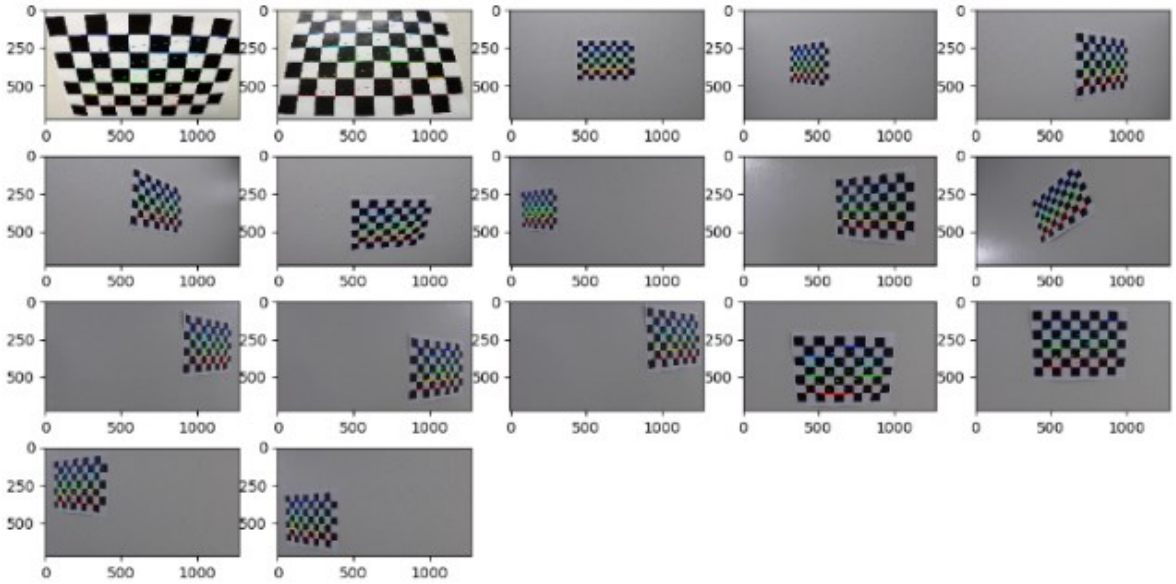

Fig. 8. Calibration pictures of chessboards with corners sketched

2) Step 2: Obtain camera matrices: To find the camera matrices, an experiment chessboard picture that has not previously been employed in detecting the corners is employed, shadowing being transformed to greyscale, along with the detected corners in the first step. This is done with the "*cv2.CalibrateCamera()*" function. To examine the calibration worthiness, use the grey testing picture along with the camera matrices to eliminate the deformation from this picture, as displayed in Fig. 9.

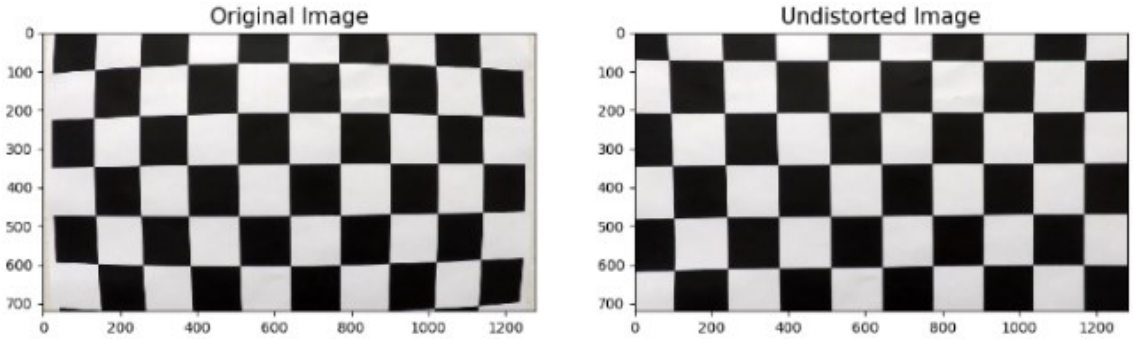

Fig. 9. A testing picture of a chessboard with deformation removed

3) Step 3 - Camera matrices saving: Employing the Pickle software package [31], the camera data (camera specs matrix and distortion coefficients) are stored in the pickle file "*camera calibration.p*" to be recovered later. Fig. 10 shows how to apply the camera calibration technique to one of the testing pictures.

Fig. 10. The influence of camera calibration (removing distortion of pictures)

6. Realization of the Support Vector Machines Classifier

In this part, the methods to develop a classifier based on the SVM machine learning technique mentioned in Section 4 will be described here in detail, while it will be given the acronym "SVMC".

6.1. Preparation of the Training Data

The following summarizes the data preparation stages for training the *SVMC*:

1) Udacity-supplied data [32][33]: Udacity-supplied data were used during the course of this investigation. The data comprises almost equalized pictures of "non-vehicles" and "vehicles":
   a) The "non-vehicle" assortments are the "GTI" [34] and "Extras" collections. Each has 8968 RGB pictures with sizes of (64, 64, 3) pixels.
   b) The "vehicles" assortments comprise the "GTI" and "KITTI". Each has 8792 RGB pictures with sizes of (64, 64, 3) pixels.
2) These assortments are 149MB in size when unzipped.
3) Augmentation of the Data: All of the images are flipped around the "Y" axis to enrich the data. As a consequence, the training data grew to 35,520 pictures.

6.2. Visualization of the Training Data

In the sequence of execution, the following phases explain the implemented data visualization steps:

1) Vehicle Data Display: As illustrated in Fig. 11, 50 pictures selected at random of automobile data have been presented. Each image has a title that corresponds to its position in the training data.
2) Non-Vehicle Data Display: As shown in Fig. 12, 50 pictures selected at random of non-automobile data have been presented. Each image has a title that corresponds to its position in the training data.
3) Presentation of HOG features of Vehicle Data: After converting the picture to grayscale, a selected picture of the automobile data was utilized to extract its hog characteristics. Furthermore, the HOG aspects of non-car cases are obtained, and the results are displayed in Fig. 13.

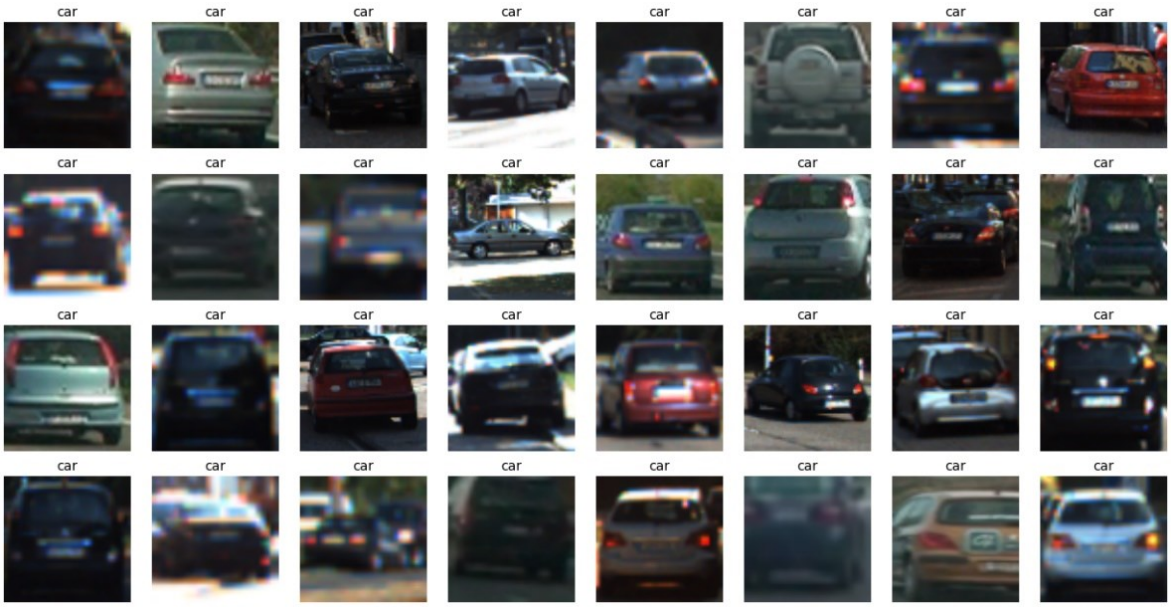

Fig. 11. 32 automobile pictures visualization (chosen randomly)

Fig. 12. 50 non-automobile pictures visualization (chosen randomly)

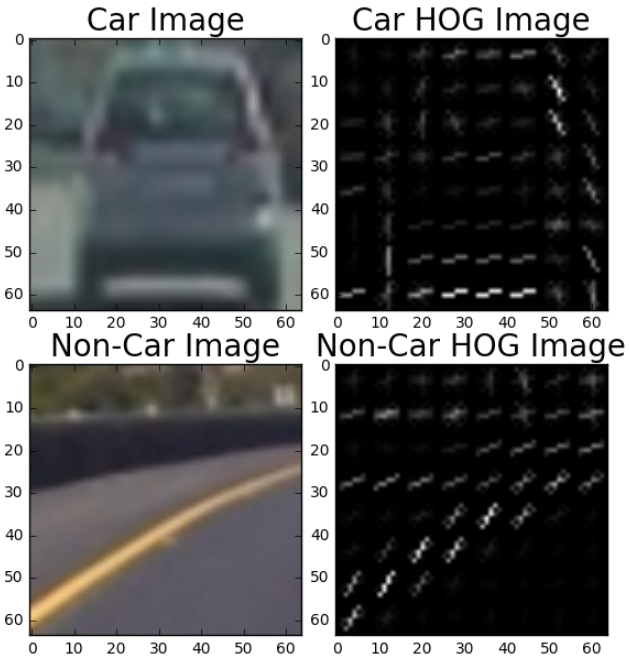

Fig. 13. HOG feature visualization for car and non-car pictures

6.3. Feature Extraction

In the sequence of execution, the following stages explain the realized picture feature extraction procedures:

1) Features of Color Spatial: A method is built to obtain the influence of each image's distinct color channels. To compute the binned color characteristics, in other words. Each image's channel is scaled to (32×32) and raveled.

2) Features of Color Histogram: A procedure is provided to calculate the histogram of each color channel in each picture using a specified count of pins, and afterward glue them together.

3) Computing HOG Properties: A procedure is created to calculate the HOG quantities of each picture channel individually, which may afterward be used independently or appended

collectively if this option is chosen. In this example, the SciKit-Image procedure "hog" [35] is utilized.

4) Integrating Everything: The following feature vectors are produced by the aforesaid feature extraction functions:

   a) Utilizing color spatial features and "spatial size = (32, 32)", a feature vector of 32×32×3 = 3072 items is obtained.

   b) Utilizing color histogram features and "histogram bins = 32" yields a feature vector with 32×3 = 96 elements.

   c) Applying the HOG extracted quantities and the values "gradient orientations cells = 9", "pixels per cell = 8×8", "cells per block = 2×2", in addition to all color channels yields a feature vector of 7×7×2×2×9 = 1764×3 = 5292 items.

   d) If all of the aforementioned procedures are employed, the final feature vector will have 3072+96+5292 = 8460 entries.

6.4. Classifier Training Steps

The procedures below are employed to construct the *SVMC* classifier that separates vehicle images from non-vehicle images and train it:

1) Creating a 35,520×8,460 training data set "X" of 35,520 vehicle/non-vehicle feature vectors of length 8,460 each. The mentioned training dataset serves as the classifier's feed-in.

2) Before joining the feature datasets, they should be scaled using the SciKit-Learn "StandardScaler().fit()" algorithm [36]. The display of unrefined and normalized feature vectors for an automobile picture is shown in Fig. 14.

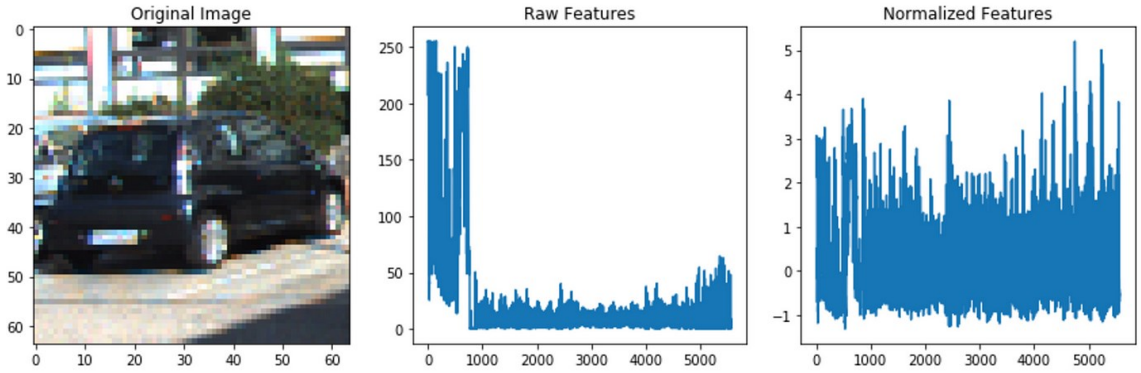

Fig. 14. Feature vector visualization for vehicle pictures

3) Creating an output training set "Y" of 35,520×1 elements, each with a binary value of "One means a vehicle image" or "Zero means non-vehicle image".

4) Using the SciKit-Learn "train test split()" method, the training datasets must be randomly shuffled and divided into 20% for validation and 80% for learning.

5) Employing the SciKit-Learn library's Linear Space Vector Machine Classifier function "LinearSVC()" [36], the constructed classifier was trained with good precision (above 97.7%) in practically all parameter combinations. The trained model is then put to the test on the provided test pictures. In some situations, the outcomes were poor. Extensive experiments were carried out with several parameter combinations, nevertheless, the outcomes were still unacceptable.

6) Following multiple tries and mistakes, it is discovered that the color spatial characteristics consume a substantial percentage of the feature vector span (>36%) short of providing a genuine contribution (often representing an ambiguous aspect) to the differentiation between cars and non-cars. Furthermore, the color histogram characteristics provide a negligible (~1.1%) contribution

to the feature vector as well as the differentiation between automobiles and non-automobiles. As a result, the color special and histogram features have been eliminated from the feature vector, leaving just the HOG features in place. As a consequence, the span of the feature vector is reduced from 8,460 to 5,292 features alone. That is, obviously, streamlines training and real-time performance of the method, resulting in significant reductions in processing and learning phases.

7) The updated Linear SVC classifier is built using various color spaces and a training dataset with a size of 35,520×5,292, as shown in Table 1.

8) With the exception of "RGB," almost other color spaces yielded equivalent results. The "LAB" color space outperforms the "YUV" color space in both learning and prediction, with the 2nd place in the precision results. Nevertheless, when tested on test photos, "YUV" yielded more false positives than "LAB". As a result, the "LAB" color space is chosen for the coming phases.

9) The SVC learning duration has no effect on the functioning of the LWVDT algorithm since it is executed offline; nevertheless, the label prediction time has an effect on the performance because it is included in the finding time for each camera frame.

Table 1. Linear SVC training results

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

7. Pipeline for Car Finding and Tracking

The below-listed stages set up the procedure (*LWVDT*) employed in the finding and tracking of the driving automobiles other than the egocar on the motorway. The following steps are stated here in the same order which will be executed:

1) Detection of Lane boundaries: this procedure is primarily to find the street borderlines (i.e. the lane lines in front of the automobile) which denote the front drivable area (exhibited in green in Fig. 2). This procedure is fully realized in [2] and employed in this text for suitability.

2) Identifying automobiles using the sliding windows technique: for each camera frame, a specialized function is constructed and invoked, using the below factors and limits:

   a) "orient = 9" specifies the count of histogram bins per cell and is employed to extract HOG features from pictures or video frames.

   b) "pix per cell = 8" indicates the count of HOG pixels per cell. The cell will be 8×8 pixels in this example.

   c) "cell per block = 2" specifies the count of HOG cells in each block. The cell will be 2×2 cells in this example.

   d) x_{start}, x_{stop}, y_{start}, y_{stop}: these four arguments establish a rectangular region of interest (ROI) on the picture or frame in which the procedure looks for an automobile using the sliding windows approach.

   e) "step size = 2" specifies the number of cells to step (or slide) to create a new search window that will overlap the previous search window.

   f) "Scale Step = 0.25" specifies the increment in search window size from one search scan to the next.

g) Scale_Multiplier_Start, Scale_Multiplier_End: a couple of parameters that define when the window sizes slowly increase while inspecting the ROI region.

The procedure applies the taught developed SVMC classifier to every built search window. Fig. 15 depicts the construction of sliding windows of various sizes to cover the required ROI. This function may be repeated numerous times with a new set of "a→g" arguments if it is needed.

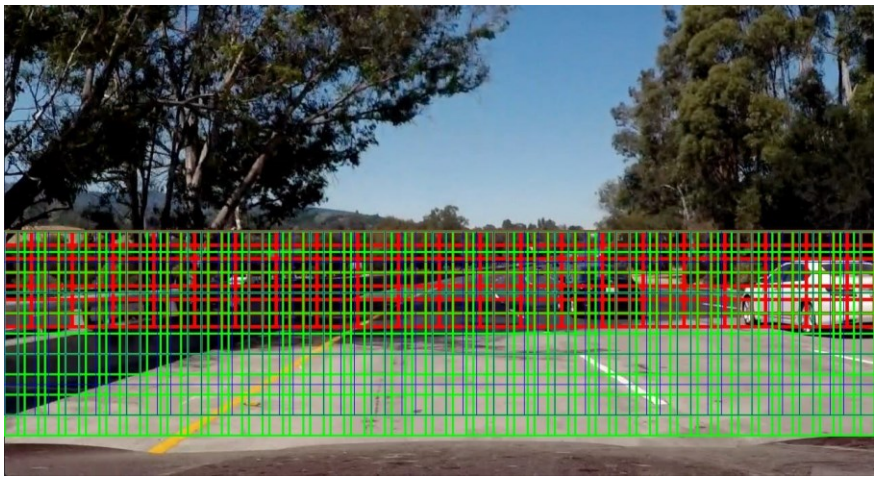

Fig. 15. Sliding panes of varying sizes scan the ROI

3) Active heatmaps Creation: The purpose is to create a heatmap for each detected automobile box for the duration of a sliding-windows scan search. This heatmap is employed to separate (or at least reduce) the number of false-positive boxes. As illustrated in Fig. 16, a specific boundary "HEAT_THRESHOLD" is utilized to merely pass (depending on its assigned amount) the automobile boxes with numerous impacts (true-positive boxes).

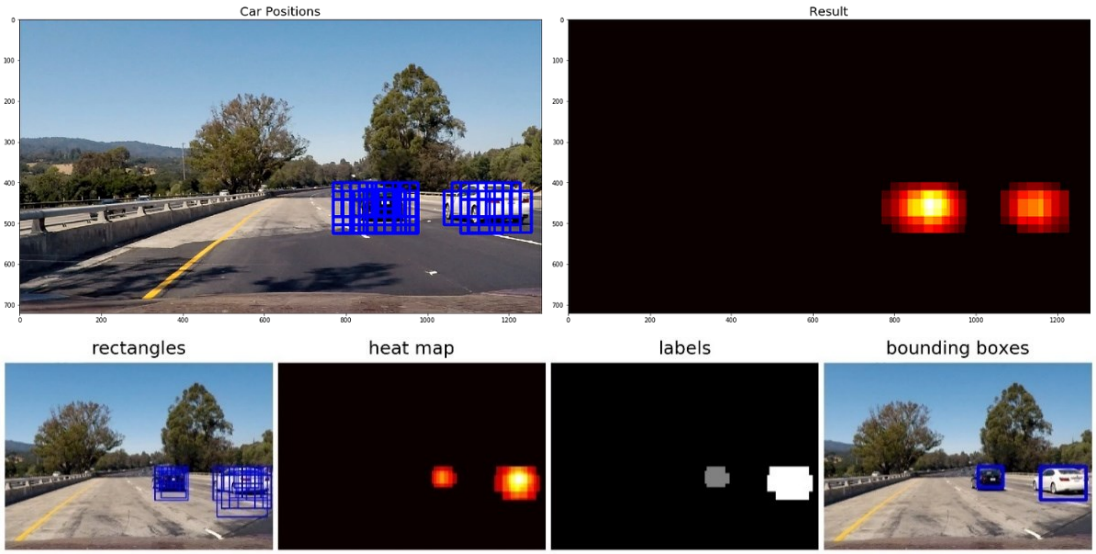

Fig. 16. Found automobile boxes and the resulting heatmaps

4) Car boxes labeling: the overlapped true-positive automobile boxes are afterward aggregated into larger boxes and labeled with the Sci-Kit Learn library's "label()" method.

5) Creating the labeled automobile boxes: Finally, the labeled boxes are sketched on the primary validation picture or frame of a video, as illustrated in Fig. 2 and Fig. 3.

Fig. 17 and Fig. 18 illustrate instances of the findings obtained after running the aforesaid algorithm on the validation pictures, which contain shadow patterns that typically mislead vision-based techniques.

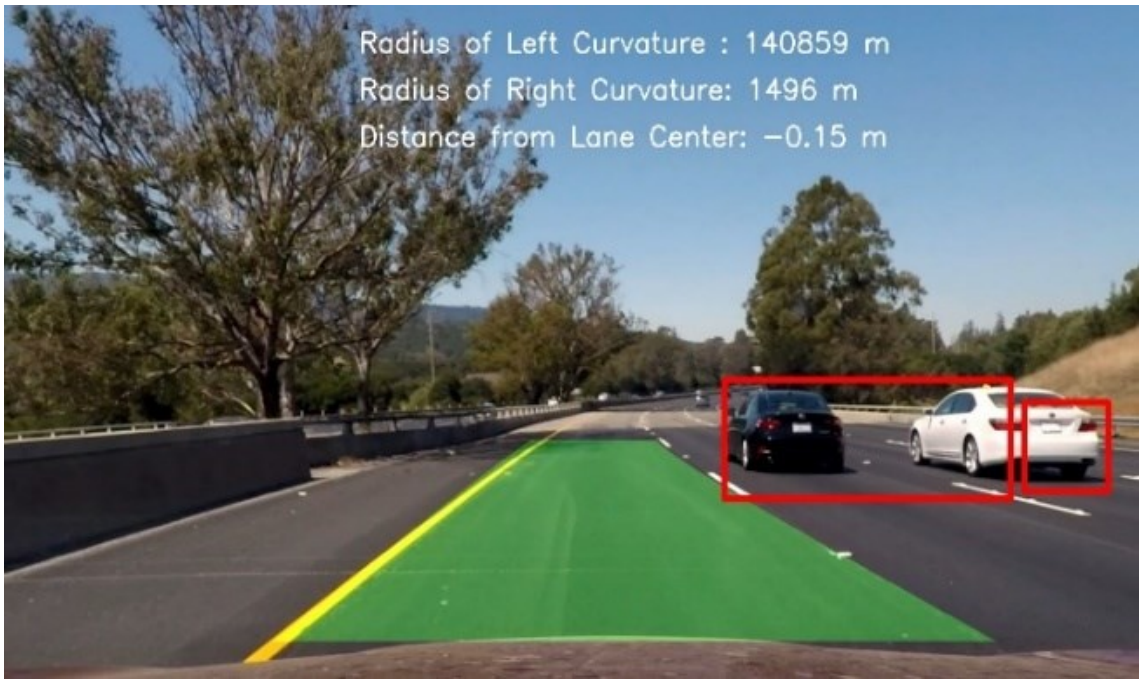

Fig. 17. Automobile identification and tracking pipeline execution

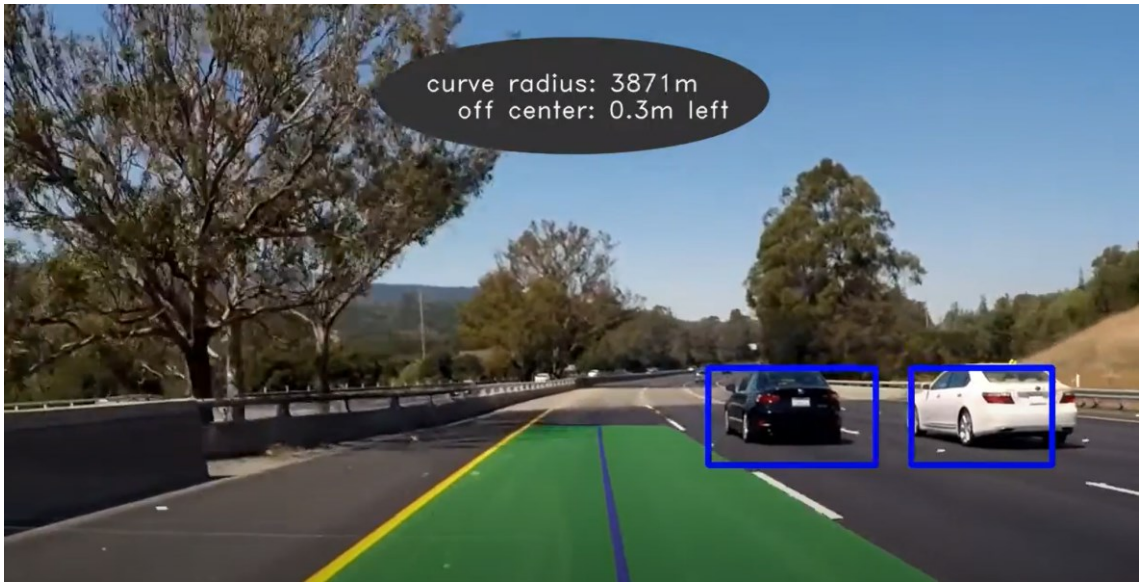

Fig. 18. Automobile identification and tracking pipeline execution

8. Validation, Assessment, and Testing

The proposed LWVDT method is then tested on several images reflecting various scenarios. The outcomes reveal that the pipeline operates admirably under various scenarios (at sunset, at full sunrise, without shadows, with shadows, with no vehicles on the adjacent lanes, and with automobiles on the adjacent lanes). Additionally, for healthiness testing and validation of the created algorithm, the pipeline is applied to many real-time video instances reflecting various driving scenarios. The LWVDT proved to be quite robust in all of the preceding situations, as illustrated in Fig. 2 and Fig. 3. Yet, as demonstrated in Fig. 17 and Fig. 18, the scattered portions of shadows influence the accuracy of constructing the automobiles' border boxes. Yet, the findings are still good enough and create effective working output.

Fig. 2, Fig. 3, Fig. 17, and Fig. 18 demonstrate the lane detection findings applied in the work of [37][38] alongside the vehicle detection results of this paper.

In real-time execution, the pipeline proved to be acceptable as per the results reported in Table 2. The below-listed quantities are measured for a couple of investigating video records (Video1 and Video2) using an Intel dual-core with 8GB RAM Core i5-4200U at 1.6 GHz, which is a pretty reasonable computational platform (very suitable for ADAS applications):

Table 2. Computing Throughput for the LWVDT Pipeline

Name of the Example	# of Frames	Entire Period Min: Sec	Frames per Second
Test Video1	1261.00	02:06	10.01
Test Video1 + Lane Finding	1261.00	03:26	06.11
Challenging Video2	0485.00	00:39	12.52
Challenging Video2 + Lane Finding	0485.00	01:24	05.77

The slowest recorded sorting-out throughput is 10.01 fps (frames/second). This is believed to be appropriate for this application's desired performance [18]. As a result, using more powerful processing gear should greatly improve the real-time outcome of the suggested procedure.

The experimental findings are evaluated for *LWVDT* performance using the three statistical measures test of a binary classification [39]: Precision, Recall, and Intersection over Union (IoU). The fraction of actual +ve samples to all positively recognized samples indicates how accurate the forecasts are. The proportion of actual +ve samples that are accurately identified is measured by recall (e.g., the percentage of automobile pictures, which are recognized as true automobile pictures). The IoU metric calculates the overlapping percentage between the predicted and ground-truth areas to determine how reliable our detector is in comparison to the ground truth. Such expressions are as in equation (23) to (25).

$$Precision = \frac{TP}{TP + FP} \tag{23}$$

$$Recall = \frac{TP}{TP + FN} \tag{24}$$

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{25}$$

where TP designates the count of true positives; in other words, the count of correctly identified vehicle pictures; The count of true negatives (TN) is the count of accurately categorized non-vehicle pictures. The count of false positives is denoted by FP. the count of photos of vehicles classed as non-automobile; The count of false negatives (FN) is the count of non-automobile pictures sorted out as vehicles.

The well-known average precision (AP) and intersection over union (IoU) metrics [39], which have been extensively employed to evaluate numerous automobile recognition techniques, are employed here to assess and compare the functioning of our proposed *LWVDT* to up-to-date methodologies [40][41][42]. The Single-Shot Detector (SSD) [40] is a cutting-edge single-stage detector that produces predictions by leveraging different feature map resolutions. Another sort of single-stage detector is You Only Look Once (YOLO) [41], which produces predictions by seeing unrefined picture information as a 7×7 grid. Furthermore, Faster RCNN [42] is a cutting-edge detector that pioneers the use of a Region Proposal Network (RPN) for the extraction of Region of Interest (RoI) candidates.

Table 3 evaluates the suggested *LWVDT* method against the up-to-date deep-learning-based techniques (e.g., Fast R-CNN, SSD, and YOLO), using the KITTI dataset [14]. The deeplearning

methods clearly outperform the *LWVDT* in terms of detection accuracy but at the penalty of massive computing expense. For instance, YOLOv2 has a high mean precision (AP) as well as real-time functioning on an expensive GPU (37 frames per second). Unfortunately, performance on a very high-end CPU is very low (0.08 frames per second), if we compare it to the 12.52 frames per second of the *LWVDT* on a low-end reasonably-priced CPU. For ADAS systems with comparatively low computational resources, the feature development and deployment cost are as critical as precision, and the car industry requires a fine balance between the two.

Table 3. Comparison of various algorithms on the KITTI vehicle-detection validation dataset

Algorithm	KITTI – Average Precision (AP) %			GPU Measure (Seconds)	CPU Measure (Seconds)
	Easy	Moderate	Hard		
LWVDT	87.19	77.40	60.60	0.058 NVIDIA Tesla K80, 13GB RAM	0.079 i5-4200U @1.6 GHz (2 Cores)
SSD [43]	97.68	93.44	80.36	0.087 NVIDIA GeForce GTX 1080Ti @1.6 GHz	16.784 i7-6820HQ (4 Cores) @ 2.70 GHz
Fast RCNN [43]	96.55	90.21	81.79	0.093 NVIDIA GeForce GTX 1080Ti @1.6 GHz	14.721 i7-6820HQ (4 Cores) with 2.70 GHz
YOLOv2 [43]	95.73	89.15	78.69	0.027 NVIDIA GeForce GTX 1080Ti @1.6 GHz	12.597 i7-6820HQ (4 Cores) with 2.70 GHz

The *LWVDT* pipeline is also executed on the cloud platform: Google Colab [44] using a couple of modes: GPU (NVIDIA Tesla K80, 13GB RAM) and TPU (v2) [44]. The top values obtained on the GPU are 0.058 seconds while on the TPU is 0.073 seconds. These tests show that there isn't much of a difference in performance when compared to CPU findings. The GPU contributed only a 27% increase in computing performance, and in the meantime, the TPU contributed merely 7.5%. The reason behind such an outcome is that the GPU is mostly used to accelerate matrix computations, and the built algorithm includes a lot of matrix calculations. Furthermore, the TPU is primarily intended to accelerate tensor-based computations, which are not employed in the formulation of the RT VDT algorithm. The RT VDT algorithm's performance is also depicted in Fig. 19 and Fig. 20.

Fig. 19. Identified automobile boundary by the LWVDT methodology on the KITTI dataset - 1

Fig. 20. Identified automobile boundary using the LWVDT methodology on the KITTI dataset - 2

9. Discussing the Employed Techniques

The below viewpoints throw more light on various technical methods and properties tried or carried out in the previously explained pipelines:

1) Decision function: As an alternative to a simple estimation function, the decision procedure [36] (from the SciKit-Learn module) is utilized after applying the accomplished SVMC prototype to each built sliding window to hunt for automobiles. The judgment procedure outputs the likelihood that the object is an automobile or not. Positive probability indicates that the identified object is at least 50% an automobile, whereas negative probabilities indicate that the thing is higher than 50% non-car. Via creating a first-hand constraint "Confidence score", that specifies an object's trust in being an automobile. The greater the positive quantity, the greater the confidence in the thing being an automobile. The use of a decision function dramatically reduced false positives.

2) Filtering heatmaps: the calculated heatmaps on each frame are not used directly, but are filtered using an FIR filter. Before applying a threshold, this FIR is meant to employ the current and earlier values of the preceding four frames. This procedure assisted in smoothing out the manufactured automobile windows as well as eliminating false positives.

3) Filtering of the Vehicle box vertices: Like heatmaps, FIRs are used to filter out the constructed vehicle boxes. The generated vertices are not immediately used; rather, they are filtered employing the computed quantities of the three preceding frames. Such a strategy significantly facilitated to diminishing of the jitter of the detected final vehicle boxes' location and size for each frame.

4) Regions of interest identification: the *LWVDT* algorithm has been designed to incorporate the identification of various ROI inspecting areas by both the x- and y-axes. Such a method improved search accuracy, reduced search time, improved search outcomes, and eliminated unwanted false positives.

5) Down-sampling of frames count: During the course of the carrying-out tests, it was discovered that it is not essential to search for automobiles in every frame at the camera's current sample rate (25 FPS) because vehicle movement between frames is very slow. As a result, the in-force search for automobiles is limited to every other frame, which cuts the video sorting out time in half and has almost little effect on the outcome.

6) Sanity checks: Certain sanity checks, for example, are employed to enhance the indicated/detected car boxes, such as:

   a) Size of the vehicle box: The detected vehicle box size is computed and confirmed before being sketched on the picture or video frame. This is accomplished by gauging the designated box's diagonal and comparing it to certain limitations.

   b) Position of the vehicle box: various validation checks are introduced to confirm the location of the indicated automobile boxes. In the testing pictures, for example, vehicle boxes cannot be found below "y = 400".

7) Color spaces: Approximately seven distinct color spaces have been tested on both test pictures and films. During the course of the investigation, both LAB and HSV delivered the greatest results in terms of automobile detection and false positives. Several color spaces, such as HLS, YCrCb, YUV, and LUV, provide equivalent results. RGB, on the other hand, generated by far the worst outcomes. As a result, during the implementation and validation phases, HSV and LAB are used.

Upon the completion of the vehicle tracking functionality. The natural continuation of this study will be to use visual [45], lidar [46], radar [47], and heuristic [48] strategies [49] to solve object detection, localization [50], navigation [51], path planning [52], and stability [53] challenges.

10. Conclusion

Thorough out this work, a trustworthy and refined automobile identification and tracking system centered around hand-crafted feature extraction techniques is proposed and referred to as *LWVDT*. *LWVDT* employs a pipeline that includes well-established color spaces like LUV, YUV, LAB, and others. Furthermore, it employs computer-vision methods such as HOG features as well as machine-learning techniques such as *Support Vector Machines*. Furthermore, the proposed procedure employs extensive picture deformation suppression and camera calibration methods to generate undeformed street pictures appropriate for more precise automobile findings. Furthermore, many sanity-check approaches are utilized to strengthen the reliability of the employed methodologies. The suggested *LWVDT* approach requires just unrefined RGB pictures from a sole CCD camera located behind the vehicle's forward-facing windscreen. The *LWVDT* algorithm's functioning is validated and assessed utilizing a large number of stationary pictures and several real-time recorded videos. The assessment findings reveal that the detection is fairly accurate and resilient, with only a minor inconsequential divergence in one situation with complicated shadow patterns. The observed performance (running time) utilizing a low-cost CPU demonstrated that the *LWVDT* is well-suited for real-time vehicle detection even without the addition of additional processing capacity such as GPUs. Additionally, on the KITTI dataset, the functioning of the *LWVDT* pipeline is compared to that of the most recent deeplearning techniques. Deeplearning methods outperform *LWVDT* in terms of performance, but at a far larger computing cost (fairly expensive GPUs). For low-end Processors, however, the *LWVDT* real-time functioning confidently demonstrates that it is suitable for ADAS tasks or autonomous automobiles. Future research will concentrate on improving the algorithm's detection and tracking of people and bikers.

Author Contribution: All authors contributed equally to the main contributor to this paper. All authors read and approved the final paper.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

References

[1] W. Farag, “Traffic signs classification by deep learning for advanced driving assistance systems”, *Intelligent Decision Technologies*, vol. 13, no. 3, pp. 305-314, 2019, https://doi.org/10.3233/IDT-180064.

[2] W. Farag and Z. Saleh, "Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems," *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 1-8, 2018, https://doi.org/10.1109/3ICT.2018.8855797.

[3] A. Zhirabok, A. Zuev, and K. Chung, "Virtual Sensors Design for Nonlinear Dynamic Systems," *International Journal of Robotics and Control Systems*, vol. 3, no. 2, pp. 134-143, 2023, https://doi.org/10.31763/ijrcs.v3i2.915.

[4] Z. Zainudin and S. Kodagoda, “Gaussian Processes-BayesFilters with Non-Parametric Data Optimization for Efficient 2D LiDAR Based People Tracking,” *International Journal of Robotics and Control Systems*, vol. 3, no. 2, pp. 206-220, 2023, https://doi.org/10.31763/ijrcs.v3i2.901.

[5] W. Farag, “A lightweight vehicle detection and tracking technique for advanced driving assistance systems,” *Journal of Intelligent & Fuzzy Systems*, vol. 39, no. 3, pp. 2693-2710, 2020, https://doi.org/10.3233/JIFS-190634.

[6] W. Farag, “Real-Time Detection of Road Lane-Lines for Autonomous Driving,” *Recent Advances in Computer Science and Communications*, vol. 13, no. 2, pp. 265-274, 2020, https://doi.org/10.2174/2213275912666190126095547.

[7] W. Farag and Z. Saleh, "An advanced road-lanes finding scheme for self-driving cars," *2nd Smart Cities Symposium (SCS 2019)*, pp. 1-6, 2019, https://doi.org/10.1049/cp.2019.0221.

[8] C. J. Manuel, M. Santos, G. G. Lenzi, and A. M. Tusset, "Longitudinal Modeling of a Road Vehicle: 4-Wheel Traction," *International Journal of Robotics and Control Systems*, vol. 2, no. 2, pp. 357-369, 2022, https://doi.org/10.31763/ijrcs.v2i2.698.

[9] F. Micheli, M. Bersani, S. Arrigoni, F. Braghin, and F. Cheli, "NMPC trajectory planner for urban autonomous driving," *International Journal of Vehicle Mechanics and Mobility*, vol. 61, no. 5, pp. 1387-1409, 2023, https://doi.org/10.1080/00423114.2022.2081220.

[10] W. Farag and Z. Saleh, "Tuning of PID Track Followers for Autonomous Driving," *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 1-7, 2018, https://doi.org/10.1109/3ICT.2018.8855773.

[11] B. B. Elallid, N. Benamar, A. S. Hafid, T. Rachidi, and N. Mrani, "A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 9, pp. 7366-7390, 2022, https://doi.org/10.1016/j.jksuci.2022.03.013.

[12] M. Nagiub, "Automatic selection of compiler options using genetic techniques for embedded software design," *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 69-74, 2013, https://doi.org/10.1109/CINTI.2013.6705166.

[13] W. Farag and Z. Saleh, "An advanced vehicle detection and tracking scheme for self-driving cars," *2nd Smart Cities Symposium (SCS 2019)*, pp. 1-6, 2019, https://doi.org/10.1049/cp.2019.0222.

[14] J. Wei, J. He, Y. Zhou, K. Chen, Z. Tang, and Z. Xiong, "Enhanced Object Detection With Deep Convolutional Neural Networks for Advanced Driving Assistance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1572-1583, 2020, https://doi.org/10.1109/TITS.2019.2910643.

[15] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231-1237, 2013, https://doi.org/10.1177/0278364913491297.

[16] X. Hu, X. Xu, Y. Xiao, H. Chen, S. He, J. Qin, and P. -A. Heng, "SINet: A Scale-insensitive Convolutional Neural Network for Fast Vehicle Detection," *IEEE Transactions on Intelligent Transportation Systems,* vol. 20, no. 3, pp. 1010-1019, 2019, https://doi.org/10.1109/TITS.2018.2838132.

[17] Y. Xiao, "Vehicle detection in deep learning," *arXiv preprint arXiv:1905.13390*, 2019, https://doi.org/10.48550/arXiv.1905.13390.

[18] D. Vajak, M. Vranješ, R. Grbić, and N. Teslić, "A Rethinking of Real-Time Computer Vision-Based Lane Detection," *2021 IEEE 11th International Conference on Consumer Electronics (ICCE-Berlin)*, pp. 1-6, 2021, https://doi.org/10.1109/ICCE-Berlin53567.2021.9720012.

[19] B. E. Rogowitz, T. N. Pappas, and S. J. Daly, "Human Vision and Electronic Imaging XII," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 6492, 2007, https://doi.org/10.1117/12.729071.

[20] S. K. Shevell. *The Science of Color*. Elsevier Science & Technology, pp. 202–206, 2003, https://books.google.co.id/books?id=-fNJZ0xmTFIC&hl=id&.

[21] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886-893, 2005, https://doi.org/10.1109/CVPR.2005.177.

[22] M. S. Kankanhalli, B. M. Mehtre, and H. Y. Huang, "Color and spatial feature for content-based image retrieval," *Pattern Recognition Letters*, vol. 20, no. 1, pp. 109-118, 1999, https://doi.org/10.1016/S0167-8655(98)00100-7.

[23] S. Sergyan, "Color histogram features based image classification in content-based image retrieval systems," *6th International Symposium on Applied Machine Intelligence and Informatics*, pp. 221-224, 2008, https://doi.org/10.1109/SAMI.2008.4469170.

[24] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273-297, 1995, https://doi.org/10.1007/BF00994018.

[25] X. Dai, "HybridNet: A fast vehicle detection system for autonomous driving," *Signal Processing: Image Communication*, vol. 70, pp. 79-88, 2019, https://doi.org/10.1016/j.image.2018.09.002.

[26] W. Farag, "Complex-Track Following in Real-Time Using Model-Based Predictive Control", *International Journal of Intelligent Transportation Systems Research*, vol. 19, pp. 112–127, 2021, https://doi.org/10.1007/s13177-020-00226-1.

[27] C-. W. Hsu and C-. J. Lin, "A comparison of methods for multiclass support vector machines," in *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415-425, 2002, https://doi.org/10.1109/72.991427.

[28] W. A. Farag, V. H. Quintana, and G. Lambert-Torres, "Genetic algorithms and back-propagation: a comparative study," *Conference Proceedings. IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.98TH8341)*, vol. 1, pp. 93-96, 1998, https://doi.org/10.1109/CCECE.1998.682559.

[29] W. Farag and A. Tawfik, "On fuzzy model identification and the gas furnace data," in *Proceedings of the IASTED International Conference Intelligent Systems and Control*, pp. 14-16, 2000, https://www.researchgate.net/publication/228707650_On_Fuzzy_Model_Identification_and_the_Gas_Furnace_Data.

[30] A. Kaehler and G. Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2016, https://books.google.co.id/books?id=LPm3DQAAQBAJ&.

[31] "Python Pickle Module", https://docs.python.org/3.1/library/pickle.html, retrieved on (24 Sept. 2022).

[32] Udacity vehicles data, https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip, retrieved on (24 Sept. 2022).

[33] Udacity non-vehicles data, https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip, retrieved on (24 Sept. 2022).

[34] GTI vehicle image database, http://www.gti.ssr.upm.es/data/Vehicle_database.html, retrieved on (24 Sept. 2022).

[35] The HOG feature descriptor, http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html, was retrieved on (24 Sept. 2022).

[36] SciKit-Learn StandardScaler Function, http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html, retrieved on (24 Sept. 2022).

[37] Linear SVM Classifier Function, http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html, retrieved on (24 Sept. 2022).

[38] K. B. Patel, H. C. Lin, A. D. Berger, W. Farag, and A. A. Khan, "EDC draft force based ride controller," *US Patent 6,196,327*, 2001, https://patents.google.com/patent/US6196327B1/en.

[39] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010, https://doi.org/10.1007/s11263-009-0275-4.

[40] W. Liu *et al*., "SSD: Single Shot multi-box Detector," in *Computer Vision–ECCV 2016: 14th European Conference*, pp. 21–37, 2016, https://doi.org/10.1007/978-3-319-46448-0_2.

[41] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 7236–7271, 2017, https://doi.org/10.48550/arXiv.1612.08242.

[42] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017, https://doi.org/10.1109/TPAMI.2016.2577031.

[43] Y. Liu, S. Cao, P. Lasang, and S. Shen, "Modular Lightweight Network for Road Object Detection Using a Feature Fusion Approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 51, no. 8, pp. 4716-4728, 2021, https://doi.org/10.1109/TSMC.2019.2945053.

[44] "Google Colaboratory", https://colab.research.google.com/notebooks/welcome.ipynb, accessed on (5 Jan 2023).

[45] J. Azimjonov and A. Özmen, "A real-time vehicle detection and a novel vehicle tracking systems for estimating and monitoring traffic flow on highways," *Advanced Engineering Informatics*, vol. 50, p. 101393, 2021, https://doi.org/10.1016/j.aei.2021.101393.

[46] Z. Liu, Y. Cai, H. Wang, and L. Chen, "Surrounding objects detection and tracking for autonomous driving using LiDAR and radar fusion," *Chinese Journal of Mechanical Engineering*, vol. 34, pp. 1-12, 2021, https://doi.org/10.1186/s10033-021-00630-y.

[47] M. Lin, J. Yoon, and B. Kim, "Self-driving car location estimation based on a particle-aided unscented Kalman filter," *Sensors*, vol. 20, no. 9, p. 2544, 2020, https://doi.org/10.3390/s20092544.

[48] A. Wischnewski, T. Stahl, J. Betz, and B. Lohmann, "Vehicle dynamics state estimation and localization for high performance race cars," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 154-161, 2019, https://doi.org/10.1016/j.ifacol.2019.08.064.

[49] W. A. Farag and M. Abouelela, "Low-Cost Active Monitoring of Attendance using Passive RFID Technology," *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika*, vol. 8, no. 4, pp. 552-564, Dec. 2022, https://doi.org/10.26555/jiteki.v8i4.25168.

[50] S. Feraco, S. Favelli, A. Tonoli, A. Bonfitto, and N. Amati, "Localization Method for Autonomous Vehicles with Sensor Fusion Using Extended and Unscented Kalman Filters," *SAE Technical Paper*, 2021, https://doi.org/10.4271/2021-01-5089.

[51] Y. Jeong and S. Yim, "Model Predictive Control-Based Integrated Path Tracking and Velocity Control for Autonomous Vehicle with Four-Wheel Independent Steering and Driving," *Electronics*, vol. 10, no. 22, p. 2812, 2021, https://doi.org/10.3390/electronics10222812.

[52] Z. Farkas, A. Mihály, and P. Gáspár, "Model Predictive Control Method for Autonomous Vehicles in Roundabouts," *Machines*, vol. 11, no. 1, p. 75, 2023, https://doi.org/10.3390/machines11010075.

[53] K. Mansour and M. ElHelw, "AiroDiag: A sophisticated tool that diagnoses and updates vehicles software over air," *2012 IEEE International Electric Vehicle Conference*, pp. 1-7, 2012, https://doi.org/10.1109/IEVC.2012.6183181.