electronics

MDPI

Article

Deep Learning-Based Stopped Vehicle Detection Method Utilizing In-Vehicle Dashcams

Jinuk Park, Jaeyong Lee, Yongju Park and Yongseok Lim *

Korea Electronics Technology Institute, Seongnam 13509, Republic of Korea; jinuk.park@keti.re.kr (J.P.); thomas96@keti.re.kr (J.L.); suede8247@keti.re.kr (Y.P.)
* Correspondence: busytom@keti.re.kr

Abstract: In complex urban road conditions, stationary or illegally parked vehicles present a considerable risk to the overall traffic system. In safety-critical applications like autonomous driving, the detection of stopped vehicles is of utmost importance. Previous methods for detecting stopped vehicles have been designed for stationary viewpoints, such as security cameras, which consistently monitor fixed locations. However, these methods for detecting stopped vehicles based on stationary views cannot address blind spots and are not applicable from driving vehicles. To address these limitations, we propose a novel deep learning-based framework for detecting stopped vehicles in dynamic environments, particularly those recorded by dashcams. The proposed framework integrates a deep learning-based object detector and tracker, along with movement estimation using the dense optical flow method. We also introduced additional centerline detection and inter-vehicle distance measurement. The experimental results demonstrate that the proposed framework can effectively identify stopped vehicles under real-world road conditions.

Keywords: stopped vehicle detection; object detection; multiple object tracking; optical flow

check for updates

Citation: Park, J.; Lee, J.; Park, Y.; Lim, Y. Deep Learning-Based Stopped Vehicle Detection Method Utilizing In-Vehicle Dashcams. *Electronics* **2024**, *13*, 4097. https://doi.org/10.3390/electronics13204097

Academic Editors: Felipe Jiménez and Yue Wu

Received: 5 September 2024
Revised: 8 October 2024
Accepted: 16 October 2024
Published: 17 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Road traffic is integral to modern life, with numerous essential services and human activities increasingly relying on it, both directly and indirectly. Consequently, effective traffic management is vital for ensuring efficient transportation, making it a priority for those responsible for traffic monitoring. In the complex environment of urban traffic, stopped or illegally parked vehicles pose a significant threat to the overall traffic system. Although surveillance through the placement of security cameras at road junctions has continuously advanced, not all roads are equipped with cameras, and it is challenging to relay information about detected stopped vehicles from closed-circuit television (CCTV) to each driving vehicle on the road. To prevent secondary accidents resulting from prior incidents or unexpectedly stopped vehicles, it is essential to detect stopped vehicles from within driving vehicles. While recent innovations include high-tech radars designed to capture speeding or stopped vehicles [1], their high cost limits their practicality for widespread deployment in all vehicles. This underscores the need for stopped vehicle detection using computer vision techniques with more cost-effective cameras.

Recently, the rapid advancements in computer vision (CV) technology have led to numerous innovations, with object detection emerging as one of the most prominent. This prominence stems from the fact that object detection often serves as a foundational element for other CV techniques [2–5], thereby enhancing its significance. In modern object detection tasks, real-time detection is essential for addressing various challenges promptly.

In the context of autonomous vehicles, object detection techniques play a crucial role in Advanced Driver-Assistance Systems (ADASs) [6]. ADASs employ sensors and cameras to mitigate risks related to forward collisions [7], lane keeping [8,9], and blind-spot collision avoidance [10]. Additionally, in autonomous vehicle research, accurately recognizing

stopped vehicles is a critical focus. According to the Levels of Driving Automation set out by the Society of Automotive Engineers (SAE), levels 4 and 4+ denote fully automated stages that operate without driver intervention [11]. At these levels, the ability to accurately detect stopped vehicles, based on the state of nearby vehicles and their lateral position on the road or highway, is vital for ensuring smooth driving and preventing accidents.

Previous research on detecting stopped vehicles has focused on two types of scenarios: stationary views [12–14] (such as security camera recordings), and dynamic views [15,16] (such as dashcam recordings). In [12], the authors proposed an algorithm that combined the Kanade–Lucas–Tomasi (KLT) feature tracker with a Self-Organizing Map (SOM) network. Also, ref. [13] employed the Selective Update of Gaussian Mixture Model (SUOG) alongside Fixed Region-of-Interest (FROI) methods for real-time road obstacle detection using security cameras. In [14], the authors introduced a technique for identifying obstacles by comparing resynthesized images with ground-truth data, utilizing three-dimensional information. The authors of [17] proposed a highly accurate object detection model and integrated the framework with an object-tracking model. While these approaches are effective at detecting stopped objects, their applicability is confined to stationary environments such as security cameras.

In contrast, recent research has increasingly concentrated on dashcam footage. For instance, ref. [15] developed a method combining centroid tracking with Mask Region-Based Convolutional Neural Network (Mask R-CNN) [18] for vehicle detection and movement estimation, including the calculation of various kinematic parameters. Also, ref. [16] integrated the YOLO [19] framework with edge detection techniques to identify car accidents using a self-produced dataset. In addition, ref. [20] introduced a fusion method of using roadside units (RSUs) and camera inertial measurement units (IMUs) to detect obstacles. The authors also proposed background subtraction methods for generalized obstacle detection. This approach enhances detection accuracy while reducing computational complexity. Furthermore, ref. [21] implemented an unsupervised traffic accident detection framework based on a Recurrent Neural Network (RNN) encoder–decoder that estimates object locations. However, these methods have not fully addressed movement factors, such as speed and optical flow.

We aim to detect stopped vehicles in adjacent lanes or in front of a moving host vehicle, which differs from the aims of most prior studies. While equipping vehicles with additional sensing devices such as LiDAR can facilitate the identification of obstacles or stopped vehicles, this approach is cost-inefficient. Although the use of security cameras can simplify the detection of stopped vehicles, stationary views cannot be installed on every road, thereby limiting the detection area. Therefore, we developed a novel framework utilizing widely installed dashcams on vehicles to identify stopped vehicles to ensure a highly cost-effective solution. Additionally, the method for assessing relative movement within the vehicle will serve as a stepping stone to accelerate future research in autonomous driving.

Detecting stopped vehicles in dashcam-based environments encounters the significant challenge of accurately calculating the relative speed between the host vehicle and observed vehicles, especially when they are in motion simultaneously. To address these limitations, we introduce a novel approach for identifying stopped vehicles in dynamic settings, such as those recorded by dashcams, by assessing the relative motion between objects and the camera. The method begins with object detection and tracking using deep learning models. Also, the movement of the host vehicle is quantified through optical flow analysis applied to consecutive frames.

For the vehicles being observed, movement estimation is performed by calculating the distance and angle traversed between two frames. Additionally, for vehicles within a specific angular range, such as those in the same lane ahead, stationarity is estimated by calculating differences in the size of the bounding box. This approach is designed to address issues where a vehicle directly in front of the host vehicle appears to exhibit minimal or no movement in the video footage. The accuracy of the proposed framework is further enhanced by auxiliary corrections, including centerline detection to ignore the opposite

lane, as well as eliminating outliers. Finally, the system guarantees real-time processing at approximately 25 frames per second through parallel processing for computationally intensive tasks, including object motion estimation and centerline detection. The simplified framework of our method is outlined in Figure 1.

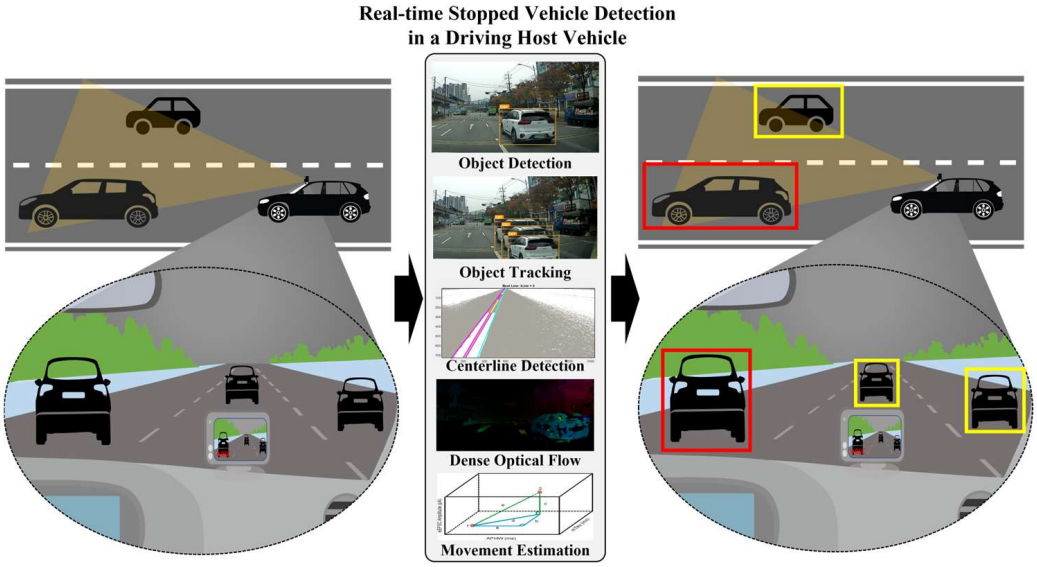


Figure 1. The proposed method for identifying stopped vehicles based on object tracking and movement estimation from a dashcam mounted on a driving vehicle.

The main contributions of this study are as follows:

- We introduce an effective method for detecting stopped vehicles using a dashcam mounted on a moving vehicle, in contrast to most existing methods, which rely predominantly on stationary cameras.
- We address the challenge of detecting stopped vehicles in real time by setting lane-specific parameters and integrating the object detection and tracking models, even in dynamic video data.
- The proposed model shows high detection performance in urban road conditions. Moreover, the comprehensive qualitative and extensive quantitative results demonstrate that the proposed method is effective for real-world applications.

The remainder of this paper is organized as follows: Section 2 briefly reviews related techniques on stopped vehicle detection tasks. In Section 3, we introduce the proposed framework including movement estimation and identification of stopped vehicles. Section 4 includes the data collection, evaluation metrics, and experimental results. In Section 5, we present comprehensive results and provide a detailed discussion through in-depth ablation studies. Finally, Section 6 provides our conclusions and considers future research directions.

2. Related Work

In the last decade, extensive studies have been performed to identify stopped vehicles on the road. Among intelligent object detection tasks related to traffic systems, stopped vehicle detection is a core problem because it is crucial for preventing sequential multiple-collision accidents in safety-critical road applications. The goal of identifying stopped vehicles necessitates key technologies such as object detection on the road, tracking, and speed estimation of observed vehicles. To achieve this goal, three main approaches are being studied: one approach is frame-differencing methods based on background subtraction, the second approach uses feature-based machine learning methods, and the third approach utilizes deep learning-based tracking methods.

The frame-differencing methods based on background subtraction mainly focus on the detection of stationary foreground objects (SFOs) [22]. The foreground (FG) detection algorithm, crucial for SFO detection, identifies objects that differ from the background scene. It primarily operates through background (BG) subtraction as the initial step in motion detection. The process of using BG subtraction to identify FG objects generally involves three steps: BG initialization, FG detection, and BG maintenance. Additionally, there are various methods that employ BG subtraction to detect SFOs. To identify a stopped or obstacle object in these methods, time-based frame differencing is widely used. These methods utilize a two-frame differencing approach, which involves calculating the pixel-wise difference between two consecutive frames [23–25]. However, they struggle with critical issues such as sudden lighting changes and noise, leading to a decline in performance in rapidly changing environments [26].

Prominent stopped vehicle detection methods are based on machine learning models including Support Vector Machine (SVM) and Gaussian Mixture Models (GMMs). In [13], the authors introduced a real-time vision system to monitor road traffic obstacles. They introduced a Selective Update of GMM for extracting forward objects and constructing the background, enabling the algorithm to handle fast-disappearing stationary objects based on a GMM. They also adopted a classification method combining the Histogram of Oriented Gradients (HOG) and SVM to sort obstructions. On the other hand, ref. [12] proposed an efficient tracking method based on calculating background gaps. However, these models are highly dependent on manual feature engineering. For this reason, they struggle to effectively model complex urban road environments.

Deep learning models, particularly those based on convolutional neural networks (CNNs), have made significant and rapid progress in computer vision tasks. Accordingly, image processing in intelligent transportation systems is also advancing along these lines and is increasingly being applied in ADASs and autonomous vehicles. In response, numerous researchers have sought to address stopped vehicle detection using CNN-based object detection frameworks [14–16]. These studies primarily utilize object detection models such as Mask R-CNN [18] and YOLO [19]. Mask R-CNN, in particular, integrates bounding-box regression with object classification, employing end-to-end methods to detect visual objects, which enhances both the detection accuracy and recognition speed. Visual object detection involves both classification and localization. Before the introduction of YOLO, these two tasks were handled separately in visual object detection. However, YOLO simplifies object detection by framing it as a regression problem. Moreover, YOLO adopts an end-to-end neural network architecture for visual object detection, enabling the simultaneous prediction of bounding-box coordinates, target confidence, and class probabilities from a single image input.

The recently proposed YOLO framework is widely used as a one-stage detector method instead of the two-stage methods to enhance computing speed, making it suitable for real-time applications while maintaining acceptable accuracy. However, it faces issues with precision in positioning and detecting fine objects. YOLO has since been upgraded through various versions—YOLOv3 [27], YOLOv4 [28], YOLOv5 [29], and YOLOv8 [30]—to improve its accuracy and speed. For instance, YOLOv5's structure significantly boosts the inference speed to 140 FPS and reduces the weight file capacity by 90%, facilitating its deployment in embedded environments. Although YOLOv5 remains highly effective for many real-time applications, challenges still remain in detecting small objects for autonomous driving. The authors of [31] introduced a YOLOv8-based object detection model specifically tailored for autonomous driving applications. It incorporates features at various scales and utilizes a bidirectional feature pyramid network. To tackle the issue of long-range detection in driving environments, a query-based model with an innovative pipeline structure was proposed.

Multiple-object tracking for stopped vehicle detection is another core method that has garnered significant research attention in recent years. The goal of tracking is to assign a unique ID to each detected object and consistently maintain this ID across consecutive

frames. One of the most successful methods in this domain is Simple Online and Realtime Tracking (SORT) [32], which utilizes Kalman filters [33] and the Hungarian algorithm [34]. SORT is designed for multi-object tracking tasks, efficiently associating detected objects to enable real-time tracking. By incorporating noise and velocity data, SORT predicts the position of objects in subsequent frames based on their current positional information. BoT-SORT [35] enhances performance by integrating improvements in three key areas and combining features from ByteTrack [36]. Unlike previous SORT-like algorithms, which typically use the original seven- and eight-tuple Kalman filter method, BoT-SORT directly computes the width and height of bounding boxes. Also, it accounts for rapid positional changes of bounding boxes in dynamic conditions, as well as environmental factors like wind and vibration in static conditions. It combines motion and appearance metrics to enhance tracking and detection performance.

Most methods for detecting stopped vehicles are tailored for stationary views, such as security camera systems, which consistently monitor the same locations. For instance, ref. [17] proposed a framework for recognizing traffic accidents by utilizing YOLO-based object detection and Deep SORT for object tracking. Since incident detection is performed from a stationary viewpoint rather than from a moving vehicle, it lacks dynamism. These stationary views are limited in their ability to detect objects in blind spots and can be costly to address. Recently, research has increasingly focused on the use of dashcams in vehicles, which provide advantages such as dynamic perspectives and a host-centered viewpoint, making them a critical area of investigation.

For instance, ref. [15] developed a new autonomous vehicle support model using deep learning methods and sensors for object detection in dashcam environments. The process begins with object detection via Mask R-CNN, followed by centroid-based tracking. This framework also estimates speed, tracking path, and acceleration. However, this method relies on bounding boxes to judge vehicle states, such as accidents or stops, and does not incorporate distance factors. Also, ref. [16] proposed a vehicle accident detection model by integrating YOLOv3 with the Canny edge detection algorithm. The model was trained on the HDCA-CS (Highway Dashcam Car Accident for Classification System) dataset, consisting of various dashcam environments. The Canny edge detection algorithm addressed data imbalance issues in the dataset by processing high-resolution images. However, the approach that solely utilizes YOLOv3 for end-to-end classification does not account for distance and lacks motion-related factors such as speed and optical flow. As a result, it has limitations in accurately representing the movements of objects.

Recently, ref. [20] introduced a framework that integrates in-vehicle equipment with road infrastructure to enhance obstacle detection on roadways. The authors utilized monocular cameras with inertial measurement units installed in vehicles, alongside security cameras positioned along the roads, to conduct YOLO-based object detection. However, since security cameras cannot cover all roadways in practice, their application remains inherently constrained. Ref. [21] proposed a methodology for future object localization applicable to egocentric driving videos. The authors aimed to predict the coordinates of future objects using a deep learning model based on an RNN. This method allows for the identification of anomalies, such as accidents or stopped vehicles, when the actual object coordinates differ from the predicted ones. However, the RNN-based coordinate prediction may struggle to determine whether the objects are the same, as it does not involve object tracking. Furthermore, the heavy computational demands of RNNs may render them unsuitable for real-time applications.

On the other hand, ref. [31] modified YOLOv8 to introduce a detection model suitable for autonomous vehicles. The proposed model features a faster inference speed than the original YOLOv8 and has a novel layered architecture specifically designed for the detection of small objects, making it suitable for autonomous driving applications. However, determining the presence of stopped vehicles solely through object detection and tracking, without estimating the speed of vehicles, can lead to inaccurate predictions.

As reviewed above, most recent studies on detecting stopped vehicles employ deep learning models for object detection and tracking. However, the majority of prior research does not account for the movement of the host vehicle. In a dynamic view, such as from a dashcam, simple object detection and tracking alone are insufficient to infer the relative speed of the observed vehicle. When both the host vehicle and the observed vehicles are simultaneously in motion, it is necessary to consider both the speed of the host and the relative movement of the observed vehicle to determine whether it is stopped. To mitigate these issues, we utilize optical flow to estimate the movement of the host vehicle. While the use of deep learning models for object detection and tracking is similar to previous studies, our key distinction is that we further estimate and incorporate the movement of both the observer and the observed vehicles. Additionally, since the size of the observed objects can be calculated differently depending on the viewing angle, such as from the front or side, we estimate the movement based on the position of the observed vehicles. To summarize, we propose a new method to estimate relative movement between the host and observed vehicles using the optical flow method in dynamic views. The proposed stopped vehicle detection method operates effectively from in-vehicle dashcams in real time and demonstrates superior performance when validated with actual road driving environments.

3. Methods

3.1. Overview

Figure 2 depicts the overall framework of the proposed method. The real-time video feed from the dashcam is processed frame by frame, with the computation divided into three primary processes: the main process, a child process for host vehicle movement, and another child process for centerline detection, all executed using a multi-process approach. Within the multi-process pipeline, a shared list is employed to exchange computational results between the processes, enabling the sharing and loading of images across different processes. This integration allows for concurrent execution, enhancing the computational speed and significantly supporting the real-time detection of stopped vehicles.

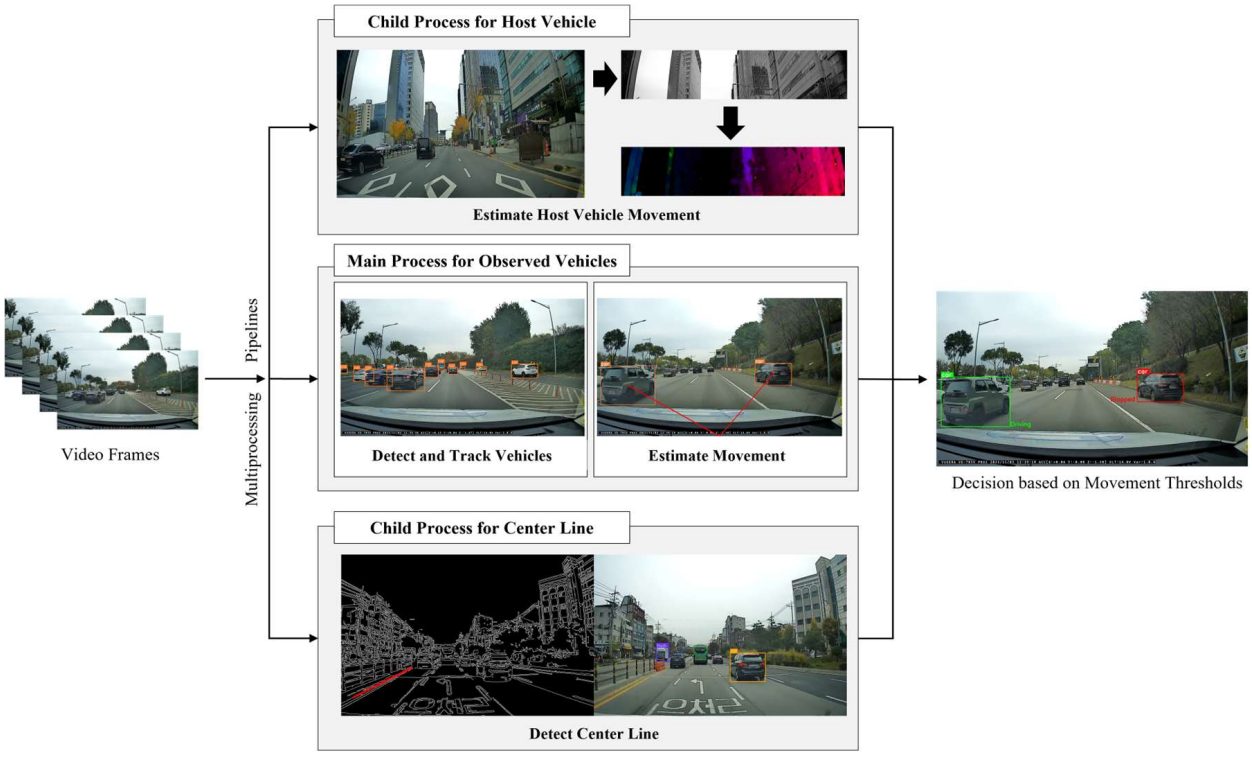


Figure 2. The overall framework of the proposed method.

Each process is assigned specific tasks. The main process focuses on object detection, tracking, speed estimation of observed vehicles, and the identification of stopped vehicles. The host vehicle process calculates the optical flow of the host vehicle, which represents the background dense optical flow. Meanwhile, the centerline process identifies the centerline to exclude vehicles in the opposite lane. Finally, the results from each process are combined to assess whether a vehicle is stopped, with the findings visually represented through images.

Figure 3 provides a more detailed representation of each process in the proposed framework. As will be discussed in the following sections, we use deep learning models to perform object detection and tracking. Simultaneously, we detect the centerline and estimate the speed of the host vehicle. For the tracked vehicles, their positions are analyzed, and their movements are estimated to determine whether they are stationary based on predefined thresholds. In this process, the trainable components, marked as "trainable" in the figure, refer to the deep learning models for object detection and tracking. Since the pre-trained models are sufficient for detecting vehicle objects, no further training was required. Rather than dividing the sections by process, we will explain the methods by categorizing them based on functionality.

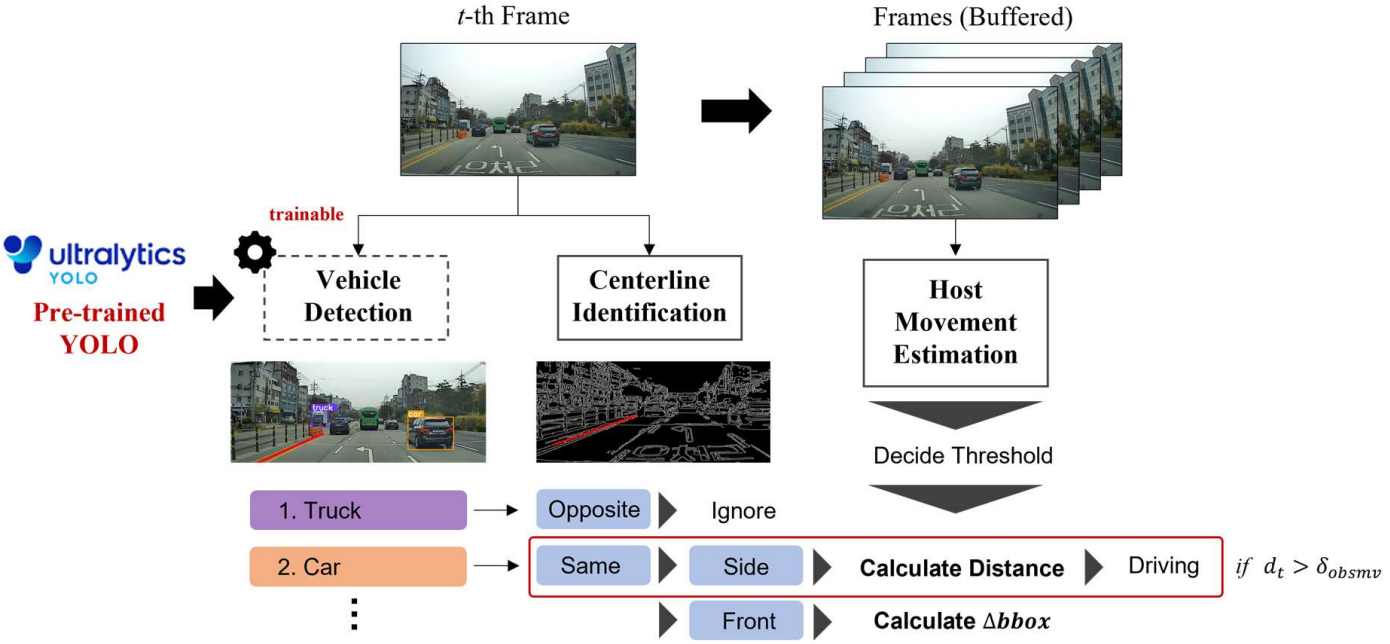


Figure 3. Detailed processes of the proposed framework.

3.2. Detecting and Tracking Observed Vehicles

To begin with, we describe the object detection and tracking processes, which are performed first in the main process. The procedure for detecting and tracking observed vehicles is straightforward. Similar to commonly used tracking frameworks, we first detect vehicles in dashcam video frames using an object detector, and then we track them using a multiple-object tracking model. When detecting objects through the dashcam, we load the video and capture it frame by frame. For object detection, we chose YOLO as our basis, since YOLO has shown powerful performance in terms of accuracy and inference speed. Indeed, YOLO has improved detection accuracy by eliminating duplicated gradient values and enhancing the neck network to boost localization accuracy and transmission performance. These improvements make YOLO suitable for real-time object detection due to its fast inference capability.

In particular, we utilized the pre-trained YOLO models provided by the official Ultralytics website [30]. All YOLO models used for comparison were downloaded from the official website as pre-trained models. Since these YOLO models already demonstrate

high accuracy in detecting vehicles and trucks on the road, no additional training was conducted. However, since the provided YOLO models include a wide range of classes, we configured the models to only detect the "car", "motorcycle", "bus", and "truck" classes for the purpose of identifying stopped vehicles.

Afterward, the detected objects are fed into the tracking model. The object tracking model assigns IDs to detected objects to ensure consistent identification across frames, facilitating effective tracking. We adopted BoT-SORT as our tracking method. BoT-SORT is a well-known tracking method that assigns inherent IDs to tracked objects, distinguishing them accurately. Once an inherent ID is assigned, we maintain each track history for each ID. The track history records the object's coordinates in a list for each frame during tracking. Figure 4 shows an example of the track history records.

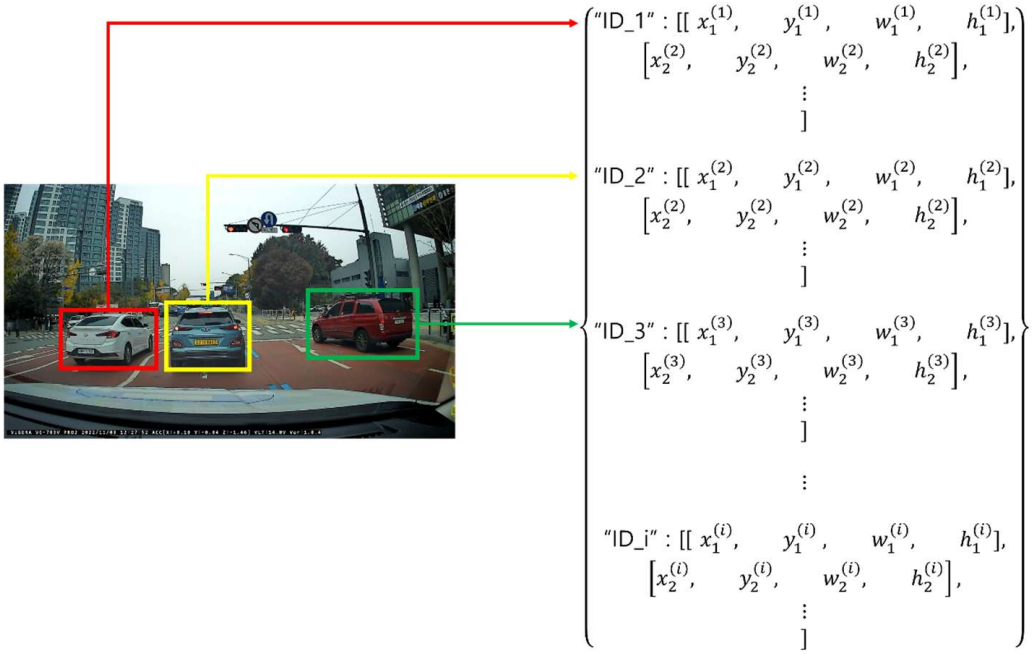


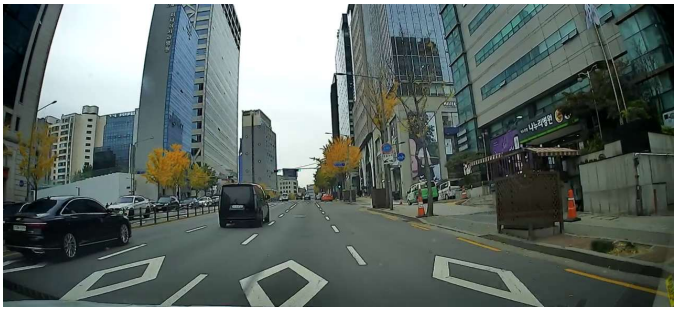
Figure 4. An example of a track history list for the tracked objects.

3.3. Estimating Optical Flow for the Host Vehicle

In this section, we describe the movement estimation of the host vehicle. Note that "host vehicle" denotes the vehicle with the dashcam mounted. We compute dense optical flow of the background to estimate the movement. Optical flow is a technique that computes vector flows between two consecutive frames to detect moving objects. We implement dense optical flow computation based on Gunnar Farnebäck's algorithm [37]. In this algorithm, optical flow can be computed in the grayscale images.

The algorithm utilizes polynomial expansion, a method of signal conversion, to approximate the neighborhood of each pixel using a quadratic polynomial. This polynomial modeling captures the local structural information of each pixel's vicinity, with the coefficients estimated via a weighted least squares method. Displacement estimation involves determining the change in the position of each pixel between two consecutive frames, achieved by contrasting the signal at the same location across different frames. The Gunnar Farnebäck algorithm can provide reliable accuracy, but it can be computationally expensive since it considers every pixel in the frame.

To mitigate this problem, we designate the region of interest (ROI) so that the algorithm only computes optical flow for the host vehicle using the background. We define the ROI above the horizon line so that only the background exists above the region without any observed vehicles. Figure 5 represents an example of the ROI and computed dense optical flow. We highlight dense optical flow based on an HSV color map for clear visualization.



(**a**) Original frame.



(**b**) Region of interest in grayscale image.



(**c**) Visualized dense optical flow.

Figure 5. Overall procedure of dense optical flow computation.

In the process of optical flow computation, we tailor the dense optical flow algorithm to align with our specific objectives after setting up the ROI section. Firstly, we check the optical flow of the ROI section using the Farnebäck algorithm, which represents optical flow in orthogonal coordinates $(u,\ v)$. However, for detecting stopped vehicles, we require polar coordinates (r, θ). Therefore, we use the *cv2.cartToPolar* function from the OpenCV Python library to perform this conversion. This function switches the coordinates using Equations (1) and (2):

$$r = \sqrt{u^2 + v^2} \tag{1}$$

$$\theta = atan2(u,\ v) \tag{2}$$

where $u,\ v$ denotes the x and y coordinates of the heading direction, r denotes the magnitude of the vector, and θ denotes the angle of the vector.

To eliminate noise from small movements, we set a threshold (δ_{mv}) for the magnitude r and retain values greater than δ_{mv}. We then analyze the remaining values to determine the vehicle's heading. Since each coordinate has its own individual value, it is challenging to ascertain the object's movement direction, so we use the median of the remaining values.

Also, the angle (θ) returns values corresponding to the magnitude as degrees, which we rearrange into new classes to define the host vehicle's direction of movement. Based on θ, we want to classify the vehicle's movement direction into the five categories: driving (backward, right, left, or forward) and stopped. We use the median of the angles of pixels with magnitudes exceeding the threshold δ_{mv} as a current direction of the host vehicle.

However, these r and θ values tend to fluctuate significantly in a moving vehicle, due to external factors such as reflections and vehicle vibrations. Therefore, we store several directions into a buffer memory and decide the current direction by majority vote. We define the buffer as a direction map. The direction map is a list initialized with zero values. The algorithm stores the direction of each frame in the direction map in real time, requiring continuous updates to the map. To achieve this, we employ a rolling method to update the direction map. This method operates over a five-frame period. For example, when the sixth frame is processed, the first frame is removed, and the sixth frame is appended after the fifth frame. While the rolling method is executed, the algorithm counts the direction that is most frequent over the frame period and determines that the top-ranked direction represents the host vehicle's movement direction.

3.4. Estimating Movement of Observed Vehicles

After finding the optical flow of the host vehicle, we estimate the speed of the observed vehicles. Before employing this method, we reset the zero-point to the center-bottom of the image. Previously, the zero-point was set to the upper-left corner, which made it difficult to calculate the observed vehicle's movement. In most cases, we estimate the magnitude of the observed vehicle's movement using the Euclidean distance between the present and past center coordinates of the observed vehicle. For direction, we calculate the difference between the previous and current y-coordinates, representing the observed

vehicle's movement direction. These movements can be calculated between two consecutive frames, and the difference may not be significant. Therefore, we calculate movement between frames separated by h intervals. The movement (d_t^i) of the i-th tracked object can be estimated following Equations (3) and (4) using its center points (c_t, c_{t+h}). We omit superscript i in the equations for better readability.

$$d_t = Euclidean(c_t, c_{t+h}) \tag{3}$$

$$= \sqrt{(x_t - x_{t+h})^2 + (y_t - y_{t+h})^2} \tag{4}$$

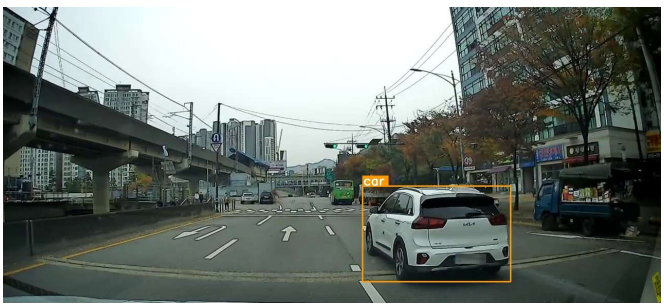
where x_t and y_t denote the object's center x and y coordinate values in the t-th frame, respectively, c denotes the observed vehicle's center coordinate value, and h denotes the intervals between frames.

When the observed vehicle is directly in front of the host vehicle, the movement distance in the image may be minimal despite the vehicle being in motion, making the abovementioned distance meaningless. Figure 6 illustrates cases where the observed vehicle is either directly in front of or beside the host vehicle. Therefore, we propose an alternative movement estimation metric specifically for vehicles in front. To implement this, the first step is to identify the front vehicle. We determine that the observed vehicle is in front when it is positioned orthogonally to the center of the host vehicle's dashcam. However, to ensure a more flexible determination, we set a soft margin of degrees (e.g., 80 to 110 degrees) when assessing orthogonality. When the observed vehicle is located directly in front of the host, we estimate movement d_t^i by using the difference in object bounding-box sizes ($bbox$) between the previous and current frames of the observed vehicle (Equations (5) and (6)). We omit superscript i (notation for object number) in the equations.

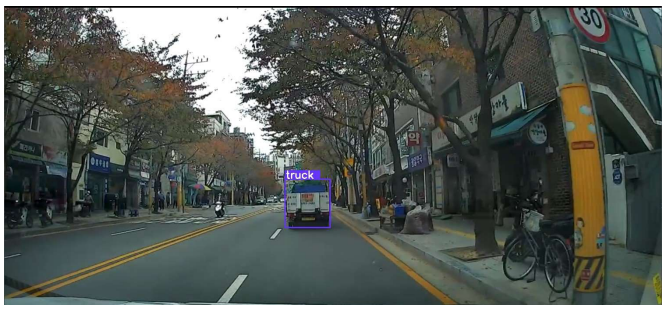
$$d_t = \Delta bbox_{t,\ t+h} \tag{5}$$

$$= bbox_{t+h} - bbox_t \tag{6}$$

where d_t denotes the observed vehicle's movement, and $bbox$ denotes the size of the bounding box computed using width and height (size).



(**a**) Positioned to the side.



(**b**) Positioned in front.

Figure 6. Possible locations of observed vehicles.

However, calculating vehicle movement for every frame can lead to misjudgments due to low inter-frame displacement. Thus, we do not estimate movement for every frame; instead, we calculate the object's movement after a certain period of time has elapsed. Additionally, when the object is too small, it indicates that the object is far from the host vehicle, which can lead to underestimation of movement and large errors. To address this, we filter out small objects by using a bounding-box size threshold.

Furthermore, the estimation method differs depending on whether the observed vehicle is in front of the host vehicle. For the front vehicle, we calculate movement using the difference in bounding-box size between sequential frames and the change in the center

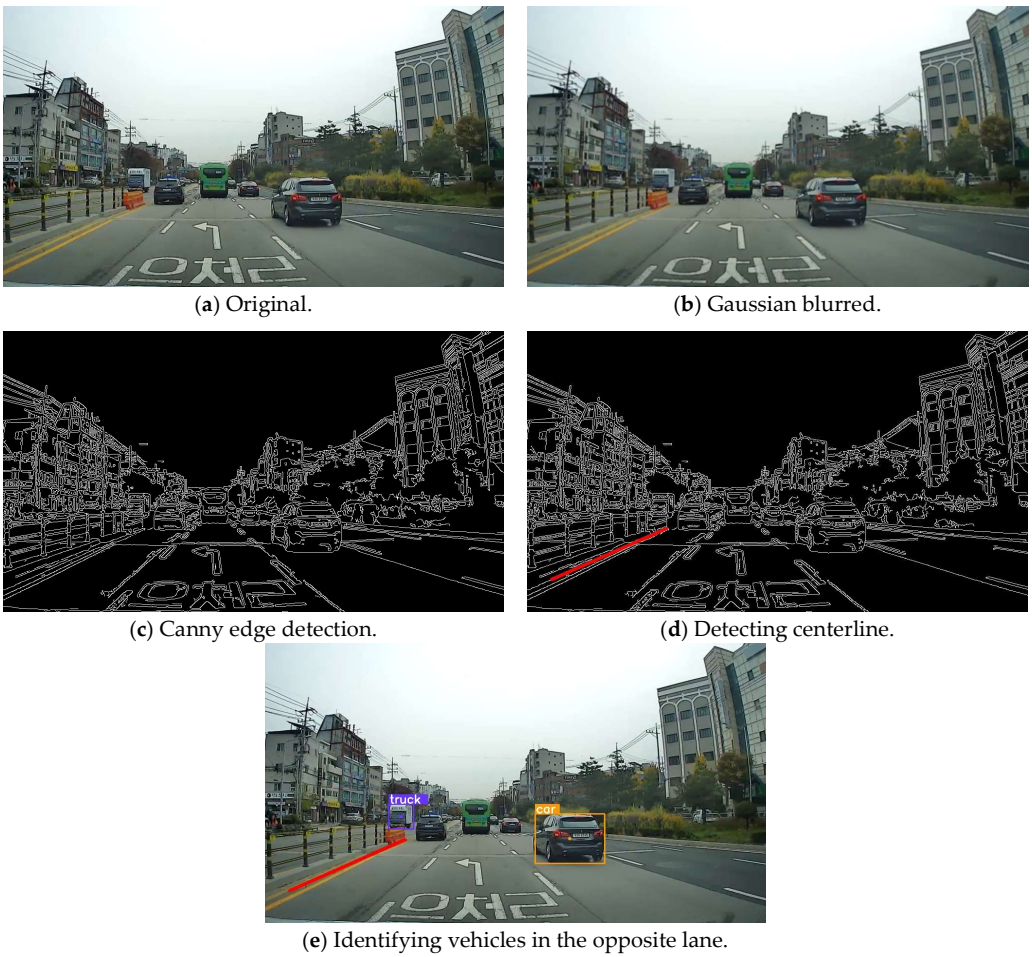
coordinates otherwise. Due to the different modality of these two metrics, we employ different thresholds and methods for judging whether the observed vehicle is stopped.

3.5. Detecting Centerline

In this section, we describe a method for detecting the centerline of the road. The purpose of detecting the centerline is to distinguish between vehicles traveling in the same lane as the host vehicle and those traveling in the opposite lane. Since our stopped vehicle detection algorithm relies on the optical flow or movement of the host vehicle and the observed vehicle, it is not suitable for vehicles moving in the opposite lane. Therefore, vehicles traveling in the opposite lane are excluded from consideration.

The key to lane separation using the centerline involves edge detection and line detection in the image, along with filtering based on the color of the centerline. The edge detection starts with applying Gaussian blur and Canny edge detection [38] to the dashcam images. After detecting edges, we employ the Hough transformation [39] for line detection and then filter the detected lines based on the predefined HSV color maps for the centerline. We developed the abovementioned centerline detection algorithm using OpenCV library.

After the centerline is identified, we extend it as an imaginary straight line to determine the lane based on whether the central coordinates of the detected objects are above or below this line. Any vehicle detected above the centerline is considered to be in the opposite lane, while objects below the centerline are deemed to be in the same lane as the host vehicle. The overall process of centerline detection and classifying the lanes of vehicles is illustrated in Figure 7.



(**a**) Original. (**b**) Gaussian blurred.

(**c**) Canny edge detection. (**d**) Detecting centerline.

(**e**) Identifying vehicles in the opposite lane.

Figure 7. Overall procedure for detecting the centerline (red) and filtering vehicles in the opposite lane.

Figure 7b,c show that applying Gaussian blur results in smoother Canny edges. Figure 7d demonstrates that detecting candidate lines using the Hough transformation and then filtering them based on the color map accurately identifies the centerline. The identified centerline is marked in red. Our method filters the candidate lines detected by the Hough transformation using a predefined HSV color map corresponding to the centerline. Through exploratory data analysis of the images for the object detection model, we predetermined that the HSV values corresponding to the centerline were in the yellow range. In Figure 7e, the purple truck (left top) is considered to be in the opposite lane since its coordinates are above the centerline, while the orange car (right bottom) can be judged to be in the same lane as the host vehicle since its coordinates are below the centerline.

3.6. Identifying Stopped Vehicles

Once the estimation of all movements of the host and observed vehicles is completed, the main process merges them and determines the status ($State_t^i$) of the i-th observed vehicle at frame t, i.e., whether the observed vehicle is stopped or driving. This process can be separated into two cases: the host vehicle is stopped, or the host vehicle is driving.

Firstly, the case where the host vehicle is stopped is straightforward. If the calculated object's motion (d_t^i) is lower than a specific threshold (δ_{obsmv}), we consider the object to be stopped (Equation (7)). For the threshold (δ_{obsmv}), we use the same value as the host vehicle's movement judging threshold (δ_{mv}) in the experiments.

$$State_t^i = \begin{cases} Stopped, \quad if\ d_t^i < \delta_{obsmv} \\ \quad (obs = \text{“}Stopped\text{”}) \\ \\ Driving, \quad otherwise \\ \quad (obs = \text{“}Driving\text{”}) \end{cases} \tag{7}$$

where $State_t^i$ denotes the status of the i-th observed vehicle at frame t.

When the observed vehicle is in front, we estimate the movement of observed objects using size of the bounding box. The approach to estimating the movement of observed vehicles depends on the position of the objects relative to the host vehicle. For vehicles directly in front, movement is estimated by analyzing the variation in the bounding-box size across successive frames, whereas for vehicles in other positions, movement is determined by tracking changes in the center coordinates. Due to the differing nature of these metrics, separate thresholds and assessment techniques are used to ascertain whether the observed vehicle is stopped. If the calculated value is lower than δ_{size}, we consider the vehicle to be driving. Conversely, if the calculated value is higher than δ_{size}, we consider the vehicle to be stopped (Equation (8)).

$$State_t^i = \begin{cases} Stopped, \ \ if\ d_t^i < \delta_{size}, \\ \quad (obs = \text{“}Stopped\text{”}) \\ \\ Driving, \quad otherwise \\ \quad (obs = \text{“}Driving\text{”}) \end{cases} \tag{8}$$

Secondly, the scenario in which the host vehicle is driving forward is more complicated compared to the first case. We address this problem by categorizing it into two possible cases: (i) the observed vehicle is in front, or (ii) the observed vehicle is moving backward in the frame, contrary to the host vehicle's direction. We use the movement direction of objects, which is computed as described in Section 3.3, to classify which of the two cases the observed vehicle falls into. Note that if the observed vehicle is moving forward within the frame, the object is obviously driving forward.

For the first case, where the observed vehicle is in front, the calculation is similar to when the host vehicle is stopped. If both the observed vehicle and the host vehicle are in motion, a problem arises due to their relative velocity. In this case, the observed vehicle

appears to move backward in the video since its relative speed is lower than that of the host vehicle. Additionally, as the vehicle draws closer to the host vehicle, the size of the object will also increase. Based on this observation, we estimate the movement of vehicles located in front of the host vehicle using changes in the size of the object's bounding box rather than distance. If this change becomes larger than the threshold δ_{size}, we can conclude that the observed vehicle is stationary. Therefore, we determine that the vehicle is stopped if the size of the bounding box for the front vehicle increases rapidly beyond the threshold, as described in Equation (9):

$$State_t^i = \begin{cases} Stopped, \ \ if\ d_t^i > \delta_{size}, \\ \quad (obs = \text{“}Stopped\text{”}) \\ \\ Driving, \quad otherwise \\ \quad (obs = \text{“}Driving\text{”}) \end{cases} \tag{9}$$

The second case involves the observed vehicle moving backward, so the object will appear to be gradually approaching the host vehicle. As with the first case, the relative speed of the observed vehicle will be slower than the host since the object is either slowing down or has already stopped. Moreover, the lower the actual speed of the observed vehicle, the faster it seems to approach the host vehicle in the video, or in other words, the faster it appears to move backward. Based on this observation, we deem that a vehicle is stopped if it moves backward at a speed exceeding a threshold δ_{dist} (Equation (10)). Although vehicles in the opposite lane always appear to move backward rapidly, we only assess the status of vehicles in the same driving lane as the host, thus excluding vehicles in the opposite lane from consideration.

$$State_t^i = \begin{cases} Stopped, \ \ if\ d_t^i > \delta_{dist}, \\ \quad (obs = \text{“}Stopped\text{”}) \\ \\ Driving, \quad otherwise \\ \quad (obs = \text{“}Driving\text{”}) \end{cases} \tag{10}$$

In summary, we employ a metric based on the distance and a metric based on the bounding-box size difference depending on the location of the observed vehicle. Since these two metrics have different modalities, they operate with different thresholds.

The procedure for identifying stopped vehicles is summarized in Figure 8.

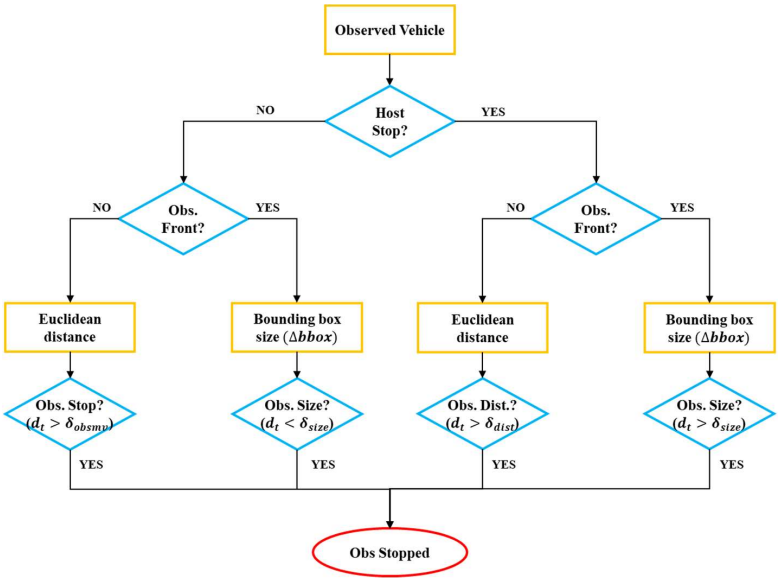


Figure 8. Summarized flowchart of stopped vehicle identification.

4. Experiments and Results

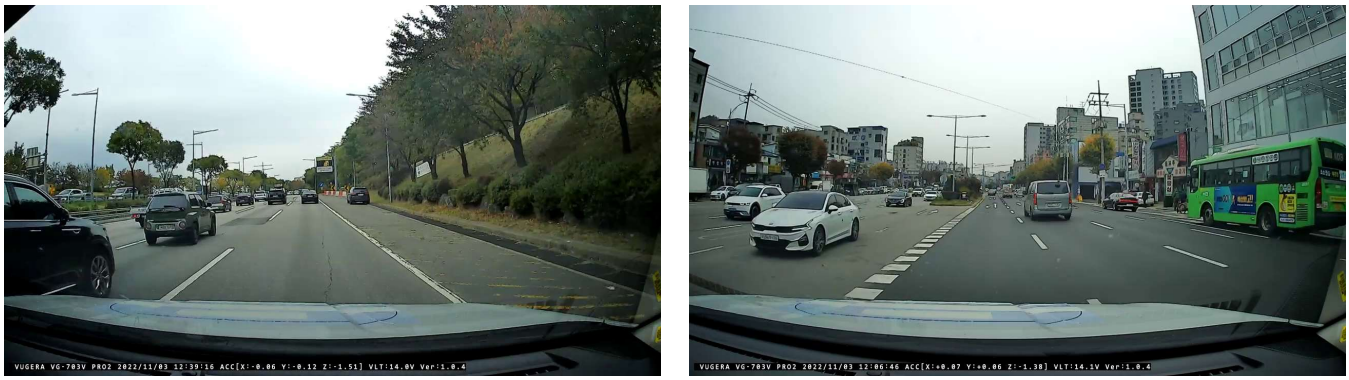
4.1. Datasets

We produced the dataset collected from our experimental vehicle's dashcam while driving through Busan, Republic of Korea, from November 2nd to November 3rd. We extracted 14 videos containing stopped vehicles and resized the footage to 1280 × 720 pixels. The labeling tool that we used was Microsoft's VoTT (Visual Object Tagging Tool, version 2). For stopped vehicles, we annotated the label as 1 using the labeling tool on our footage. The details of the datasets are shown in Table 1.

Table 1. Statistics of datasets.

#	Length	# of Frames	# of Stopped	# of Labeled
1	0:37	1098	9	173
2	0:38	1141	25	479
3	0:37	1104	4	112
4	0:36	545	10	214
5	0:34	1012	4	127
6	0:34	516	2	37
7	0:46	699	3	295
8	0:37	536	1	32
9	0:34	1044	4	130
10	0:34	1020	7	141
11	0:45	1352	1	28
12	0:41	624	6	170
13	0:47	1141	8	630
14	0:38	1139	23	358

In the domain of image-based object detection tasks, typically, a single label is annotated per object. However, in the context of video-based tasks, multiple labels were affixed to a single object, resulting in approximately 20 to 30 labeled images for each vehicle. Samples from this dataset are depicted in Figure 9.



(**a**) Example of a stopped vehicle (right road shoulder). (**b**) Example of urban road conditions.

Figure 9. Samples of the collected datasets.

4.2. Experimental Settings

We implemented the framework using Python3.10, Ultralytics [30], and OpenCV. For the YOLO models, we utilized pre-trained weights from the official website [30] and did not perform fine-tuning. Since the pre-trained YOLO models are designed to detect a wide range of object classes, we reconfigured the models to focus solely on detecting the "car", "motorcycle", "bus", and "truck" classes in order to accurately identify stopped vehicles. The inference was performed using the PyTorch 2.1 deep learning framework and employed an NVIDIA GeForce RTX 3090 Ti GPU.

Additionally, we selected the necessary hyperparameters (thresholds) for the framework through grid search. We applied the same grid of values {3, 5, 8, 10, 13, 15, 18, 20} to all three thresholds (δ_{obsmv}, δ_{size}, and δ_{dist}), selecting the optimal thresholds based on the highest F1 scores. A detailed comparison is provided in Section 5.3.

To evaluate the performance of our proposed framework, we chose widely used metrics for object detection tasks: precision, recall, mAP, and F1 score (Equations (11)–(14), respectively). Also, we used frames per second (FPS) to evaluate whether the framework is suitable for real-time applications.

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

$$mAP = \frac{1}{C}\sum_{K=i}^{N} P(k)\Delta R(k) \tag{13}$$

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{14}$$

4.3. Main Results of Identifying Stopped Vehicles

In this section, we evaluated our framework using 14 sets of video footage with four evaluation metrics. Table 2 reports the results for each video per metric. The average values for each indicator—precision, recall, mAP, and F1 score—were 0.8691, 0.8543, 0.7869, and 0.8550, respectively. The mAP values can show relatively low numerical values compared to other indicators due to the discrepancy between YOLO's bounding boxes and our manually labeled bounding boxes.

Table 2. Results of identifying stopped vehicles in the collected datasets.

#	Precision	Recall	mAP	F1 Score	FPS
1	0.8222	0.7048	0.6223	0.7590	35.84
2	0.7850	0.8760	0.7156	0.8280	26.23
3	0.7778	0.9978	0.8021	0.8742	27.15
4	0.9798	0.9327	0.9179	0.9557	26.08
5	0.7500	0.9996	0.8361	0.8570	23.20
6	0.9986	0.8895	0.8895	0.9409	24.68
7	0.8152	0.6276	0.6129	0.7092	24.77
8	0.9978	0.9991	0.9848	0.9984	22.26
9	0.8760	0.9899	0.8981	0.9295	23.95
10	0.8243	0.6100	0.5980	0.7011	23.87
11	0.9819	0.9997	0.9848	0.9907	24.78
12	0.9892	0.7419	0.7402	0.8479	24.55
13	0.7343	0.7932	0.6875	0.7626	25.30
14	0.8351	0.7980	0.7267	0.8161	23.93
Average	0.8691	0.8543	0.7869	0.8550	25.47

In Table 2, our framework shows consistently high precision across all videos. We excluded vehicles in the opposite lane, leading to a reduction in the number of objects that the model evaluated, which, in turn, caused high precision performance. The F1 score also showed high performance due to the high precision and recall values, indicating that the model minimizes errors and provides reliable results.

Without doubt, the recall score is more important than precision in the context of safety. From this perspective, our framework performs well, with a high recall score of 85.43%. Moreover, the FPS of the proposed framework demonstrates commendable outcomes, with

an average FPS of 25.47, which is similar to the input footage's FPS. This negligible decline in frame rates indicates that our system maintains a high level of real-time performance.

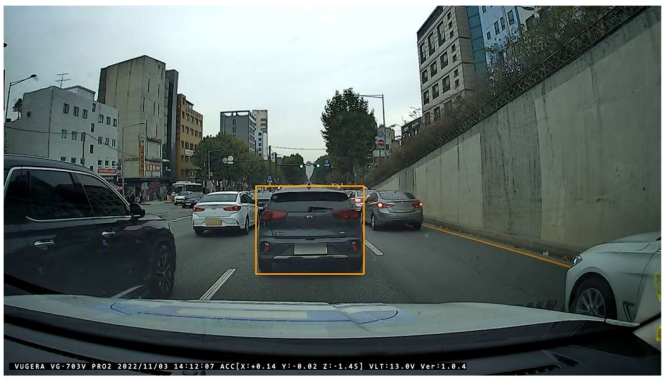
5. Discussion

5.1. Qualitative Analysis

5.1.1. Observed Vehicle in Front of Host Vehicle

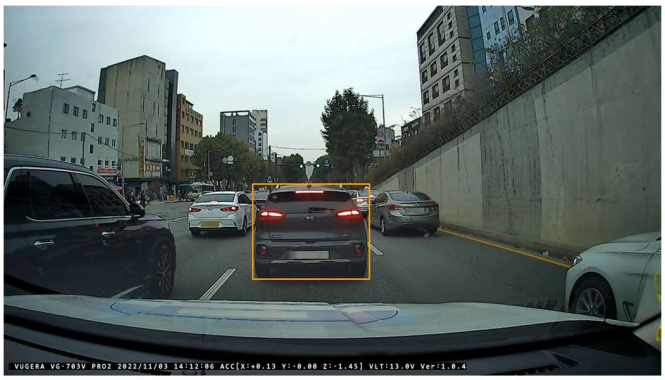
To validate the effects of movement estimation methods, we conducted an additional ablation study with varied metrics for the movement of observed vehicles directly in front of the host vehicle. In cases where the observed vehicle was directly in front of the host vehicle, we computed changes in the bounding boxes to measure the movements. We determined the status of the observed vehicles by comparing the size of the bounding boxes, which differs from the Euclidean distance calculation used for objects located alongside the host. Vehicles adjacent to the host vehicle are usually classified as stopped based on movement distance. Conversely, for vehicles in front, we should not use the movement distance to determine the vehicle's status.

Figure 10 illustrates an example of a vehicle gradually coming to a stop in front of the host vehicle. For the front (center) vehicle, we compared the two metrics mentioned above: the change in distance and the change in box size.



Center point $(x, y, w, h) = (595, 436, 210, 170)$

(**a**) Driving in front of host.



Center point $(x, y, w, h) = (597, 439, 227, 184)$

(**b**) Stopped in front of host.

Figure 10. An example of a stopped vehicle in front of the host vehicle.

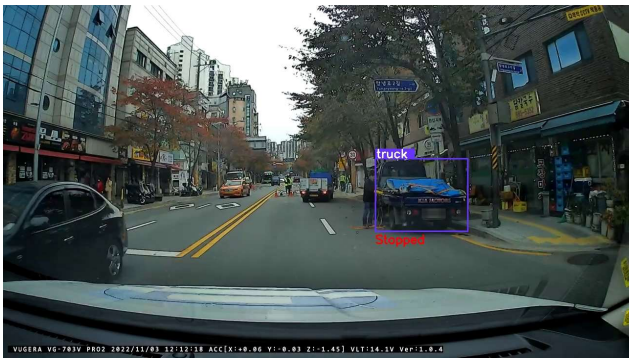
The center coordinates (x, y) of the object representing the front vehicle while it is in motion are (595, 436), and the size of the bounding box (height, width) is (227, 184). Similarly, when the vehicle has stopped, the detected object's center coordinates and size (x, y, w, h) in the image are (595, 436, 210, 170). Using the object detection results from the two frames, the movement distance (d_t^i) based on the movement of the center coordinates was calculated as $d_t^i = 3.61$ using Equation (3). On the other hand, after recognizing that the vehicle is in front, the metric based on changes in the bounding-box size for the two objects was calculated as $d_t^i = 6068$ using Equation (5).

In this case, the criterion for determining a stopped vehicle in the proposed framework is based on exceeding the thresholds, as specified in Equations (9) and (10). This is grounded in the simple theory that a vehicle coming to a stop relative to the host vehicle will appear to move backward in the frame due to its lower relative speed. In practice, even though the vehicle is stopped, the distance-based metric results in a small value, leading to the incorrect conclusion that the vehicle is not stopped. However, the metric based on the change in the bounding-box size calculates a significantly large value, allowing the system to correctly identify the vehicle as stopped. Therefore, for the observed vehicles in front of the host, using the change in the bounding-box size is proven to be highly effective in determining whether the object is stopped.

5.1.2. Observed Vehicle Beside the Host Vehicle

To evaluate the overall performance of the proposed framework, we also investigated other scenarios where the observed vehicle was positioned in locations other than directly in front of the host vehicle. In the collected dataset, possible locations of stopped vehicles included vehicles located on the roadside next to the host vehicle and vehicles stopped at traffic signals, either in front or to the side. Vehicles positioned to the side were assessed for stationarity based on Equations (3) and (10). For vehicles located in front, the determination was based on Equations (5) and (9). When stopped vehicles both at the side and in front were present within a single frame, each metric was selected and calculated based on the location of each observed vehicle.

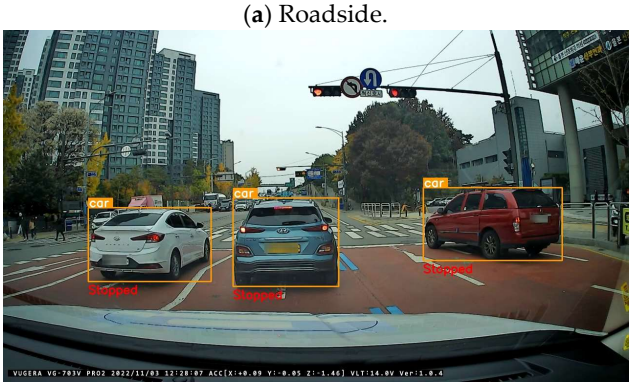
Figure 11 shows examples of detected results using the proposed framework, demonstrating that the proposed framework can be effectively applied to a variety of moving objects on the road, including trucks and motorcycles, rather than just standard vehicles. In Figure 11a, a truck temporarily stopped on the roadside was accurately detected, while Figure 11d shows the detection of a motorcycle waiting at a red light. Additionally, even when the host vehicle is driving on a highway, the framework accurately detects vehicles stopped on the shoulder, as shown in Figure 11b. Figure 11c,d illustrate that vehicles stopped both in front of and to the side of the host vehicle can be precisely identified using each metric based on the location of objects in an urban road environment. Notably, Figure 11d shows that the framework can accurately determine whether a taxi (bottom right) is stopped or not, even when the bounding box has been cropped. This is why the framework employs the distance-based metric of the centroid to measure movements.



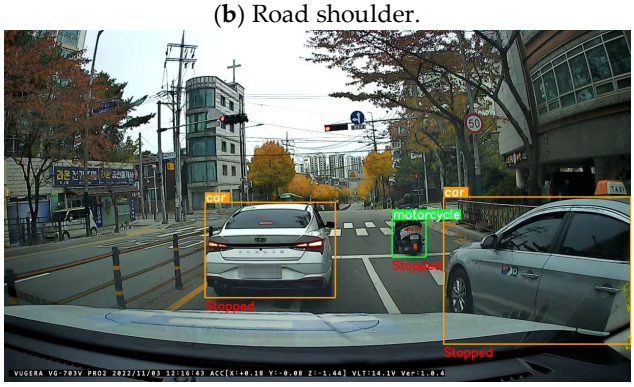
(**a**) Roadside.



(**b**) Road shoulder.



(**c**) Traffic signal.



(**d**) Traffic signal with cropped bounding box.

Figure 11. Examples of detected stopped vehicles.

5.1.3. Obscured Vehicles

The proposed framework determines whether an object is stopped based solely on dashcam video footage, without the use of additional sensors like LiDAR. As a result, it inherits common challenges associated with image-based computer vision processing, such as image distortion, light reflection, and occlusion. This is especially prevalent in road

environments where multiple vehicles are moving simultaneously, making it inevitable for some vehicles to be partially obscured by others. Figure 12 illustrates an example of a stopped vehicle being obscured by a nearby vehicle. In the figure, a large bus blocks the view of the stopped vehicle (right) parked on the roadside.



(**a**) Bus blocking the stopped vehicle. (**b**) Stopped vehicle after a few frames.

Figure 12. An example of an obscured vehicle.

The proposed stopped vehicle detection framework begins with detecting and tracking vehicles using a YOLO detector. In cases where a vehicle is obscured in the given frame, as shown in the previous example, the framework cannot detect stopped vehicles on the roadside.

While this limitation can be seen as a drawback of our framework, it is an inherent challenge that cannot be overcome without incorporating additional sensors. In particular, implementing a surrounding view through the installation of multiple cameras for perspective mapping would help to minimize blind spots, thus aiding in the detection of hidden vehicles that have come to a stop. For hidden vehicles, even precise sensors like LiDAR may not provide significant assistance. However, when a previously concealed vehicle comes into view of the sensors, it can be detected and responded to immediately.

Nevertheless, as shown in Figure 12b, the vehicle previously obscured by the bus reappears in the frame and becomes trackable after a few frames. Therefore, while immediate tracking is not possible, the framework is ultimately able to detect the stopped vehicle since the moving bus quickly passes the stationary vehicle.

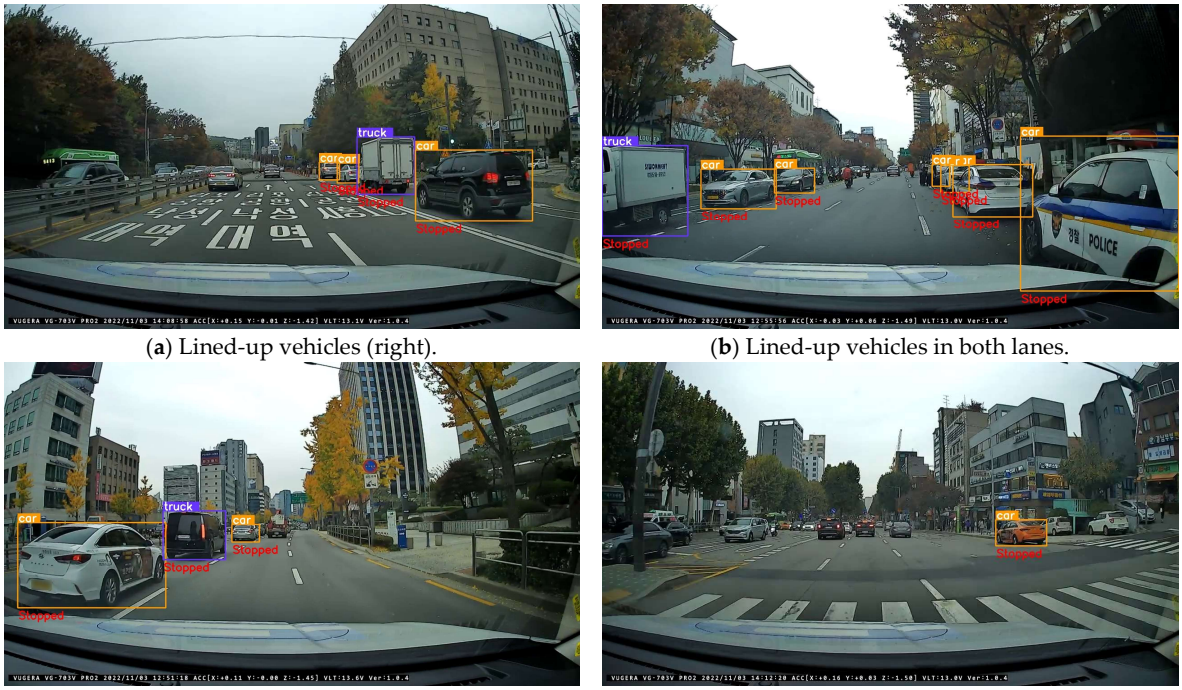
5.1.4. Extreme Road Conditions

In this section, we investigated the performance of identifying stopped vehicles in extreme road conditions. In smart transportation systems, effectively managing extreme situations is crucial. While our model demonstrates strong detection capabilities for stopped vehicles under typical conditions, its ability to maintain high performance in more challenging scenarios, such as traffic congestion, remains uncertain. Evaluating its performance in such extreme conditions will be essential to confirm the robustness of the framework.

Figure 13 illustrates examples of detected stopped vehicles in extreme road conditions. The results for an image of vehicles waiting at a signal, as shown in Figure 11c,d, already demonstrate the accurate identification of stopped vehicles. Therefore, we present the results for scenarios where vehicles are continuously stopping due to road congestion, where vehicles in the opposite lane are stationary, and where many vehicles are present in the frame.

Figure 13a shows that the model accurately identifies vehicles lined up on the right as stopped. In Figure 13b, the model detects several police cars parked on the right and accurately recognizes the vehicle partially obscured by people in front of the police cars as stopped as well. However, it also detects vehicles waiting for a U-turn on the left, across the centerline, as stopped. Although these vehicles are indeed stationary, they should not be included in the detection since they are in the opposite lane. These results occurred because the road's centerline was not accurately recognized due to the distortion in the image's field

of view and the color map. Nevertheless, all of the vehicles stopped on the right side were accurately detected. In Figure 13c, all lined-up vehicles were correctly detected as stopped. In Figure 13d, despite the presence of many vehicles in the frame, the model successfully identified the taxi on the right edge as stopped. These qualitative analyses confirm that the proposed framework is robust enough to accurately detect stopped vehicles across various road environments.



(**a**) Lined-up vehicles (right). (**b**) Lined-up vehicles in both lanes.

(**c**) Lined-up vehicles (left). (**d**) Congested with vehicles.

Figure 13. Case analysis for extreme road conditions.

5.2. Centerline Detection

5.2.1. Performance of Centerline Detection

In this section, we conducted an additional experiment to evaluate the framework's performance in detecting the centerline. The proposed framework detects the road's centerline and ignores vehicles driving in the opposite lane. Since the framework determines whether a vehicle is stopped based on its relative motion, vehicles that appear to approach the host vehicle rapidly may be mistakenly identified as stopped. Thus, it is crucial for the algorithm to exclude vehicles from the opposite lane. However, before applying this method, the high performance of the centerline detection module must be ensured. To validate this, we examined the centerline detection accuracy on subsets of the acquired datasets that included road centerlines.

Table 3 presents the centerline detection results for video subsets containing centerlines. Based on the F1 scores, the centerline detection accuracy was approximately 90% on average, demonstrating high performance. However, some sets showed lower scores. For example, sets 6 and 10 displayed lower accuracy. This is reflected in the main results of Table 2, where the detection performance of stopped vehicles in sets 6 and 10 is comparatively low. Notably, set 10 exhibited the poorest stopped vehicle detection performance across all datasets. These findings suggest that accurate centerline detection significantly impacts the performance of the stopped vehicle detection framework. This is to be expected, as the framework tends to classify vehicles approaching the host vehicle as decelerating or stopped. Nevertheless, for the majority of the datasets, the framework shows high centerline detection performance, which, in turn, contributes to the overall high stopped vehicle detection accuracy.

Table 3. Performance results of centerline detection.

#	Precision	Recall	F1 Score
2	0.8858	0.9746	0.9280
3	0.9808	0.9299	0.9547
4	0.9861	0.9725	0.9793
5	0.9348	0.9585	0.9465
6	0.9153	0.6702	0.7738
10	0.9617	0.5102	0.6667
12	0.9919	0.9838	0.9878
14	0.9716	0.9567	0.9641
Average	0.9535	0.8696	0.9001

We further investigated the reasons behind the lower performance in detecting the centerline for videos 6 and 10, despite achieving high accuracy in most cases. The frames extracted from these two videos are shown in Figure 14 below. In the case of video 6 (Figure 14a), the centerline detection module failed to detect the short yellow line in the middle because it did not meet the minimum length criterion set by the module. For video 10 (Figure 14b), the host vehicle was driving on the far right, and as a result, the centerline on the far left was not captured by the predefined yellow HSV color map. Future research could explore centerline detection methods that dynamically respond to color variations, which could further improve detection performance. Nevertheless, in most cases, as shown in Figure 14c,d, the centerline was accurately detected, and excluding videos 6 and 10, the framework achieved an average F1 score of 96%.



(**a**) Short centerline. (**b**) Color distortion.

(**c**) Detected centerline 1. (**d**) Detected centerline 2.

Figure 14. Case analysis of centerline detection.

5.2.2. Effect of Centerline Detection

In this section, we performed an ablation study to validate the effect of detecting the centerline and filtering out vehicles in the opposite lane in the proposed framework. When detecting stopped vehicles in a dashcam environment, we only detect vehicles traveling in the same direction as the host car. However, in the real-world road environment, vehicles can be both in the same driving direction as the host car and on the opposite side of the centerline. The vehicles in the opposite lane can easily be confused with vehicles

moving forward and backward, degrading the performance of the stopped vehicle detection framework. When the host vehicle is moving forward, we filter out vehicles on the opposite side of the centerline. Filtering is performed using a centerline detection algorithm, which identifies vehicles on the left side of the centerline as opposite-lane vehicles. To validate the proposed method, we detected stopped vehicles in the datasets with and without the centerline detection.

Table 4 reports the results of performance differences without applying the centerline detection algorithm using the collected videos. Also, "w/o cline" denotes results that do not include centerline detection, while "w/cline" denotes results that include it. We estimate whether vehicles are stopped based on their movement distance.

Table 4. Results of stopped vehicle detection with and without centerline detection.

#	Precision	Recall	mAP	F1 Score	FPS
1	0.7611	0.6935	0.5982	0.7257	30.06
2	0.6533	0.9420	0.6412	0.7715	24.12
3	0.5223	0.9987	0.5369	0.6859	24.07
4	0.7527	0.9272	0.7850	0.8309	24.62
5	0.5300	0.9060	0.6163	0.6688	26.30
6	0.8350	0.9430	0.8946	0.8857	32.30
7	0.8386	0.7571	0.7326	0.7958	26.24
8	0.5614	0.9847	0.7399	0.7151	23.91
9	0.6772	0.9817	0.7306	0.8015	24.17
10	0.7264	0.7700	0.6812	0.7476	24.33
11	0.9686	0.9887	0.9869	0.9785	25.38
12	0.9896	0.7661	0.7644	0.8636	26.74
13	0.5898	0.8724	0.6520	0.7038	26.95
14	0.5890	0.9400	0.6518	0.7242	25.07
Avg (w/o cline)	0.7162	0.8928	0.7161	0.7785	26.02
Avg (w/cline)	0.8691	0.8543	0.7869	0.8550	25.47

In Table 4, it can be seen that the F1 score dropped sharply (about 9%) after eliminating centerline detection and filtering out opposite-lane vehicles. The original proposed framework results in an average of 0.8550, while the performance without centerline detection shows an average of 0.7785. For vehicles moving in the opposite direction, the movement direction will always be backward, making it difficult to determine whether the vehicle is stopped. Indeed, we judge an observed vehicle as stopped based on Equation (10), which the vehicles in the opposite lane always satisfy. Consequently, the algorithm estimates all opposite-lane vehicles as stopped, resulting in extremely low precision. The recall score decreased slightly when ignoring vehicles in the opposite lane. However, if the model states that all vehicles in the opposite lane are stopped, it is evident that the recall scores should be high due to the precision–recall trade-off. From this result, we can confirm that detecting the centerline and filtering out the vehicles in the opposite lane play an important role in the proposed framework.

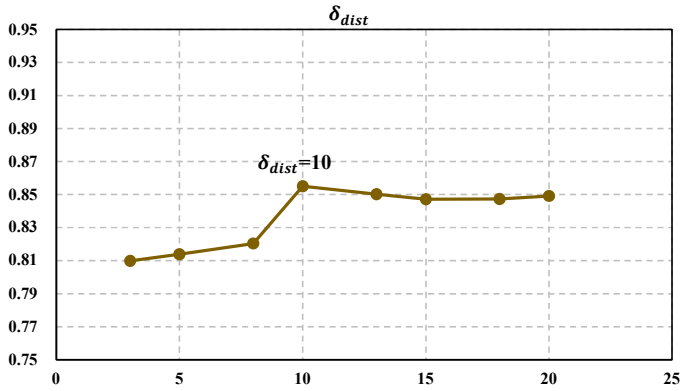
The FPS decreased slightly due to the resource usage of the centerline detection algorithm. Despite this result, a value of 25.47 FPS is sufficient for real-time processing, indicating that the framework can detect stopped vehicles in real time.

Through the ablation experiment, we proved that centerline detection significantly impacts the overall performance of the algorithm. Accurate centerline detection contributes to increased accuracy and assists in accurately detecting stopped vehicles. In other words, false detection can occur if the centerline detection is incorrect. Based on these findings, improving the reliability of centerline detection could further enhance the recall score of stopped vehicle detection. Enhancing the overall performance of the algorithm will be part of our future work.

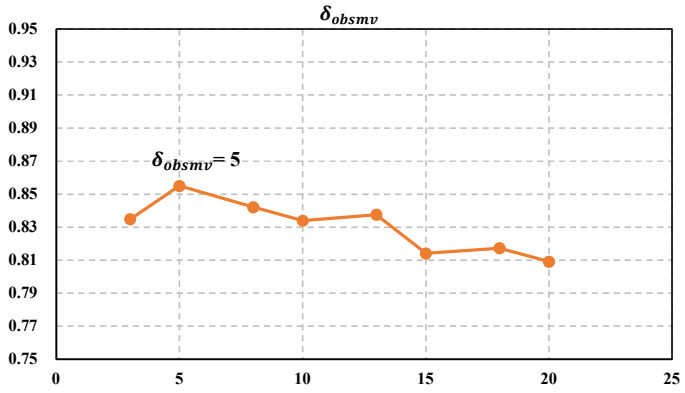
5.3. Hyperparameter Sensitivity

The proposed framework requires three specific thresholds to accurately determine whether a vehicle is stopped. These thresholds, δ_{obsmv}, δ_{size}, and δ_{dist}, correspond to the absolute motion of the observed vehicle, the change in the size of its bounding box, and the distance it moves between frames, respectively. To determine the optimal values, we conducted a grid search, using the same grid of {3, 5, 8, 10, 13, 15, 18, 20} for all three thresholds, and chose the values that yielded the highest F1 score. We evaluated the performance using the entire dataset and calculated the average F1 score.

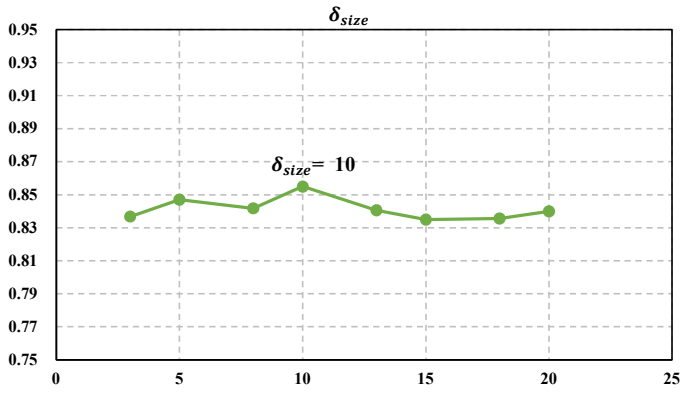
Figure 15 shows the grid search results for each threshold. Note that a higher F1 score indicates better performance. Among the thresholds, the most stable one in response to changes is the δ_{size} threshold in Figure 15c. However, when the size threshold exceeds 13, the performance drops sharply, since a large δ_{size} threshold can cause misclassification.



(**a**) δ_{dist}.



(**b**) δ_{obsmv}.



(**c**) δ_{size}.

Figure 15. Results of grid search for three thresholds δ_{dist}, δ_{obsmv}, and δ_{size}.

Similarly, for the δ_{dist} threshold in Figure 15a, the model shows less sensitivity to values above 10, with the performance remaining consistently high. However, if the threshold is set too high (above 15), there is a slight decline in performance. Setting the distance threshold too low can result in the framework exceeding the threshold in most cases, causing it to falsely classify most vehicles as stopped. Therefore, a threshold of 10 or higher seems to appropriately assess the motion of observed vehicles.

Lastly, the δ_{obsmv} threshold in Figure 15b performs best at a value of 5, with a gradual decline in performance as the threshold increases. The δ_{obsmv} threshold is identical to the movement threshold used for estimating host movement. The threshold used to estimate host movement is applied to reduce noise. When the host is stopped, the absolute movement of the observed vehicle is computed, and a certain amount of motion above the noise threshold is required to classify the observed vehicle as driving. As the δ_{obsmv} threshold increases, the noise threshold for both the host and the observed vehicle also increases, which may lead to both vehicles being misclassified as stopped, resulting in decreased performance.

Through this grid search, we ultimately set the final thresholds to 10, 5, and 10 for δ_{dist}, δ_{obsmv}, and δ_{size}, respectively. However, the relative motion threshold δ_{dist} could remain effective at values above 10, and the bounding-box threshold δ_{size} could maintain similar performance with values below 10.

5.4. Comparison of Object Detectors

In this section, we conducted ablation experiments using various versions of YOLO detectors to investigate the performance differences between each version. We exploited YOLOv5s from Ultralytics as the basis detector in the proposed framework, but there are several other YOLO models available off-the-shelf. For instance, one could utilize the bigger model, YOLOv5m, which reports higher mAP scores in public datasets. Alternatively, YOLOv8 can be used for its unique head architecture, as it has no anchor.

Table 5 displays the comparison results of different YOLO detectors. Note that the model with *n* has the fewest layers, while the number of layers increases sequentially in *s*, *m*, and *l*. As the number of layers in a deeper model increases, the number of parameters requiring computation also rises. Consequently, the computational load increases proportionally with the number of layers, leading to an obvious reduction in FPS. Although deeper models offer the advantage of more accurate object detection, in edge computing environments such as in-vehicle systems it is crucial to carefully balance the trade-off between speed and predictive performance.

Table 5. Comparison results of various YOLO detectors.

Model	Precision	Recall	mAP	F1 Score	FPS
3	0.8085	0.8502	0.7431	0.8194	20.84
5n	0.8751	0.8212	0.7655	0.8367	28.88
5s	0.8690	0.8498	0.7827	0.8520	23.68
5m	0.8601	0.8602	0.7744	0.8513	18.42
8n	0.8785	0.8299	0.7750	0.8454	21.85
8s	0.8252	0.8420	0.7533	0.8249	18.36
8m	0.8110	0.8564	0.7460	0.8269	14.03
8l	0.8091	0.8604	0.7461	0.8279	14.93

In terms of F1 score, the YOLOv5s model results in the best performance, with the second-highest FPS. YOLOv3 shows the worst F1 score, since it has the lowest object detection ability. We highlight that the recall scores are similarly high in most models. This indicates that, once the object detection performance of the model reaches a certain level, detecting stopped vehicles becomes sufficiently feasible in the proposed framework.

Surprisingly, YOLOv8 variants generally exhibit low precision scores, except for 8*n*. This result occurs because the high performance of YOLOv8 detects very small vehicles

within the frame and parts of vehicle edges in the screen corners. As a result, this causes the stopped vehicle detection algorithm to perform poorly. Additionally, the bounding boxes for each frame fluctuate since it has no anchor, leading to a high rate of false alarms in the proposed framework.

When comparing each version of YOLO, a meaningful comparison can be found in the FPS section. For large models such as YOLOv5m, YOLOv8m, and YOLOv8l, the inference time is extraordinarily high, causing a decrease in processing speed. This results in frame-drop phenomena, making these models unsuitable for real-time applications. Conversely, smaller models such as YOLOv3, YOLOv5n, and YOLOv8n exhibit faster processing speeds due to their shorter inference times.

Therefore, in an in-vehicle environment that operates with limited computing resources and requires real-time processing, we recommend the models YOLOv5n, YOLOv5s, and YOLOv8n. Considering all indices, we chose YOLOv5s, which demonstrated excellent performance for the proposed framework.

6. Conclusions

Recent advancements in intelligent transportation systems have concentrated on enhancing object detection and tracking technologies using computer vision, with a particular focus on real-time object detection for Advanced Driver Assistance Systems. Previous studies have primarily concentrated on detecting and tracking vehicles from stationary viewpoints, such as CCTV, but these methods have limitations in dynamic vehicle environments. Accurately detecting stopped vehicles using a dashcam in a driving vehicle, in particular, remains a significant unresolved challenge.

To tackle this issue, we propose a deep learning-based framework that integrates the YOLO object detector, the BoT-SORT tracker, and movement estimation including the dense optical flow method. This combination enables precise estimation of the relative motion between the host vehicle and observed vehicles. We also introduced additional features, such as centerline detection and inter-vehicle distance measurement, to effectively differentiate between stationary and moving vehicles under real-world road conditions. These enhancements markedly improved the overall performance of stopped vehicle detection, particularly in terms of precision and recall.

The proposed method achieved a real-time processing rate of approximately 25 frames per second, demonstrating its suitability for real-time systems. The experimental results indicated high precision, recall, and F1 scores, validating the robustness and reliability of the framework even in complex driving scenarios. To support the performance, we further investigated the proposed framework through in-depth quantitative and qualitative analysis. We have discussed the effectiveness of both distance-based and bounding-box-size-based metrics. Also, we have proven the impact of detecting the centerline and filtering out vehicles in the opposite lane through ablation experiments. To expand the compatibility of the framework and prompt practical usage, we compared various detectors in terms of their detection performance and FPS.

In summary, the proposed framework demonstrates a significant technical advancement in real-time stopped vehicle detection, achieving both fast speed and high accuracy. Nevertheless, challenges remain in detecting stopped vehicles that are obscured from view. Future research will aim to address this issue by expanding the applicability of the framework to a wider range of road conditions. Also, refining the centerline detection to further enhance the accuracy of stopped vehicle detection would be an interesting avenue for future study.

Author Contributions: Conceptualization, J.P. and Y.L.; data curation, J.P. and J.L.; formal analysis, J.P. and Y.P.; funding acquisition, Y.L.; investigation, J.P. and J.L.; methodology, J.P.; project administration, Y.P.; resources, Y.L.; software, J.P.; supervision, J.P. and Y.L.; validation, Y.P.; visualization, J.P. and J.L.; writing—original draft, J.L. and J.P.; writing—review and editing, J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a grant from the Korea Agency for Infrastructure Technology Advancement (KAIA), funded by the Ministry of Land, Infrastructure, and Transport (Grant RS-2021-KA160853).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Baselice, F.; Ferraioli, G.; Matuozzo, G.; Pascazio, V.; Schirinzi, G. 3D automotive imaging radar for transportation systems monitoring. In Proceedings of the 2014 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems Proceedings, Naples, Italy, 17–18 September 2014; pp. 1–5.
2. Hariharan, B.; Arbeláez, P.; Girshick, R.; Malik, J. Hypercolumns for object segmentation and fine-grained localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 447–456.
3. Karpathy, A.; Fei-Fei, L. Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3128–3137.
4. Wu, Q.; Shen, C.; Wang, P.; Dick, A.; Van Den Hengel, A. Image captioning and visual question answering based on attributes and external knowledge. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 1367–1381. [CrossRef]
5. Kang, K.; Li, H.; Yan, J.; Zeng, X.; Yang, B.; Xiao, T.; Zhang, C.; Wang, Z.; Wang, R.; Wang, X.; et al. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *IEEE Trans. Circuits Syst. Video Technol.* **2017**, *28*, 2896–2907. [CrossRef]
6. Wei, J.; He, J.; Zhou, Y.; Chen, K.; Tang, Z.; Xiong, Z. Enhanced object detection with deep convolutional neural networks for advanced driving assistance. *IEEE Trans. Intell. Transp. Syst.* **2019**, *21*, 1572–1583. [CrossRef]
7. Nedevschi, S.; Vatavu, A.; Oniga, F.; Meinecke, M.M. Forward collision detection using a stereo vision system. In Proceedings of the 2008 4th International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, 28–30 August 2008; pp. 115–122.
8. Saranya, M.; Archana, N.; Reshma, J.; Sangeetha, S.; Varalakshmi, M. Object detection and lane changing for self driving car using cnn. In Proceedings of the 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 10–11 March 2022; pp. 1–7.
9. Waykole, S.; Shiwakoti, N.; Stasinopoulos, P. Review on lane detection and tracking algorithms of advanced driver assistance system. *Sustainability* **2021**, *13*, 11417. [CrossRef]
10. Muzammel, M.; Yusoff, M.Z.; Saad MN, M.; Sheikh, F.; Awais, M.A. Blind-Spot Collision Detection System for Commercial Vehicles Using Multi Deep CNN Architecture. *Sensors* **2022**, *22*, 6088. [CrossRef]
11. SAE International Recommended Practice, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, SAE Standard J3016_202104, Revised April 2021, Issued January 2014. Available online: https://www.sae.org/standards/content/j3016_202104/ (accessed on 1 September 2024).
12. Bevilacqua, A.; Vaccari, S. Real time detection of stopped vehicles in traffic scenes. In Proceedings of the 2007 IEEE Conference on Advanced Video and Signal Based Surveillance, London, UK, 5–7 September 2007; pp. 266–270.
13. Lan, J.; Jiang, Y.; Fan, G.; Yu, D.; Zhang, Q. Real-time automatic obstacle detection method for traffic surveillance in urban traffic. *J. Signal Process. Syst.* **2016**, *82*, 357–371. [CrossRef]
14. Lis, K.; Nakka, K.; Fua, P.; Salzmann, M. Detecting the unexpected via image resynthesis. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–November 2019; pp. 2152–2161.
15. Chand, D.; Gupta, S.; Kavati, I. Computer vision based accident detection for autonomous vehicles. In Proceedings of the 2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, 10–13 December 2020; pp. 1–6.
16. Chung, Y.L.; Lin, C.K. Application of a model that combines the YOLOv3 object detection algorithm and canny edge detection algorithm to detect highway accidents. *Symmetry* **2020**, *12*, 1875. [CrossRef]
17. Karim, A.; Raza, M.A.; Alharthi, Y.Z.; Abbas, G.; Othmen, S.; Hossain, M.S.; Nahar, A.; Mercorelli, P. Visual Detection of Traffic Incident through Automatic Monitoring of Vehicle Activities. *World Electr. Veh. J.* **2024**, *15*, 382. [CrossRef]
18. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.
19. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
20. Ding, S.; Xu, Y.; Zhang, Q.; Yu, J.; Sun, T.; Ni, J.; Shi, S.; Kong, X.; Zhu, R.; Wang, L.; et al. Obstacle detection method based on RSU and vehicle camera fusion. *Sensors* **2023**, *23*, 4920. [CrossRef]
21. Yao, Y.; Wang, X.; Xu, M.; Pu, Z.; Wang, Y.; Atkins, E.; Crandall, D.J. Dota: Unsupervised detection of traffic anomaly in driving videos. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *45*, 444–459. [CrossRef]
22. Cuevas, C.; Martínez, R.; García, N. Detection of stationary foreground objects: A survey. *Comput. Vis. Image Underst.* **2016**, *152*, 41–57. [CrossRef]

23. Abdulrahim, K.; Salam, R.A. Traffic surveillance: A review of vision based vehicle detection, recognition and tracking. *Int. J. Appl. Eng. Res.* **2016**, *11*, 713–726.
24. Xu, L.; Bu, W. Traffic flow detection method based on fusion of frames differencing and background differencing. In Proceedings of the 2011 Second International Conference on Mechanic Automation and Control Engineering, Inner Mongolia, China, 15–17 July 2011; pp. 1847–1850.
25. Zhang, H.; Wu, K. A vehicle detection algorithm based on three-frame differencing and background subtraction. In Proceedings of the 2012 Fifth International Symposium on Computational Intelligence and Design, Hangzhou, China, 28–29 October 2012; Volume 1, pp. 148–151.
26. Buch, N.; Velastin, S.A.; Orwell, J. A review of computer vision techniques for the analysis of urban traffic. *IEEE Trans. Intell. Transp. Syst.* **2011**, *12*, 920–939. [CrossRef]
27. Redmon, J. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
28. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
29. Jocher, G. YOLOv5 by Ultralytics, Version 7.0. 2020. Available online: https://zenodo.org/records/7347926 (accessed on 8 August 2024).
30. Jocher, G.; Chaurasia, A.; Qiu, J. YOLO by Ultralytics, Version 8.0.0; 2023. Available online: https://github.com/ultralytics/ultralytics (accessed on 1 May 2024).
31. Wang, H.; Liu, C.; Cai, Y.; Chen, L.; Li, Y. YOLOv8-QSD: An improved small object detection algorithm for autonomous vehicles based on YOLOv8. *IEEE Trans. Instrum. Meas.* **2024**, *73*, 2513916. [CrossRef]
32. Bewley, A.; Ge, Z.; Ott, L.; Ramos, F.; Upcroft, B. Simple online and realtime tracking. In Proceedings of the 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 25–28 September 2016; pp. 3464–3468.
33. Welch, G.; Bishop, G. An Introduction to the Kalman Filter. 1995. Available online: https://dl.acm.org/doi/book/10.5555/897831 (accessed on 1 August 2024).
34. Kuhn, H.W. The Hungarian method for the assignment problem. *Nav. Res. Logist. Q.* **1955**, *2*, 83–97. [CrossRef]
35. Aharon, N.; Orfaig, R.; Bobrovsky, B.Z. BoT-SORT: Robust associations multi-pedestrian tracking. *arXiv* **2022**, arXiv:2206.14651.
36. Zhang, Y.; Sun, P.; Jiang, Y.; Yu, D.; Weng, F.; Yuan, Z.; Luo, P.; Liu, W.; Wang, X. Bytetrack: Multi-object tracking by associating every detection box. In *European Conference on Computer Vision*; Springer Nature: Cham, Switzerland, 2022; pp. 1–21.
37. Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, 29 June–2 July 2003 Proceedings 13*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 363–370.
38. Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *6*, 679–698. [CrossRef]
39. Matas, J.; Galambos, C.; Kittler, J. Robust detection of lines using the progressive probabilistic hough transform. *Comput. Vis. Image Underst.* **2000**, *78*, 119–137. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.