Wiley
Journal of Advanced Transportation
Volume 2024, Article ID 3679014, 16 pages
https://doi.org/10.1155/2024/3679014

Research Article

Intelligent Infrastructure for Traffic Monitoring Based on Deep Learning and Edge Computing

Jaime Villa,[1] Franz García,[1] Rubén Jover,[2] Ventura Martínez,[2] and José M. Armingol[1]

[1]*Intelligent Systems Lab (LSI) Research Group, Universidad Carlos III de Madrid, Leganés 28911, Spain*
[2]*Sacyr Concesiones S.L., Madrid 28027, Spain*

Correspondence should be addressed to Jaime Villa; javillap@pa.uc3m.es

Received 19 January 2024; Revised 26 April 2024; Accepted 10 May 2024; Published 23 May 2024

Academic Editor: Ziyuan Gu

In the field of traffic management and control systems, we are witnessing a symbiotic evolution, where intelligent infrastructure is progressively collaborating with smart vehicles to produce benefits for traffic monitoring and security, by rapidly identifying hazardous behaviours. This exponential growth is due to the rapid development of deep learning in recent years, as well as the improvements in computer vision models. These technologies allow for monitoring tasks without the need to install numerous sensors or stop the traffic, using the extensive camera network of surveillance cameras already present in worldwide roads. This study proposes a computer vision-based solution that allows for real-time processing of video streams through edge computing devices, eliminating the need for Internet connectivity or dedicated sensors. The proposed system employs deep learning algorithms and vision techniques that perform vehicle detection, classification, tracking, speed estimation, and vehicle geolocation.

1. Introduction

In the 21st century, where mobility is at the forefront of our daily lives, the demand for effective traffic monitoring systems has never been more pressing. Traditionally, traffic monitoring relied on a large variety of sensors and technologies, each with its own set of advantages and limitations. To obtain information about traffic from the road surface, different types of sensors have been employed. On-road systems use sensors that require complex installation procedures, often necessitating disruptive measures such as preparing the asphalt or interrupting traffic for extended periods of time. This category of sensors, classified as intrusive, includes inductive loop detectors, magnetometers [1, 2], microloops, pneumatic tubes, and piezoelectric cables. While these technologies have reached a mature stage of development, their installation and maintenance incur high costs and are challenging to relocate, making them less adaptable to evolving traffic needs.

Over-road systems, on the other hand, offer a distinct advantage by leveraging nonintrusive sensors that do not require road closures during installation, resulting in significantly reduced costs and minimal disruption. These nonintrusive sensors fall into two main categories: active and passive. Active sensors employ signal transmission and reception to measure the distance between the sensor and an object, utilizing the TOF [3] (Time of Flight) principle.

In contrast, passive sensors can gather information about traffic without direct interaction, and they are primarily represented by traditional cameras. Video cameras offer high-resolution imaging capabilities and can be deployed almost everywhere, enabling us to capture detailed visual data of traffic scenarios. Embracing this technology choice in our investigation is justified by the versatility and effectiveness of cameras.

This research proposes an edge computing platform capable of monitoring traffic in real time through computer vision, without the need of cloud or hybrid computing [4, 5] or depending on specialised sensors [1–3]. The solution should be able to (1) detect and classify vehicles, (2) track detection, (3) geolocate tracked vehicles and measure their speed, and (4) run in real time on an edge computing device.

The structure of this article is divided into several sections, each dedicated to a specific aspect of our research. In

Section 2 (Literature Review), we comprehensively review SOTA (State-of-the-Art) techniques for object detection, tracking, and speed estimation. Moving forward, Section 3 elaborates on the methodology used for dataset development and outlines the design process for our global solution. Section 4 provides a detailed explanation of our solution, while Section 5 showcases the results of our experiments and evaluations, offering insights into the system's performance. Finally, Section 6 summarizes our conclusions, discussing the implications of our findings and potential paths for future research. This paper aims to provide a structured and informative exploration of our work in the field of intelligent infrastructure for traffic analysis.

2. Literature Review

In this section, we will conduct a thorough review of current methods in computer vision-based vehicle detection, tracking, and speed estimation, highlighting their application in traffic monitoring. By presenting an in-depth exploration of these techniques, this section will provide a solid foundation for understanding the SOTA in computer vision-based traffic monitoring.

2.1. Object Detection Algorithms. Object detection algorithms are a basic component of computer vision-based traffic monitoring systems. Traditional approaches have emphasized background subtraction, effectively distinguishing moving vehicles from stationary environment. Others focus on vehicle features like shape, symmetry, casted shadow [6], headlights [7], or license plates [8]. However, these traditional methods often face limitations in terms of adaptability and accuracy in varied traffic scenarios. This led to the exponential growth of deep learning solutions for object detection applications, setting a new standard in accuracy and efficiency that benefited traffic monitoring tasks.

Computer vision has been profoundly shaped by deep learning algorithms like the R-CNN family and the YOLO (You Only Look Once) series. Starting at 2013, R-CNN [9] combined CNNs (Convolutional Neural Networks) with region proposals for object localization and segmentation. This evolution continued with Fast R-CNN [10] and then Faster R-CNN [11], the latter introducing the Region Proposal Network (RPN) for more efficient object location prediction.

In 2015, Redmond et al. developed the original YOLO model [12, 13], which was revolutionary, as it was able to process images in a single pass, unifying the tasks of object localization and classification. Subsequent versions brought substantial improvements: YOLOv2 [13, 14] introduced batch normalization and anchor boxes, while YOLOv3 [13, 15] optimized performance with an improved backbone network and multiple anchors. YOLOv4 [13, 16] continued this trend with enhanced training methods like mosaic data augmentation and a new head architecture using anchor-free detection.

YOLOv5 [17], released by Ultralytics and designed by G. Jocher, not only offered improvements but also is the first model of the family to be implemented in PyTorch and to add size variations for diverse accuracy and computing requirements. It utilizes advanced data augmentation techniques such as scaling, colour space adjustments, and mosaic augmentation, broadening the model's exposure to different scenarios. Furthermore, YOLOv5 marks a notable evolution in anchor box optimization, featuring autolearning bounding box anchors from the training data.

The introduction of YOLOv6 [18] marks a significant milestone in the YOLO series. Developed with a focus on industrial applications, YOLOv6 achieves high trade-off between accuracy and speed. Following this advancement, YOLOv7 [19] emerged as a breakthrough, extending beyond object detection to achieve pose estimation on the COCO key point dataset.

YOLOv8 [20] and YOLO-NAS [21] represent the pinnacle of this series. YOLOv8 improved upon YOLOv7 with architectural modifications, like the anchor-free detection heads and new features, such as pose estimation, semantic segmentation, and pure object classification. YOLO-NAS, developed by Deci AI, stands out with its Neural Architecture Search [21] technology, excelling in accuracy-latency trade-offs and featuring a robust approach to feature extraction and real-time object detection.

In Table 1, different versions of YOLO after 2020 are shown. There, the models with the highest AP (Average Precision) on the COCO2017 dataset are YOLOv7, YOLOv5, and YOLOv8, respectively. However, in the development of object detection algorithms, precision alone is not the only concern. Balancing latency and precision is also important in real-world applications, as well as the size that models occupy in memory.

In the plots of Figure 1, a comparison is shown between YOLOv5-6-7-8 models and their respective subversions based on the number of parameters. YOLOv5n [17] is the one that occupies the least memory space, with a mAP (mean Average Precision) on COCO below 30%. On the other hand, YOLOv8n [20] and YOLOv6n [18] achieve similar accuracy, with version 8 being slightly more precise with less parameters. Regarding larger model sizes, YOLOv8 consistently surpasses its predecessors in mAP, yet it exhibits more latency.

Since its release, YOLOv5 has remained in widespread use, even after versions 6 and 7 were proposed. This is because the environment in which it was developed has allowed for training, validation, and deployment in a fast and simple way, greatly expanding the support for this model on numerous platforms. In 2023, Ultralytics followed the same strategy and launched YOLOv8 with an even simpler environment.

2.2. Multiple Object Tracking Algorithms. Object tracking algorithms are a crucial component in computer vision applications, where differentiating between subjects is key. Multiobject Trackers (MOTs) assign a unique tracking identifier to each object, making them distinguishable over time. Diverse approaches have been developed, each improving upon its predecessors in unique ways. These advancements have allowed the implementation of such algorithms in traffic monitoring tasks.

Table 1: YOLO architecture summary.

Version	Date	Framework	AP (%)
YOLOv4 [16]	2020	Darknet [13]	43.5
YOLOv5 [17]	**2020**	**PyTorch**	**55.8**
YOLOv6 [18]	2022	PyTorch	52.5
YOLOv7 [19]	**2022**	**PyTorch**	**56.8**
YOLOv8 [20]	**2023**	**PyTorch**	**53.9**
YOLO-NAS [21]	2023	PyTorch	52.2

The AP metric (Average Precision) is calculated on COCO2017, and data are taken from [22]. Bold values indicate the top 3 models for Average Precision (AP) in COCO2017.

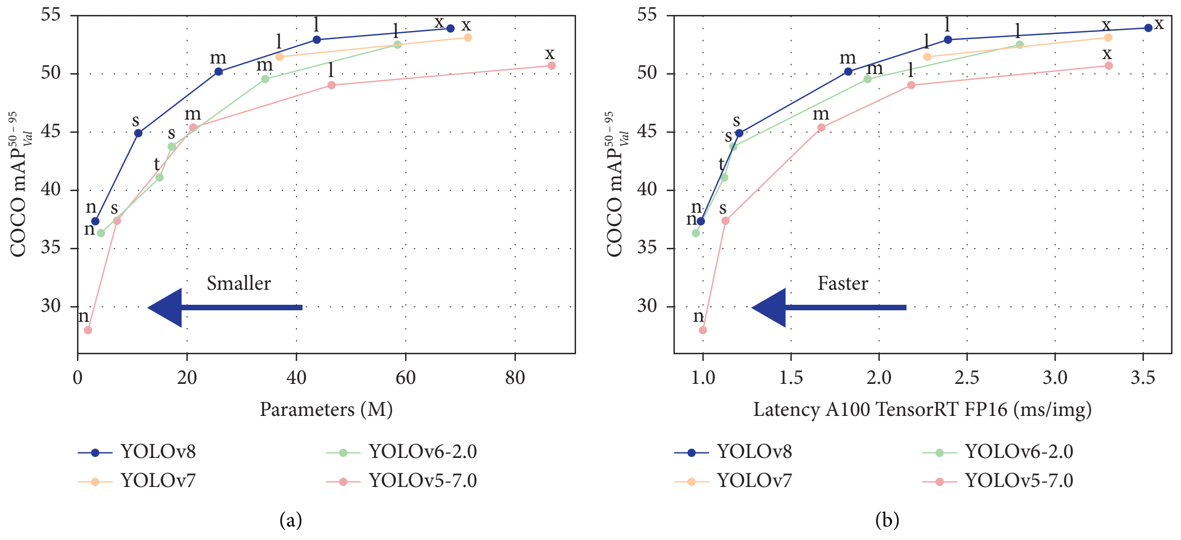


Figure 1: Performance comparison for different YOLO model sizes and versions. In (a) the size-mean average precision relationship is illustrated, while (b) shows the latency-mean average precision trade-off. YOLOv5-6-7-8 models are compared in different size versions. Plots were taken from [23].

In 2016, Bewley et al. proposed SORT [24] (Simple Online and Realtime Tracking), which employs a tracking-by-detection framework. Objects are initially detected by an object detection algorithm, and then the tracker assigns the target identifiers. The tracking process is subdivided in two separated tasks: motion estimation and data association. For motion estimation, a Kalman Filter [25] is used to perform a state-space estimation on a linear velocity model, proposing new position candidates for previous detection. The data association task is performed by the Hungarian algorithm [26], assigning current detection to existing targets. In SORT, occlusions, both short-term and long-term, are not directly addressed. Instead, the algorithm implicitly manages short-term occlusions with the intersection-over-union (IOU) metric during the data association process.

Deep SORT [27] built upon SORT incorporates a deep appearance descriptor that enhances the data association process. Like SORT, this algorithm also employs a Kalman filter to predict target movements. For data association, it computes the appearance descriptor of every detection and tracked targets using a CNN, and similarity is obtained using the Cosine distance. Movement is also considered for the association task, and it is evaluated through the Mahalanobis distance [28]. Both metrics improve performance in occlusion and reidentification scenarios.

Byte Track [29] further advanced the field by adopting a novel strategy for associating almost every detection box, including those with low scores, thereby significantly reducing object missing and trajectory fragmentation. This is achieved by matching high confidence detection first, using IoU (intersection-over-union) and appearance features. Low confidence detection is matched after, using only IoU.

BoT SORT [30]stands out in its approach by not only addressing the limitations found in its predecessors but also by introducing innovative enhancements. First it integrates a camera motion compensation, crucial for more accurate tracking in dynamic scenes. Furthermore, BoT SORT refines the use of the Kalman filter for state estimation, offering more precise predictions of object trajectories, and improves the appearance and movement association metrics.

2.3. Speed Estimation Techniques. Traffic monitoring systems employ different strategies for measuring vehicle speeds, some consist in sensors directly installed on roads [2], others consist in TOF measurements [3], and others even deploy smartphones inside vehicles to measure actual speed [4, 5].

In the computer vision field, various methods are based on distance estimation techniques coupled with timestamps. There are several approaches for acquiring these measurements, for example, one approach involves using stereo systems [31], where distances are obtained from disparity maps. However, these systems demand highly precise camera calibration for effective operation.

Alternatively, there are methods that rely on a single camera, known as monocular systems. Unlike stereo solutions, monocular methods do not require intricate camera calibration. Instead, they often use environmental measurements [32], such as the distances between road symbols, lines [33], streetlights, poles, or signs for accurate distance estimation. The monocular solution proposed in [34] computes a BEV (Bird's Eye View) perspective, transforming 2D data into 3D by adding depth information. These transformations can be extended to other different coordinate systems [35], which is useful not only for distance and speed measurement but also for positioning vehicles with precision in GPS coordinate systems.

2.4. Summary. The most modern object detection solutions based on computer vision use the latest YOLO versions [17–21]. In addition, various algorithms have been developed for object tracking such as Deep SORT, Byte Track, or BoT SORT. Finally, best distance or speed inference techniques employ stereo strategies [31] or perspective transformations [34, 35] in monocular vision systems.

3. Materials and Methods

This section details the methodology employed in the development of our solution. We specify the data acquisition process, dataset design and construction, model training procedures, tracking algorithm, geolocation and speed estimation techniques, and hardware selection criteria.

3.1. Data Acquisition and Dataset Design. Training high-quality object detection models requires datasets that contain a representative number of instances per class, enabling the model to comprehend the distinctions between each one of them. It is also crucial to balance the dataset as much as possible, to mitigate the biases that imbalance may introduce. To address the vehicle classification problem, we have chosen to categorize the vehicles into seven distinct types. Since we did not have a preexisting dataset that met our requirements, we created one from scratch (see Figure 2).

Our data consisted of various video sequences, recorded across different locations in Spanish territory. The clips were acquired through numerous devices, ranging from IP security cameras to mobile phones, which provided a diverse range of image resolutions and scenarios. Each frame underwent postprocessing using computer vision tools, to add padding while maintaining 1 : 1 aspect ratio, and filtering frames with damaged information.

After obtaining the postprocessed images, we started an initial labelling process in which a portion of the dataset was labelled concurrently with the ongoing data acquisition through the recording and postprocessing of new videos. Upon completing the initial labelling, we proceeded with training a YOLO model using the existing data. This made the labelling process easier and faster, as the model generated candidate bounding boxes, which were later reviewed by our team to ensure data quality.

The video sequences recorded by our team typically showcase landscapes with favourable weather conditions, which has forced us to generate artificial data [36] to diversify, simulating different weather conditions as shown in Figure 3. After this final process, we obtained over 16,000 labelled images which featured over 35,000 vehicles (see Figure 4). Finally, we divided the data randomly, allocating 30% for the validation set and 70% for the training set.

3.2. Model Training. Despite the comparisons between SOTA YOLO versions stated in Section 2, which demonstrate advancements in versions 6, 7, 8, and NAS, YOLOv5 [17] was the best and most widely used version at the time of dataset labelling and training (2021-2022). Ultralytics did a great job developing a simple environment for training PyTorch-based YOLO models. In addition, YOLOv5 came out in different sizes, which was a great advantage when embedding the nanoversion (YOLOv5n) into edge computing devices.

As previously explained, the training process was developed in parallel with the dataset labelling tasks. We adopted an iterative and gradual approach, by meticulously labelling our custom dataset, a process as laborious as it is critical. To ensure that the model learns accurately from reliable information, we manually supervised labels after each session. As our dataset grew larger and became more refined, we carried out successive training cycles and evaluated the model's performance on a separate test set. This iterative methodology has allowed us to monitor and fine-tune the model's parameters, ensuring that improvements in labelling effectively translate into increased accuracy and robustness of the YOLOv5n detection system.

To achieve robust and efficient training, our training server is equipped with two Nvidia RTX 3090 Ti graphics cards. While the RTX 4000 series represents the next step in hardware innovation, the RTX 3090 Ti cards offer an optimal balance between availability, cost, and performance, suitable for our current needs. The ability to operate these GPUs in parallel has not only maximized processing speed but has also provided a solid platform for experimenting with data-intensive training strategies.

The progression in training outcomes is depicted in Table 2; it points to a notable improvement in the performance metrics across successive training iterations on every stage of the dataset. The marked improvement in precision (P), recall (R), mean Average Precision at 50% IoU (mAP50), and mean Average Precision from 50% to 95% IoU (mAP50-95) underscores the efficacy of the iterative training and dataset refinement process employed. The transition from YOLOv5n-1 to YOLOv5n-4 demonstrates a significant climb in performance, with the overall mAP50-95 jumping from 0.479 to an impressive 0.8, which can be attributed to the enriched quality of the dataset over time.

On the other hand, there is a noticeable variation in accuracies for each vehicle class. This happens because many of these vehicles are not as frequent as others on the roads;

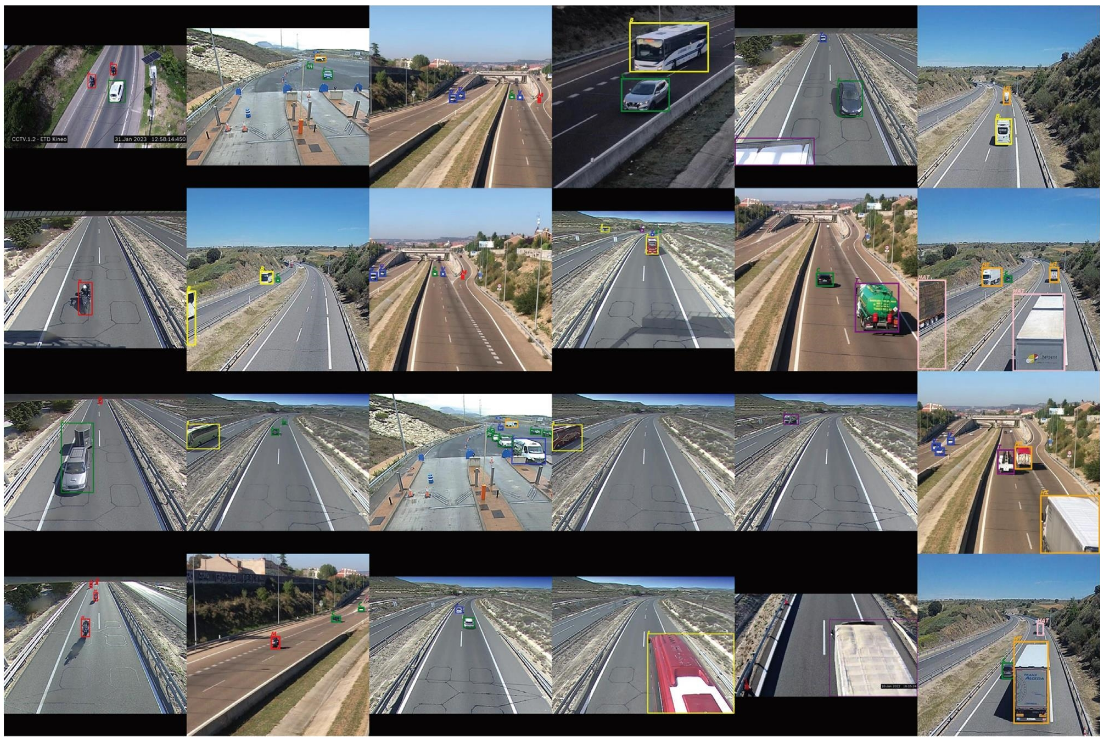


Figure 2: Dataset preview in mosaic format. A collection of different instances of vehicles obtained at locations distributed throughout Spanish territory.

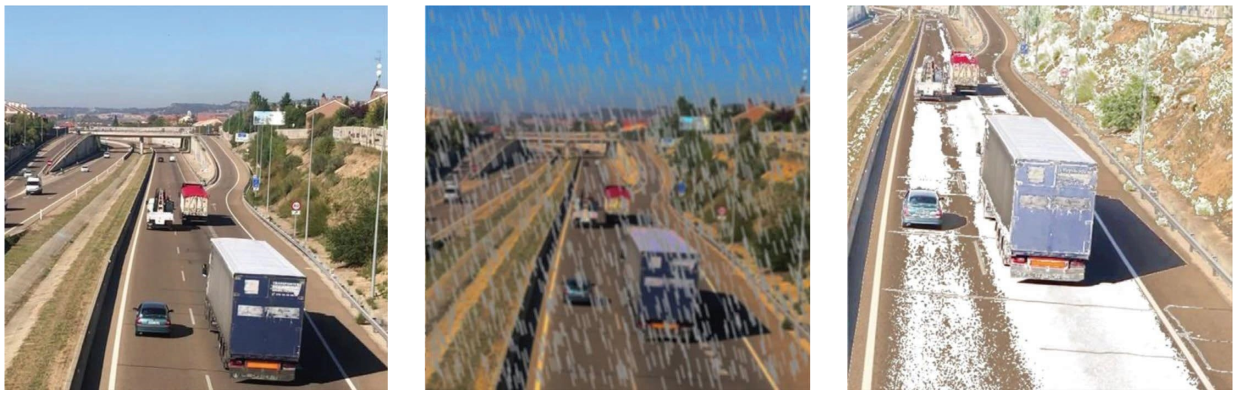


Figure 3: Visualization of weather data augmentation [36]. Normal, rainy, and snowy conditions.

capturing cars in winter videos is more common than recording motorcycles. In addition, distinguishing between cars and vans or different types of trucks from a distance presents a significant challenge. In summary, both the data diversity and balance, as well as vehicle appearance similarity, can introduce biases that are reflected in these metrics.

3.3. Tracking Algorithm. The evolution of object tracking in computer vision has led to the development of numerous algorithms, each with unique strengths and applications. Although we acknowledge the higher capabilities of Byte Track [29] and BoT SORT [30], we have focused on the implementation of the Deep SORT [27] (Simple Online and Realtime Tracking with a Deep Association Metric) algorithm. This choice was motivated by Deep SORT's proven effectiveness in real-time tracking scenarios during our tracking algorithm implementation stage.

Deep SORT, building upon the foundational SORT [24] algorithm, enhances tracking capabilities by integrating appearance information through deep learning. This approach addresses the limitations of SORT, particularly in maintaining consistent object identities through periods of occlusion and interaction. By leveraging a CNN [27], Deep SORT achieves a significant reduction in identity switches and improves overall tracking robustness.

When managing long-term occlusions, Deep SORT struggles to reidentify vehicles. This usually happens when small vehicles such as cars and motorbikes hide behind large trucks or buses for long periods of time. To mitigate this, we try to place cameras in a position that benefits

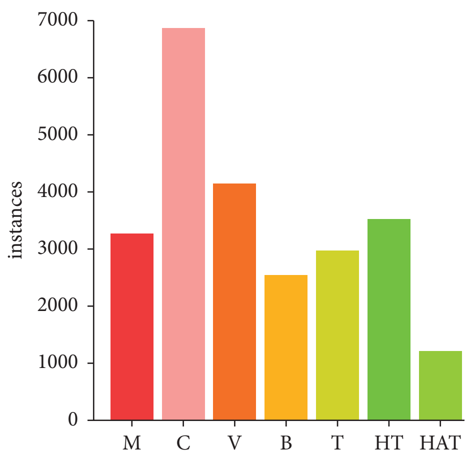


Figure 4: Vehicle instance dataset distribution. The labels correspond to motorbike (M), car (C), van (V), bus (B), truck (T), heavy truck (HT), and heavy articulated truck (HAT).

Table 2: YOLOv5 validation benchmark progress.

Model	Class	Images	Instances	*P*	*R*	mAP50	mAP50-95
YOLOv5n-1	All	4992	10616	0.845	0.577	0.655	0.479
	M		1406	0.949	0.780	0.844	0.606
	C		2983	0.851	0.505	0.591	0.422
	V		1771	0.866	0.565	0.679	0.517
	B		1046	0.890	0.543	0.645	0.479
	T		1324	0.610	0.638	0.627	0.486
	HT		1561	0.870	0.477	0.560	0.398
	HAT		525	0.878	0.533	0.641	0.447
YOLOv5n-2	All	4992	10616	0.891	0.837	0.883	0.699
	M		1406	0.839	0.939	0.956	0.657
	C		2983	0.839	0.769	0.804	0.599
	V		1771	0.865	0.727	0.849	0.668
	B		1046	0.926	0.910	0.944	0.825
	T		1324	0.797	0.827	0.845	0.683
	HT		1561	0.987	0.710	0.792	0.639
	HAT		525	0.988	0.975	0.992	0.820
YOLOv5n-3	All	4992	10616	0.917	0.840	0.888	0.735
	M		1406	0.954	0.922	0.969	0.697
	C		2983	0.857	0.759	0.808	0.633
	V		1771	0.885	0.772	0.855	0.695
	B		1046	0.924	0.904	0.947	0.857
	T		1324	0.816	0.823	0.850	0.713
	HT		1561	0.993	0.716	0.794	0.682
	HAT		525	0.992	0.980	0.993	0.865
YOLOv5n-4	All	4992	10616	0.983	0.929	0.954	0.800
	M		1406	0.989	0.985	0.994	0.762
	C		2983	0.960	0.849	0.927	0.714
	V		1771	0.981	0.938	0.962	0.810
	B		1046	0.997	0.962	0.976	0.892
	T		1324	0.985	0.985	0.993	0.878
	HT		1561	0.993	0.798	0.835	0.708
	HAT		525	0.978	0.987	0.993	0.837

analysis and reduces the chance of facing such situations. In Figure 5, we show Deep SORT performing reidentification after a vehicle is occluded behind other and after a few instants reappeared. Different scenarios need different parameter tuning, as high-speed roads do not need to store in memory old targets for a long time, and highly congested lanes should be capable of maintaining context without mixing up identifiers. These parameter adjustments are crucial in occlusion handling and need to be adapted to each situation.

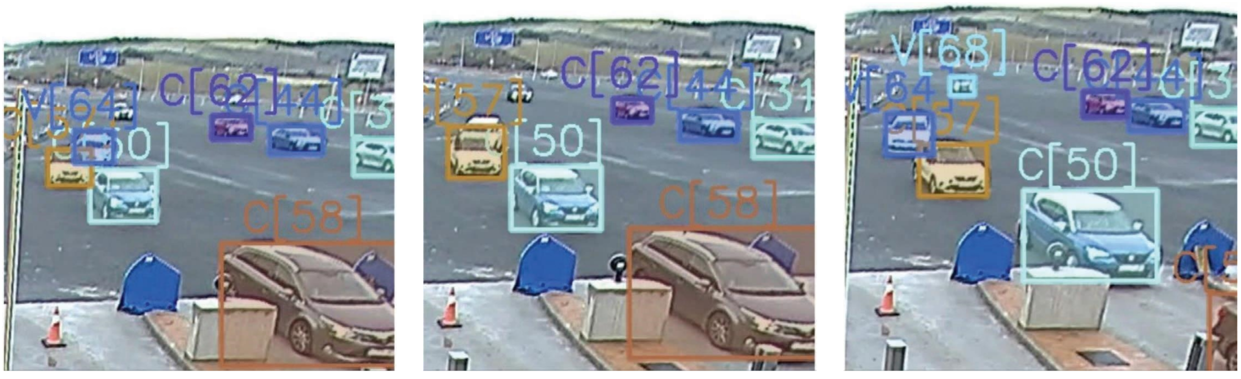


Figure 5: Occlusion handling. This composite image demonstrates how Deep SORT [27] deals with occlusions. On (a) vehicle with track id 64 is detected and tracked successfully, although it is not detected on (b), as it hides behind vehicle with id 57. Finally, vehicle 64 is reidentified on (c).

3.4. Geolocation and Speed Estimation Technique. Geolocation technology has become an essential asset of modern robotics, self-driven vehicles, and intelligent infrastructures, facilitating advanced applications across unstructured and urban environments. The synergy of unmanned vehicles with geolocative capabilities provides a framework for autonomous navigation and object tracking, leveraging the advances in sensor fusion, computer vision, and artificial intelligence. Pioneering research such as that presented in [37] demonstrates the integration of unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs) for object detection and global tracking using custom AI subsystems and multispectral sensor data.

Further advancing the field, Ulicny et al. [38] have proposed innovative methodologies for geolocation and height estimation of objects utilizing street-level RGB imagery. Their approach combines the precision of CNNs for object detection with depth map prediction and Markov Random Field (MRF) optimization, revealing the methodology's potential for enhancing urban environment applications and contributing to more accurate road management systems.

Our solution employs the method proposed by Blake [35], which stands out for its algorithmic simplicity, making it exceptionally suitable for edge computing environments. By employing straightforward and efficient perspective transformation techniques, the method provides robust geolocation while conforming to the constraints of computational resources inherent in edge computing. This approach lays a solid foundation for deploying advanced vision-based geolocation systems within a resource-limited hardware.

$$\begin{bmatrix} \mathrm{lon}' \\ \mathrm{lat}' \\ w' \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix} \mathrm{lon}_{\mathrm{normalized}} \\ \mathrm{lat}_{\mathrm{normalized}} \end{bmatrix} = \begin{bmatrix} \dfrac{\mathrm{lon}'}{w'} \\ \dfrac{\mathrm{lat}'}{w'} \end{bmatrix}. \tag{1}$$

To transform the representation of a vector from one plane to another, a change of basis is applied. This involves multiplying the vector by a transformation matrix that encodes the relationship between the two coordinate systems. The transformation matrix is constructed from the basis vectors of the destination plane relative to the original plane.

When dealing with two-dimensional image coordinate points, these are extended to three dimensions as homogeneous coordinates, represented as $[x, y, 1]$ instead of the standard two-dimensional $[x, y]$. The transformation process involves multiplying these extended pixel coordinates by a 3×3 perspective transformation matrix, resulting in the homogeneous GPS coordinate system, in the form $[\mathrm{lon}', \mathrm{lat}', w']$. To convert these back to the conventional two-dimensional GPS format, a normalization process is applied using the w' scaling factor, as shown in (1). The resulting normalized coordinates, $\mathrm{lon}_{\mathrm{normalized}}$ and $\mathrm{lat}_{\mathrm{normalized}}$, accurately represent geographical positions in two dimensions.

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{array}{l} \mathrm{lon}'_i \\ \mathrm{lat}'_i \\ w' \end{array} \begin{array}{l} \\ = \\ \\ \end{array} \begin{array}{l} m_{11}x_i + m_{12}y_i + m_{13}, \\ m_{21}x_i + m_{22}y_i + m_{23}, \quad \text{where } i \in \{0, 1, 2, 3\}. \\ m_{31}x_i + m_{32}y_i + m_{33}. \end{array} \tag{2}$$

In the use case proposed by Blake [35], the M matrix is computed using four sets of pixel points $[x, y]$ and their corresponding set of four longitude and latitude points $[\text{lon}, \text{lat}]$. Using this information, the elements in the M matrix are calculated from the equation system described by (2).

$$\begin{bmatrix} x' \\ y' \\ \mu' \end{bmatrix} = M^{-1} \begin{bmatrix} \text{lon} \\ \text{lat} \\ 1 \end{bmatrix} \begin{bmatrix} x_{\text{normalized}} \\ y_{\text{normalized}} \end{bmatrix} = \begin{bmatrix} \dfrac{x'}{\mu'} \\ \dfrac{y'}{\mu'} \end{bmatrix}. \tag{3}$$

After M is calculated, the M^{-1} matrix can also be computed inverting the original transformation matrix. With these two mathematical tools, we can move from the pixel plane to the geolocation plane with ease, as shown in (1) and (3). In the inverse transformation, the vector $[x', y', \mu']$ represents the homogeneous coordinates, where μ' is the scaling factor that allows the reduction of the image coordinates into two dimensions.

$$d = 2 \times R \times \arctan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right), \tag{4}$$

$$a = \sin^2\left(\frac{\Delta\text{lat}}{2}\right) + \cos(\text{lat}_1) \times \cos(\text{lat}_2) \times \sin^2\left(\frac{\Delta\text{lon}}{2}\right). \tag{5}$$

The current method effectively converts pixel data into geographical coordinates (see Figure 6). However, these raw data have limited utility for speed calculations, which are typically expressed in km/h (kilometres per hour). To transform this geographical information into a kilometre-based measurement, another technique must be used. The Haversine formula (4) is crucial for calculating the great-circle distance between two geographical points on the Earth's surface. To determine the actual distance, we calculate an intermediate value (5), which represents the central angle between the pair of points on the spherical surface. Finally, the arch distance is obtained by multiplying the Earth's mean radius (R) with the arc tangent function of the central angle.

$$v_{\text{avg}} = \frac{\sum_{i=1}^{n} \Delta d_i}{\sum_{i=1}^{n} \Delta t_i}. \tag{6}$$

This approach is especially advantageous in the context of speed estimation, as it provides instantaneous measurements between each pair of consecutive frames. These readings can be used afterwards to compute an average value. Our system processes the travelled distance between frames for each tracked vehicle and stores the elapsed time. After n readings, we compute the average speed value as shown in (6), where Δd_i is the distance increment during the i-th interval and Δt_i is the time increment during the i-th interval. This approach helps filtering noisy readings and stabilizing short-term fluctuations.

Other studies [4, 5] also calculate distances between GPS locations using the same Haversine formula, although the position readings come from IoT (Internet of Things) devices and smart phones installed directly on the vehicles, instead of computer vision-based methods. These IoT solutions claim high accuracies in speed estimation and no Internet connection needed for speed calculations, although they need to install a dedicated device on each vehicle.

Now that we have described the implemented technique in detail, it is pertinent to introduce our validation methodology for the geolocation and speed estimation system. We have employed an experimental setup (see Figure 7) that combines the use of two components. The first component consists of two inductive loop sensors embedded in the roadway. These sensors detect disruptions in the magnetic field as a vehicle passes over them, thereby enabling the measurement of the vehicle's speed. The second component of our system is a camera mounted on an overhead gantry prior to the location of the inductive loops. This camera is strategically aimed to cover the area just beyond the loops, focusing directly on the roadway; this angle provides Deep SORT [27] enough time to perform the tracking correctly.

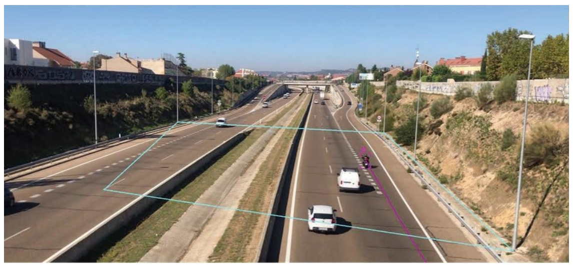
3.5. Hardware Selection. The developed system is based on the premise of not relying on cloud processing, so the software must be embedded in an edge computing device. To achieve this, we have explored various types of devices in the edge computing paradigm and have concluded that the Jetson family is among the most suitable options. To select the best choice within this Nvidia series we collected technical information from datasheets, as presented in Table 3.

We carried out a series of empirical tests to juxtapose the theoretical capabilities of edge computing devices with their actual performance in real-world scenarios. In our first test, each device was tasked with running a YOLOv5n [17] model on a two-minute traffic surveillance video captured at 25 frames per second (FPS) to perform object detection. The findings, depicted in Figure 8, indicate that the Jetson Nano falls short of the system requirements for our specific application.

To further evaluate Jetson Orin AGX and Jetson Xavier AGX, we conducted a second test running the complete system on a 2-hour traffic surveillance video. Each device had to detect vehicles using YOLOv5n, track them using Deep SORT, and estimate their GPS position and speed. The results of this experiment (Figure 9) show that, in terms of FPS stability, Jetson Orin AGX is the best option; it also presents the highest FPS mean rate (23.36 Frames per Second), with a maximum power consumption of 60W. On the other hand, Xavier shows a more pronounced fluctuation and lower FPS mean rate (22.77 Frames per Second) with a lower power consumption (maximum of 30W).

4. System Overview

This section provides a comprehensive overview of the system architecture, integrating the components discussed in Section 3 into a cohesive whole. Our system is designed to process a continuous stream of video data through a cascade of advanced algorithms, which collectively contribute to a reliable vehicle monitoring solution.


(a)

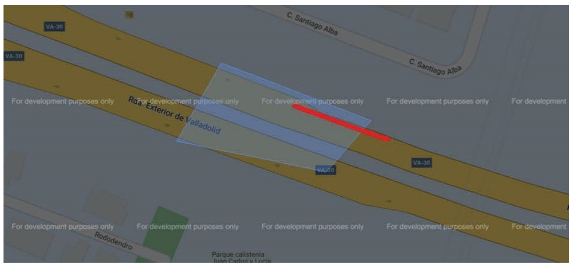

(b)

FIGURE 6: Geolocation technique [35] demonstration on motorbike trajectory. The image on (a) shows a motorbike being tracked, with its trajectory annotated in pink. In (b), the same motorbike is geolocated using the described technique on a map (©2024 Google, Inst. Geogr. National) and represented using gmplot. In both images, a blue polygon is annotated, whose corners are the calibration points used to obtain M in this example.

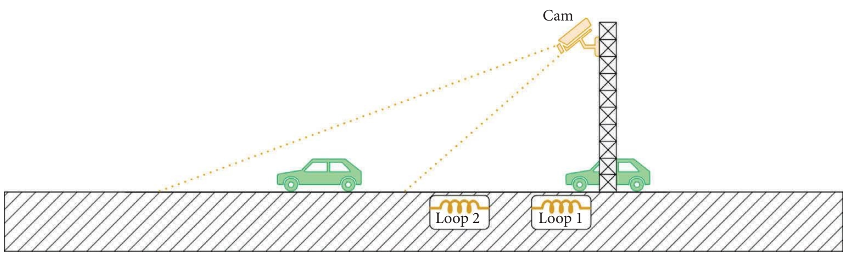


FIGURE 7: Speed validation setup. This build employs a pair of magnetic loops as ground truth and a camera mounted on a gantry to feed the vision system. Both systems record vehicles speeds, the inductive loops detect magnetic disruptions when vehicles drive over them, and the vision system can detect and track the same vehicles as they pass through. Readings are stored and can be compared to test the system accuracy.

TABLE 3: Jetson developer kit device specifications.

Jetson	GPU	CPU	Memory	Power (W)
AGX Orin	Ampere	12-core	64 GB LPDDR5	15–60
AGX Xavier	512-Core Volta	8-core	16 GB LPDDR4x	10–30
Nano	128-Core Maxwell	4-core	4 GB LPDDR4	5

As depicted in Figure 10, video input serves as the initial step in our system; this is achieved through a surveillance IP camera that supports RTSP (Real-Time Streaming Protocol). Then our pretrained YOLOv5n model performs vehicle detection and infers vehicle bounding boxes and classes, which are essential for the tracking phase. The vehicle bounding boxes and their associated data serve as input for the Deep SORT algorithm, which performs the tracking task. The tracker output then feeds into the geolocation module, which utilizes the tracking data to determine the precise GPS location of each vehicle. Finally, by applying the Haversine formula, the system calculates the distance travelled by each vehicle between frames, enabling accurate instantaneous and average speed estimations.

The deployment phase for our system utilizes the Nvidia Jetson Orin AGX 64 GB Developer Kit, chosen for its computational efficiency as justified previously. We use Docker to containerize the system (Figure 11), ensuring a consistent environment across various devices. Network port forwarding configuration is critical for our setup, necessary for RTSP video streaming, and allows for direct communication between the IP camera and the Docker container.

Figure 12 illustrates the entire system in action, showcasing its capabilities in detecting, tracking, and estimating speed. This figure depicts the integration of our key modules: the YOLOv5n model, for initial vehicle detection, Deep SORT algorithm, for robust vehicle tracking, and the visual geolocation module, for accurate speed estimation.

5. Results and Discussion

In this section, we present the results and discussions of our experiments, examining the performance of our vehicle detection models, geolocation validation experiments, and the detailed analysis of speed validation conducted.

5.1. Vehicle Detection Model. Previously, we outlined an iterative training approach and the importance of dataset refinement in enhancing vehicle detection performance. This method is exemplified in the YOLOv5n model progression on our test set, particularly with the YOLOv5n-4 version, as detailed in Table 2. The overall mAP50 increased to 0.954, demonstrating a substantial improvement over earlier versions. Furthermore, that same model achieved an overall mAP50-95 of 0.8, indicating a robust performance across various IoU thresholds. This metric improvement underscores the success of our training methodologies and

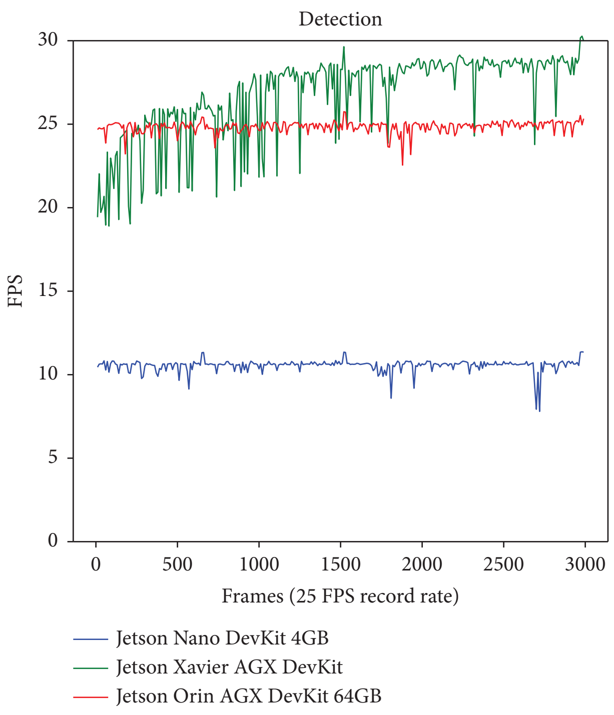


FIGURE 8: Jetson devices' performance running YOLOv5n [17] on a traffic surveillance video. The Jetson Orin AGX (in red) shows the most stable performance, while the Jetson Xavier AGX (in green) is occasionally the fastest; it is quite unstable as its frame rate oscillates strongly. Finally, the Jetson Nano (in blue), although a good option for low power consumption, does not deal well with real-time requirements.

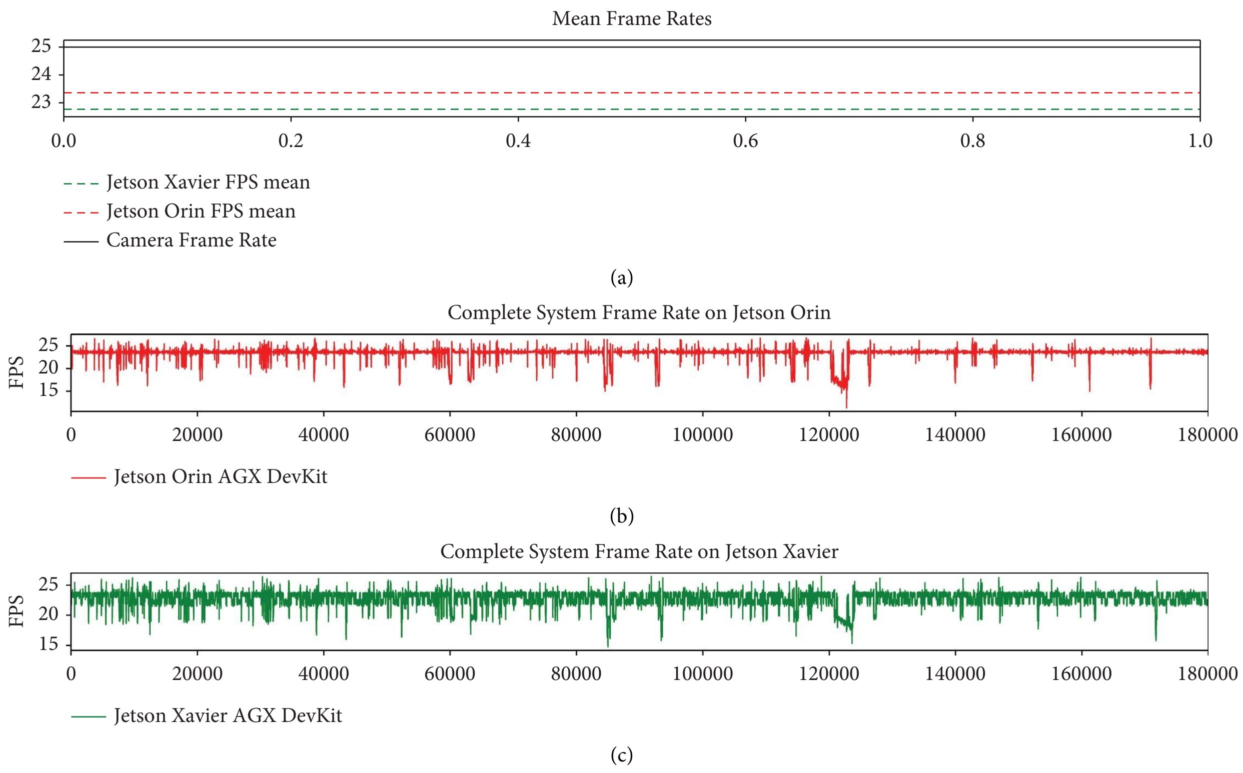


FIGURE 9: Jetson Orin AGX and Jetson Xavier AGX process 2-hour traffic surveillance video using the complete system (vehicle detection [17], tracking [27], geolocation [35], and speed estimation). Jetson Orin ((b) in red) is the most stable option and achieves the highest FPS mean rate ((a) in red). On the other hand, Jetson Xavier ((c) in green) is also a good option for this task, as it also achieves a high and consistent FPS mean rate ((a) in green) throughout the video.

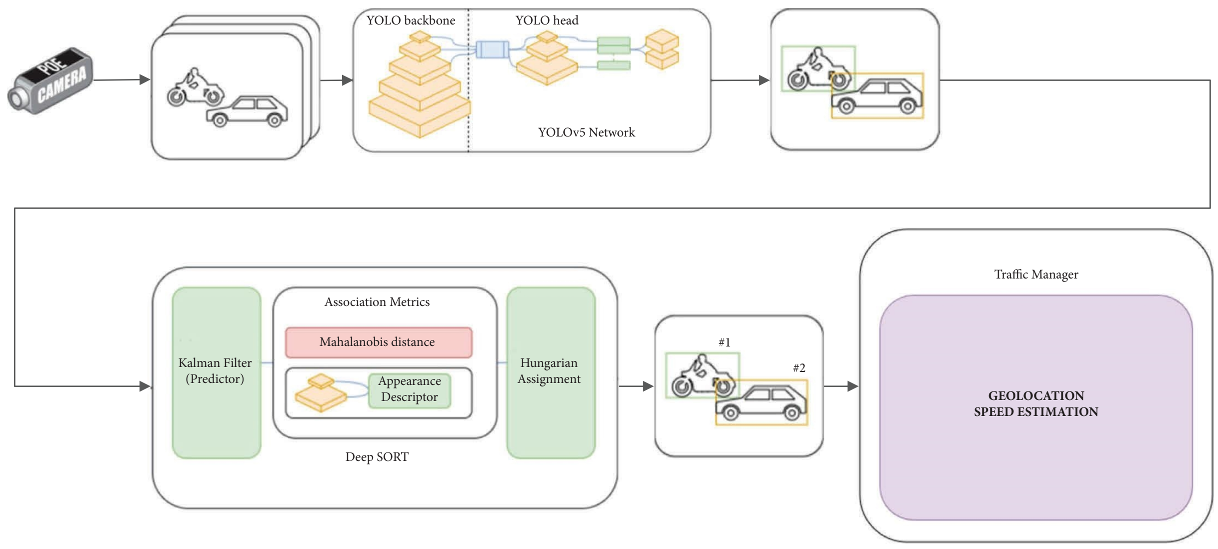


Figure 10: System architecture overview. This diagram shows the data processing pipeline. First the video frames are gathered using RTSP (Real-Time Streaming Protocol), and then they are processed by the YOLOv5n [17] model, which infers bounding boxes and vehicle classes. After that, Deep SORT [27] assigns target identifiers to each detection and finally the information is introduced in the traffic manager module. There, the vehicles are geolocated [35] and their instantaneous speeds are stored to provide average speed readings.

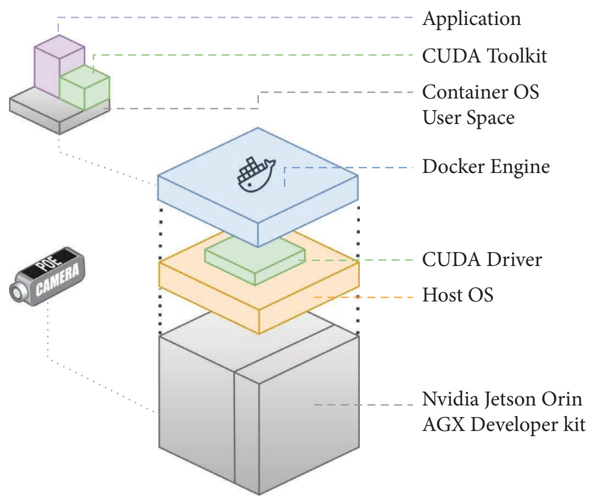


Figure 11: Software deployment. The algorithms are containerized using Docker and installed in the Jetson Orin AGX device, ensuring stability and reducing installation and deployment efforts to a minimum.

the quality of the refined dataset in achieving a high-quality object detection system.

Class-specific performance further demonstrates the YOLOv5n-4 model's enhanced detection capabilities. Noteworthy achievements include mAP50 scores of 0.989 for motorbikes, 0.960 for cars, 0.981 for vans, 0.997 for buses, 0.985 for trucks, 0.993 for heavy trucks, and 0.978 for heavy articulated trucks. Respectively, mAP50-95 scores for these classes were 0.762, 0.714, 0.810, 0.892, 0.878, 0.708, and 0.837. As we already discussed, there is a noticeable variation in accuracies for each vehicle type. This happens because some vehicles are more common than others, and their unique visual characteristics are also difficult to distinguish sometimes. All these factors influence model's biases and affect directly on classification performance.

In our training procedures, we stated that we employed artificial rain and snow data augmentation on our original images. To test our model on real data, we employed the DAWN dataset [39], which features rain, snow, fog, and sandstorm traffic scenarios. We adapted labels to match our vehicle classification distribution. We assumed bicycles and motorbikes to be the same vehicle class and removed pedestrian annotations.

Finally, we performed validation on the rain, fog, and snow images, obtaining Table 4 results. The overall mAP50-95 metrics are 0.332 with snow, 0.404 with fog, and 0.397 with rain. Although the results are modest, they underscore the model's ability to generalize in low visibility conditions, as many DAWN images [39] feature vehicles that are difficult to discern with the naked eye.

5.2. Geolocation Validation Experiment

5.2. Geolocation Validation Experiment. To validate the geolocation technique implemented in our system, we have conducted two types of experiments. On the one hand, we used elements of the road to manually select their pixel coordinates in the image and geolocate them. Then we obtained the ground truth GPS coordinates via satellite imagery and measured the error using Haversine. An example of this type of experiment can be observed in Figure 13, where 19 points corresponding to road paint symbols are used as key points to geolocate, and all of them are approximately 35 m away from the camera. The results of this experiment are shown in Table 5, with a maximum error of 0.85 m and a mean error of 0.534 m. Considering that, according to the Official U.S. Government information about the GPS [40], mobile devices have an accuracy of a 4.9 m radius under open sky, solutions like [5] or [4] do not seem to provide the same reliability in exact position accuracy.

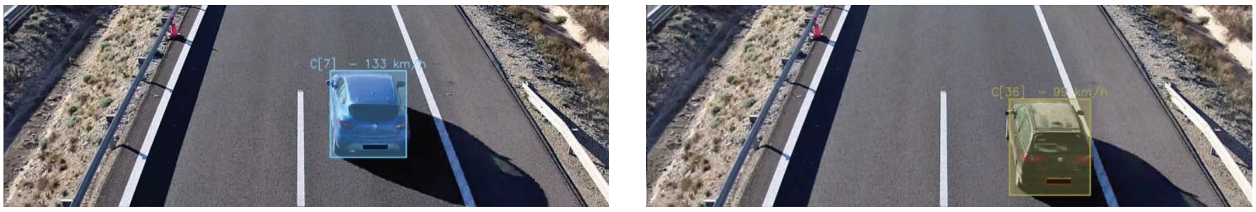


Figure 12: Complete system showcase. The system performs vehicle detection, tracking, geolocation, and speed estimation. Bounding boxes contain vehicle class information, tracker identifier, and speed in km/h.

Table 4: YOLOv5n [17] validation on DAWN dataset [39].

Class	Snow		Fog		Rain	
	Instances	mAP50-95	Instances	mAP50-95	Instances	mAP50-95
All	1917	0.332	1831	0.404	1559	0.397
M	3	0.206	35	0.249	4	0.295
C	1710	0.518	1521	0.558	1327	0.548
V	66	0.268	51	0.355	27	0.302
B	30	0.503	66	0.435	12	0.348
T	25	0.200	22	0.385	37	0.285
HT	76	0.494	134	0.444	148	0.404
HAT	7	0.084	2	0.400	4	0.598

This dataset consists of traffic scenarios under different inclement weather situations, such as snow, fog, or rain.

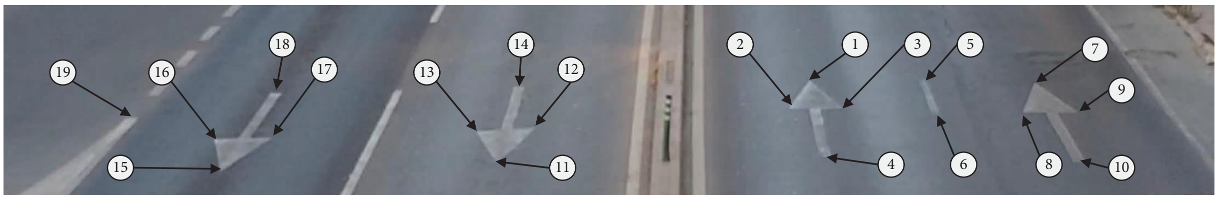


Figure 13: Validation of key point geolocation on road symbols. This experiment assesses the error in GPS position estimation using the previously described vision-based method. The image illustrates various points, corresponding to road lines and directional arrows on the asphalt, which have been attempted to be localized using this method.

Table 5: Key point geolocation results over road symbols.

Sample	Measured latitude	Measured longitude	Real latitude	Real longitude	Error (m)
1	36.838143	−2.446986	36.838144	−2.446992	0.5342
2	36.838141	−2.447006	36.838141	−2.447010	0.3450
3	36.838134	−2.447004	36.838133	−2.447006	0.3603
4	36.838126	−2.447039	36.838121	−2.447042	0.6226
5	36.838130	−2.446978	36.838130	−2.446984	0.4588
6	36.838121	−2.447005	36.838121	−2.447006	0.0520
7	36.838114	−2.446978	36.838110	−2.446980	0.5170
8	36.838111	−2.446998	36.838110	−2.447000	0.2407
9	36.838105	−2.446998	36.838100	−2.447000	0.5795
10	36.838097	−2.447033	36.838090	−2.447030	0.8570
11	36.838161	−2.447055	36.838160	−2.447060	0.4550
12	36.838164	−2.447033	36.838158	−2.447037	0.8119
13	36.838170	−2.447038	36.838167	−2.447041	0.5122
14	36.838179	−2.447003	36.838176	−2.447009	0.6692
15	36.838190	−2.447072	36.838186	−2.447073	0.4663
16	36.838193	−2.447050	36.838187	−2.447051	0.7188
17	36.838207	−2.447020	36.838200	−2.447021	0.7229
18	36.838200	−2.447054	36.838195	−2.447057	0.6142
19	36.838216	−2.447044	36.838211	−2.447045	0.6193

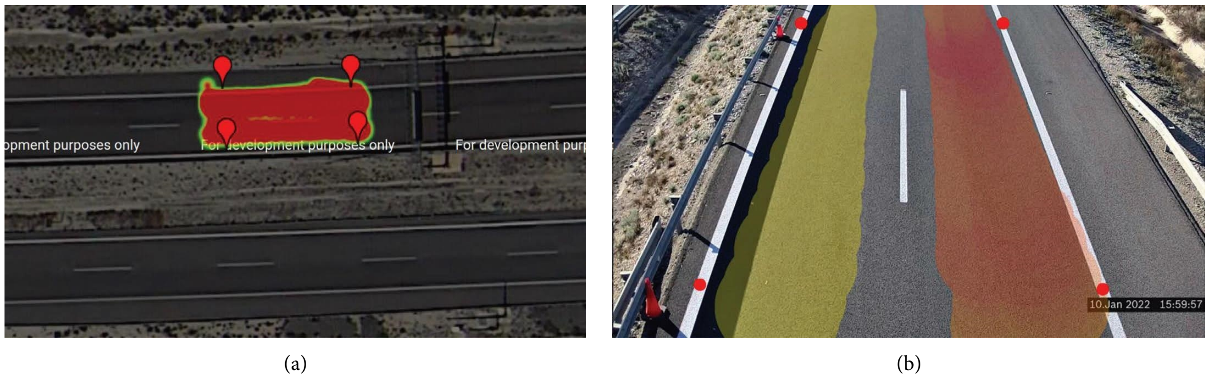


FIGURE 14: Geolocation visual validation experiment. On (a), a heatmap is represented on a satellite map (©2024 Google ©2024 Airbus) along with the four calibration landmarks (in red). On (b), the same heatmap and landmarks are represented on the camera perspective.

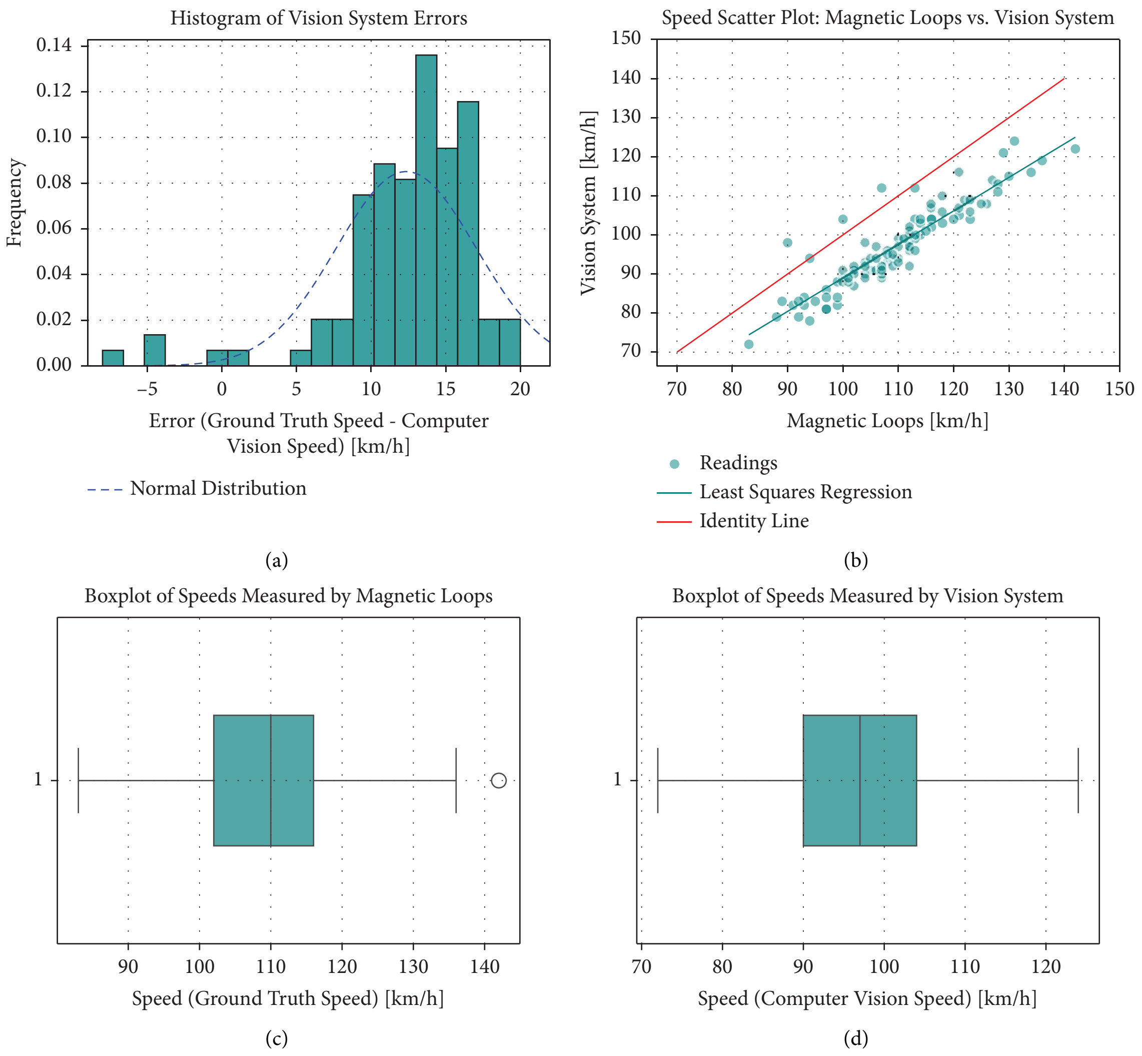


FIGURE 15: Speed validation experiment results. On (a) an error histogram is represented. On (b) the actual readings are compared between both strategies. On (c, d) both box graphs represent data distribution for each system's readings.

On the other hand, to further demonstrate the geolocation capabilities of the system, we have employed the scenario described in Figure 7. We performed an experiment that consisted of visual verification, by constructing a heatmap in both satellite map and video camera image format, using the vehicle detection obtained from a one-hour video. This experiment is depicted in Figure 14, which offers a comparative visualization of the same heatmap overlaid on two distinct coordinate systems. The left image showcases the heatmap plotted against geographical coordinates, providing a macroscopic view of the area of interest on a satellite map. The right image displays the same heatmap in relation to pixel coordinates, showing the camera perspective. Both heatmaps visually reinforce the numerical accuracy demonstrated in the experiment results of Table 5.

5.3. Speed Validation Experiment. Utilizing the experimental setup outlined in Figure 7, we have validated our implementation of the velocity estimation system. As already mentioned, the validation experiment scenario included two inductive loops capable of measuring the speed of vehicles travelling on both lanes in the same direction and a video camera connected to our edge computing unit.

The quantitative analysis of the velocity estimation system was conducted on a sample of approximately 100 vehicles, revealing several key statistics, which are depicted in the histogram of Figure 15: a mean absolute error of 12.68 km/h, a median absolute error of 13.00 km/h, a minimum absolute error of 0.00 km/h, and a maximum absolute error of 20.00 km/h. Correspondingly, the mean percentage error was 11.58%, median percentage error was 11.88%, minimum percentage error was 0.00%, and maximum percentage error was 17.86%.

In [5], a 96.08% accuracy was obtained using smartphones to estimate speed, while our experiments obtained a mean 88.42% accuracy. It is important to state that speed readings in [4, 5] are low speeds ranging from 40 to 70 km/h, computed and obtained from devices installed inside the vehicles. On the other hand, our implementation of [35] employs computer vision to gather readings ranging from 70 to 120 km/h.

The speed scatter plot of Figure 15 suggests a systematic underestimation of vehicle speeds by the proposed vision system. Despite a clear positive correlation between the two systems, the data spread widens with increasing speed, hinting at a reduction in accuracy potentially due to technical constraints like motion blur or sampling rate limitations. The boxplots provide a visual summary of the distribution of speeds as measured by the magnetic loop system and the computer vision system. The boxplot corresponding to the computer vision solution shows a wider interquartile range than the magnetic loop solution boxplot, implying greater variability in the speed measurements.

6. Conclusions

In this study, we successfully developed an edge computing platform designed for real-time traffic monitoring, meeting the objectives set forth at the introduction of this article. Our integrated system architecture, combining a YOLOv5 [17] network for vehicle detection and the Deep SORT [27] algorithm for tracking, demonstrates a cohesive and efficient approach to vehicle monitoring.

Regarding the object detection model, the accuracy requirements for localization and classification have been successfully met in most scenarios. Conversely, although the model has been able to generalize, the metrics obtained on DAWN [39] are not sufficient for robust vehicle detection in poor weather conditions. We managed to handle short-term occlusions but struggled with reidentification of overlapping bounding boxes of similar appearance vehicles. Also, long-term occlusions represent a huge challenge, which typically happen when large trucks or buses occlude cars, vans, or motorbikes.

Our geolocation validation experiments provide critical insights into the performance and limitations of the system. While we have obtained positioning errors in the centimetre scale (see Table 5), we still have high underestimation errors in speed inference experiments. It is also important to consider that well-calibrated inductive loop systems exhibit a margin of error ranging from 3 to 5% [41].

This solution has also met soft real-time requirements, with an average frame rate of 23.36 FPS and 22.77 FPS achieved on the Jetson Orin AGX and the Jetson Xavier AGX devices, respectively.

In conclusion, our research has established a robust foundation for real-time traffic monitoring using edge computing, with successful demonstrations in vehicle detection, tracking, geolocation, and speed estimation.

6.1. Future Work. Our future work will address enhancing the vehicle detection network's capability. This will involve improving the dataset used for training the model. Specifically, we aim to enrich the dataset with nighttime images, which are crucial for ensuring the robustness of the system under low-light conditions. We will also focus our efforts on gathering real rain, snow, fog, and even sandstorm situations to strengthen model's performance. Exploring the latest YOLO versions, such as YOLOv8 [20] and YOLO-NAS [21], holds potential for significant improvements, offering advancements in accuracy and inference latency.

Secondly, we plan to upgrade the tracker by integrating advanced algorithms like BoT SORT [30] and Byte Track [29]. These trackers have demonstrated improved performance in complex scenarios, which is essential for enhancing the overall accuracy and reliability of our system, especially in dynamic and challenging traffic environments.

Improving speed estimation accuracy is particularly crucial, given the identified limitations in our current system. To address this, we plan to refine our visual geolocation techniques and ensure better geographical and image calibration. Also, a third speed measurement system should be utilized for better contrast when validating the vision system along with ground truth methods.

Given the demonstrated accuracy of the implemented geolocation technique (see Table 5), a new research path could focus on vehicle dimension estimation. Additionally, we aim to expand our analysis to encompass more traffic parameters, which includes the direction of vehicle movement, enabling the detection of wrong-way drivers (Kamikaze), identifying the presence of vehicles in specific lanes within the roadway, along with other factors, to provide a more comprehensive understanding of traffic behaviour and patterns.

Data Availability

Due to confidentiality agreements between the participating organizations in this work, there are strict limitations on the dissemination of the dataset, models, and code developed during this research. As a result, all the resources associated with this study are not available for public access.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was funded by MCIN/AEI/10.13039/501100011033, Grant PID2022-140554OB-C32 and PDC2022-133684-C31. Grant Project Movilidad 2030 funded by Centre for the Development of Industrial Technology (CDTI). Open Access funding was enabled and organized by CRUE-Madroño 2024.

References

[1] W. Balid, H. Tafish, and H. H. Refai, "Intelligent vehicle counting and classification sensor for real-time traffic surveillance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1784–1794, 2018, https://api.semanticscholar.org/CorpusID:44160492.

[2] S. Taghvaeeyan and R. Rajamani, "Portable roadside sensors for vehicle counting, classification, and speed measurement," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 1, pp. 73–83, 2014.

[3] J. Liu, Q. Sun, Z. Fan, and Y. Jia, "TOF lidar development in autonomous vehicle," in *Proceedings of the 2018 IEEE 3rd Optoelectronics Global Conference (OGC)*, pp. 185–190, Shenzhen, China, September 2018, https://api.semanticscholar.org/CorpusID:53283382.

[4] Y. A. Daraghmi, M. A. Helou, E. Y. Daraghmi, and W. Abu-ulbeh, "IoT-based system for improving vehicular safety by continuous traffic violation monitoring," *Future Internet*, vol. 14, no. 11, p. 319, 2022.

[5] Y. A. Daraghmi, "Vehicle speed monitoring system based on edge computing," in *Proceedings of the 2021 International Conference on Promising Electronic Technologies, ICPET 2021*, pp. 14–18, Institute of Electrical and Electronics Engineers Inc, Deir El-Balah, Palestine, State of, November 2021.

[6] J. M. Collado, C. Hilario, A. de la Escalera, and J. M. Armingol, "Model based vehicle detection for intelligent vehicles," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 572–577, Parma, Italy, June 2004.

[7] I. Sina, A. Wibisono, A. Nurhadiyatna, B. Hardjono, W. Jatmiko, and P. Mursanto, "Vehicle counting and speed measurement using headlight detection," in *Proceedings of the 2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pp. 149–154, Sanur Bali, Indonesia, September 2013.

[8] D. Luvizon, B. Nassu, and R. Minetto, "Vehicle speed estimation by license plate detection and tracking," in *Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing- Proceedings*, Florence, Italy, April 2014.

[9] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013, http://arxiv.org/abs/1311.2524.

[10] R. B. Girshick, "Fast R-CNN," 2015, http://arxiv.org/abs/1504.08083.

[11] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," 2015, http://arxiv.org/abs/1506.01497.

[12] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only Look once: unified, real-time object detection," 2015, http://arxiv.org/abs/1506.02640.

[13] J. Redmon, "Darknet: open source neural networks in C," 2013, https://pjreddie.com/darknet/.

[14] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," 2016, http://arxiv.org/abs/1612.08242.

[15] J. Redmon and A. Farhadi, "YOLOv3: an incremental improvement," 2018, http://arxiv.org/abs/1804.02767.

[16] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: optimal speed and accuracy of object detection," 2020, https://arxiv.org/abs/2004.10934.

[17] G. Jocher, "Ultralytics YOLOv5," 2020, https://hub.docker.com/r/ultralytics/yolov5.

[18] C. Li, "YOLOv6: a single-stage object detection framework for industrial applications," 2022, https://arxiv.org/abs/2209.02976.

[19] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing network design strategies through gradient path analysis," *Journal of Information Science and Engineering*, 2023.

[20] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023, https://github.com/ultralytics/ultralytics.

[21] A. Shay, "Super-gradients YOLO-NAS," 2021, https://www.supergradients.com/.

[22] J. Terven, D. Cordova-Esparza, and J. A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: from YOLOv1 to YOLOv8 and YOLO-NAS," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023.

[23] Ultralytics, "YOLOv8- Ultralytics YOLOv8 docs," https://docs.ultralytics.com/models/yolov8/#overview.

[24] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," 2016, http://arxiv.org/abs/1602.00763.

[25] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[26] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, 1955.

[27] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3645–3649, Beijing, China, September 2017.

[28] R. De Maesschalck, D. Jouan-Rimbaud, and D. L. Massart, "The Mahalanobis distance," *Chemometrics and Intelligent Laboratory Systems*, vol. 50, no. 1, pp. 1–18, 2000.

[29] Y. Zhang, "ByteTrack: multi-object tracking by associating every detection box," 2021, https://arxiv.org/abs/2110.06864.

[30] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, "BoT-SORT: robust associations multi-pedestrian tracking," 2022, https://arxiv.org/abs/2206.14651.

[31] M. Jalalat, M. Nejati, and A. Majidi, “Vehicle detection and speed estimation using cascade classifier and sub-pixel stereo matching,” in *Proceeding of the 2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*, pp. 1–5, Tehran, Iran, December 2016.

[32] S. Javadi, M. Dahl, and M. Pettersson, “Vehicle speed measurement model for video-based systems,” *Computers & Electrical Engineering*, vol. 76, pp. 238–248, 2019.

[33] J. Collado, C. Hilario, A. de la Escalera, and J. M. Armingol, “Adaptative road lanes detection and classification,” in *Lecture Notes in Computer Science*, Springer, Bali, Indonesia, 2006.

[34] C. Maduro, K. Batista, P. Peixoto, and J. Batista, “Estimating vehicle velocity using rectified images,” in *Proceedings of the Third International Conference on Computer Vision Theory and Applications*, pp. 551–558, San Diego, CA, USA, October 2008.

[35] S. Blake, “How to track objects in the real world with TensorFlow, SORT, and OpenCV,” *The Medium: HAL24K TechBlog*, 2019, https://medium.com/hal24k-techblog/how-to-track-objects-in-the-real-world-with-tensorflow-sort-and-opencv-a64d9564ccb1.

[36] U. Saxena, “Image Augmentation: make it rain, make it snow. How to modify photos to train self-driving cars,” *freeCodeCamp*, 2018.

[37] D. Guttendorf, “UGV-UAV object geolocation in unstructured environments,” 2022, https://arxiv.org/abs/2201.05518.

[38] M. Ulicny, V. A. Krylov, J. Connelly, and R. Dahyot, “Combining geolocation and height estimation of objects from street level imagery,” 2023, https://www.researchgate.net/publication/370775409_Combining_geolocation_and_height_estimation_of_objects_from_street_level_imagery.

[39] M. Kenk, *Dawn*, Mendeley, London, UK, 2020.

[40] GPS, “Official U.S. government information about the Global Positioning System (GPS) and related topics,” https://www.gps.gov/systems/gps/performance/accuracy/.

[41] D. L. Woods, B. P. Cronin, and R. A. Hamm, “Speed measurement with inductance loop speed traps,” 1994, https://api.semanticscholar.org/CorpusID:107074922.