A Modified Lightweight DeepSORT Variant for Vehicle Tracking

Ayoub El-alami[1], Younes Nadir[2], Khalifa Mansouri[3]
Signaux, systèmes distribués et Intelligence Artificielle (M2S2I), Equipe Systèmes Informatiques Distribués (SID), Ecole Normale Supérieure de l'Enseignement Technique (ENSET), Université Hassan II de Casablanca (UH2C), Mohammedia, Morocco[1, 3]
Signaux, systèmes distribués et Intelligence Artificielle (M2S2I), Équipe Technologies de l'Information et Intelligence Artificielle (T2IA), Ecole Nationale Supérieure de l'Art et de Design (ENSAD), Université Hassan II de Casablanca (UH2C), Mohammedia, Morocco[2]

***Abstract*—Object tracking plays a pivotal role in Intelligent Transportation Systems (ITS), enabling applications such as traffic monitoring, congestion management, and enhancing road safety in urban environments. However, existing object tracking algorithms like DeepSORT are computationally intensive, which hinders their deployment on resource-constrained edge devices essential for distributed ITS solutions. Urban mobility challenges necessitate efficient and accurate vehicle tracking to ensure smooth traffic flow and reduce accidents. In this paper, we present a modified lightweight variant of the DeepSORT algorithm tailored for vehicle tracking in traffic surveillance systems. By leveraging multi-dimensional features extracted directly from YOLOv5 detections, our approach eliminates the need for an additional convolutional neural network (CNN) descriptor and reduces computational overhead. Experiments on real-world traffic surveillance data demonstrate that our method reduces tracking time to 25.29% of that required by DeepSORT, with only a minimal increase over the simpler SORT algorithm. Additionally, it maintains low error rates between 0.43% and 1.69% in challenging urban scenarios. Our lightweight solution facilitates efficient and accurate vehicle tracking on edge devices, contributing to more effective ITS deployments and improved road safety.**

Keywords*—Distributed systems; intelligent transportation systems; edge computing; object tracking

I. Introduction

In the era of smart cities, integrating distributed Intelligent Transportation Systems (ITS) with edge computing marks a significant advancement in urban mobility. By processing data closer to the source, using embedded devices with traffic cameras rather than relying solely on centralized servers, ITS can enable real-time decision-making and responsiveness, even in resource-constrained environments. This decentralized approach improves system scalability, reduces latency, and enhances data privacy by processing sensitive information locally. Multiple Object Tracking (MOT), a core component of ITS, refers to tracking the trajectories of multiple objects across video frames. Recent advances in object detection and deep learning have made tracking-by-detection the dominant approach in MOT [1]. This approach formulates MOT as a data association task, linking newly detected objects with those already being tracked.

Recent Multiple Object Tracking (MOT) systems are generally composed of three key components: the object detector, the embedding model, and the data association algorithm. The object detector localizes and identifies objects of interest within each frame, commonly using deep learning models such as YOLO [1], and Faster R-CNN [2]. The embedding model extracts representative features from the detected objects to capture their appearance characteristics. These embeddings combined with spatiotemporal parameters, are used to associate objects across frames and maintain consistent track identities. Finally, the data association algorithm links objects based on their appearance and spatiotemporal similarities. This is typically achieved through methods such as the Hungarian algorithm, as used in DeepSORT [3], or via deep learning-based techniques like Siamese networks or recurrent neural networks [4].

Current object detection and tracking algorithms, notably YOLOv5 and DeepSORT, have achieved high accuracy in multi-object tracking tasks. However, their computational intensity poses significant challenges for deployment on resource-constrained edge devices essential for distributed ITS. This limitation creates a challenge for the practical application of ITS, where real-time processing is crucial for urban mobility and road safety enhancements. To bridge this gap, we propose a lightweight variant of the DeepSORT algorithm that reduces computational overhead while maintaining tracking accuracy, making it suitable for edge computing environments [5], [6], [7], [8], [9], [10].

In this work, we propose a lightweight tracking algorithm that follows a similar workflow to DeepSORT, with a key modification in the appearance embedding step. Traditionally, this step relies on a dedicated CNN model as a descriptor to calculate appearance similarity between objects. While effective, CNN descriptors are computationally intensive, particularly for resource-constrained devices. To enhance the computational efficiency of the tracking process, we eliminate the CNN descriptor and introduce an alternative strategy for object appearance extraction and association.

Our strategy replaces the appearance features provided by the embedding model with data directly derived from the feature map generated during the initial detection phase. These feature maps, produced by the intermediate layers of the detector,

contain high-level abstract representations of the objects. Experiments demonstrate that these representations are a viable substitute for CNN descriptor features when calculating appearance similarity, reducing computational overhead without sacrificing tracking accuracy.

The rest of this paper is structured as following. Section II discusses related work in object detection and multi-object tracking. In Section III, we present our proposed tracking approach, starting with its background methods (i.e. YOLOv5 and DeepSORT), followed by a detailed explanation of our lightweight DeepSORT variant, including the modified detection and appearance similarity processes. Section IV describes our experimental setup and shows the results. In conclusion section, we summarizing our contributions and suggesting future research directions.

II. Related Work

In this section, we present a review of key works related to object detection and multi-object tracking.

A. Object Detection

Since the introduction of deep convolutional neural networks (CNNs), numerous object detection approaches have emerged, leveraging large open datasets such as COCO [11] and Pascal VOC [12]. Early detectors followed a two-stage process, beginning with region proposal generation, followed by object classification and refinement. Models like Fast R-CNN [13] and Faster R-CNN enhanced both speed and accuracy by improving feature extraction and region proposal methods. Despite their effectiveness, two-stage detectors often face computational inefficiencies due to their sequential processing structure.

The introduction of YOLO (You Only Look Once) [1] in 2016 marked a significant shift towards one-stage detectors, which focus on simultaneous detection and classification. Subsequent versions of YOLO have made considerable strides in performance, particularly in real-time applications, by refining this one-stage detection approach. Recent YOLO versions, such as YOLOv5, YOLOv7-tiny, YOLOv8s, and YOLOv9s, offer lightweight models optimized for devices with limited computational power, making them well-suited for edge devices and real-time object detection tasks [15], [16], [17].

B. Multi-Object Tracking

In recent years, tracking-by-detection (TBD) has emerged as the dominant paradigm in Multiple Object Tracking (MOT), driven by the increasing capabilities of object detectors. In the TBD approach, tracking begins with the detection of target objects in each frame, followed by the extraction of image crops and spatial parameters based on the objects' bounding boxes. Tracking is then achieved by establishing correspondences between objects in consecutive frames using the extracted data. To facilitate this, a similarity matrix is constructed using parameters such as distance, IoU (Intersection over Union), and appearance features. Algorithms like the Hungarian algorithm or greedy algorithm are commonly used to solve this assignment problem.

To further improve tracking performance, state estimation algorithms are employed to predict the positions of tracked objects based on their spatiotemporal characteristics. Notable state estimation methods include Kalman filters, particle filters, and Gaussian processes. Two of the most widely recognized tracking algorithms in MOT are SORT (Simple Online and Realtime Tracking) [18] and DeepSORT.

SORT is an efficient algorithm that uses a Kalman filter for state estimation and Hungarian matching for data association. It focuses solely on objects' positions, making it highly effective in real-time scenarios. However, in more challenging conditions, such as complex motion patterns, occlusions or missed detections, SORT may produce incorrect associations, leading to identity switches and reduced tracking robustness. To address these limitations, DeepSORT extends SORT by incorporating cascade matching for enhanced data association. In addition to spatial similarity, DeepSORT uses appearance similarity to refine associations between detected objects and existing tracks, improving tracking robustness in more complex environments [8], [9], [14], [19].

III. Proposed Tracking Approach

Our proposed approach consists of an optimized tracking algorithm based on DeepSORT, designed to run efficiently in terms of processing time while maintaining high tracking accuracy. Our main contribution is the development of a modified tracking algorithm based on YOLOv5 and DeepSORT, incorporating novel methods for appearance feature extraction and appearance similarity calculation. This innovation allows us to eliminate the need for the CNN model traditionally used in DeepSORT for the appearance-embedding task. By removing the CNN model, we significantly reduce the vehicle tracking time, making the algorithm more suitable for deployment on devices with limited computational resources.

A. Background

1) YOLOv5: In our implementation, we utilized the publicly available YOLOv5s detector, trained on the COCO dataset, with an input image size of 416 × 416. YOLOv5 is a widely-used single-stage object detector, known for its clear and flexible architecture, offering high precision and speed. It is available in five different scales: N, S, M, L, and X, representing Nano, Small, Medium, Large, and Xlarge models, respectively. Each variant maintains the same overall structure, scaling the depth and width to improve detection performance.

A key feature of the YOLO architecture is its grid-based approach, which divides the input image into spatial cells. This division allows YOLOv5 to perform object detection at multiple spatial resolutions simultaneously, enabling the model to detect objects of varying sizes and aspect ratios across the image. Each cell is responsible for predicting the bounding box coordinates and associated class probabilities for objects within its spatial region, as illustrated in Fig. 1. For our experiments, we chose the small model (YOLOv5s) due to its optimal trade-off between precision and speed.

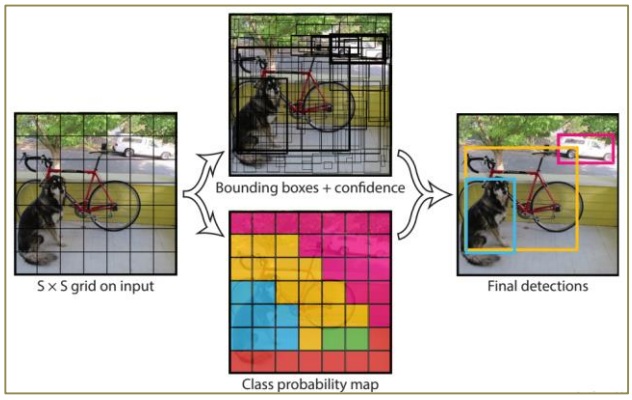

Fig. 1. Example of multiple steps in grid-based approach used in YOLO.

As shown in Fig. 2, the three main components of the YOLOv5 architecture typically include:

- Backbone: The backbone is responsible for extracting features from the input image and is composed of multiple convolutional layers arranged hierarchically. These layers progressively capture higher-level features from the image, enabling the detection of objects of interest by analyzing patterns, edges, and textures at different levels of abstraction.
- Neck: Positioned between the backbone and head, the neck is responsible for additional feature fusion and refinement. Its primary function is to aggregate data extracted at various scales and levels of abstraction by the backbone, ensuring that the model can effectively capture both spatial and contextual information. This step enhances the overall detection performance by combining fine details and broader contextual features before passing them to the detection head.
- Head: The head receives the refined feature maps from the neck and processes them to generate the final detection results. Its layers are designed to predict bounding boxes, objectness scores, and class probabilities, which are used to determine the locations, presence, and categories of objects within the image.

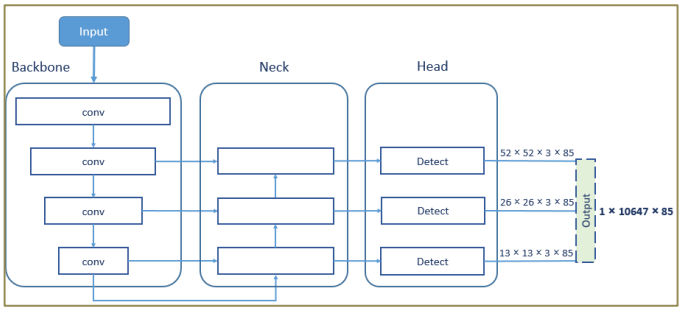

Fig. 2. Abstract overview of YOLOv5 default structure with an input size of 416×416.

2) DeepSORT: DeepSORT builds on the efficiency of the traditional SORT algorithm while incorporating the discriminative power of deep appearance features. Its workflow integrates object detection, feature extraction, data association, and track management to deliver robust tracking results. The DeepSORT workflow can be summarized as follows:

- Detection: The process starts with detecting objects in each frame using an object detector, providing bounding boxes around the detected objects, which serve as input for the tracking process.
- State Estimation: The Kalman Filter (KF) is employed for state prediction, estimating the future states of detected objects based on their current states. This estimation is important for the tracking of each object across frames, especially in cases where an object's detection might be shortly missed.
- Similarity measurement: Multiple metrics are used to calculate object similarity. Mahalanobis distance measures the spatial relationship between predicted states and new detections, while appearance descriptors assess visual similarity. Intersection Over Union (IoU) evaluates the overlap between bounding boxes. These metrics are combined to ensure accurate object association for effective tracking.
- Data Association: The Hungarian algorithm is used to associate detected objects with existing tracks, based on predicted states from the Kalman filter and appearance similarities. This assignment problem combines two metrics: object location and appearance information. DeepSORT's cascade matching prioritizes more frequently seen objects, reducing unstable tracks caused by missed detections. In the final stage, an IoU association is performed for any remaining unmatched targets.
- Track Management: After data association, tracks are updated with their associated detections. New tracks are initialized for unassociated detections, and existing tracks are updated for the next frame using the Kalman filter.

B. Proposed Lightweight Deep SORT Variant

To implement our lightweight variant of DeepSORT, we modified existing modules and introduced new ones to optimize the overall tracking process. The key adjustments made to transition from DeepSORT to our lightweight version include:

- Simultaneous detection and appearance feature extraction: We streamlined the detection process by enabling the simultaneous extraction of both detection data and appearance features. This eliminates the need for a separate embedding model, significantly reducing the computational overhead.
- Adaptation of the appearance similarity process: We adapted the appearance similarity calculation to match the new format of the features obtained from the detection phase. This adjustment ensures seamless integration with our new feature extraction process.

In the following, we provide a detailed explanation of the key adjustments made to our algorithm.

1) Modified detection process: Originally, DeepSORT employs two CNN models: the first functions as an object detector, returning bounding boxes around targets, while the second is a simple CNN descriptor that generates appearance features for those bounding boxes. As mentioned earlier, our approach eliminates the need for a separate CNN model. To maintain appearance association in our algorithm without using a CNN descriptor, we replace the appearance features generated by this descriptor with feature map data associated with the bounding boxes. These feature maps are extracted by YOLOv5's backbone during the detection phase. To extract both feature map data and detection data simultaneously, we implemented several key steps (as shown in Fig. 3), including:

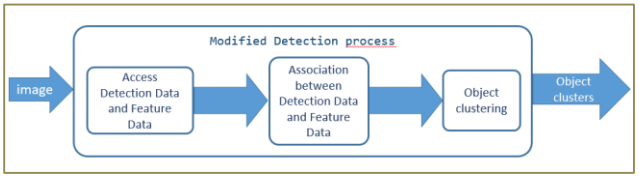

Fig. 3. Workflow of our proposed tracking system.

a) Access detection data and feature data from YOLO: This step involves modifying the model inference process to access intermediate layers of the YOLOv5 architecture, which contain feature map data, in addition to the final detection layer (as shown in Fig. 4). This modification produces an output that includes both detection data and feature map data.

- Detection Data: The detection output includes bounding boxes and class probabilities for small, medium, and large objects, aggregated into a tensor of shape (1, 10647, 85). Specifically, the dimensions 52×52×3×85, 28×28×3×85, and 13×13×3×85 correspond to the detection outputs for small, medium, and large objects, respectively. Here, the grid cell size (e.g., 52×52) refers to the resolution of the detection, the number 3 represents the number of anchor boxes per grid cell, and the 85 elements represent the prediction output for each detection.
- Feature Map Data: The feature map output consists of three tensors representing the features extracted from the backbone layers for each object size. These feature maps are organized in grids of vectors, where each vector is a feature vector for a grid cell. The backbone generates three different vector sizes (128, 256, and 512) for small, medium, and large objects, respectively.

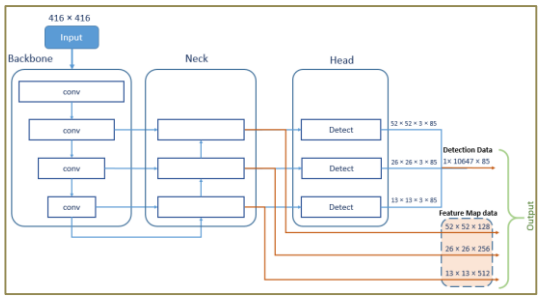

Fig. 4. A representation of the output data extracted from YOLOv5 layers in our algorithm.

b) Association between detection data and feature data: After the model makes its predictions, an association step is performed to map each detected object to its corresponding grid cell in the feature maps to identify its appearance feature. This process involves identifying the spatial locations of object centers within the grid. Since YOLOv5 uses different grid sizes for detection (small (26x26), medium (52x52), and large (13x13)), each object may be represented by multiple detection instances across different grids. Each detection instance is associated with a vector of dimension 85, which includes the object's bounding box coordinates and class information. The feature map data from the grid cell that contains the center of the bounding box is used to represent the object (Fig. 5). Subsequently, the grid cells are mapped to bounding boxes where the box center lies within the grid cell region. This step ensures that all relevant backbone feature vectors are accessible for each detected object.

c) Object clustering: Typically, a non-maximum suppression (NMS) process is applied to remove redundant bounding boxes and retain only the most confident detection. However, in our approach, we aim to extract both bounding box and feature map data. To achieve this, we replace NMS with an Object Clustering (OC) module. The OC module works similarly to NMS, using the IoU (Intersection over Union) metric to compare object overlap. However, instead of eliminating redundant bounding boxes, the OC clusters detections of the same object. This approach allows us to retain all associated feature data, which contributes more effectively to the appearance-based association process.

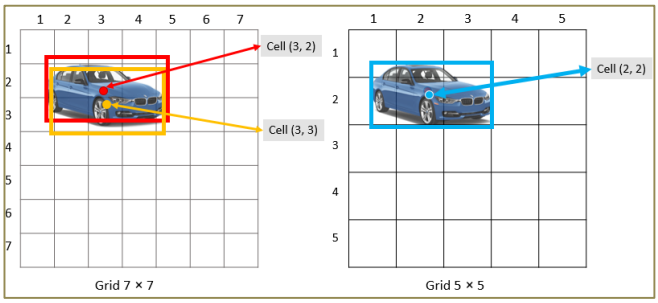

Fig. 5. Example of cell location on different feature grids.

2) Modified appearance similarity process: In the original DeepSORT, a feature bank (or gallery) mechanism is employed to retain long-term appearance information about targets, aiding in recovering object identities after long-term occlusions. This mechanism calculates the appearance similarity between a detection and a tracked target by storing up to 100 appearance descriptors for each track. The similarity is determined by the smallest cosine distance between the detection's appearance vector and the stored vectors in the feature bank. These appearance descriptors are obtained through a pre-trained CNN embedding model. However, this CNN model in DeepSORT contains 2,800,864 parameters and processing a batch of 32 bounding boxes takes approximately 30 ms on an Nvidia GeForce GTX 1050 mobile GPU. On embedded devices or devices with limited resources, especially those without GPU capability, the processing time may increase significantly.

To optimize the tracker for such devices, we propose extracting the appearance similarity without relying on the CNN model. Instead, we leverage the feature map vectors generated during the detection process. As discussed earlier, our detection process associates each detected object with its corresponding feature map vectors extracted from the YOLOv5 neck. The number of feature vectors varies depending on the number of predicted detections for that object, and these vectors differ in dimensionality based on the three grid sizes used in YOLOv5's detection process. These feature vectors are stored for each object in what we refer to as an object cluster. An object cluster consists of all the feature vectors associated with the multiple predicted detections of the same object resulted from the previous modified detection phase.

For example, in YOLOv5s, the grid for small objects (52x52) generates feature vectors with a dimensionality of 128, the grid for medium-sized objects (26x26) produces vectors with a dimensionality of 256, and the grid for large objects (13x13) creates vectors with a dimensionality of 512. Considering these feature vectors, we propose an approach that evaluates appearance similarity by comparing vectors of the same size between two objects.

Algorithm 1 outlines the steps for calculating appearance similarity between a detection D and a track T. First, we divide the feature vectors into three sets based on their dimensions (128, 256, and 512). Then, we compute the similarity score for each set as the minimum cosine distance between corresponding vectors. The final appearance similarity score is determined by taking the overall minimum of these three scores.

$$Simil(DF_{Dim}, TF_{Dim}) = \min\{\cos(df_i, tf_j) \mid df_i \in DF_{Dim}, tf_j \in TF_{Dim}\} \quad \text{(Eq.1)}$$

Algorithm 1 Appearance Similarity between two objects

Input: $DF = \{df_1, \ldots, df_n\}$, // features of object 1
$TF = \{tf_1, \ldots, tf_n\}$ // features of object 2

1: $(DF_{128}, DF_{256}, DF_{512})$ = *separate_by_size*(DF)
2: $(TF_{128}, TF_{256}, TF_{512})$ = *separate_by_size*(TF)
3: $simil_{128}$ = Simil(DF_{128} , TF_{128}) using Eq. 1
4: $simil_{256}$ = Simil(DF_{256} , TF_{256}) using Eq. 1
5: $simil_{512}$ = Simil(DF_{512} , TF_{512}) using Eq. 1
6: **return** min{ $simil_{128}$, $simil_{256}$, $simil_{512}$ }

IV. EXPERIMENTS

To evaluate the performance of our proposed tracking algorithm, we conducted a series of experiments using real-world highway surveillance videos. We compared the tracking accuracy, robustness, and computational efficiency of our algorithm against the state-of-the-art SORT and DeepSORT algorithms. The video sequences used for testing captured a variety of challenging traffic scenarios, including high-density traffic and occlusions, to thoroughly assess the algorithm's performance under realistic conditions.

For our experiments, we utilized a locally collected traffic video dataset comprising over 10,000 frames. This dataset captures real-world urban traffic scenarios with varying numbers of vehicles, ranging from light traffic to highly congested conditions. It includes challenging situations such as shadowing caused by varying lighting conditions throughout the day and occlusions resulting from vehicles overlapping or obstructing each other. The videos were recorded at a resolution of 1920x1080 and a frame rate of 30 fps, ensuring high-quality imagery for accurate detection and tracking. The dataset encompasses different times of the day and weather conditions, providing a comprehensive set of challenges for evaluating the robustness and scalability of our proposed tracking approach.

The experiments were performed on a Windows machine equipped with an Intel Xeon Silver 4110 CPU @ 2.10 GHz and 32 GB of RAM. To simulate low-resource environments, we deployed our tracker within a Docker container configured with reduced resources, specifically 4 CPU cores and 4 GB of RAM, to mimic the capabilities of edge devices. This setup allowed us to assess the feasibility of deploying our algorithm on resource-constrained platforms without compromising its real-time performance.

To evaluate the computational efficiency of our algorithm, we measured the average processing time required to track varying numbers of vehicles in the scene. This processing time represents the duration needed to perform the tracking task for each frame, excluding the object detection phase. This metric provided valuable insights into the performance of our algorithm, particularly in terms of calculating appearance similarity and data association, under different workload conditions.

To assess the scalability of our method, we analyzed its performance across scenes with varying numbers of vehicles. Table I illustrates the processing time per frame (in milliseconds) relative to the number of detected vehicles compared to DeepSORT and SORT. Additionally, Fig. 6 provides a graphical representation of the data in Table I, offering a visual comparison of the processing times between the algorithms. Our findings revealed that, on average, the processing time of our tracker is about 25 % of the time required by DeepSORT, and approximately 10% higher than SORT. These results underscore the significant computational efficiency of our tracker, as it maintains near-constant processing time as the number of vehicles increases, demonstrating efficient scalability. This consistent performance is critical for real-time applications in ITS, where traffic density can fluctuate significantly.

TABLE I. MEASURED PROCESSING TIMES IN MILLISECONDS FOR EACH ALGORITHM WITH VARYING NUMBERS OF VEHICLES

# Number of Vehicles	DEEP SORT	SORT	Our Tracker
1	24,58	2,23	**2,97**
2	39,75	3,82	**7,01**
3	52,43	5,80	**12,77**
4	65,14	7,85	**20,71**
5	73,83	10,37	**31,21**
6	87,07	12,38	**38,08**
7	96,49	12,69	**42,06**
Avg Proce. time	**45,01**	**6,83**	**11,38**

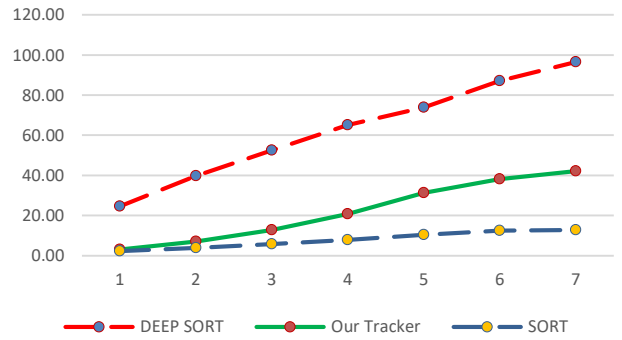

Fig. 6. Visual comparison of the processing times between our algorithm and the DeepSORT and SORT.

To evaluate the accuracy of our tracker, instead of using manually annotated video sequences with ground truth object trajectories, we opted to compare our tracking results directly with those of DeepSORT, which achieved a perfect score in our scenarios. We employed an error rate metric, calculated by summing the number of misses and false matches, then dividing by the total number of matches. This metric provides a comprehensive assessment of the accuracy and reliability of our algorithm under various conditions, with lower error rates indicating higher tracking accuracy.

To assess the robustness of our tracker, we introduced real-world uncertainty by simulating missed detections with a certain probability. We utilized a parameter called the skip detection rate, which we tested with three values: 0%, 25%, and 50%. This parameter simulates scenarios of missed detections by occasionally sending an empty detection output to the tracker. This test allowed us to evaluate the tracker's ability to re-identify objects through appearance association, even after missed detections, providing valuable insights into its reliability for real-world applications.

TABLE II. A COMPARISON BETWEEN THE ERROR RATES OF OUR TRACKER AND SORT REGARDING VARIOUS SKIP RATE VALUES

Skip rate	**0%**		**25%**		**50%**	
Algorithm	Our tracker	SORT	Our tracker	SORT	Our tracker	SORT
Miss matches	34	92	53	1048	50	1164
False matches	2	0	6	2	14	0
Total matches	8369	8362	6119	5949	3783	3661
Error Rate	**0,43%**	1,10 %	**0,96 %**	17,65 %	**1,69 %**	31,79 %

Table II presents the accuracy performance results of our tracker compared to SORT across three scenarios of skip detection rates: 0%, 25%, and 50%. DeepSORT was excluded from the comparison due to its perfect accuracy score in all scenarios.

The results show that under normal conditions (i.e., without skip detections) both our tracker and SORT achieve high accuracy with very low error rates. Notably, our tracker achieves an error rate of 0.43%, outperforming SORT, which records an error rate of 1.10%. In more challenging scenarios with non-zero skip detection rates, our tracker demonstrates a significant advantage. At skip rates of 25% and 50%, our tracker achieves error rates of 0.96% and 1.69%, respectively, while SORT shows much higher error rates of 17.65% and 31.79%. These results highlight the superior robustness of our tracker in handling missed detections compared to SORT (Fig. 7).

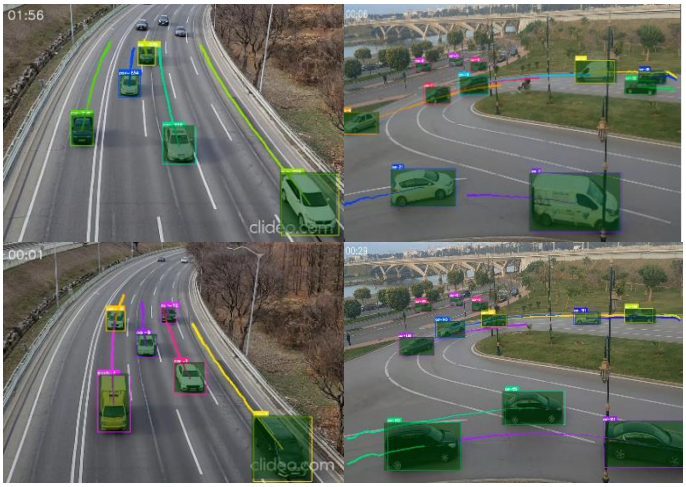

Fig. 7. Sample of tracking results visualization using our lightweight tracker.

V. DISCUSSION

The experimental results confirm that our modified lightweight DeepSORT variant significantly reduces computational requirements while maintaining high tracking accuracy. This improvement addresses a critical barrier in deploying effective ITS solutions on edge devices, enabling more widespread and efficient traffic monitoring systems. Our approach demonstrates that leveraging features directly from YOLOv5 detections is an effective strategy for reducing overhead without sacrificing performance. This finding suggests a paradigm shift in multi-object tracking, where the integration of detection and tracking components can lead to more streamlined and efficient algorithms.

One of the key insights from our study is that integrating detection and appearance features can lead to more efficient multi-object tracking solutions suitable for real-time deployment. However, we acknowledge that testing on a single, localized dataset may limit the generalizability of our findings. Evaluating the method on additional datasets, possibly from different regions or with different characteristics, would further substantiate its scalability.

One challenge we observed is that in highly congested scenes, the absence of an additional CNN descriptor slightly affected the appearance similarity process, occasionally leading to some miss matching and ID switches. This issue could be addressed in future work by incorporating further improvements to the processing method. Some of these improvements could include exploring optimized adaptive association mechanisms and integrating more recent and enhanced detection models to further improve performance. We also we aims to investigate the integration of our tracking algorithm with other ITS components, such as traffic prediction models and anomaly detection systems.

A promising direction for improvement is training the detector on a specialized dataset enriched with detailed annotations. By labeling objects with attributes like distinct colors, shapes, and varied viewing angles, the detector's backbone network can learn to recognize more discriminative features. This enhancement would significantly improves its ability to differentiate between objects, particularly in complex traffic scenarios with similar-looking vehicles or difficult perspectives, leading to more accurate and reliable tracking.

VI. CONCLUSION

In this paper, we present a novel lightweight tracking algorithm designed specifically for vehicle tracking in Intelligent Transportation Systems (ITS). By leveraging the predictability of vehicle trajectories and optimizing the DeepSORT workflow, we propose an algorithm that strikes a balance between accuracy and computational efficiency, making it well-suited for deployment on resource-constrained edge devices. Our approach utilizes features extracted from the detector network as a source of appearance information for the appearance matching task, effectively eliminating the need for the traditional CNN descriptor used in DeepSORT. This significantly reduces the computational load of our tracking algorithm. Experiments with real-world traffic surveillance data reveal that our tracker not only outperforms traditional methods like SORT in tracking accuracy but also significantly reduces processing time compared to DeepSORT. This efficiency is crucial for real-time applications on resource-constrained devices. However, some scenarios when a large number of features are extracted with objects, the algorithm can encounter some challenges, as the similarity calculation becomes computationally demanding. To address this, we plan in future works to add a new feature limitation mechanism that adjust the number of features used in the similarity processing, allowing users to tradeoff between efficiency and accuracy. Additionally, we aim to enhance the detector by training it on specialized datasets to improve object distinction. We also intend to evaluate the effectiveness of our method on devices like the Raspberry Pi to assess its performance on typical edge computing hardware and explore its applicability in more general use cases within ITS.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, et A. Farhadi, « You Only Look Once: Unified, Real-Time Object Detection », in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA: IEEE, juin 2016, p. 779-788. doi: 10.1109/CVPR.2016.91.

[2] S. Ren, K. He, R. Girshick, et J. Sun, « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks », in Advances in Neural Information Processing Systems, Curran Associates, Inc., 2015. Consulté le: 19 septembre 2022. [En ligne]. Disponible sur: https://proceedings.neurips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html

[3] N. Wojke, A. Bewley, et D. Paulus, « Simple Online and Realtime Tracking with a Deep Association Metric », 21 mars 2017, arXiv: arXiv:1703.07402. doi: 10.48550/arXiv.1703.07402.

[4] Z. Zhang et H. Peng, « Deeper and Wider Siamese Networks for Real-Time Visual Tracking », présenté à Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, p. 4591-4600. Consulté le: 28 avril 2024. [En ligne]. Disponible sur: https://openaccess.thecvf.com/content_CVPR_2019/html/Zhang_Deeper_and_Wider_Siamese_Networks_for_Real-Time_Visual_Tracking_CVPR_2019_paper.html

[5] C. Duan et X. Li, « Multi-target Tracking Based on Deep Sort in Traffic Scene », J. Phys.: Conf. Ser., vol. 1952, no 2, p. 022074, juin 2021, doi: 10.1088/1742-6596/1952/2/022074.

[6] H. A. Abdelali, H. Derrouz, Y. Zennayi, R. O. H. Thami, et F. Bourzeix, « Multiple Hypothesis Detection and Tracking Using Deep Learning for Video Traffic Surveillance », IEEE Access, vol. 9, p. 164282-164291, 2021, doi: 10.1109/ACCESS.2021.3133529.

[7] M. Anandhalli, V. P. Baligar, P. Baligar, P. Deepsir, et M. Iti, « Vehicle detection and tracking for traffic management », IJ-AI, vol. 10, no 1, p. 66, mars 2021, doi: 10.11591/ijai.v10.i1.pp66-73.

[8] A. El-Alami, Y. Nadir, L. Amhaimar, et K. Mansouri, « An efficient hybrid approach for vehicle detection and tracking », in 2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM), oct. 2023, p. 1-8. doi: 10.1109/WINCOM59760.2023.10322924.

[9] A. El-Alami, Y. Nadir, et K. Mansouri, « A hybrid vehicle tracking System for Low-power Embedded Devices », in 2024 International Conference on Circuit, Systems and Communication (ICCSC), juin 2024, p. 1-6. doi: 10.1109/ICCSC62074.2024.10617125.

[10] Z. Charouh, A. Ezzouhri, M. Ghogho, et Z. Guennoun, « A Resource-Efficient CNN-Based Method for Moving Vehicle Detection », Sensors, vol. 22, no 3, Art. no 3, janv. 2022, doi: 10.3390/s22031193.

[11] T.-Y. Lin et al., « Microsoft COCO: Common Objects in Context », 20 février 2015, arXiv: arXiv:1405.0312. Consulté le: 18 juin 2023. [En ligne]. Disponible sur: http://arxiv.org/abs/1405.0312

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, et A. Zisserman, « The Pascal Visual Object Classes (VOC) Challenge », Int J Comput Vis, vol. 88, no 2, p. 303-338, juin 2010, doi: 10.1007/s11263-009-0275-4.

[13] R. Girshick, « Fast R-CNN », 27 septembre 2015, arXiv: arXiv:1504.08083. Consulté le: 19 septembre 2022. [En ligne]. Disponible sur: http://arxiv.org/abs/1504.08083

[14] D. V. Tu, P. M. Quang, H. P. Nghi, et T. N. Thinh, « An Edge AI-Based Vehicle Tracking Solution for Smart Parking Systems », in Intelligence of Things: Technologies and Applications, N.-N. Dao, T. N. Thinh, et N. T. Nguyen, Éd., in Lecture Notes on Data Engineering and Communications Technologies. Cham: Springer Nature Switzerland, 2023, p. 234-243. doi: 10.1007/978-3-031-46573-4_22.

[15] A. Benjumea, I. Teeti, F. Cuzzolin, et A. Bradley, « YOLO-Z: Improving small object detection in YOLOv5 for autonomous vehicles », 3 janvier 2023, arXiv: arXiv:2112.11798. Consulté le: 7 avril 2024. [En ligne]. Disponible sur: http://arxiv.org/abs/2112.11798

[16] A. El-Alami, Y. Nadir, et K. Mansouri, « A review of object detection approaches for traffic surveillance systems », International Journal of Electrical and Computer Engineering (IJECE), vol. 14, no 5, Art. no 5, oct. 2024, doi: 10.11591/ijece.v14i5.pp5221-5233.

[17] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, et B. Lee, « A Survey of Modern Deep Learning based Object Detection Models », 12 mai 2021, arXiv: arXiv:2104.11892. Consulté le: 7 septembre 2022. [En ligne]. Disponible sur: http://arxiv.org/abs/2104.11892

[18] A. Bewley, Z. Ge, L. Ott, F. Ramos, et B. Upcroft, « Simple Online and Realtime Tracking », in 2016 IEEE International Conference on Image Processing (ICIP), sept. 2016, p. 3464-3468. doi: 10.1109/ICIP.2016.7533003.

[19] M. Elhoseny, « Multi-object Detection and Tracking (MODT) Machine Learning Model for Real-Time Video Surveillance Systems », Circuits Syst Signal Process, vol. 39, no 2, p. 611-630, févr. 2020, doi: 10.1007/s00034-019-01234-7.