Tiny-PIRATE: A Tiny model with Parallelized Intelligence for Real-time Analysis as a Traffic countEr

Synh Viet-Uyen Ha[1,2,*] Nhat Minh Chung[1,2], Tien-Cuong Nguyen[1,2], Hung Ngoc Phan[1,2]
[1] School of Computer Science and Engineering, International University, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam *

Abstract

Due to the rapid growth in the number of vehicles over the last decade, there has been a dramatic increase in demand for highway capacity analysis. Vehicle counting, in particular, has become a key element of vision-based intelligent traffic systems deployed across metropolitan areas. Most methods solved the vehicle counting problem under the assumption of state-of-the-art computing systems. However, large-scale deployment of such systems for multi-camera processing is very inefficient. With the recent advancement of cost-efficient Internet-of-Things (IoT) devices alongside machine learning methods developed specifically for such devices, solving the vehicle counting problem for real-time traffic analysis on IoT edge devices, and thereby facilitating its large-scale deployment have become highly favorable. In this paper, we propose a framework of vehicle counting designed specifically for IoT edge computers which follows the detection-tracking-counting (DTC) model. The proposed solution aims at addressing the multimodality of contextual dynamics in traffic scenes with a small detector model, a robust tracker and a counting process that accurately estimate both a vehicle's motion of interest and its exit time from observation areas. Experimental results on AI City 2021 Track-1 Dataset showed that ours outperformed related methods with promising results regarding both accuracy and execution speed.

1. Introduction

The past decade has experienced a dramatically increasing demand for analyses of traffic capacity on highways, owing to the rapid growth in the number of vehicles. Particularly, vehicle counting has become a pivotal component in vision-based traffic surveillance systems across metropolises. This function cohesively associates with a wide range of research works that focus on extracting static and dynamic attributes regarding vehicles' appearances and motion characteristics, including locations, shapes, sizes, categories, trajectories, paths of movement, and time-series records of motion within the observational views of surveillance cameras. So far, these advances have improved the overall accuracy of digital traffic analytic systems. However, in real-world applications, taking into account scaling-up strategies of practical implementations, the vehicle counting task needs to be carried out on computationally limited platforms at real-time execution efficiency. As a result, mimicking in-road hardware sensor-based counting on IoT devices necessitates that vision solution settings well utilize hardware resources in terms of computations and memory complexities.

AI City Challenge is a competition that focuses on efficient transportation systems, and pushes technologies closer to practical integration. Our research work deals with the one of the tracks introduced by the AI City Workshop at CVPR 2021: *Track1: Multi-Class Multi-Movement Vehicle Counting Using IoT Devices*. In this challenge, teams are required to develop class-wise, motion-specific vehicle counting systems that can efficiently record vehicles in two categories, i.e., cars and trucks, in different traffic paths and scenes in real time on a single-board computer. In Track 1 of [20], the vehicle counting challenge was met with astounding successes by many proposed methods [18, 21, 31, 34], where the methods well satisfy the effectiveness and efficiency requirements when executed on a high-end processing system with an exceptional graphic processing unit (GPU). However, scaling up these models to accommodate expansions of multiple traffic scenarios requires expensive infrastructural developments.

Recent years have seen many technological advances of IoT devices and edge computing. These devices' computing capability has had striking improvements which, in turn, allows large scale deployment of machine learning models, and encourages the machine learning research community

*Corresponding email: hvusynh@hcmiu.edu.vn

to develop efficient methods and libraries for IoT devices. Hence, it is favorable to efficiently solve the vehicle counting problem for real-time traffic analysis on these IoT edge devices. In this paper, anchoring on the computational capabilities of IoT devices like NVIDIA Jetson Xavier NX, we propose a framework of vehicle counting that follows the detection-tracking-counting (DTC) model. Our work performs multi-vehicle tracking via associating vehicle detections that are extracted from a separated YOLOv4-Tiny [29], a robust and light-weighted object detector for mobile embedded devices. From the trajectories, we record moving vehicles in terms of estimated real paths and exit times. Our contributions in this paper are described as follows:

- Firstly, addressing contextual dynamics in traffic scenes, we trained and inferenced a vehicle detector on our custom hand-labelled AI City 2021 Track-1 dataset with a mosaic data augmentation [32], where we also exploit major parallelism at lower levels of CUDA efficiency for batch processing at high speed.
- Secondly, in traffic scenes, there is a fact that vehicles' motions usually accompany sudden changes in velocity and positions. Addressing these problems, we incorporate significant modifications by engaging with IoU, Mahalanobis distance and visual histogram distribution difference on a Kalman-Filter-based, multi-vehicle tracker called SORT [2]. We especially avoid deep features such as in DeepSORT [33] to resolve computational costs of feature extraction.
- Thirdly, we propose a counting component that estimates each vehicle's direction of motion and its exit time from observational areas by exploiting its trajectory and momentum. In this component, we use predefined scope fields to limit the domain of analysis to only vehicles of interest, and from the numerical motion trajectories of the vehicles, we estimate their real-life paths via a directional pattern-trajectory matching. We then record the vehicles of interest at the moment they are expected to leave the regions of interest via a Kalman Filter's linear constant velocity model.
- Finally, in our DTC solution, based on the fact that the three modules are mostly independent from one another, and that most of our processing flow adopts a sequential paradigm, we extend the proposed solution's implementation with thread-level parallelism. Our experiments suggest that this implementation not only increases the efficiency of resource usage on an IoT device, but it also effectively leverages our highly accurate DTC approach to traffic density analysis.

The rest of the paper is structured as follows. Section 2 provides a recapitulation of previous related modular research that is capable of execution on embedded devices. In Section 3, our proposed model of vehicle counting for embedded devices are elaborated. Our experimental evaluation are then presented in Section 4 and discussions are concluded in Section 5.

2. Related Works

Object Detection: Object detection has an essential role in the analysis of object behaviors, where the detection module can simplify the domain of interest and support a variety of tasks such as tracking vehicles, traffic density analysis, anomaly recognition, etc. In the past decade, the research community has shifted attention to deep learning to tackle the object detection problem. Research results have indisputably demonstrated the effectiveness of deep neural networks (DNN) [12, 13, 23] and [3] for accurately recognizing image objects, but largely by greatly trading off speed for accuracy on high-end machines. Therefore, in order to promote wide-spread industrial scalability, many researchers seek to propose similar DNN-based approaches for embedded IoT edge devices such as MobileNet-SSD [14, 17], YOLOv3-tiny [22], YOLOv4-Tiny [29], NAS-FPNLite [11], and EfficientDet [26]. These models have met ever-increasing success as they reduce the computational complexity while preserving accuracy.

Multi-Object Tracking: Multi-Object Tracking (MOT) is one of the most widely investigated problems as it is a vital component for many applications in computer vision when we can reserved the time-series information of desired objects. The typical motif of MOT is the tracking-by-detection model, where objects of interest are localized by detectors before tracking. Regarding MOT problems, recently-published methods are categorized into two representative approaches: offline tracking and online tracking. Offline techniques use the detection results of a batch of consecutive frames in the video on a global optimization algorithm [1, 7, 8, 16, 27, 30]. Online MOT, on the other hand, often uses only the previous and current video frame since long-term movements of objects being tracked are embedded into a state space for memorization. This further leads to a data association formulation between the object being tracked with the newly received detection of the MOT task. In this kind of formulation, it is crucial to find a good mechanism that can effectively evaluate the similarity of the same objects from various viewpoints, e.g. by following mathematical foundations [2, 18], or deep-learning practices [33]. The literature of deep learning approaches for MOT has introduced mechanisms to estimating the similarity that includes the attention mechanism [5, 10, 36], neural networks like Recurrent Neural Networks [9, 19, 24] and Siamese Networks [6, 25]. However, incorporating these mechanisms poses a dilemma between good accuracy and huge computational overheads to the systems. When good detections already have high computational demands, it is less reason-

able to run neural network methods than those with explicit mathematical formulations on embedded devices.

Vehicle Counting: Vehicle counting is a fascinating problem as it enables traffic engineers to analyze traffic conditions to devise plans for controlling traffic flow and expanding road networks. This problem was revisited by the AI City 2020 [20], where a variety of different methods have been proposed to tackle the problem [4, 28]. Most of the proposed method follows a sequential DTC paradigm, where independent improvements of the modules can be made, i.e. error mitigation on separate modules. There are generally two strategies to tackle the vehicle counting problem in the DTC framework. The first strategy is to use an accurate, computationally complex detector to support the following tracker and counter. The representative methods for this strategy are introduced in [18, 21, 31, 34]. While better and often more heavy-weighted state-of-the-art detectors do improve counting accuracy, they are not capable of running on a IoT device which is strictly constrained in computational capability. The second strategy is to formulate rigorously the subsequent tracking and counting module to supplement the detector. This strategy allows for the scaling down of detectors to improve efficiency with acceptable trade-offs in terms of overall effectiveness. This second strategy was also tested in the vehicle counting task of AI City 2020 to allow a smaller detector such as YOLOv3 used in [28]. This strategy is more favorable for IoT solutions by scaling down the computational demands of detectors.

3. Methodology

Our proposed approach for analyzing traffic behaviors includes a system sequence of 3 modules, where each is responsible for a specific task: 1) vehicle detection, 2) vehicle tracking, and 3) path-specific vehicle counting. As demonstrated in Fig. 1, in a multi-threaded manner, we receive a traffic video containing multiple vehicles of interest and preprocess it along with predefined ROIs (regions-of-interest) and MOIs (motions-of-interest) settings; then we produce a list of detected vehicles via performing batch object detection recognize vehicles in view (i.e. cars or trucks); then we perform our proposed three-fold matching scheme for tracking vehicles' trajectories; finally, we record vehicles of interest that have exitted the field of observation based on MOIs configurations that represent real-life paths. The design of the proposed solution is not only robust and effective for a wide array of scenarios but also capable of making efficient use of available computing resources.

3.1. Vehicle Detection on Batch Inputs

Simplifying traffic vision problems by means of clustering locally connected image pixels to only a number of 2D bounding boxes representing vehicles of interest, our solution takes advantage of one of the state-of-the-art object detectors, the YOLOv4-Tiny object detector [29], as the first step to analyzing vehicle behaviors on multiple street scenarios in an online manner. Given an RGB image I recorded at time step t, the detector function produces a list of detections D_i, with i ($1 \leq i \leq P$) being the i-th detection in P detections returned by the detector after non-max-suppression on confidence scores:

$$\mathrm{D}_i = \{\mathrm{d}_i, \mathrm{c}_i\} \tag{1}$$

where d_i and c_i denotes respectively the i-th detection's bounding box geometry vector $[x, y, a, h]^T$ (center at x and y, aspect ratio a of width over height and height of h) and its corresponding class (i.e. car or truck). Our strategy with the object detector includes batch processing where we append a list of consecutive 450×450 frames, from time step t to $t+N$ (where N is the batch dimension of an image batch of shape $N \times H \times W \times C$) together as input, and produce the corresponding time-step coherent list of appended detection lists using the YOLOv4-Tiny model.

The YOLOv4-Tiny model is light-weighted and highly accurate even in occlusion. Futhermore, it possesses an advantage over many state-of-the-art detectors (e.g. YOLOv4 [3]) in terms of deployment scalability, due to its suitability for IoT devices. In training the YOLOv4-Tiny detector, we labelled 7000 images manually to tune the COCO-pretrained YOLOv4-Tiny model via transfer learning. The training configuration is the same as the one set by YOLOv4-Tiny authors at an 80%-20% training-testing split, except that we added Mosaic data augmentation [32] to enhance the training data. In this augmentation, batches of four training images are combined in specific ratios to avoid overfitting and facilitate multi-scale adaptability. The model parameters are determined by training the COCO-pretrained YOLOv4-Tiny model for 100000 iterations with stochastic gradient descent, in which the best weights evaluated on the test set are selected for inference.

On inference with the 5 hours of video dataset, the resultant model is capable of detecting a diverse array of vehicles from 31 different scenarios, including bird-eye view, teleport view, or the different weather like dawn, snow or rain. By leveraging batch processing, our choice with the YOLOv4-Tiny model is further rationalized experimentally as the object detector has acceptable trade-offs between accuracy, resource demands and speed on IoT devices, such as a Jetson Xavier NX. Specifically, it is able to achieve good mAP@0.5 accuracy of 86% on a separate test set, and it is capable of running by batches of 16 continuously with a specially allocated thread and an NVIDIA Jetson Xavier NX at approximately 700 fps on average.

3.2. Online Directional Multi-Vehicle Tracking

The tracking algorithm is one of our major contributions in the proposed system. It is responsible for connecting

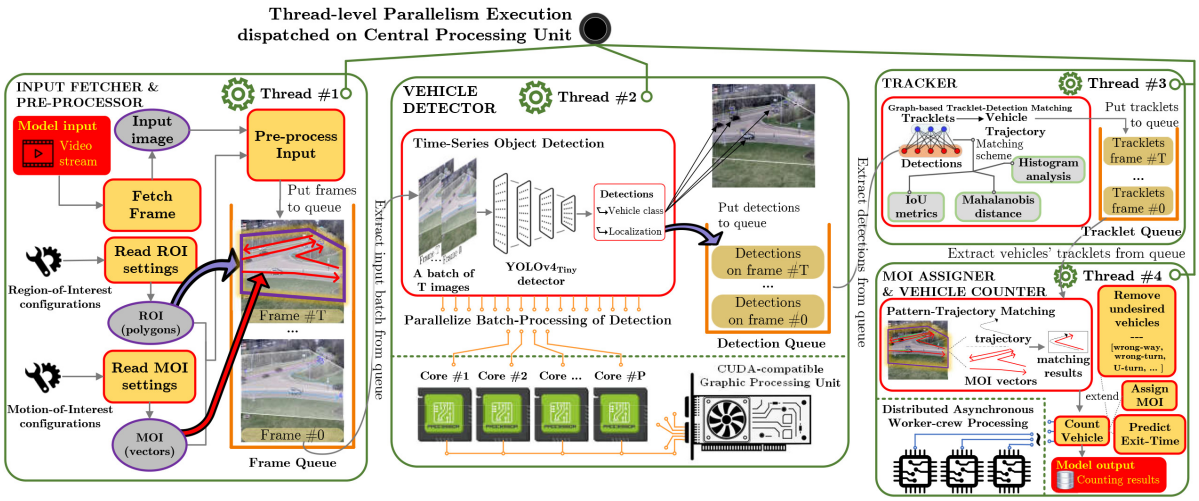


Figure 1. The overview workflow of our proposed approach for vehicle counting with thread-level paralellism. Thread 1 handles input featching and proprocessing, Thread 2 handles batch vehicle detection with the GPU, Thread 3 handles sequential tracking and Thread 4 handles path-based vehicle counting using worker-crew processing. The threads communicate via intermediary queues.

vehicle detections representing the same real-life moving cars or trucks across the temporal axis, thereby extracting their trajectories of motion for further analysis (e.g. behavior understanding). Inspired by the lean approaches of the [2] and [33] algorithms, we only make use of rudimentary techniques (i.e. Kalman Filter [15], Hungarian Matching [35]) and object features (i.e. bounding boxes, positions, and color encodings) to facilitate the tracing process of each vehicle. As shown in Fig. 2, our tracking algorithm is formulated around a three-fold data association scheme supported by inter-frame predictions of vehicle positions. Specifically, by leveraging vehicle detections estimated by the object detector, our algorithm employs the Kalman Filter (KF) to predict vehicles' positional displacements throughout a scene, thereby enabling effective data association of each vehicle using the Hungarian Matching algorithm on account of a target's bounding box geometry, its directional position with respect to the vehicle's Kalman Filter state, and its RGB color histogram.

3.2.1 The Kalman Filter Motion Estimation Model

Our adoption of the Kalman Filter is both similar to the original formulation in [2] and that of DeepSORT [33], where vehicles of interest are called tracklets with trajectories represented by lists consisting of 2D bounding box positions, and with a class label signifying a car or truck. The tracking scenario is defined on an 8-D state space of the attributes $[x, y, a, h, \dot{x}, \dot{y}, \dot{a}, \dot{h}]^T$ that contains a tracklet's current center at (x, y), and (a, h) being the aspect ratio width divided by height and the actual height value of the bounding box surrounding the tracklet's appearance. The latter 4 features $\dot{x}$, $\dot{y}$, $\dot{a}$, $\dot{h}$ represent the former features' respective velocities, i.e. changes of the tracklet's center coordinate and bounding box geometry in image coordinates using the Kalman Filter model of linear constant velocity. The KF-state of a tracklet is updated on an observed vehicle $[x, y, a, h]^T$ recognized by the object detector module.

At each time step t, prior to associating existing tracklets of the tracking module with observed vehicle detections, every tracklet T's current estimated bounding box geometry, called $\mu = [x, y, a, h]^T$, is computed by predicting its new image coordinates for the current time step. By exploiting momentums with the Kalman Filter in this manner, cars and trucks that are temporarily being occluded by rain blots or other vehicles can be re-tracked once their respective appearances are re-observed.

3.2.2 Three-fold Data Association Scheme

The core of our tracking algorithm is a three-fold, sequential matching procedure that connects existing tracklets to newly observed vehicle detections, as shown in Fig. 2. We use the Hungarian Matching to facilitate optimal assignment of detection observations to tracklet predictions in terms of IoU costs for all tracklets and observed detections, then in terms of Mahalanobis distance costs, then histogram distribution distance costs respectively for unmatched ones. This order of matching metrics not only can balance between their computational demands and actual input sizes for each metric, but each metric also serves a role:

IoU Matching: The use of IoU metric as the cost criteria for optimal matching originates from the SORT approach [2], where it has significantly demonstrated its fast and effective performance when used alongside the Kalman Filter. It exploits an observation in practice that most vehicles are recorded to be moving slowly enough that their trajectories consist of bounding boxes that are close to one another. Thus, for a detection D_i, its cost of being matched with any tracklet T_j is computed using IoU that is thresh-

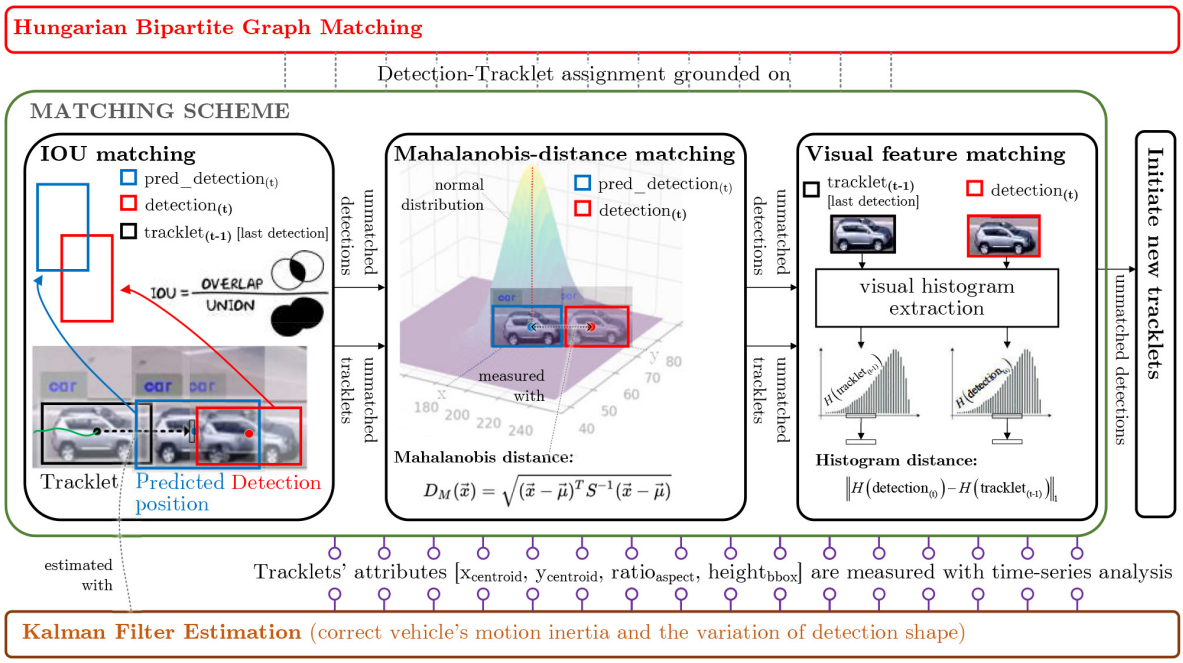


Figure 2. The proposed three-fold sequential vehicle tracking method. With Hungarian Matching and Kalman Filter predictions, tracklets and observed detections are matched by IOU, otherwise they are matched directionally by Mahalanobis distance, or by visual feature.

olded by $\epsilon_{IoU} = 0.1$ at minimum.

$$\text{cost}_{IoU}(i,j) = \frac{bbox(d_i) \cap bbox(\mu_j)}{bbox(d_i) \cup bbox(\mu_j)} \tag{2}$$

Centroid Matching: As there are some vehicles that move quickly through observation areas, where the assumption of slow movement of the IoU metric fails, we propose using the Mahalanobis distance to accommodate the matching of fast vehicles. The Mahalanobis distance measures the proximity of a detection point D_i from the distribution represented by the KF-state of a tracklet T_j, which grows as D_i is further away from the mean μ (also the current bounding box location) of T_j's KF-state along each of its principal component axis of bounding box motion, independently of the 2D image space. Thus, the tracklet vehicles' tendencies of motions are exploited to enable directional matching, where axes of large standard deviations can correspond with the vehicles' fast motion along those axes, and axes of smaller spread can correspond with slower displacements.

$$\text{cost}_{\text{centroid}}(i,j) = (d_i - \hat{\mu}_j)^T S_j^{-1} (d_i - \hat{\mu}_j) \tag{3}$$

where S_j is tracklet T_j's covariance matrix, and $\hat{\mu}_j$ is the mean of the projection of the j-th tracklet's distribution into the measurement space of $(\hat{\mu}_j, S_j)$.

Histogram Feature Matching: In much fewer cases of sudden movements where vehicles suddenly jump from one spot to another in a non-linear motion that the thresholded centroid distance matching procedure may miss, we take advantage of their normalized 3-channel histogram distribution of 8 bins extracted from the image based on the bounding box geometry of a detection D_i and a tracklet T_j to facilitate matching using Manhattan distance, thresholded by $\epsilon_H = 3$ at maximum.

$$\text{cost}_{\text{histogram}}(i,j) = \|H(d_i) - H(\mu_j)\|_1 \tag{4}$$

where $H(\cdot)$ is a function mapping a vehicle's appearance at the time of extraction constrained by its bounding box to a vector of normalized histogram feature values.

For centroid matching and histogram feature matching criteria, we extend them with a Euclidean distance thresholding constant empirically chosen at 50 image pixels to avoid great-distance matching in some cases, and a forward directional suppression scheme on tracklets to avoid erroneous matching with vehicle detections behind them or to their sides as they exit the observational areas. For the latter, we proceed with an angular approach using bounding box centers (only x, y coordinates) on each pivot $p_j = [x, y]^T$ for a tracklet T_j, which is defined by averaging its five latest center positions on the input space. The forward directional suppression procedure is described in Algo. 1. As our implementations are largely in vectorized forms, computational overheads are well traded-off in terms of accuracy.

While matched tracklets are updated in terms of their KF-state by the respective detections with the new positions appended to their trajectories, unmatched tracklets are removed if they are still in the ROIs after 35 frames or if they have just been initialized by no more than 3 time steps prior. Otherwise, the unmatched tracklets are simply vehicles temporarily missing observed detections and awaiting re-tracking. On the other hand, unmatched detections are used to initialize new tracklet vehicles at zero velocities.

Algorithm 1: Forward directional suppression

Input:
Tracklet indices $T_j = \{1, ..., A\}$;
Tracklet pivots p; Tracklet current centers T_c;
Detection indices $D_i = \{1, ..., B\}$;
Detection current centers D_c;
Angle threshold θ_0;
Output:
Binary mask M of shape (A, B), M[j, i]=1 if tracklet j-th can be matched with detection i-th;

Procedure:
- Initialize M of shape (A, B) at zeros;
- **foreach** $j \in T_j$ **do**
  - **foreach** $i \in D_i$ **do**
    - Compute vector of $v_j \leftarrow T_c[j] - p_j$;
    - Compute vector of $v_i \leftarrow D_c[i] - p_j$;
    - Calculate $\theta_{j,i} \leftarrow angle(v_j, v_i)$;
    - $M[j, i] = \theta_{j,i} < \theta_0$;
  - **end**
- **end**
- **return** M;

3.3. Path-specific Vehicle Counting

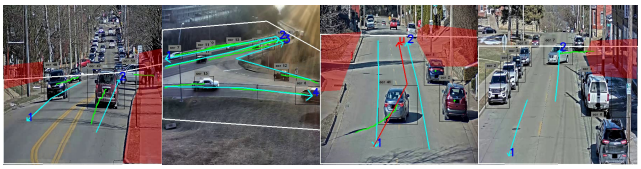
Given trajectories of vehicles at time t, we record them in terms of label (car or truck), motion path and time of exit as they exit the ROIs. By defining scopes that constrain vehicles' legal locations, we assign motion paths to the vehicles using a vector-based MOIs assignment on their trajectories. A vehicle is determined to have exitted if its expected position at the current time step is out of scope.

3.3.1 Vector-based Motions-of-interest Assignment

In order to assign MOIs that represents real-life paths to all vehicles, we propose an approach that only takes into account vehicles that fit within the scope of the MOIs labels, then performs assignments of moving directions based on the trajectories of all vehicles.

Scope Regions-of-Interest: Firstly, in order to define the suitable scopes of interest, we construct two modified types of ROIs based on those provided by the AI City 2021 challenge. These custom ROIs serve as a suppression approach for disregarding vehicles that are not within the field of observation, and those that are within the field of observation but are moving in illegal directions.

Particularly, with the former type, we impose certain restrictions called extended ROIs (eROIs) on vehicle positions by manipulating the ROIs defined for the challenge, as demonstrated by the white boundary polygon in the subfigures of Fig. 3. Accordingly, detected vehicles whose centers are not bound by the eROIs are removed. Thus, the domain is more constrained as the number of vehicles that the algorithm needs to consider are fewer, thereby effectively eliminating out-of-scope vehicles altogether. To construct eROIs, we typically modify the given ROIs by manipulating their areas to avoid high-occlusion areas and accommodate illegal directions for removal, while allowing us to leverage KF-predictions of exit time. Otherwise, we leave them as how they were originally.



(a) (b) (c) (d)

Figure 3. Example of our tracking process with directional assignment settings: eROIs (white polygons), iROIs (red overlays), MOIs (blue, red arrows), and trajectories (green lines).

For the latter type, where vehicles are moving in eROIs but not in legal directions (e.g. turning from or into small streets, parking lots, etc.), we track and exploit the tracking trajectories of those moving vehicles by marking them for later removal (without counting) if they enter illegal ROIs (iROIs). An example is illustrated in subfigure (a) and (d) in Fig. 3, where the red overlays signal regions where only illegal paths will go through.

A vehicle is suppressed by mapping their position, or trajectory, on the binary indicator functions of eROIs and iROIs, respectively denoted as $eROI(\cdot)$ and $iROI(\cdot)$. Vehicle detections are ignored if their centers of (x, y) correspond to $eROI(x, y) = 0$. Vehicle tracklets are discarded their trajectory has at least one positional center (x, y) where $iROI(x, y) = 0$.

Vector-Point Motion Assignment: Having defined the scope, we assign labels on motion trajectories following a proposed two-point vector model. In this approach, we manually label for each movement a vector denoting the general path that vehicles in that movement would travel. Specifically, by leveraging the results of our proposed tracking module, which would produce a trajectory for each vehicle from entering to exiting ROIs, we label for its corresponding movement a vector consisting of an exit point E_m that is close to the vehicle's point of exit, and a start point S_m which is close to the vehicle's point of entrance. Hence, as shown in Fig. 3, in order to assign a movement, we simply extract from a tracklet T_j's trajectory a point $\hat{S}_m$ which is closest to S_m from the first β portion of the trajectory, and one other $\hat{E}_m$ closest to E_m from the latest β. Specifically, β is chosen to be 10% to facilitate fair consideration against all movements in cases of fluctuating bounding box predictions on large vehicles (e.g. freight trucks, close-up cars). Thus, the multi-point trajectory with respect to the movement vector is simplified as a vector.

The assignment of movement labels necessitates that every trajectory vector is thresholded against its potential movement vector by angle, and whose sum of distances be-

Algorithm 2: MOIs assignment

Input:
Tracklet indices $T_j = \{1, ..., A\}$;
Tracklets T;
Movement indices $M_k = \{1, ..., Q\}$;
Movements M;
Portion ratio β; Angle threshold θ_0;
Output:
Tracklets are matched to movement labels;

Procedure:
  foreach $j \in T_j$ **do**
    $max\_distance \leftarrow inf$;
    foreach $k \in M_k$ **do**
      $S_m, E_m \leftarrow M[k].points()$;
      $\hat{S}_m \leftarrow T[j].closest\_in\_start(S_m, \beta)$;
      $\hat{E}_m \leftarrow T[j].closest\_in\_end(E_m, \beta)$;
      $v1 \leftarrow E_m - S_m$;
      $v2 \leftarrow \hat{E}_m - \hat{S}_m$;
      $\theta \leftarrow angle(v1, v2)$;
      if $\theta < \theta_0$ **then**
        $d \leftarrow dist2D(S_m, \hat{S}_m)$;
        $d \leftarrow d + dist2D(E_m, \hat{E}_m)$;
        if $d < max\_distance$ **then**
          $max\_distance \leftarrow d$;
          $T[j].assign\_motion(M[k].name())$;
  return T;

tween the heads and tails of vectors is minimized. Our procedure is shown in Algo. 2. An example of our MOIs labels is shown in Fig. 3. We also define a few unwanted movements in eROIs, which are labeled 'u' for signaling discard, but we avoid defining so many that efficiency is affected.

KF-aided Counter: In order to record vehicles of interest, we output the vehicles' labels (car or truck) along with their motion descriptions the moment their predicted KF-projections completely exit observational areas. As detections are suppressed if they are not within eROIs, vehicles come very close to the inner boundaries without fully exiting them via data association. Thus, by employing the Kalman Filter to estimate exit times, vehicles are counted following their linear constant velocity momentums.

3.4. DTC Framework on Thread-level Parallelism

Elaborating Fig. 1, we implement our proposed DTC vehicle counting solution from video input in a multi-threaded CPU-, GPU-utilizing manner. As the three aforementioned modules are mostly independent from one another, we take advantage of current hardware development for IoT devices of both multi-core CPU and CUDA-compatible GPU support, and propose a thread-level parallelism framework for tackling the AI City 2021 challenge.

Our approach best performs with at least 5 threads and a GPU, where each will be processing inputs continuously throughout the given video sequence. From the main thread (which is not Thread #1), we deploy our object detection model on the GPU, initialize all necessary components of the solution (i.e. input configurations, tracker and counter), and deploy the other 4 threads of continuous processing that only terminates once they receive ending flags:

Thread #1: We receive as input a traffic video containing multiple vehicles of interest and preprocess it along with predefined ROIs and MOIs settings. This thread is responsible for receiving input frame batches from video files, and pushing them into a 'Frame Queue' that is used for batch object detection.

Thread #2: From the 'Frame queue', we extract batches of input images and produce the corresponding time-step consistent lists of detected vehicles via performing batch object detection with the GPU. This thread is used for efficiently recognizing vehicles (cars, trucks) within view, and continuously pushing each time-step list of detected vehicles into 'Detection Queue' for sequential tracking.

Thread #3: From the 'Detection Queue', we perform our proposed three-fold matching scheme for tracing vehicles (tracklets) in a temporally sequential manner. Tracklets that are predicted to have exitted the ROIs are pushed into a 'Tracklet Queue' for matching their trajectories with real paths, and for recording their expected exit time.

Thread #4: From the 'Tracklet Queue', this thread removes tracklets that are no longer within observation, and further makes use of a distributed worker-crew processing to asynchronously record them. Thus, the thread deploys multiple child threads where each will record a vehicle of interest that has exitted the field of observation, after algorithmically assigning it a path label based on its numerical trajectory and MOIs configurations that represent real-life paths. After recording all vehicles of interest, Thread #4 will wait for its deployed child threads to terminate.

After all execution steps for the input video, the main thread will wait for Thread #1 to #4 to terminate and exit the algorithm. This design of the proposed solution is experimentally demonstrated to be not only robust and effective for a wide array of scenarios, but it is also highly capable of making efficient use of available computing resources.

4. Experiments

4.1. Experimental Setup

Our experiments are performed using the dataset provided by The AI City Challenge 2021. The dataset contains 31 videos of different traffic scenarios which were captured from 20 unique camera scenes in high resolution at the total length of 5 hours. The dataset presents an array of different traffic densities and vantage points, includ-

ing full intersections, highway segments, and city streets in different light intensities, weather conditions and recording distances. Taking into account a neural network's learnability, we utilize it to approximate a generalized mathematical model for recognizing vehicles on scenarios specific to the dataset. For effectively counting vehicles from input videos, our solution's implementations are then constructed in such a way that can efficiently take advantage of the domain comprehension. The resultant solution best performs on a multi-threaded, GPU-supported machine.

We tested our solution on an NVIDIA Jetson Xavier NX, which is an embedded AI mini-computer. With GPU-support and a multicore processor, the Jetson Xavier NX device (whose specification provides a CUDA-accelerated GPU with a shared memory of 8GB) is adequately capable of running our solution in real-time speed. Taking into account hardware developments, we also perform our solution on a configuration of Intel Core i7 with a NVIDIA GeForce RTX 2060 GPU to assess how the solution's efficiency will be boosted given more CPU and GPU computability.

4.2. Metrics

We judge the results with AI City Challenge evaluation score which is defined as:

$$S1 = 0.3 * S1_{Efficiency} + 0.7 * S1_{Effectiveness} \quad (5)$$

where $S_{Efficiency}$ is calculated based on the execution time and is adjusted by the Base Factor which is dependent on the running system's CPU and GPU computability. It assesses the solution's ability to execute online within its computing environment and resources. On the other hand, $S_{Effectiveness}$ is computed for vehicle counts as a weighted average of normalized weighted root mean square error scores (nwRMSE) across all videos, movements, and vehicle classes of the test sets. By splitting each video into segments for reducing jitters due to labeling discrepancies, it penalizes errors via the weighted cumulative vehicle counts from the start of the video to each segment's end. Detailed illustrations of all evaluation metrics can be found on the official website of AI City 2021 Challenge[1].

4.3. Results

4.3.1 Results on Computability

Evaluations of our solution on both computing environments show high ratings of accuracy (with deviations due to hardware architectural differences) at 0.94 $S1\_Effectiveness$. As illustrated by $S1\_Efficiency$ in Table 1, the rating for our solution significantly increases in an environment of higher computability, where total execution time significantly decreases from 58.4701 minutes to 14.0679 minutes as the base factor increases from 0.6948 (on Jetson) to 0.9782 (on PC). Vehicle detections are extracted by batch at very high speed even on the Jetson device, so it is likely that sequentially processing on CPU is a bottleneck, and that efficiency improvements for our solution can be made through streamlining CPU computations.

[1]https://www.aicitychallenge.org/2021-data-and-evaluation/

Table 1. The comparision of our proposed scheme on different hardware configurations when evaluated on AI City 2021 Dataset

Device	Exec. Time	$S1_{Effectiveness}$	$S1_{Efficiency}$	$S1$
Jetson NX	58.4701 mins	0.9403	0.8763	0.9211
RTX 2060	14.0679 mins	0.9406	0.9581	0.9459

4.3.2 Final ranking

Our team achieved a very competitive final score of S1 = 0.9459 at the second place, and is only 0.0008 lower than that of the top team. The final ranking results of the challenge are shown in Table 2. Nevertheless, supposing all other teams run their solutions on the Jetson Xavier NX device, ours may only be at 5^{th} place.

Table 2. The overall ranking on $S1$ score of the vehicle counting task in AI City 2021 Track 1

Rank	Team ID	S1 score
1	37	0.9467
2	**5 (Ours)**	**0.9459**
3	8	0.9263
4	19	0.9249
5	118	0.9235
6	42	0.9157

5. Conclusion

In this paper, we have presented a DTC framework for counting vehicles specific to many travel paths and scenes. Our solution consists of a fast batch vehicle detector, a three-fold matching scheme for vehicle tracking, and predictive counter that assigns motion labels on vehicles' trajectories in a vectorized manner. Both the effectiveness and efficiency of our solution are experimentally illustrated.

Acknowledgment

We owe a big thank to Ho Chi Minh City International University—Vietnam National University (HCMIU-VNU) for facilitating our work. Our sincerest appreciations also go to all our colleagues for their support that have significantly improved the manuscript.

References

[1] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence, 33(9):1806–1819, 2011. 2

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016. 2, 4

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. 2, 3

[4] K. N. Bui, H. Yi, and J. Cho. A vehicle counts by class framework using distinguished regions tracking at multiple intersections. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2466–2474, 2020. 3

[5] Q. Chu, W. Ouyang, H. Li, X. Wang, B. Liu, and N. Yu. Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4846–4855, 2017. 2

[6] B. Cuan, K. Idrissi, and C. Garcia. Deep siamese network for multiple object tracking. In *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, Aug 2018. 2

[7] A. Dehghan, S. M. Assari, and M. Shah. Gmmcp tracker: Globally optimal generalized maximum multi clique problem for multiple object tracking. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4091–4099, 2015. 2

[8] A. Dehghan, Y. Tian, P. H. S. Torr, and M. Shah. Target identity-aware network flow for online multiple target tracking. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1146–1154, 2015. 2

[9] K. Fang, Y. Xiang, X. Li, and S. Savarese. Recurrent autoregressive networks for online multi-object tracking. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 466–475, 2018. 2

[10] Xu Gao and Tingting Jiang. Osmo: Online specific models for occlusion in multiple object tracking under surveillance scene. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, page 201–210, New York, NY, USA, 2018. Association for Computing Machinery. 2

[11] G. Ghiasi, T. Lin, and Q. V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7029–7038, 2019. 2

[12] R. Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 2

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 2

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 2

[15] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. 4

[16] Li Zhang, Yuan Li, and R. Nevatia. Global data association for multi-object tracking using network flows. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. 2

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. 2

[18] Z. Liu, W. Zhang, X. Gao, H. Meng, X. Tan, X. Zhu, Z. Xue, X. Ye, H. Zhang, S. Wen, and E. Ding. Robust movement-specific vehicle counting at crowded intersections. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2617–2625, 2020. 1, 2, 3

[19] Anton Milan, S. Hamid Rezatofighi, Anthony Dick, Ian Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4225–4232. AAAI Press, 2017. 2

[20] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M. Chang, X. Yang, L. Zheng, A. Sharma, R. Chellappa, and P. Chakraborty. The 4th ai city challenge. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2665–2674, 2020. 1, 3

[21] A. OSPINA and F. TORRES. Countor: count without bells and whistles. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2559–2565, 2020. 1, 3

[22] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. 2

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. 2

[24] A. Sadeghian, A. Alahi, and S. Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 300–311, 2017. 2

[25] Bing Shuai, Andrew G. Berneshawi, Davide Modolo, and Joseph Tighe. Multi-object tracking with siamese track-rcnn, 2020. 2

[26] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020. 2

[27] S. Tang, B. Andres, M. Andriluka, and B. Schiele. Subgraph decomposition for multi-target tracking. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5033–5041, 2015. 2

[28] M. Tran, T. V. Nguyen, T. Hoang, T. Le, K. Nguyen, D. Dinh, T. Nguyen, H. Nguyen, X. Hoang, T. Nguyen, V. Vo-Ho,

T. Do, L. Nguyen, M. Le, H. Nguyen-Dinh, T. Pham, X. Nguyen, E. Nguyen, Q. Tran, H. Tran, H. Dao, M. Tran, Q. Nguyen, T. Nguyen, T. Vu-Le, G. Diep, and M. N. Do. itask - intelligent traffic analysis software kit. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2607–2616, 2020. 3

[29] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network, 2021. 2, 3

[30] X. Wang, E. Türetken, F. Fleuret, and P. Fua. Tracking interacting objects using intertwined flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2312–2326, 2016. 2

[31] Z. Wang, B. Bai, Y. Xie, T. Xing, B. Zhong, Q. Zhou, Y. Meng, B. Xu, Z. Song, P. Xu, R. Hu, and H. Chai. Robust and fast vehicle turn-counts at intersections via an integrated solution from detection, tracking and trajectory modeling. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2598–2606, 2020. 1, 3

[32] Zhiwei Wei, Chenzhen Duan, Xinghao Song, Ye Tian, and Hongpeng Wang. Amrnet: Chips augmentation in aerial images object detection, 2020. 2, 3

[33] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, 2017. 2, 4

[34] L. Yu, Q. Feng, Y. Qian, W. Liu, and A. G. Hauptmann. Zero-virus*: Zero-shot vehicle route understanding system for intelligent transportation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2534–2543, 2020. 1, 3

[35] Y. Zeng, X. Wu, and J. Cao. The research and analysis of hungarian algorithm in the structure index reduction for dae. In *2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering Science*, pages 446–450, 2012. 4

[36] Ji Zhu, Hua Yang, Nian Liu, Minyoung Kim, Wenjun Zhang, and Ming-Hsuan Yang. Online multi-object tracking with dual matching attention networks. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 379–396, Cham, 2018. Springer International Publishing. 2