Proceedings of Engineering and Technology Innovation, vol. 31, 2025, pp. 15-28

Enhancing Synchronization of YOLO-Based Traffic Detection on Low-End Devices by Using the COID Algorithm

Cheng-Hsiu Li*, Hao-Sheng Hou

Department of Information Management, National Taitung Junior College, Taitung, Taiwan, ROC

Received 11 June 2025; received in revised form 28 August 2025; accepted 29 August 2025

DOI: https://doi.org/10.46604/peti.2025.15264

Abstract

The rapid increase in population and vehicle usage intensifies traffic congestion, creating a pressing need for accurate real-time vehicle detection. While the you only look once (YOLO) algorithm enables efficient end-to-end detection, its performance is constrained by hardware limitations, leading to desynchronization on low-end devices. To address this limitation, this study proposes the catch one image detection (COID) algorithm, which restores synchronization without altering the YOLO architecture or requiring retraining. By dynamically adjusting the frame capture interval according to hardware capability, COID ensures real-time alignment between detection and live video streams while reducing deployment complexity. Experimental evaluations on high-, mid-, and low-end devices, including validation on multi-intersection surveillance footage under low-visibility conditions, confirm the robustness and applicability of COID. These findings demonstrate COID's practicality as a scalable solution for real-world intelligent traffic monitoring.

Keywords: YOLO, traffic congestion, low-end devices, COID algorithm

1. Introduction

With the rapid growth of the global population, the number of vehicles on the road has increased significantly, resulting in severe traffic congestion in urban areas. Traditional traffic control technologies are increasingly inadequate for maintaining smooth and efficient traffic flow [1]. In such high-density traffic environments, accurate and real-time vehicle detection plays a crucial role in preventing violations, reducing accidents, and mitigating congestion. Early vehicle detection techniques mainly utilized handcrafted features and classical machine learning algorithms, including the Viola–Jones image classification algorithm, histogram of oriented gradients (HOG) combined with support vector machines (SVM), and deformable part-based models (DPM) [2]. These methods, however, are often constrained in terms of accuracy and computational efficiency.

Recent advancements in deep learning have introduced you only look once (YOLO), an object detection framework that formulates detection as a single regression problem. YOLO predicts bounding boxes and class probabilities in a single pass through a convolutional neural network, achieving high processing speeds of up to 45 frames per second (fps) in its base model and 155 fps in Fast-YOLO, while maintaining competitive accuracy [3]. Due to its high computational efficiency, YOLO has been widely adopted in traffic monitoring [4-7], intelligent transportation systems [1, 8], and autonomous driving applications [9-12]. Typically, YOLO is used for object recognition, while platforms such as OpenCV handle annotation and visualization tasks [1]. Despite its advantages, the implementation of YOLO in real-time applications faces a major limitation: it demands substantial computational power. Most deep learning-based object detection systems require high-end hardware to ensure real-time performance [5-6, 9]. On low-end devices, YOLO may lose synchronization with live surveillance footage, significantly reducing its practicality in time-sensitive traffic control applications.

* Corresponding author. E-mail address: chsiu.lee@gmail.com

To address the synchronization problem, this study proposes a novel algorithm termed catch one image detection (COID). The primary components influencing computing performance are the graphics processing unit (GPU) and central processing unit (CPU), with network bandwidth and memory capacity also playing critical roles. Three computer systems with distinct hardware configurations—classified as high-end, mid-range, and low-end based on GPU, CPU, and memory—were selected as experimental platforms. The COID algorithm enhances YOLO's real-time performance on low-end hardware without altering the model architecture by dynamically adjusting the frame capture interval according to measured processing time. Overall, the proposed method provides a practical solution to synchronization issues between detection outputs and live video streams, thereby enhancing the accessibility of real-time vehicle detection for traffic monitoring and facilitating the timely delivery of traffic information to intelligent traffic control systems.

2. Literature Review

Recent developments have focused on deploying YOLO-based detection on embedded platforms to achieve real-time performance in traffic scenarios. Koay et al. [13] addressed the challenges of object detection in aerial imagery, where high scene variability and small object sizes often reduce detection accuracy. While deep learning models like YOLO have enhanced detection performance, their deployment on low-cost edge devices remains challenging. To overcome these limitations, the authors proposed YOLO-RTUAV, an improved version of YOLOv4-Tiny optimized for edge hardware. When evaluated on the VAID and COWC datasets, YOLO-RTUAV achieved superior mean average precision (mAP) and inference speed compared to other lightweight models.

Farid et al. [14] proposed an enhanced vehicle detection and classification approach using a fine-tuned YOLOv5 model, addressing the limitations posed by insufficiently labeled training data. By employing transfer learning, the authors adapted a pre-trained YOLOv5 to diverse traffic conditions, including varying densities, occlusions, and weather conditions. Extensive image and video datasets were manually annotated to support this adaptation.

Zhang et al. [15] proposed an improved YOLOv5-based vehicle detection method aimed at reducing false detection rates caused by occlusions in complex traffic scenarios. A Flip-Mosaic data augmentation algorithm was introduced to enhance the network's detection performance for small targets. The model was trained on a custom multi-class vehicle dataset collected under various traffic and environmental conditions. Experimental results confirmed that the proposed approach improved detection accuracy and significantly reduced false positives.

Ali et al. [16] explored the efficient deployment of various YOLO architectures—YOLOv3, v3-tiny, v4, and v4-tiny—for vehicle detection, with a particular focus on Jetson Nano devices. Their study compared the models' performance in terms of accuracy and computational efficiency. Among all evaluated versions, YOLOv4 achieved the highest accuracy at 95.41%. The paper serves as a practical reference for deep learning-based vehicle detection, covering both model selection and hardware-specific optimization for embedded environments.

Lee and Hwang [17] proposed an adaptive frame control (AFC) mechanism to enhance the real-time performance of YOLO-based object detection on embedded devices such as Jetson Nano and TX2. By discarding stale frames before passing them to the YOLO core, their method maintains synchronization with real-time video streams without modifying YOLO itself. The AFC layer reduces cumulative service delay and ensures smooth detection under hardware constraints.

However, a significant research gap persists in achieving real-time synchronization of YOLO-based detection with live video streams on general-purpose low-end devices, which are widely used in budget-constrained environments. To address this issue, the present study proposes the COID algorithm, which enables synchronized, real-time traffic detection using YOLO on low-end hardware without requiring any modification to the original model architecture.

3. Methodology

In this study, the image processing techniques YOLO and OpenCV are employed to count, classify, and identify vehicles. Surveillance video streams processed through the OpenCV software library are accurately detected and recognized. This research focuses on the detection of cars within traffic lanes, with particular emphasis on mitigating real-time detection delays caused by low-end hardware.

3.1. Traffic data

This study uses publicly available surveillance footage from intersection traffic cameras. As a case example, the proposed system is applied to footage from an intersection on Section 1, Zhonghua Road (northeast direction) in Taitung City. The video stream is accessible at the following URL: https://trafficvideo.ttcpb.gov.tw/9e79f5d0. To ensure robustness and generalizability, the collected surveillance footage encompasses diverse real-world conditions, including daytime and nighttime scenarios as well as varying traffic densities such as congested and free-flowing traffic.

3.2. Basic YOLO detection approach

In the original object detection approach using YOLO, frames are periodically extracted from traffic surveillance video streams, and the pretrained YOLOv8 model is employed to detect and label vehicles. The detection and labeling process is described as follows:

Assuming that the real-time video stream is sourced from a traffic camera URL at a specific intersection, the stream is represented as a sequence of consecutive frames as:

$$F = \{f_1, f_2, f_3, \ldots, f_m\} \tag{1}$$

Each frame $f_t \in R^{H\times W\times 3}$ is a color image captured at time t, composed of H pixels in height, W pixels in width, and three color channels. For each extracted frame f_t, the YOLO model performs the following inference:

$$R_t = \text{YOLO}(f_t) \tag{2}$$

where $R_t = \{b_1, b_2, \cdots, b_n\}$ and each b_i is a detected object bounding box, defined as:

$$b_i = (x_{i1},\ y_{i1},\ x_{i2},\ y_{i2},\ conf_i,\ class_i) \tag{3}$$

where (x_{i1}, y_{i1}) and (x_{i2}, y_{i2}) denote the pixel coordinates of the top-left and bottom-right corners of the bounding box, respectively, thereby defining the spatial extent of the detected object. The confidence score $conf_i$, ranging from 0 to 1, quantifies the model's certainty that the bounding box contains an object of the specified category. Finally, the semantic label $class_i$ indicates the object category (e.g., "car," "truck," "pedestrian") assigned by the YOLO model. Collectively, this tuple provides both the geometric localization and semantic classification of each detected object.

Because the focus is on vehicle detection, only objects classified as "car" ($class_i$ = car) are retained. For each detected car b_i, its center coordinates (c_{xi}, c_{yi}) are computed as:

$$c_{xi} = \frac{x_{i1} + x_{i2}}{2},\ c_{yi} = \frac{y_{i1} + y_{i2}}{2} \tag{4}$$

The tracking algorithm *tracker.update*() is employed to process all vehicle bounding boxes and thereby assign a unique ID to each vehicle as shown in the formula below. Finally, each frame f_t is rendered with bounding boxes, center points (c_{xi}, c_{yi}), and corresponding ID labels id_i, completing the vehicle detection and labeling process.

$$\{b_i\}_{i=1}^{n} \rightarrow \{x_{i1},\ y_{i1},\ x_{i2},\ y_{i2},\ id_i\} \tag{5}$$

3.3. COID algorithm

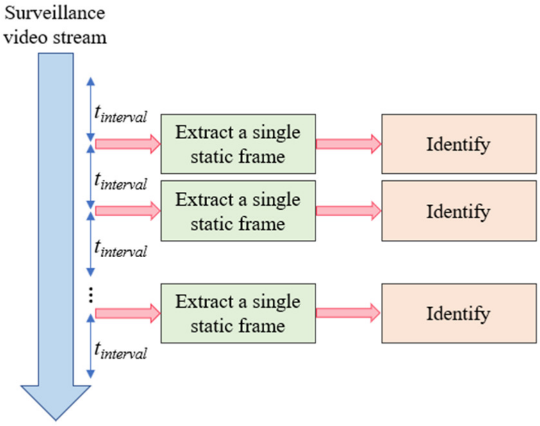


Fig. 1 Processing flow diagram of the COID algorithm

This study proposes a novel detection approach named the COID algorithm. The core concept of this algorithm is that it dynamically adjusts the frame capture interval based on the processing time required for detection. Specifically, the detection and recognition processing time is calculated. By decoupling the frames in the original surveillance video stream, the system captures individual frames for recognition at intervals is defined by the measured processing time. The overall workflow is illustrated in Fig. 1, and the primary steps are described as follows:

(1) Computing the hardware processing time: The processing time of the system is measured during the initial image recognition task, yielding two values: t_{det}, the time requires for performing detection, and $t_{non\text{-}det}$, the time spent without executing detection. The algorithmic procedure is described as follows:

Algorithm 1
1. If detection has not yet been executed then
2. Save current frame as "photo_name.jpg"
3. Measure execution time for running traffic detection on the frame
4. Output the first detection time
5. Mark detection as executed
6. End if

(2) Calculating the interval for static image capture from the hardware processing time: In practice, video streams are subject to buffering, which introduces additional latency between frame delivery and processing. The primary role of the interval is to align each static frame capture with this buffering delay, ensuring that the detection process remains synchronized with the effective video stream rather than lagging behind it. Subsequently, the interval uses for capturing static images, denoted $t_{interval}$, is computed using the formula $t_{interval} = t_{det} / t_{non\text{-}det} \times t_{fine\text{-}tune}$. This interval determines the frequency at which images are captured for subsequent recognition. The constant $c_{fine\text{-}tune}$ is a fine-tuning parameter, set to 3 by default in the experiments, and can be adjusted based on the synchronization conditions.

The rationale for this formula is that t_{det} represents the per-frame detection time, which includes YOLO inference and post-processing, whereas $t_{non\text{-}det}$ denotes the time spent on non-detection operations, such as I/O, rendering, and bookkeeping. The ratio t_{det} / $t_{non\text{-}det}$ reflects the extent to which detection dominates the processing loop, thereby indicating the potential backlog in frame handling. Multiplying this ratio by $c_{fine\text{-}tune}$ compensates for residual time drift and hardware-specific latency, ensuring that the frame capture cadence aligns with the system's processing capability. In practice, a larger $c_{fine\text{-}tune}$ increases the safety margin against backlog, enhancing synchronization under high computational load, whereas a smaller value provides smoother visuals but risks reintroducing delay when the workload spikes. Preliminary tests indicate that setting $c_{fine\text{-}tune} = 3$ provides robust synchronization on mid- and low-end systems, whereas a reduced value of 2.5 yield optimal performance on the high-end system by balancing synchronization with frame smoothness. The algorithmic procedure is described as follows:

Algorithm 2

```
If the calculation has not yet been performed then
    Compute time_setting = round ((execution_time / (elapsed − execution_time)) × finetune)
    Mark calculation as completed
End if
```

(3) Dynamic adjustment of the capture interval: The algorithm adjusts the capture interval based on changes in hardware performance to ensure optimal processing efficiency. The algorithmic procedure is described as follows:

Algorithm 3

```
If time_counter equals time_setting then
    Save current frame as "photo_name.jpg"
    Perform traffic detection on the saved frame
End if
```

(4) Perform inference using the YOLO model: Each frame is subjected to YOLO, which detects and annotates objects within the image. The algorithmic procedure is described as follows:

Algorithm 4

```
Function Traffic_Detection (frame):
    Run YOLO model to predict objects in the frame
    Extract detection results (bounding box coordinates and class IDs)
    Initialize an empty list for detected cars
    For each detected object do
        Retrieve bounding box (x1, y1, x2, y2) and class ID
        If class corresponds to 'car' then
            Append bounding box to the car list
    End for
    Update tracker with detected car bounding boxes
    For each tracked object do
        Compute center point (cx, cy)
        Draw a circle at the object center
        Draw a bounding box rectangle around the object
        Display object ID near bounding box
    End for
    Display the total number of detected cars on the frame
    Show updated frame as "Traffic Detection"
End function
```

(5) Pseudocode of COID:

Algorithm 5 COID (catch one image detection)

Input: Video stream V, YOLO detector M, fine-tuning constant $c_{fine\text{-}tune}$

Output: Synchronized detection results

```
Initialize system parameters
Measure initial detection time t_det
Measure non-detection time t_non-det
Compute capture interval:
      t_interval = (t_det / (t_non-det)) × c_fine-tune
while V is active do
      if time_counter == t_interval then
        Capture one frame F from V
        Perform detection using YOLO model M on F
        Display or record detection results
            Reset time_counter
          else
            Increment time_counter
      end if
end while
```

4. Results

To verify the effectiveness of the COID algorithm in addressing synchronization issues between the detection system and surveillance footage under limited hardware capabilities, experiments are conducted using three computer hardware configurations. These configurations are classified as high-end, mid-range, and low-end according to GPU, CPU, and memory, with Computers #1, #2, and #3 corresponding to these categories, respectively. The aim is to evaluate the algorithm's ability to enhance the real-time detection performance of YOLO when processing surveillance video streams. The hardware specifications of the test systems are detailed in Table 1.

In this context, the synchronization issue is quantitatively defined as the time difference, measures in seconds (s), between the vehicle's position in the detection output and its position in the live surveillance feed. All experiments are conducted under identical network conditions to eliminate the impact of network speed and related factors on the accuracy of the results.

Table 1 Computer system specifications

Specification / Computer ID	Computer #1	Computer #2	Computer #3
Central processing unit (CPU)	Intel(R) Core(TM) i7-9750H CPU @ 2.60 GHz	Intel(R) Core(TM) i5-5200U CPU @ 2.20 GHz	Intel(R) Celeron(R) N4100 CPU @ 1.10 GHz
Random access memory (RAM)	16.0 GB	8.0 GB	8.0 GB
Graphics processing unit (GPU)	NVIDIA GeForce GTX 1050	Intel(R) UHD Graphics 5500	Intel(R) UHD Graphics 600
Dedicated GPU	Yes	No	No
Relative level	High-end	Mid-range	Low-end

4.1. Basic YOLO detection results

In this study, the basic YOLO model is first tested. For the evaluation on Computer #1 (high-end device), a white car (indicated by the detection bounding box) gradually exits the surveillance footage, but the detection result continues to display the vehicle within the frame. This indicates clear asynchrony between the detection output and the actual surveillance feed. The detected results do not correspond to the real-time state of the monitored scene. The test outcome is presented in Fig. 2.

Tests are then conducted on Computers #2 and #3 (mid-range and low-end devices). The results reveal the same issue as for Computer #1, namely a lack of synchronization between the detection output and surveillance footage. However, the synchronization delays are even larger. The results are illustrated in Fig. 3.

Quantitatively, the average per-frame delays for the basic YOLO model are measured at 0.265 s for Computer #1, 0.415 s for Computer #2, and 0.966 s for Computer #3. These measurements confirm that lower hardware specifications led to greater desynchronization.

Fig. 2 YOLO detection results (Computer #1)

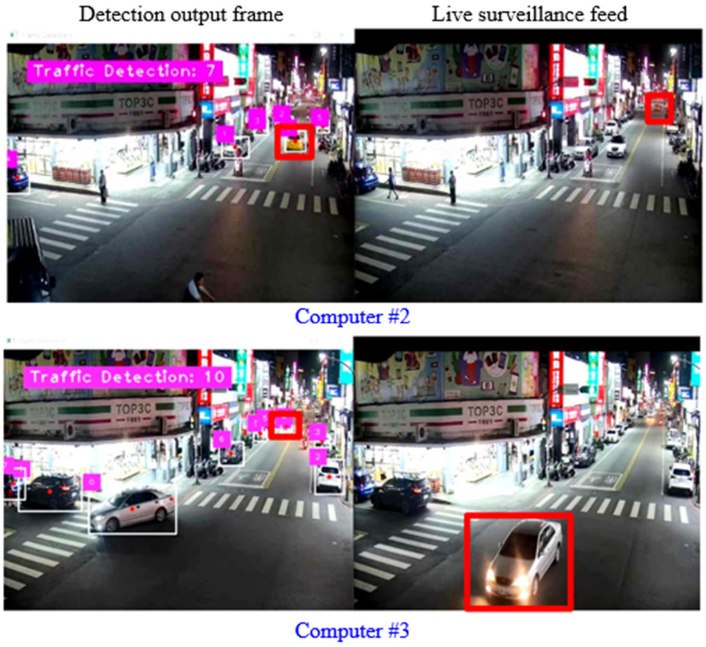


Fig. 3 YOLO detection results (Computers #2 and #3)

4.2. COID-YOLO detection results

The original YOLO detection method is subsequently replaced by the proposed COID algorithm. In the test on Computer #1, the synchronization of the detection output with the surveillance footage is evaluated by comparing the position of a black car (indicated by the detection bounding box). The position of the vehicle is found to be nearly identical in the surveillance feed and the detection output, indicating that the detection output matches the real-time surveillance frame. The test results are displayed in Fig. 4.

The first 11 data entries from the algorithm's execution process are extracted and analyzed. The analysis results are presented in Fig. 5. On Computer #1, the average image recognition time per frame (t_{det}) is 0.174 s, the average time interval between frame captures ($t_{interval}$) is 0.146 s, the frame rate is 6.849 fps, and the fine-tuning constant ($c_{fine\text{-}tune}$) was 2.5.

Fig. 4 YOLO-COID detection results (Computer #1)

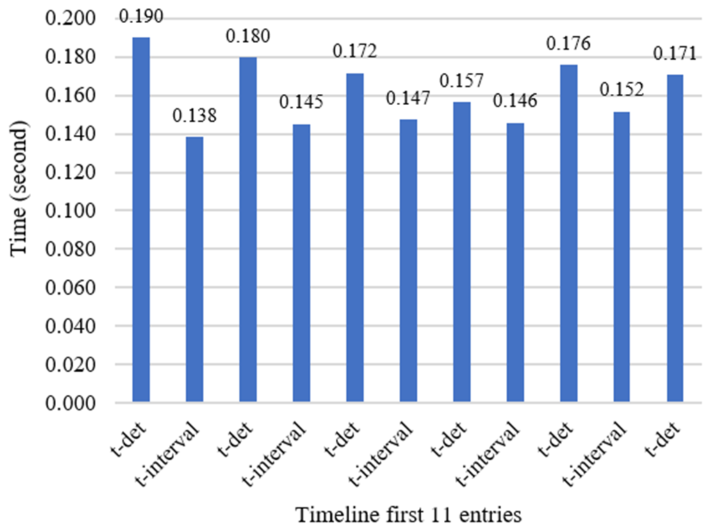


Fig. 5 Time taken to execute YOLO-COID algorithm (Computer #1)

Tests are then conducted on Computers #2 and #3, and the results are shown in Figs. 6 and 7, respectively. The detection output is nearly identical to the real-time surveillance footage. The less-powerful computer requires more time for image recognition, leading to longer intervals between captured frames. Although real-time detection is achieved, the visual output is less smooth than on the more powerful computer.

The analysis involving the first 11 data entries from the algorithmic process is repeated; the results are presented in Figs. 8 and 9. For Computer #2, t_{det} is 0.539 s, $t_{interval}$ is 0.304 s, the frame rate is 3.289 fps, and $c_{fine\text{-}tune}$ is 3. On Computer #3, t_{det} is 1.159 s, $t_{interval}$ is 0.544 s, the frame rate is 1.838 fps, and $c_{fine\text{-}tune}$ is 3.

Fig. 6 YOLO-COID detection results (Computer #2)

Fig. 7 YOLO-COID detection results (Computer #3)

To further assess the overall performance of the detection system, the evaluation is extended to include mAP metrics in accordance with the common objects in context (COCO) evaluation protocol. Specifically, mAP@0.5 (Intersection over Union, IoU threshold = 0.5) and mAP@0.5:0.95 (averaged across IoU thresholds from 0.5 to 0.95 in 0.05 increments) are calculated to measure detection accuracy under both lenient and strict localization criteria. The detection tests are performed using the pretrained YOLOv8 model without any modifications to its network architecture or parameters. The proposed COID algorithm solely adjusts the frame capture interval to synchronize the acquisition of video frames with the detection process. This design reduces redundant detections and misses targets that arise from frame sampling mismatches.

Experimental results show that the COID-enhanced YOLOv8 achieves a mAP@0.5 of 0.622 and a mAP@0.5:0.95 of 0.406 on the test dataset. These results demonstrate that even without altering the YOLOv8 model itself, optimizing the temporal sampling of input frames can improve detection robustness. Moreover, this approach maintains competitive accuracy, especially in scenarios are constrained by processing resources.

To quantify the input characteristics of the network stream, the study inspects the metadata using FFmpeg, which reports an encodes frame rate of 25 fps for the surveillance feed. Real-time probing with a lightweight OpenCV client under the same network conditions as the experiments yield an effective delivery rate of 5.51 fps. This value reflects the combined effects of network latency, buffering, and server-side throttling for public video streams. With COID enables, the capture cadence is adaptively adjusted according to each device's processing capability, resulting in 6.849 fps on Computer #1, 3.289 fps on Computer #2, and 1.838 fps on Computer #3. These lower capture rates effectively reduce the average per-frame delay and improve synchronization with the live feed. Importantly, despite the lower capture rate compares with the 25 fps source, detection accuracy on the annotated subset remaines stable (mAP@0.5 = 0.622, mAP@0.5:0.95 = 0.406). This indicates that the COID strategy retained essential traffic information while mitigating temporal lag.

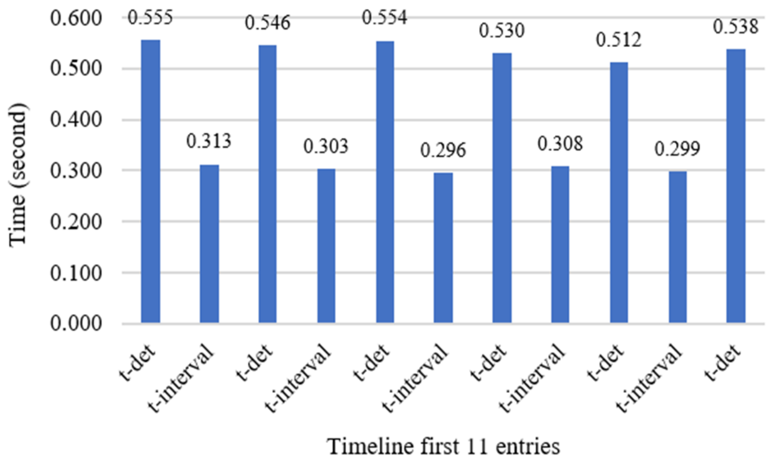


Fig. 8 Time taken to execute YOLO-COID algorithm (Computer #2)

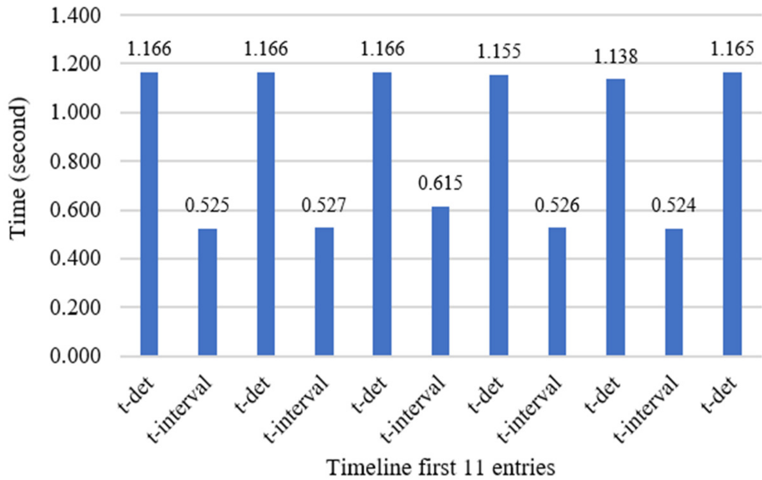


Fig. 9 Time taken to execute YOLO-COID algorithm (Computer #3)

5. Discussion

The discussion section synthesizes the effectiveness and practical implications of the proposed COID algorithm based on the experimental findings. Three aspects are highlighted to provide a comprehensive analysis. First, COID is compared with existing synchronization methods to underscore its advantages in deployment and integration. Second, its ability to maintain real-time alignment between detection outputs and surveillance streams is assessed through experiments at multiple intersections, including challenging nighttime scenarios. Finally, the influence of hardware configurations and streaming conditions on performance is examined, with results contextualize using both system-level benchmarks and the intrinsic properties of the surveillance feed. Together, these analyses demonstrate the robustness, efficiency, and applicability of COID in real-world traffic monitoring environments.

5.1. Comparative analysis with existing methods

Several prior studies have addressed the challenge of maintaining real-time synchronization in YOLO-based detection under hardware constraints. For instance, Lee and Hwang [17] proposed an AFC mechanism that discards stale frames before passing them to the YOLO core, thereby preserving alignment with the video stream. While AFC effectively reduces service delay, it requires modifications to the pre-processing pipeline of YOLO, which may increase integration complexity. Similarly, Koay et al. [13] developed YOLO-RTUAV, an optimized YOLOv4-Tiny model for low-cost edge devices. This approach achieved notable improvements in inference speed and detection accuracy. However, it required model-level modifications and retraining, limiting its direct applicability to existing deployments.

In contrast, the proposed COID algorithm achieves synchronization improvements without altering the YOLO model architecture or requiring retraining. By dynamically adjusting the frame capture interval based on measured processing time, COID can be seamlessly integrated into existing YOLO-based systems across various hardware configurations. Experimental results demonstrate that COID delivers synchronization performance comparable to AFC on low-end devices, while offering significantly lower deployment complexity, as summarized in Table 2.

Table 2 Comparison of existing synchronization methods and COID

Method	Hardware dependency	Model modification	Retraining required	Synchronization improvement
AFC [17]	Medium	Yes	No	High
YOLO-RTUAV [13]	Low	Yes	Yes	Medium
COID (Proposed)	Low	No	No	High

5.2. *Synchronization performance of YOLO versus YOLO-COID*

Experimental results show that all three computers, with different hardware configurations, experience synchronization issues when using the basic YOLO algorithm. Specifically, the alignment between detection outputs and the original surveillance footage is inconsistent (Figs. 2 and 3). When YOLO is replaced with the YOLO-COID algorithm and an image capture interval based on the formula $t_{interval} = t_{det} / t_{non\text{-}det} \times c_{fine\text{-}tune}$ is employed, with $c_{fine\text{-}tune}$ set to 2.5 and 3, synchronization improves considerably on all three computers. This adjustment enables real-time synchronization between the detection outputs and the surveillance video (Figs. 4, 6, and 7).

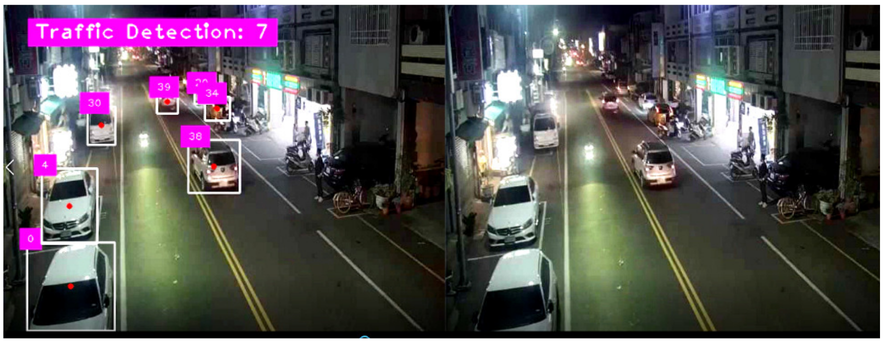
Table 3 Surveillance footage sampled for synchronization testing

ID	Intersection name	Surveillance URL
1	Kaifeng St.	https://trafficvideo.ttcpb.gov.tw/f1392a2c
2	Binlang Rd., Ln. 283	https://trafficvideo.ttcpb.gov.tw/d4676640
3	Zhonghua Rd., Sec. 2	https://trafficvideo.ttcpb.gov.tw/7847ac3c
4	Xinsheng Rd.	https://trafficvideo.ttcpb.gov.tw/6247ab39
5	Zhonghua Rd., Sec. 1	https://trafficvideo.ttcpb.gov.tw/39eab823

Table 4 Synchronization test results on low-end device (Computer #3)

ID	Intersection name	Detection results (left: detection output frame; right: live surveillance feed)
1	Kaifeng St.	
2	Binlang Rd., Ln. 283	

Table 4 Synchronization test results on low-end device (Computer #3) (continued)

ID	Intersection name	Detection results (left: detection output frame; right: live surveillance feed)
3	Zhonghua Rd., Sec. 2	
4	Xinsheng Rd.	
5	Zhonghua Rd., Sec. 1	

To further validate the effectiveness of the YOLO-COID algorithm, an additional synchronization test is conducted using surveillance footage randomly sampled from five intersections in Taitung County, Taiwan (https://tw.live/county/taitungcounty/). The evaluation focuses on the low-end device (Computer #3) to examine performance under constrained hardware conditions. Details of the five surveillance feeds are summarized in Table 3, and the experimental results are shown in Table 4. The findings demonstrate that even in nighttime scenarios with limited visibility, the YOLO-COID algorithm successfully maintains synchronized vehicle detection across all tested intersections on the low-end device (Computer #3). This confirms its robustness and applicability in real-world traffic monitoring environments. Importantly, the detection accuracy on the annotated subset remains stable (mAP@0.5 = 0.622, mAP@0.5:0.95 = 0.406), further supporting the reliability of YOLO-COID in retaining essential traffic information while mitigating temporal lag.

5.3. Processing speeds for different hardware configurations

Deep learning models are inherently computationally demanding. Three computer systems with different hardware specifications—in terms of GPU, CPU, and memory—are selected for experimental comparison. Computers #1 to #3 are labeled high-end, mid-range, and low-end, respectively. The performance of the COID and YOLO algorithms is then evaluated on each system.

Fig. 10 summarizes the COID processing times for the three computers. All systems achieve real-time synchronized detection, but the most powerful computer is considerably faster. Specifically, the high-end system (Computer #1) has an average detection time of 0.174 s, whereas the mid-range and low-end systems (Computers #2 and #3) require 0.539 and 1.159 s, respectively.

Similarly, image capture intervals are inversely correlated with hardware level. Computers #1, #2, and #3 achieve capture intervals of 0.146, 0.304, and 0.544 s, respectively. The capture interval directly influences the smoothness of the real-time detection. Computer #1 achieve the highest frame rate at 6.849 fps, resulting in the smoothest display. Computer #2 had 3.289 fps, while Computer #3 reaches 1.838 fps (least smooth).

To further contextualize these results, the metadata of the surveillance stream is first inspected using FFmpeg, which reports an encoded frame rate of 25 fps. Real-time probing with a lightweight OpenCV client under the same network conditions reveal an effective delivery rate of only 5.51 fps, due to latency, buffering, and server-side throttling. With COID enables, the frame capture cadence is adaptively aligned to each device's processing capability, lowering per-frame delay and improving synchronization with the live feed.

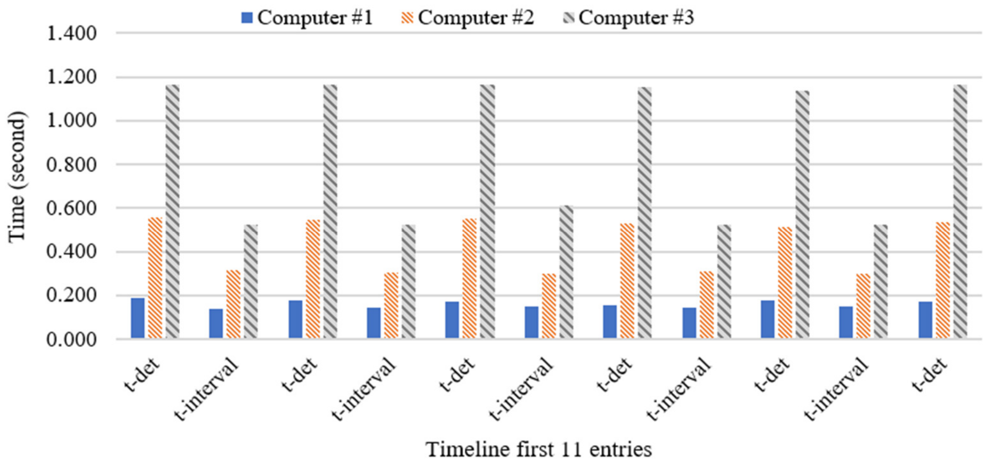


Fig. 10 Performance of YOLO-COID detection on different computers

6. Conclusions

This study developed and implemented the COID algorithm to address synchronization challenges encountered when applying the YOLO framework on devices with different hardware capabilities. COID was designed to enable real-time vehicle detection on low-end devices by dynamically adjusting the frame capture interval, restoring synchronization without modifying the YOLO architecture or requiring retraining. The algorithm achieves comparable performance to more complex approaches while reducing deployment complexity and enabling seamless integration into existing systems. Experimental evaluations across different hardware configurations and multiple surveillance feeds validated its robustness and applicability in real-world traffic monitoring. The main conclusions of this study are as follows:

(1) Overcoming integration complexity: Unlike methods such as AFC or YOLO-RTUAV, which require preprocessing modifications or retraining, COID restores synchronization through adaptive adjustment of the frame capture interval. This approach achieves comparable performance while minimizing integration complexity.

(2) Validation for real-time monitoring: Experimental results revealed that basic YOLO suffered from desynchronization. In contrast, COID maintained synchronous alignment between detection outputs and surveillance streams. Tests using surveillance footage from five intersections further confirmed its robustness, including under low-visibility nighttime conditions.

(3) Performance across hardware configurations: On high-, mid-, and low-end devices, COID achieved average detection times of 0.174 s, 0.539 s, and 1.159 s. The corresponding frame rates were 6.849, 3.289, and 1.838 fps. Although the surveillance stream was encoded at 25 fps, real-time probing revealed an effective rate of only 5.51 fps due to latency and buffering. COID adaptively matched the capture cadence to each device's capability, thereby reducing per-frame delay and improving synchronization.

Current evaluations relied on publicly available surveillance feeds, and COID was tested only with YOLO-based detectors. Moreover, while it improves synchronization, it does not directly accelerate inference speed. Future work will explore hardware-aware scheduling, lightweight detection models, and adaptive frame selection to further enhance performance under constrained environments.

Acknowledgments

This work was supported by the Ministry of Science and Technology (MOST) (formerly known as the National Science Council, NSC) under Grant No. NSTC 114-2637-E-602 -002 -.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] B. Gomathi and G. Ashwin, "Intelligent Traffic Management System Using YOLO Machine Learning Model," Disruptive Technologies for Big Data and Cloud Applications, Lecture Notes in Electrical Engineering, vol. 905, pp. 119-128, 2022.

[2] S. Kang, Z. Hu, L. Liu, K. Zhang, and Z. Cao, "Object Detection YOLO Algorithms and Their Industrial Applications: Overview and Comparative Analysis," Electronics, vol. 14, no. 6, article no. 1104, 2025.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779-788, 2016.

[4] A. R. Kalva, J. S. Chelluboina, and B. Bharathi, "Smart Traffic Monitoring System Using YOLO and Deep Learning Techniques," 7th International Conference on Trends in Electronics and Informatics, pp. 831-837, 2023.

[5] R. Shreeshayana, L. Lakshmaiah, P. O. Kavya, and D. Anil, "Smart Traffic Control and Management System Using Machine Learning Techniques," Tuijin Jishu/Journal of Propulsion Technology, vol. 44, no. 5, pp. 1889-1896, 2023.

[6] A. P. Rangari, A. R. Chouthmol, C. Kadadas, P. Pal, and S. Kumar Singh, "Deep Learning Based Smart Traffic Light System Using Image Processing with YOLO v7," 4th International Conference on Circuits, Control, Communication and Computing, pp. 129-132, 2022.

[7] L. Dhake, A. Talekar, and K. Bembade, "Traffic Management Using YOLO and Convolutional Fizzy Neural Network," International Journal of Engineering Research & Technology, vol. 13, no. 07, article no. IJERTV13IS070074, 2024.

[8] P. Kunekar, Y. Narule, R. Mahajan, S. Mandlapure, E. Mehendale, and Y. Meshram, "Traffic Management System Using YOLO Algorithm," Engineering Proceedings, vol. 59, no. 1, article no. 210, 2023.

[9] P. Azevedo and V. Santos, "YOLO-Based Object Detection and Tracking for Autonomous Vehicles Using Edge Devices," ROBOT2022: Fifth Iberian Robotics Conference, pp. 297-308, 2022.

[10] J. Hu, P. Wang, J. Yang, and L. Ni, "An Improved Algorithm for Moving Object Detection in YOLO UAV Videos," Journal of Computers, vol. 33, no. 3, pp. 147-158, 2022.

[11] H. Fujiyoshi, T. Hirakawa, and T. Yamashita, "Deep Learning-Based Image Recognition for Autonomous Driving," IATSS Research, vol. 43, no. 4, pp. 244-252, 2019.

[12] F. I. Abd-AL Sahib, H. B. Taher, and R. F. Ghani, "Detection of the Autonomous Car Robot Using Yolo," Journal of Physics: Conference Series, vol. 1879, no. 3, article no. 032129, 2021.

[13] H. V. Koay, J. H. Chuah, C. O. Chow, Y. L. Chang, and K. K. Yong, "YOLO-RTUAV: Towards Real-Time Vehicle Detection through Aerial Images with Low-Cost Edge Devices," Remote Sensing, vol. 13, no. 21, article no. 4196, 2021.

[14] A. Farid, F. Hussain, K. Khan, M. Shahzad, U. Khan, and Z. Mahmood, "A Fast and Accurate Real-Time Vehicle Detection Method Using Deep Learning for Unconstrained Environments," Applied Sciences, vol. 13, no. 5, article no. 3059, 2023.

[15] Y. Zhang, Z. Guo, J. Wu, Y. Tian, H. Tang, and X. Guo, "Real-Time Vehicle Detection Based on Improved YOLO v5," Sustainability, vol. 14, no. 19, article no. 12274, 2022.

[16] A. Ali, M. I. Sheikh, A. Ashraf, and A. A. Raja, "Development of Real Time AI-Based Vehicle Detection System for Traffic Surveillance," International Conference on Communication, Computing and Digital Systems, pp. 1-6, 2023.

[17] J. Lee and K. I. Hwang, "YOLO with Adaptive Frame Control for Real-Time Object Detection Applications," Multimedia Tools and Applications, vol. 81, no. 25, pp. 36375-36396, 2022.