

---

# Journal of Informatics and Web Engineering

Vol. 2 No. 2 (September 2023)

eISSN: 2821-370X

---

## Traffic Impact Assessment System using YOLOv5 and ByteTrack

**Jin Jie Ng<sup>1</sup>, Kah Ong Michael Goh<sup>2\*</sup>, Connie Tee<sup>3</sup>**

<sup>1,2,3</sup>Faculty of Information Science & Technology, Multimedia University, Jalan Ayer Keroh Lama, 75450 Bukit Beruang, Melaka, Malaysia.

*\*corresponding author: (michael.goh@mmu.edu.my; ORCID:0000-0002-9217-6390)*

*Abstract* - Monitoring software for traffic is not too much in this era of digital. Even cheaper is decent traffic monitoring software. You can gauge the quality of the software. It should be possible to assess the code's performance outside of a test environment. The most useful metrics are frequently those that support the program's ability to fulfil business requirements. Therefore, this project is planning to develop a traffic assessment system. The main purpose of development is to improve heavy traffic in this country – Malaysia. This system includes function vehicle detection using YOLOv5, vehicle counting with a different type (such as bus, car, truck), vehicle classification, vehicle idling time by each region, and vehicle counting for each junction. Users can draw regions and lines for each camera/video to count and record vehicles. After the traffic analysis, intelligent signal light systems that respond to loads and timing can be helpful in easing traffic congestion. Smart traffic lights may adapt to the patterns of bustle at junctions and other important road traffic places based on the number of cars, data from queue detectors, and images from cameras. Also, this report includes comparisons with StrongSORT, OC-SORT and ByteTrack and accuracy test for vehicle counting.

*Keywords*—YOLO, Vehicle Detection, Computer Vision, Vehicle Counting, Vehicle Tracking

Received: 13 June 2023; Accepted: 20 August 2023; Published: 16 September 2023

### I. INTRODUCTION

Due to incomplete public transport, vehicle ownership is gradually more and more. One person having one vehicle is very common in our lover country Malaysia [1]. When the quantity of vehicles becomes many, we cannot avoid heavy traffic problem. Therefore, traffic impact analysis (TIA) is very important for every country.

The technical investigation of traffic challenges and safety concerns associated with a given development is called a "traffic impact analytic" (TIA) [2]. Finding out if a certain development project would influence the security and effectiveness of nearby roadways is the main goal of TIA reports. The paper also sets the path for further research and the creation of traffic management strategies that reduce traffic congestion. The TIA studies the consequences of traffic on all other road users as well, including how effective and safe the road network is, such as passengers of



Journal of Informatics and Web Engineering

<https://doi.org/10.33093/jiwe.2023.2.2.13>

© Universiti Telekom Sdn Bhd. This work is licensed under the Creative Commons BY-NC-ND 4.0 International License.

Published by MMU Press. URL: <https://journals.mmupress.com/jiwe>

public transportation, walkers, and cyclists. The TIA considers the effects on both transportation infrastructure physically and operationally.

Firstly, this project will let the user insert a traffic video or direct connection to the camera. After, the user needs to drop the line for each direction and the box for car idling based on the image from the video or camera. Then, the system automatically detects the vehicle using YOLOv5. The system can be classification types of vehicles such as motorcycles, cars, trucks and so on. At the same time, the system must calculate vehicle idling time for each road. On top of that, the system counts the number of vehicles on the road when vehicles through the line which is dropped by users.

Furthermore, this system will show all data on the table on the website. One of the tables shows the number of vehicles passing through for each road. It includes the type of vehicle for every road and direction, vehicle direction and so on.

## II. RELATED RESEARCH

Vehicle recognition and statistics in video sequences of traffic monitoring are crucial for intelligent traffic management and control of the traffic [3]. The widespread installation of traffic surveillance cameras has resulted in the collection of a vast archive of video material for analysis. In general, a higher viewing angle can be used to consider a more distant road surface. It is challenging to see a little thing far from the road from this vantage point since the vehicle's object sizes vary greatly. It is essential to swiftly fix the problems before applying them to trickier camera settings [4].

### A. Vehicle Detection

A study on traffic video monitoring was conducted in 2015 by two researchers [5]. They found that employing artificial neural networks would be more effective than using other models. Their suggested approach calls for the use of Gaussians for background removal and Artificial Neural Network (ANN) re-modelling for vehicle detection. Re-modelling in this context refers to improving outputs by adjusting the standard ANN characteristics for vehicle detection.

According to their research, a standard Histogram Oriented Gradient (HOG) with a K-Nearest Neighbors (k-NN) model would provide an accuracy of 75.1%, however a modified ANN model with a HOG has produced a classification accuracy of 82.5%. So, the improved ANN model, which was recorded at 0.015 with 65 frames/second post-training, offers greater classification accuracy in a shorter amount of computing time. While the SVM model's computational time for 42 frames per second, equals 0.023 and the k-NN model's computational time has a frame rate of 28 and is 0.03. This demonstrates unequivocally that their study was successful, resulting in the ANN model winning the race with an 82.5% classification accuracy. However, without comparing many classifiers and models that are highly competitive with one another, what use is a machine learning algorithm? In short, ANN model and HOG feature extraction is a very good combination to detect vehicle.

M. Fachrie (2020) [6] YOLOv3 for counting vehicles for car, motor, bus, and truck. Detections were used separately for 1080p resolution video at 30 FPS and 1080p resolution at 15 FPS video. They provide a straightforward method for counting the cars without needing to monitor their progress from frame to frame. As shown in Figure 1, our approach simply measures the distance between the centroid of the vehicle and the boundary line. If the distance is less than or equal to the previously determined threshold value, one vehicle is counted. After several observations, the threshold value in this experiment is selected to be 1.5% of the video resolution. Based on the vehicle's centroid distance from the border, it is tallied. The problem of this straightforward approach is that if a single vehicle occupies many near positions to the border line over the course of several frames, the algorithm will count that car as two or even three vehicles.

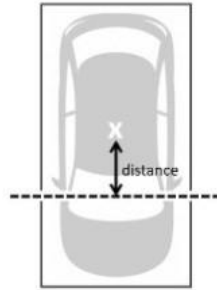


Figure 1. Distance Counted Based In Its Centroid Distance [6]

As a result, they used an extra technique, as seen in Figure 2, in which they examined three successive frames and assessed the positions of the identical cars in relation to the boundary. By measuring their centroid in relation to the previous frame, identical cars may be recognized. The same vehicles in different frames must have the closest distance among the other vehicles. This approach can reduce counting mistakes. Additionally, they noted that the identical cars' minimum separation from the video resolution is 4%.

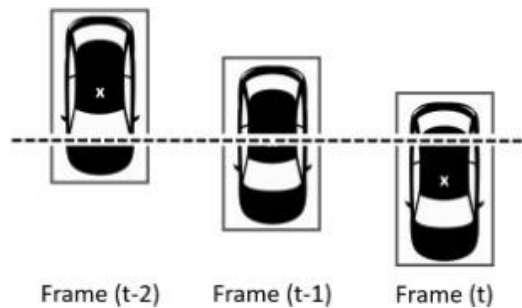


Figure 2. Non-tracking Vehicle Counting [6]

Without monitoring the movements of the vehicles, a vehicle counting system has been constructed using YOLOv3. By measuring the distance from the centroid of the vehicle to the boundary line, the counting is easily carried out. When using frontside-1x zoom footage, it was able to reach the greatest accuracy of 97.72%. Since counting is restricted to objects that are identified, YOLOv3 plays a key role in identifying cars. The object "car" has the best counting precision, followed by "motor" and "bus," while "truck" has the poorest. The frame rate of the video influences how well the system performs since it symbolizes the accuracy of the information it processes. Overall, this research was effectively finished and had positive results.

### B. Tracking Methods

SORT (Simple Online and Realtime Tracking) is a popular algorithmic framework for multiple objects tracking in videos or real-time applications [7]. It provides a simple yet effective solution for tracking objects across consecutive frames. The basic idea behind SORT is to use a combination of detection and tracking to estimate the state of each object in the video. The algorithm operates in an online and real-time manner, meaning it processes the frames as they arrive and updates the object tracks accordingly.

DeepSORT is a computer vision tracking technique that assigns a distinct ID to each tracked object proposed by N. Wojke et al. [7]. It enhances the SORT algorithm by incorporating deep learning, which helps reduce identity shifts and improves tracking effectiveness. SORT performs exceptionally well in terms of tracking precision and accuracy. However, it struggles when dealing with occlusions and often produces a significant number of ID changes due to the limitations of the association matrix used. In contrast, DeepSORT employs a more efficient association metric by combining motion and appearance descriptors [8]. This allows it to maintain object tracks not only based on their motion and velocity but also on their visual appearance [9]. Figure 3 shows the DeepSORT architecture.

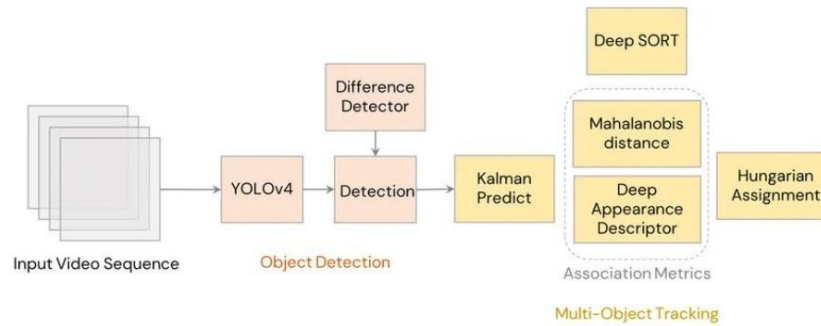


Figure 3. The DeepSORT Architecture [7]

Du et al. [10] developed the StrongSORT method, which, like DeepSORT, utilizes Multi-Object Tracking (MOT) techniques based on tracking-by-detection and joint-detection-association paradigms. They assert that, despite the latter approach gaining more attention and performing comparably to the former, the tracking-by-detection paradigm remains the most advantageous option for tracking accuracy [11]. The researchers reviewed the well-known DeepSORT tracker and made several improvements to its association, embedding, and detection methods, resulting in the creation of the StrongSORT tracker. Figure 4 shows the comparison between DeepSORT and StrongSORT.

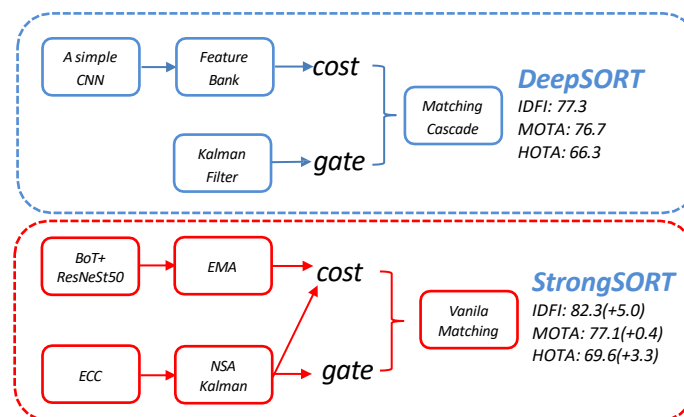


Figure 4. Comparison Between DeepSORT And StrongSORT [10]

ByteTrack is a real-time object tracking algorithm introduced by Zhang et al. [12]. It is designed to efficiently track objects in video sequences. They suggest ByteTrack, a straightforward, efficient, and general data association technique. ByteTrack does not preserve all the detection boxes; instead, it keeps virtually all of them and separates them into boxes for high- and low-score detection [13]. It initially links the tracklets to the high score detecting boxes. Because the appropriate high score detection box does not match them, certain tracklets become mismatched. This frequently happens whenever there is motion blur, occlusion, or size change. To get the items out of the low-scoring detection boxes and get rid of the background noise, it then connects these mismatched tracklets with the low score detection boxes. ByteTrack adopts a one-shot detection-based approach, where object tracking, and detection are combined in a single model. It achieves high tracking speed by sharing the computation between detection and tracking.

The suggested technique by Cao [14] is called Observation-Centric SORT, or OC-SORT for short, and it greatly outperforms occlusion and non-linear motion in terms of robustness while being straightforward, online, and real-time. They suggest two key advances in this study to mitigate the detrimental effects of these constraints. To minimise the cumulative error during the track's loss in a backcheck method, they first create a module that utilises object state observations. To be exact, they add a step of re-update to the customary processes of forecast and update to rectify the accumulated inaccuracy. When a track is reactivated by attaching to an observation after going untracked for a while,

the re-update is triggered. To avoid mistake accumulating, the re-update makes advantage of virtual observations on the prior time steps. The virtual observations originate from a trajectory that was created utilising the most recent observation that reactivated this track as well as the last observation observed before it was untracked as anchors. Observation-centric Re-Update (ORU) [15] is the term given to it.

In addition to ORU, the assumption of linear motion offers uniformity in the direction of object motion. However, due of the significant uncertainty in direction estimation, it is challenging to employ this cue in SORT's association. However, they suggest an observation-centric approach to include the direction consistency of tracks in the association's cost matrix. They refer to it as OCM, or observation-centric momentum. They examine the well-known motion-based tracker SORT and identify its inherent Kalman filter shortcomings. These restrictions have a major negative impact on tracking accuracy when the tracker is unable to get observations for supervision, which is usually due to defective detectors, occlusion, or quickly and non-linearly moving targets. They suggest OC-SORT as a solution to these problems.

While being straightforward, online, and real-time, OC-SORT is more resistant to occlusion and non-linear object motion. Trials on several datasets show that OC-SORT performs considerably better than the state-of-the-art. The gain is particularly important for multi-object tracking with non-linear object motion and occlusion.

### III. PROPOSED SOLUTION

First, users are provided with the option to connect a camera or upload a video. Once the camera or video is connected, the system generates video data including parameters such as video FPS and length. Next, the user is required to draw lines and regions, if necessary, for vehicle counting. Additionally, users can configure the detection settings such as the model, inference size, tracking method, and other thresholds. If no specific setup is performed, the system will utilize default settings. When the user clicks the start button, the detection process begins. Initially, YOLO detection is applied. Moreover, the system displays a line and region on the screen, which are then utilized for vehicle counting. Once the counting process is complete, the system updates the results in the database and presents them on the webpage. If the video ends or the user manually stops the detection, the process is terminated; otherwise, it returns to the YOLO detection step. After detection ends or is stopped, the user can insert additional videos or cameras for further detection. If no further inputs are provided, the program concludes. The flowchart of the proposed solution is illustrated in Figure 5.

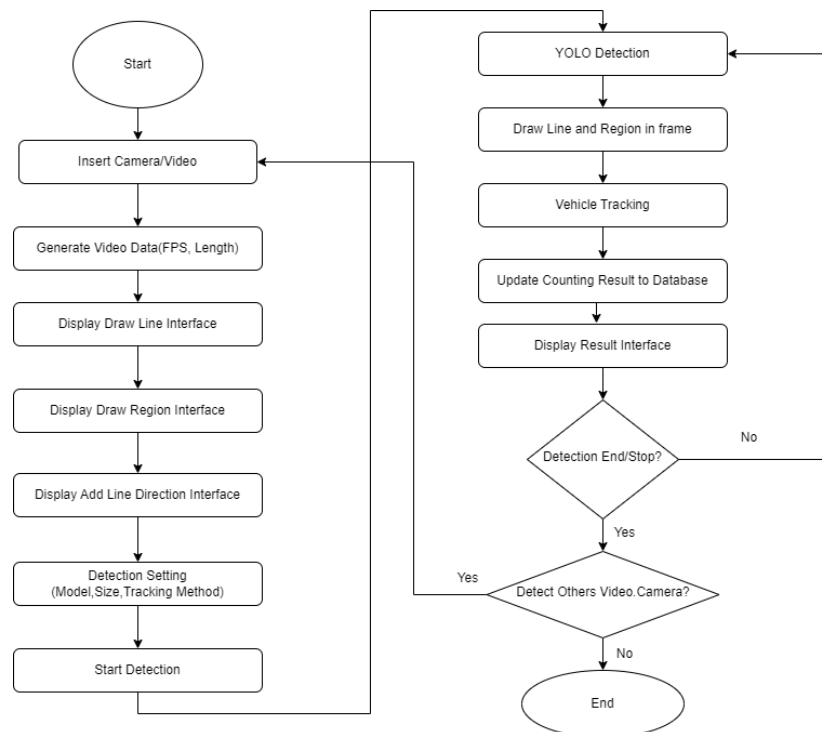


Figure 5. Flowchart Of The Proposed Solution

### A. Line and Region Drawing

The proposed system provides a user-friendly interface for users to draw lines. Users simply need to click and drag the line across the road or lane in the image to create a line for vehicle counting. Figure 6 illustrates the interface of adding a new line or region for a video or camera.

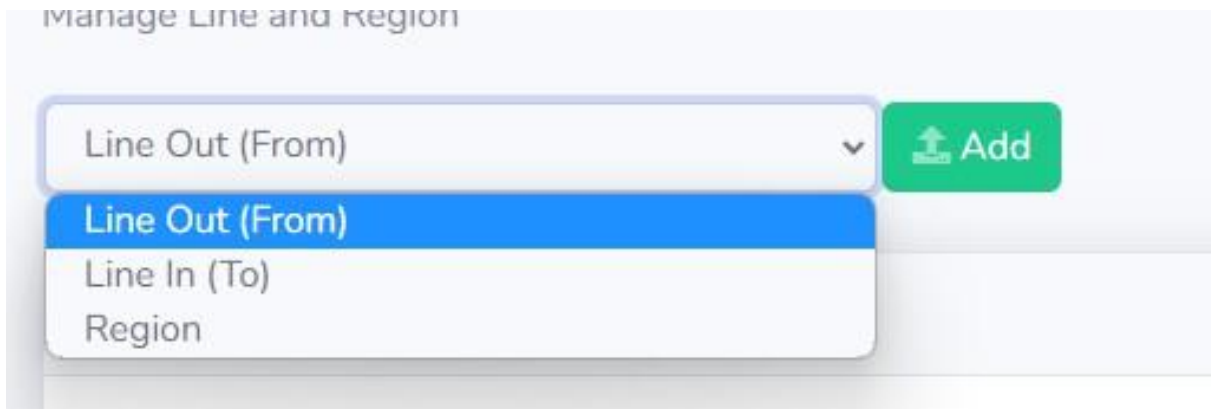


Figure 6 . Screen Shot For Add Line And Region

There are three options available: "Line Out" indicates the direction of vehicles leaving the road or junction, "Line In" signifies the direction of vehicles entering the road or junction, and "Region" allows the user to define a specific area. Upon selecting the desired option, the user needs to click the "Add" button, which will navigate them to a new page for drawing a new region or line. For "Line In" and "Line Out," the user can specify the direction using the buttons located on the right side. Figure 7 denotes the interface of users drawn a line in the frame.



Figure 7. Screen Shot For After Draw Line

For the draw region, the user requires placing at least three to a maximum six points on the image, to create a polygon region, Figure 8(a) and Figure 8(b) the sample of images with three points and six points regions.

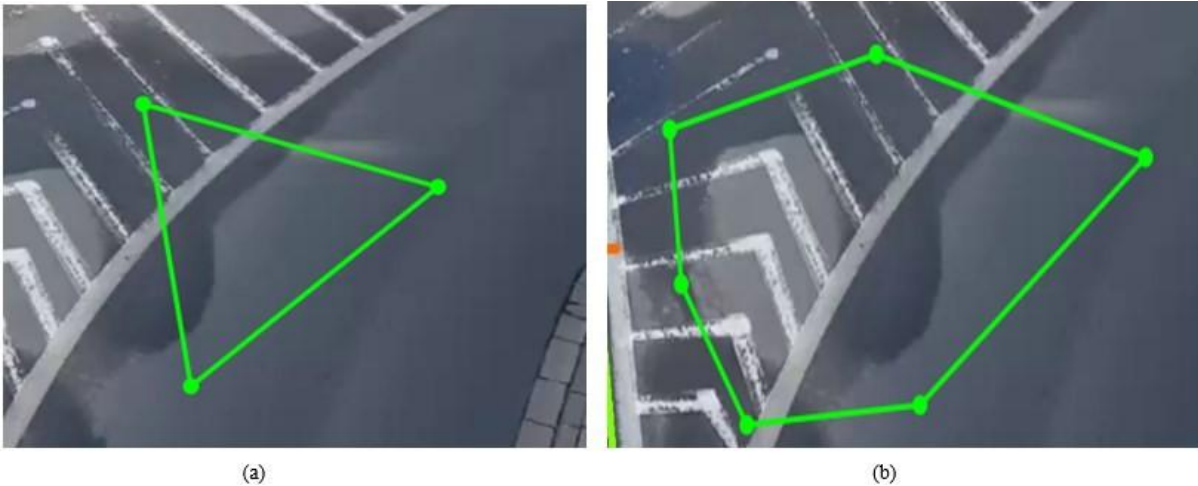


Figure 8. Creating Polygon Regions On Image

Figure 9 shown the interface of the add new line direction table. In the direction section, the user has four options: straight, turn left, turn right, and U-turn. For instance, if a vehicle exits from line L4 to enter line L1 with a left turn, the user needs to locate the "From (Line Out)" column in L4, select "To (Line In)" as L1, and choose "Turn Left" as the direction.

Add New Line Direction			
From (Line Out)	To (Line In)	Direction	Save
L1	L2	Straight	Save

Figure 9. Screen Shot For Add New Line Direction Table

### B. Setting

In the Figure 10, users can configure the detection settings for each individual video or camera. This means that users have the flexibility to set specific settings for each video or camera. For instance, video A can utilize YOLOv5 model A, while video B can use YOLOv5 model C. Additionally, the detection process can run simultaneously with different models for different videos or cameras, allowing for simultaneous detection using different models across multiple sources.

Figure 10. Screen Shot For Setting Page

The model options available are based on the file names in the YOLOv5 folder. Developers can add additional models to this folder, and any model with a file extension of ".pt" within the folder will be displayed on the settings page shown in Figure 11.

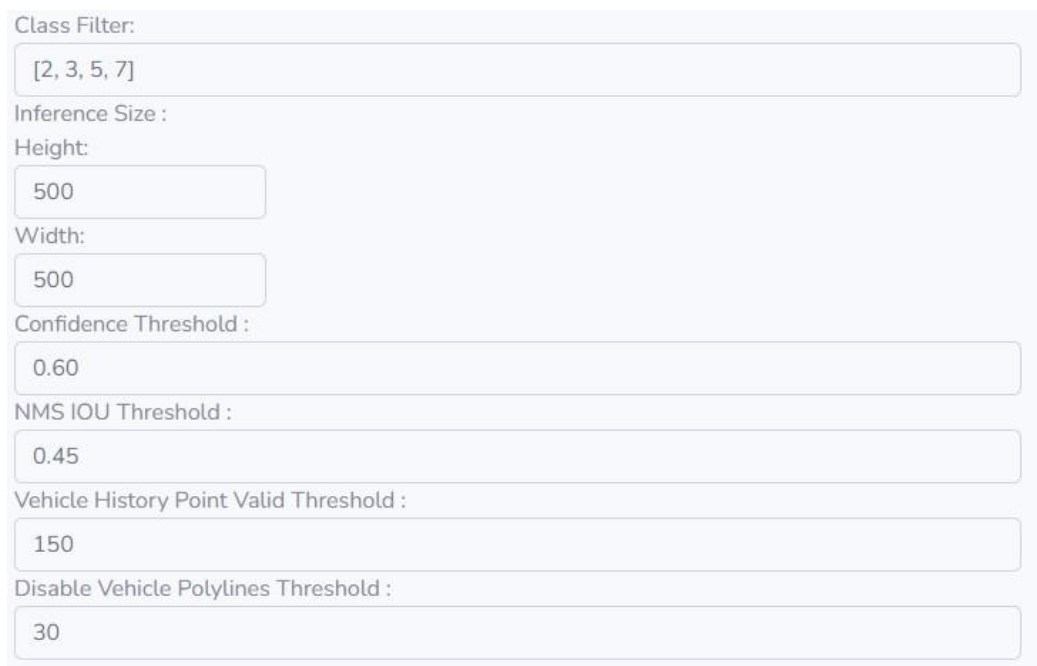
Figure 11. Screen Shot For Model Selection

Figure 12 presents three options for the tracking method: ByteTrack, OC-SORT, and StrongSORT. ByteTrack and OC-SORT are faster and more suitable for camera-based tracking. On the other hand, StrongSORT offers higher accuracy but may operate at a slower speed.

Figure 12. Screen Shot For Tracking Method Selection



In Figure 13, the class filter is responsible for filtering the detection classes. For example, when using the YOLOv5 Ultralytics [16] model, classes 2, 3, 5, and 7 might represent on-road vehicles. By setting the class filter, the detection system will only consider these four classes for detection. The inference size refers to the size at which the detection is performed. Smaller sizes result in faster detection speeds but may compromise accuracy. The confidence threshold determines the minimum required accuracy for a detection result. It ranges from 0.01 to 1. For instance, if the confidence threshold is set to 0.7, only detections with an accuracy higher than 0.7 will be considered. Non-Max Suppression (NMS) Intersection over Union (IoU) threshold is a technique employed by YOLO to filter out redundant bounding boxes. In the code provided, the NMS threshold is set to 0.45. Bounding boxes with detection probabilities below this threshold will be discarded. The vehicle history point valid threshold is utilized for tracking the vehicle's location. If the distance between consecutive points exceeds this threshold, the point will not be considered valid for tracking purposes. Higher thresholds are more suitable for faster vehicles or when the detection speed of the camera is slow. The disabled vehicle polylines threshold controls the display of vehicle location history lines. If the time gap between consecutive detections exceeds this threshold, the corresponding vehicle's location history line will not be drawn. Higher thresholds may result in a cluttered display frame.



The screenshot displays a configuration interface for object detection settings. It includes the following fields and values:

- Class Filter:** [2, 3, 5, 7]
- Inference Size :** (Label only)
- Height:** 500
- Width:** 500
- Confidence Threshold :** 0.60
- NMS IOU Threshold :** 0.45
- Vehicle History Point Valid Threshold :** 150
- Disable Vehicle Polylines Threshold :** 30

Figure 13. Screen Shot For Others Detection Setting

### C. Object Detection

For object detection, the system utilizes YOLOv5 with tracking by [17]. Users have the flexibility to modify various settings, such as changing the YOLOv5 mode, adjusting the image processing size, configuring the confidence threshold, adjusting Non-maximum Suppression (NMS) Intersection over Union (IOU) threshold, and selecting the device for detection, including options like Central Processing Unit (CPU) or Compute Unified Device Architecture (CUDA) devices like Graphic Processing Units (GPU)s. Utilizing GPUs for detection can significantly improve speed and accuracy, especially for real-time detection. Once the device is set, the YOLOv5 model is initialized, and the image size is checked.

Additionally, the system includes a model warm-up process. Delaying certain initializations until the model receives its first few inference requests is dependent on the model runtime or execution back end. Without proper warm-up, setup and optimizations may be postponed, resulting in significantly slower response times, potentially even hundreds

of times longer for the first few requests. Therefore, ensuring an adequate warm-up period is essential to avoid production traffic delays.

To enhance detection accuracy, the system incorporates a vehicle history point mechanism for vehicle counting. Several filters are implemented to exclude incorrect vehicle points. Firstly, if the new point of a vehicle is too far from the last point, it is ignored. Additionally, the system combines multiple points to calculate an average point, ensuring accuracy. Moreover, the line angle is calculated using the new point and the last two points, and if the angle does not fall within the range of 135 to 225 degrees, the point is disregarded. These filters significantly improve the accuracy of the points, consequently enhancing the subsequent tracking process. Figure 14 shows four vehicle classes detected by the system.

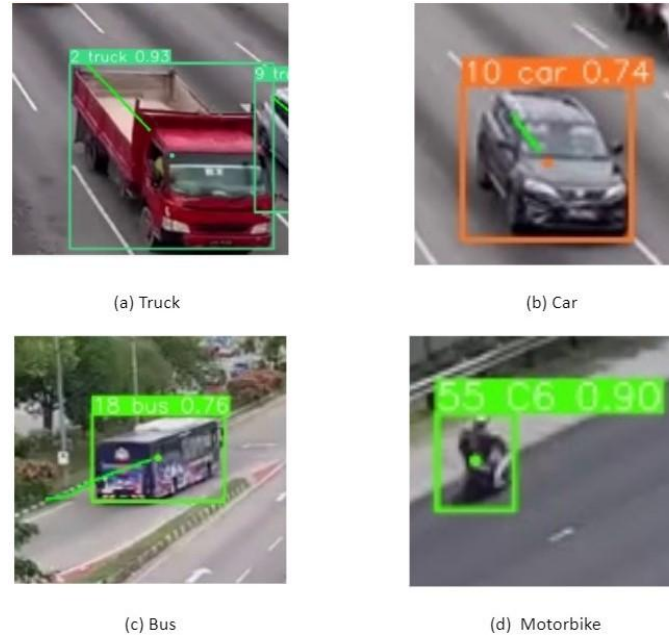


Figure 14. Sample Vehicle Detection

#### D. Object Counting

In the counting function, the system will be looping all regions that have been drawn. In the looping, the system will get region coordinates from the database for counting vehicles in each region. For counting, the system uses a point polygon test function in OpenCV to count the vehicle in the polygon. Let us consider a polygon with vertices  $V_1, V_2, \dots, V_n$ , where  $n$  is the number of vertices. The point in question is denoted as  $P(x, y)$ . If  $d$  larger than 0, the point  $P$  is outside the polygon. Formula to calculate the position of the point  $P$  with respect to the polygon is shown in Equation (1).

$$d = (V_2 - V_1) \cdot (P - V_1) \quad (1)$$

If the return value bigger or equal to 0, the vehicle is entering the region. To avoid counting multiple times for each vehicle, the system uses a list to store the vehicle that has been counted for each region. For idle time counting, video counting using the frame. For instance, if there are 100 frames for the vehicle in the region, the system will be using 100 frames to divide video FPS like  $100/30\text{FPS}$  equal to 3.33 seconds. For the camera, it will be counting using real time such as using current time to minus vehicle last seen time in this region. For vehicle counting for line, intersection convex is used to check whether the vehicle history point has across the line or not as shown in Figure 15. For example,

let  $A$  and  $B$  be two convex polygons. To calculate the intersection between  $A$  and  $B$ , resulting in a new polygon  $C$  as shown in Equation (2).

$$C = A \cap B \quad (2)$$

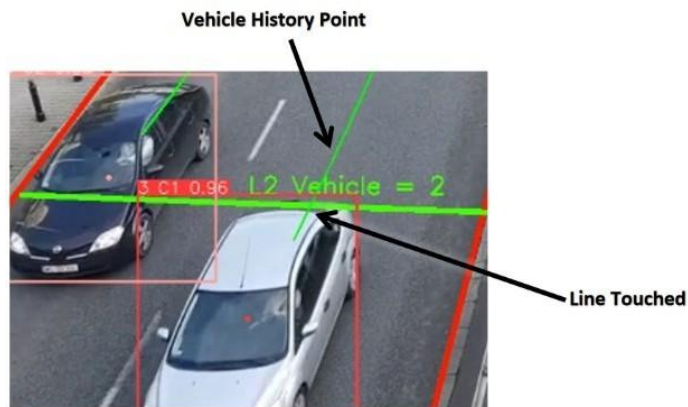


Figure 15. Screen Shots For Example Of Vehicle History Point Touch The Line

#### E. Displaying Object Information

After the detection, the system will generate object coordinates, accuracy, object ID, and object type. At the same time, the system will calculate the center point of coordinates for the object. It will make it system easier to count the number of vehicles when the history point touches the line or in the region. The object detected result is displayed in the interface in Figure 16.

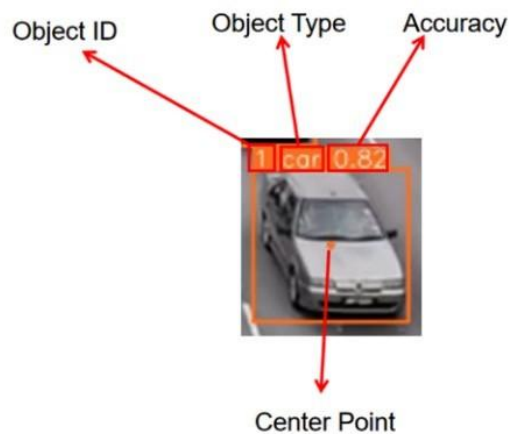


Figure 16. Display Object Result

Besides, the system will display the line, line number, and vehicle counting number for the line as Figure 17. The vehicle counting number here is real-time. Region has region number, region and vehicle counting number as Figure 18. This vehicle counting number for region is real-time.

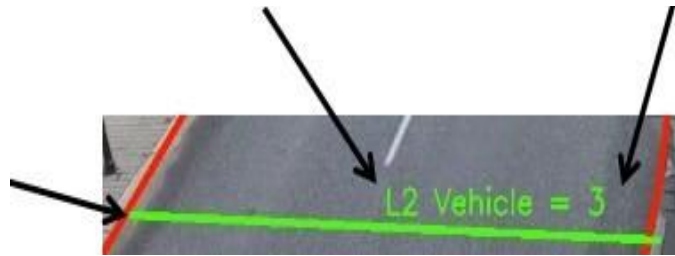


Figure 17. Display A Line

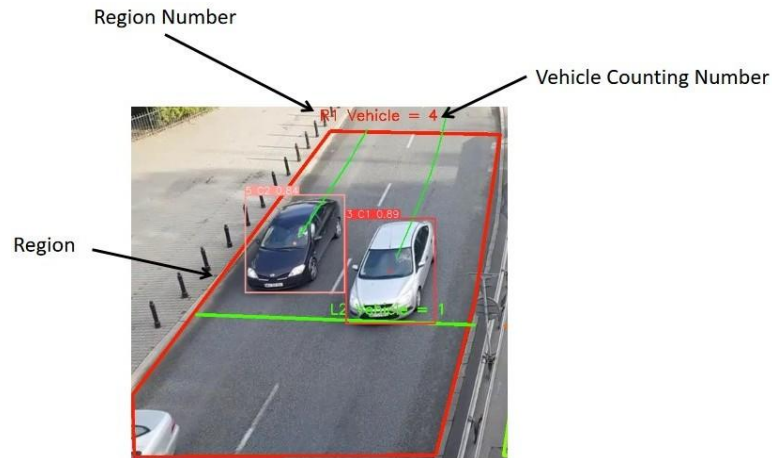


Figure 18. Display A Region

Figure 19 shows the FPS (frames per second) information in the top-left corner. The count number for L1 is 8 and L2 is 2, with an FPS of 21. Moreover, in regions R1 and R2, the number of counted vehicles is 10 and 5, respectively. The system distinguishes each detected vehicle by drawing a box in a different color. The class name and color assigned to each vehicle are determined by the YOLOv5 model.

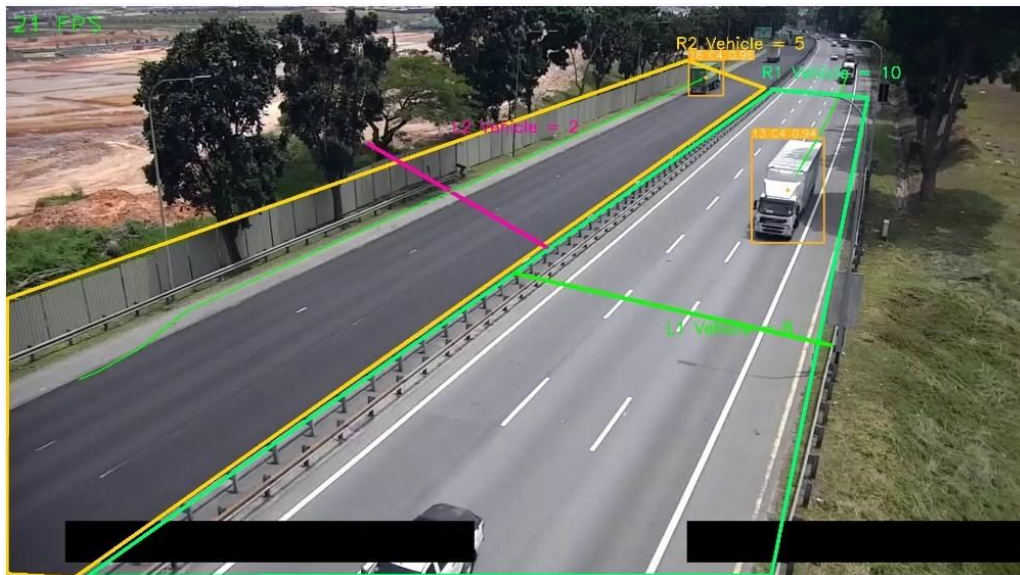


Figure 19. Lines, Regions, And Object Information

### F. Displaying Tracking Result

After the detection starts, the table data will be updated every 10 seconds. In the line table, the line number represents the line number in the frame. Line colours indicate the colours of the lines shown in the detection frame, allowing users to easily distinguish each line using the line number and line colour. Additionally, the line type refers to the selected type of line used for drawing, such as "LineIn" or "LineOut". The vehicle number indicates the total number of vehicles that have passed through the line since the detection started.

In the in-vehicle data, there is a small table for each line. This table displays the vehicle ID and vehicle type for each vehicle that has passed through the line. The length of this table is determined by the number of vehicles.

Figure 20 illustrates the structure of this table.

Data update every 10 seconds

Line

Line Number	Line Color	Line Type	Vehicle Number	Vehicle Data												
L1		LineOut	2	<table><tr><th>Vehicle ID</th><th>Type</th></tr><tr><td>25</td><td>truck</td></tr><tr><td>46</td><td>car</td></tr></table>	Vehicle ID	Type	25	truck	46	car						
Vehicle ID	Type															
25	truck															
46	car															
L2		LineOut	5	<table><tr><th>Vehicle ID</th><th>Type</th></tr><tr><td>18</td><td>car</td></tr><tr><td>21</td><td>car</td></tr><tr><td>31</td><td>car</td></tr><tr><td>32</td><td>car</td></tr><tr><td>34</td><td>car</td></tr></table>	Vehicle ID	Type	18	car	21	car	31	car	32	car	34	car
Vehicle ID	Type															
18	car															
21	car															
31	car															
32	car															
34	car															

Figure 20. Screen Shot For Line Result Table

In Figure 21, the region table includes the following information: number, colour, vehicle number, vehicle data, and idle time. The vehicle data in the region table contains an additional item called "Idle Time," which indicates how long a vehicle has stayed in the region. The idle time is measured in seconds.

Region

Region Number	Region Color	Vehicle Number	Vehicle Data																								
R1		7	<table><tr><th>Vehicle ID</th><th>Type</th><th>IdleTime(s)</th></tr><tr><td>3</td><td>car</td><td>18.67</td></tr><tr><td>4</td><td>car</td><td>18.67</td></tr><tr><td>5</td><td>car</td><td>18.07</td></tr><tr><td>6</td><td>car</td><td>18.27</td></tr><tr><td>7</td><td>car</td><td>17.90</td></tr><tr><td>8</td><td>car</td><td>18.07</td></tr><tr><td>9</td><td>car</td><td>16.40</td></tr></table>	Vehicle ID	Type	IdleTime(s)	3	car	18.67	4	car	18.67	5	car	18.07	6	car	18.27	7	car	17.90	8	car	18.07	9	car	16.40
Vehicle ID	Type	IdleTime(s)																									
3	car	18.67																									
4	car	18.67																									
5	car	18.07																									
6	car	18.27																									
7	car	17.90																									
8	car	18.07																									
9	car	16.40																									
R2		1	<table><tr><th>Vehicle ID</th><th>Type</th><th>IdleTime(s)</th></tr><tr><td>25</td><td>truck</td><td>6.77</td></tr></table>	Vehicle ID	Type	IdleTime(s)	25	truck	6.77																		
Vehicle ID	Type	IdleTime(s)																									
25	truck	6.77																									

Figure 21. Screen Shot For Region Result Table

The vehicle table shown in Figure 22, the following information is provided: vehicle ID, vehicle type, idle region, idle time, line out, line in, direction, and last seen time. The "idle region" indicates the region through which the vehicle has passed. The "idle time" represents the amount of time the vehicle has spent in that region, measured in seconds. The "line out" field denotes whether the vehicle has exited the line. Conversely, the "line in" field indicates whether the vehicle has entered the line. The "direction" of the vehicle is determined by analyzing the "line in" and "line out" data. If the vehicle has passed through both lines, the direction is determined based on these inputs. The "last seen time" indicates the time at which the vehicle was last detected in the frame. In the case of video, the "last seen time" is determined using the video's timestamp. For camera-based systems, the time is calculated in real-time and includes the date and time information.

Vehicle							
Vehicle ID	Vehicle Type	Idle Region	Idle Time	Out	In	Direction	Last Seen Time
1	car	R1	2.70		L2		00:00:02
2	car						00:00:00
3	car	R1	1.70		L2		00:00:02
4	car	R2	2.83	L1	L3	Turn Left	00:00:05
5	car	R2	1.53		L3		00:00:01
6	car	R2	2.23	L1	L3	Turn Left	00:00:06
7	car	R2	3.00		L3		00:00:03
8	car	R1	1.57	L1	L2	Straight	00:00:03
9	car	R1	1.47		L2		00:00:01

Figure 22. Screen Shot For Vehicle Result Table



### G. Experimental and Results Discussion

In the experimental results, we employed two videos, namely Video 1 and Video 2, to evaluate the counting accuracy and tracking accuracy. Both videos had a duration of two minutes, a frame rate of 24 FPS, and a resolution of  $1280 \times 720$ . The inference size was set to  $720 \times 480$ , and the detection process was performed using a GPU, specifically the NVIDIA RTX 2060 Super model, with the YOLOv5 model. Preview frame for Video 1 and Video 2 shown as Figure 23 and Figure 24.

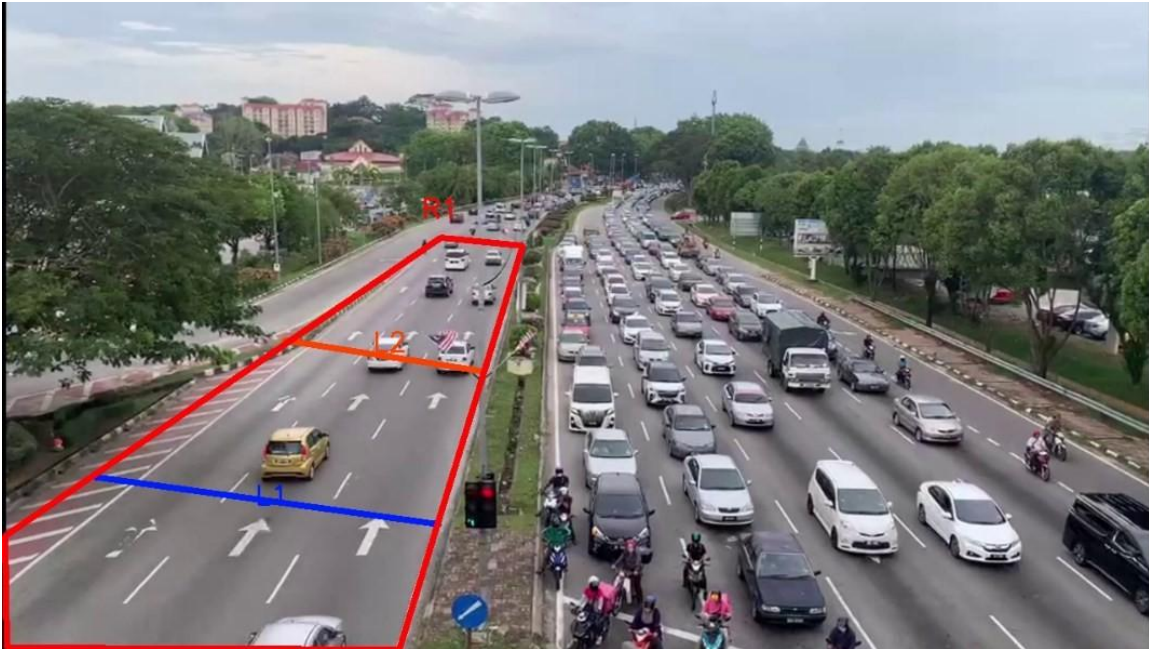


Figure 23. Preview Frame for Video 1

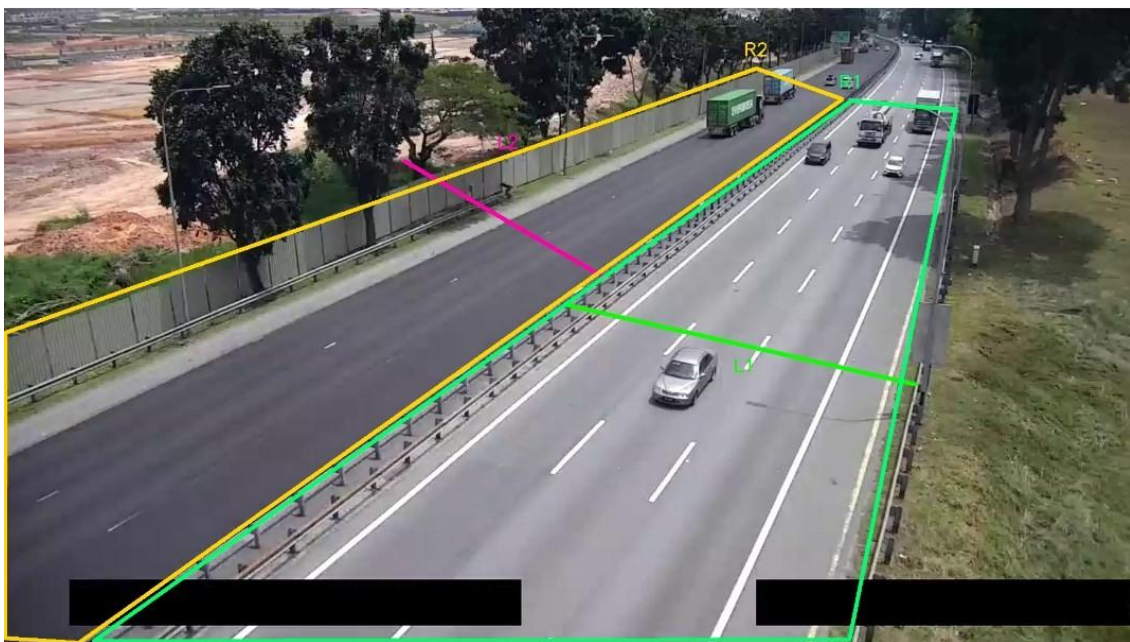


Figure 24. Preview Frame for Video 2

#### IV. EXPERIMENTAL AND RESULTS DISCUSSION

In the experimental results, we employed two videos, namely Video 1 and Video 2, to evaluate the counting accuracy and tracking accuracy. Both videos had a duration of two minutes, a frame rate of 24 FPS, and a resolution of  $1280 \times 720$ . The inference size was set to  $720 \times 480$ , and the detection process was performed using a GPU, specifically the NVIDIA RTX 2060 Super model, with the YOLOv5 model [18]. Preview frame for Video 1 and Video 2 shown as Figure 23 and Figure 24.

##### A. Tracking Test

For the tracking time test, we will compare the performance of StrongSORT, OC-SORT, and ByteTrack. DeepSORT is not included in the comparison because StrongSORT incorporates improvements from DeepSORT. We utilized two videos for each tracking method and measured the time taken for the processing to complete. All testing was conducted under identical conditions, with different tracking methods employed, the extra time required in comparison to the duration of the original video was measured and recorded in seconds for each method. We have tracking accuracy Test 1 and Test 2 with different vehicle in Video 1. The tracking accuracy for three tracking methods are shown in Figure 25 and Figure 26.

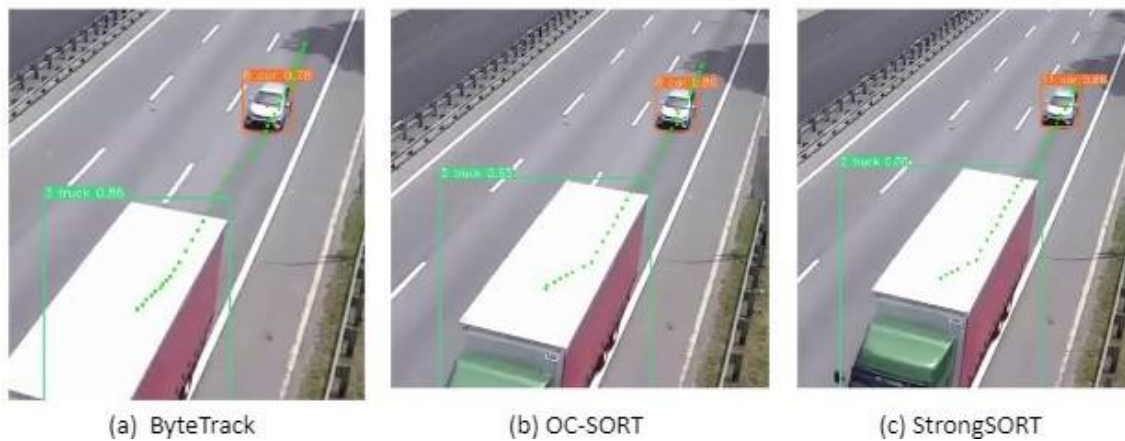


Figure 25. Tracking Accuracy Test 1

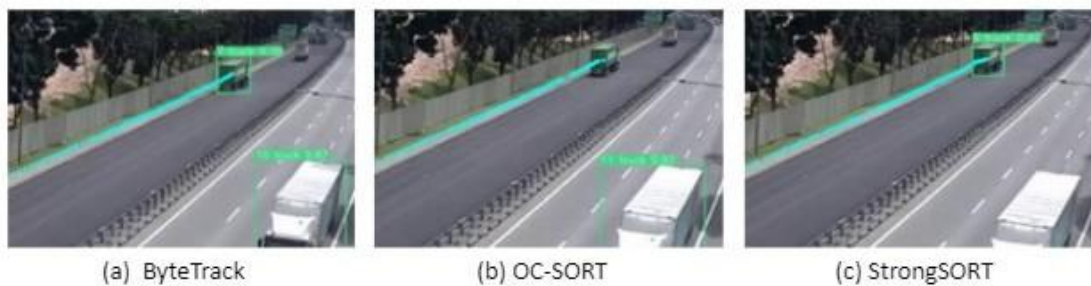


Figure 26. Tracking Accuracy Test 2

In the tracking accuracy test, the results of the three tracking methods are almost the same. In Test 1, ByteTrack was the smoothest, while OC-SORT and StrongSORT had breakpoints in Test 2. Therefore, users can choose their preferred tracking method from these three options in the settings. An analysis was conducted to calculate and compare



the tracking time needed for each method to process the two Test 1 and 2 videos compare with the two minutes video length. The additional tracking time needed to process the 2 minutes video length in percentage is denoted by  $P_n$  as shown in Equation (3), where  $n$  is the video number,  $T_{vn}$  is the video length time and  $T_{pn}$  is the tracking time in minutes.

$$P_n = (T_{vn} - T_{pn}) / T_{vn} \quad (3)$$

The tracking times for each method are shown in Table 1. It can be observed that the results indicated that the processing times for OC-SORT and ByteTrack were nearly identical. In Video 1, the processing times were 3.38 minutes for OC-SORT and 3.37 minutes for ByteTrack. In Video 2, the processing times were 2.27 minutes for OC-SORT and 2.30 minutes for ByteTrack. On the other hand, StrongSORT exhibited slower processing times, with 12.47 minutes in Video 1 and 10.55 minutes in Video 2 and the tracking process takes an additional time of 523% of the original video time. It is evident that slower tracking methods are not suitable for real-time tracking applications.

Table 1. Tracking Time With Different Method For Video 1 And Video 2

Track Methods	Video 1		Video 2	
	$T_{p1}$	$P_1$	$T_{p2}$	$P_2$
<b>StrongSORT</b>	12.47	523%	10.55	428%
<b>OC-SORT</b>	3.38	69%	2.27	14%
<b>ByteTrack</b>	3.37	68%	2.30	15%

An analysis was conducted to check on the tracking velocity on four vehicles as shown in Figure 28, the results were based on four vehicles tracked using three tracking methods. Furthermore, we also tested the velocity of each tracking method. The tracking velocity,  $V$  is denoted in Equation (4), where  $D$  is the distance between two pixels of two frames and  $t$  is the time in milliseconds using the formula:

$$V = D / t \quad (4)$$

Distance is the measurement of the distance between the last vehicle's center point and the new vehicle's center point. The tracking velocity is using four vehicles in Video 1. Vehicle 1 and Vehicle 2 are trucks, Vehicle 3 is a sedan car, and Vehicle 4 is a pickup truck shown in Figure 27. The tracking velocity for each vehicle is represented in graphs shown in Figure 28.



Vehicle 1



Vehicle 2



Vehicle 3



Vehicle 4

Figure 27. Vehicle Types Selected For Tracking Velocity Analysis

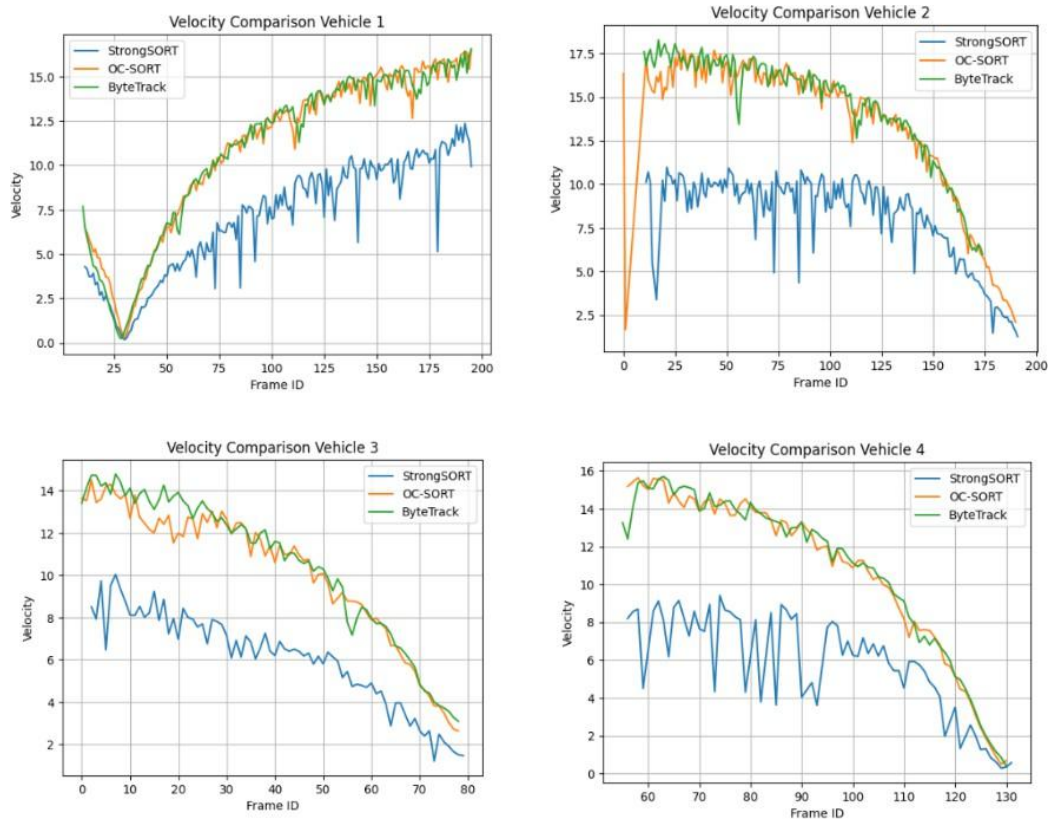


Figure 28. Tracking Velocity Comparison For StrongSORT, OC-SORT And ByteTrack For Each Vehicle Type

In comparison velocity test for OC-SORT, StrongSORT and ByteTrack, there are getting the same result with tracking time. ByteTrack and OC-SORT get similar results and StrongSORT are slower. It can be observed that the velocity is not affected by the vehicle size, whether it is a truck or a small car.

### B. Counting Accuracy

The counting accuracy test involves the usage of ByteTrack to count vehicles. The system's actual vehicle counting results are compared with the expected result achieved through manual counting by humans. By obtaining both sets of counting results, the accuracy score can be calculated. For Video 1, lines and regions are drawn as shown in Figure 23. The accuracy result for Video 1 is shown in Table 2.

Table 2. Accuracy Result for Video 1

Item	Actual Result	Expected Result	Accuracy
Line 1	71	77	0.92
Line 2	68	77	0.83
Region 1	95	84	0.88

In test 1 of the counting test, line 1 achieved an accuracy score of 0.92, while line 2 scored 0.83. The counting accuracy of the lines in this test is unsatisfactory, as there are several instances of missed counting. Line 2 performs worse than line 1 due to the blurrier vehicle sizes observed in line 2, which adversely affects the counting results. Additionally, the counting of vehicles passing through the historical line is affected if the detected historical points are inaccurate, as only the historical points are considered for counting.

As for the regions, they achieved an accuracy score of 0.88. Unlike the lines, the regions tend to result in extra counting, meaning there are instances where vehicles are counted multiple times. For Video 2, lines and regions are drawn as shown in Figure 24. The accuracy result for Video 2 is shown in Table 3.

Table 3. Accuracy Result for Video 2

Item	Actual Result	Expected Result	Accuracy
Line 1	83	83	1
Line 2	49	49	1
Region 1	102	86	0.84
Region 2	63	52	0.82

In Test 2, both lines achieved a perfect score for line counting. However, the two regions produced the same result as in Test 1, counting non-existing vehicles and resulting in extra counting. Several factors can influence the counting results, such as the performance of the YOLO model and the quality of the analyzed video, among others. Based on the conducted tests, it is evident that line counting is highly accurate, achieving nearly perfect scores in both Test 1 and Test 2. However, region counting requires improvement as it tends to include extra vehicles in the count.

## V. CONCLUSION

In conclusion, this is a system that allows users to count vehicles automatically and analyze the time and number of vehicles. Users can upload a video or connect to a camera. Based on the picture from the video or camera, the user must remove the line for each direction and the box for the automobile to idle. Then, the system uses YOLOv5 to automatically recognize vehicles. The system may classify many types of vehicles, such as trucks, cars, and motorcycles. The system must simultaneously determine how long each car will idle on each road. Additionally, the system counts the number of cars that pass through a road as they pass across a line that has been dropped by users. Additionally, this system will display all data on a real-time website table. The number of vehicles passing via each road is displayed in one of the tables. The type of vehicle for each route and direction, the vehicle's direction, and other information are included.

As for the future work, there are many possibilities for improvement. The system can calculate the vehicle speed based on maps. Additionally, users can be allowed to add their own YOLO model, and the system can also support other versions of YOLO. Furthermore, the system can detect vehicles that commit traffic violations, such as illegal U-turns, overspeeding, and running red lights, among others. We could enhance our software design process by integrating the semi-automated approach proposed in [19]. Additionally, we can optimize our development methodology by adopting the most effective agile practices outlined in [20]. This combined strategy will enable us to create the system with greater efficiency and precision. Finally, we believe that the knowledge gained from working on this system will be useful in future endeavors.

## ACKNOWLEDGEMENT

This work was supported by the TM R&D Fund (Project no. RDTTC/221054 and SAP ID: MMUE/220023) and MMU IR Fund (Project ID MMUI/220041).

A version of this paper was presented at the third International Conference on Computer, Information Technology and Intelligent Computing, CITIC 2023, held in Malaysia on 26th-28th July 2023

## REFERENCES

- [1] E. Teh, "Traffic Congestions in Malaysia and the Lessons We Must Learn | Heinrich Böll Foundation | Southeast Asia Regional Office", Heinrich-Böll-Stiftung, 2022. <https://th.boell.org/en/2022/07/22/traffic-malaysia>.
- [2] M. Armit, "Traffic Impact Assessment Report: All you Need to Know", Medium, 2016. <https://medium.com/@michaelarmit3/traffic-impact-assessment-report-all-you-need-to-know-7613c81b8587>.
- [3] G. Wiecek, S. B. Ud Din Tahir, I. Akhter, J. Kurek, "Vehicle Detection and Recognition Approach in Multi-Scale Traffic Monitoring System via Graph-Based Data Optimization", *Sensors*, vol. 23, no. 3, pp. 1731, 2023. <https://doi.org/10.3390/s23031731>.
- [4] H. Song, H. Liang, H. Li, Z. Dai, X. Yun, "Vision-based vehicle detection and counting system using deep learning in highway scenes", *European Transport Research Review*, vol. 11, no. 1, 2019. <https://doi.org/10.1186/s12544-019-0390-4>.
- [5] S. K. B and S. G., "Traffic video surveillance: Vehicle detection and classification", 2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum, India, 2015, pp. 516-521, doi: 10.1109/ICCC.2015.7432948.
- [6] M. Fachrie, "A Simple Vehicle Counting System Using Deep Learning with YOLOv3 Model", *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 4, no. 3, pp. 462 - 468, 2020.
- [7] N. Wojke, A. Bewley, D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric", *arXiv (Cornell University)*, 2017, doi: 10.48550/arxiv.1703.07402.
- [8] "Understanding Multiple Object Tracking using DeepSORT", 2022. <https://learnopencv.com/understanding-multiple-object-tracking-using-deepsort/>.
- [9] M. Gaikwad, "Deep Sort: Simple Online and Real-time Tracking with Deep Associative Metric", Medium, 2023. [https://medium.com/@mohit\\_gaikwad/deep-sort-simple-online-and-real-time-tracking-with-deep-associative-metric-94138d528ff1](https://medium.com/@mohit_gaikwad/deep-sort-simple-online-and-real-time-tracking-with-deep-associative-metric-94138d528ff1) (accessed Aug. 18, 2023).
- [10] Y. Du, Z. Zhao, Y. Song, Y. Zhao, F. Su, T. Gong, H. Meng, "StrongSORT: Make DeepSORT Great Again", *IEEE Transactions on Multimedia*, vol. 25, pp. 1–14, 2023. <https://doi.org/10.1109/tmm.2023.3240881>.
- [11] S. Sah, "Real time Object tracking and Segmentation using YoloV8 with Strongsort, Ocsort and Bytetrack", Medium, 2023. <https://siddharthsah.medium.com/real-time-object-tracking-and-segmentation-using-yolov8-with-strongsort-ocsort-and-bytetrack-180eef43354a> (accessed Aug. 18, 2023).
- [12] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, X. Wang, "ByteTrack: Multi-Object Tracking by Associating Every Detection Box", *arXiv (Cornell University)*, Oct. 2021, <https://doi.org/10.48550/arxiv.2110.06864>.
- [13] B. Le, "An Introduction to BYTETrack: Multi-Object Tracking by Associating Every Detection Box", *DataTure*. <https://www.datature.io/blog/introduction-to-bytetrack-multi-object-tracking-by-associating-every-detection-box> (accessed Aug. 18, 2023).
- [14] J. Cao, X. Weng, Rawal Khirodkar, J. Pang, K. M. Kitani, "Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking", *arXiv [cs.CV]*, 2022, <https://doi.org/10.48550/arxiv.2203.14360>.
- [15] I. Berrios, "Introduction to OC-SORT", Medium, 2023. <https://medium.com/@itberrios6/introduction-to-ocsort-c1ea1c6adfa2#25ad> (accessed Aug. 18, 2023).

- [16] G. Jocher et al., “ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations”, Semantic Scholar, 2021. <https://www.semanticscholar.org/paper/ultralytics%2Fyolov5%3A-v5.0-YOLOv5-P6-1280-models%2C-and-Jocher-Stoken/fd550b29c0efee17be5eb1447fddc3c8ce66e838>.
- [17] M. Broström, “Real-time multi-object, segmentation and pose tracking using YOLOv8 with DeepOCSORT and LightMBN”, GitHub, 2023. [https://github.com/mikel-brostrom/yolo\\_tracking](https://github.com/mikel-brostrom/yolo_tracking).
- [18] G. Jocher, “ultralytics/yolov5”, GitHub, 2020. <https://github.com/ultralytics/yolov5>.
- [19] F. F. Chua, T. Y. Lim, B. Tajuddin, A. P. Yanuarifiani, “Incorporating Semi-Automated Approach for Effective Software Requirements Prioritization: A Framework Design,” *Journal of Informatics and Web Engineering (JIWE)*, vol. 1, no. 1, pp. 1-15, 2022, doi: 10.33093/jiwe.2022.1.1.1.
- [20] S. L. M. Keong and Z. C. Embi, “A Systematic Review on Non-Functional Requirements Documentation in Agile,” *Journal of Informatics and Web Engineering (JIWE)*, vol. 1, no. 2, pp. 19-29, 2022.