Vehicle Speed Estimation Using YOLO, Kalman Filter, and Frame Sampling

Asif Hummam Rais
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
hashshura@gmail.com

Rinaldi Munir
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rinaldi@informatika.org

***Abstract*—Vehicle speed estimation based on video feed can be used to enforce road rules and give traffic insights without the need of physical interference. Common methods are background subtraction, motion detection, and/or convolutional neural network (CNN). The first two methods suffer from inability to differentiate classes and occlusion, whereas CNN suffer from computational complexity. A system based on You Only Look Once (YOLO) as detector and Kalman filter as tracker is proposed. TensorRT and frame sampling are used to further optimize the inference time. From experiment using BrnoCompSpeed dataset on i5 9600k and RTX 2060 Super environment, proposed system runs the entire process at 118 FPS with mean average error (MAE) of 0.96 km/h and [-3, 2] error interval at 93.81%. Frame sampling can be used to further improve the FPS, with 1/5 sampling improves the speed by 50% to 177 FPS with only 0.11 km/h MAE tradeoff to 1.07 km/h.**

***Keywords*—vehicle speed estimation; yolo; tensorrt; kalman filter; frame sampling**

I. Introduction

As the cost to purchase and operate vehicles gets cheaper year by year, the number of road users will also increase. This can potentially cause traffic jams and higher accident rate. Almost 1.2 million people die yearly, with millions more suffer from physical injury or disability due to traffic incidents. This figure is mostly affected by low- and mid-income countries and is caused by excessive speed through reaction factors [1]. Vehicle speed itself can be defined in the context of both microscopic and macroscopic traffic characteristics [2]. Speed as microscopic characteristic weighs the speed of individual vehicles, in which case it is used to tighten speed limit regulations. Speed as macroscopic characteristic weighs the flow of vehicles; hence it can be utilized to serve as additional data to give further insights whom the government can make decisions with.

Vehicle speed can be monitored from the vehicle's internal (speedometer) or through an external detector that estimates vehicle speed through vehicle movement. The external detector itself is divided into intrusive and non-intrusive systems [3]. In intrusive systems, which generally use inductive loop detectors, complex installation of the road foundation is required, having to deal with physical interference and degradation to the road components. On the other hand, non-intrusive systems, generally using laser meter sensors and Doppler radar, overcome the problems of intrusive systems at a higher cost and require regular maintenance. Computer vision within its sub-field object tracking aims to estimate the state of movement of a target object in image sequences [4]. This state of motion can be the velocity of the object considering its trajectory. Image sequences can be obtained by taking frame by frame on a video file obtained from a video camera. As the cost of video cameras is getting smaller with improved video quality, video-based vehicle speed detectors can be a solution in the context of non-intrusive speed detectors.

Vehicle speed detector systems from video have become a topic of considerable research attention. One of the previously proposed systems uses motion detection model and object tracking by vehicle license plate similarity [3]. This limits the possibility of using low-res camera or specific camera placements, as vehicle license plate features always need to be visible. Also, as the motion detection still requires license plate detection, it adds up as an overhead to lower the inference speed. Another proposed system uses Mask-RCNN object detection and Deep-SORT (Kalman filter with deep association metric) object tracker [5]. This system overcomes the previous limitations due to the similarity evaluated from all parts of the vehicle object, therefore allowing the camera parameters to be unbounded. However, Mask-RCNN is known for having slow inference speed. Furthermore, as vehicles have predictive movements on the road, deep association metric can be substituted to faster Hungarian algorithm association method with only intersect-over-union as the cost function [6].

As an effort towards detecting fast vehicle speeds obtained in real time, a new design is proposed which is based on the object detection and object tracking methods. Rectification to world coordinates is done via homography matrix whose values are calculated based on four reference points in the image. As a baseline, Mask-RCNN + Deep-SORT will be tested. The object detector will be substituted with YOLOv4. In the case of speed estimation, segmentation features from Mask-RCNN are not needed. To speed up the inference time even further, the trained YOLO model will be converted into TensorRT framework-based and video frame sampling will be used to speed up the entire pipeline process. CPU-based YOLO using Darknet and several methods of selecting reference points in the bounding box will also be explored.

II. Related Research

A. Video-Based System for Vehicle Speed Measurement in Urban Roadways

A non-intrusive, video-based system for measuring vehicle speed proposed by Luvizon et al. [3] used a motion detector model and a novel system to detect similarities in license plates in the image region containing motion. The different features are then selected based on the license plate area, traced on multiple frames, and the perspective distortion is corrected. Vehicle speed is measured by comparing the trajectory of the tracked feature with measures available in the real world. This system resulted in an average error of -0.5 km/h in the experiments and 96% of the measured speeds still fall within the acceptable speed estimation error interval by US standards ([-3, 2]). The license plate detector outperforms two state-of-the-art text detector and license plate detector with a precision of 0.93 and recall of 0.87.

B. A Semi-Automatic 2D Solution for Vehicle Speed Estimation from Monocular Videos

Kumar et al. [5] designed an architecture to estimate vehicle speed from monocular videos. The pipeline consists of three main modules: (1) multi-object detection using Mask-RCNN, (2) tracking using SORT/Deep-SORT, and (3) speed estimating using a homography matrix. To detect vehicles, Mask-RCNN based on FPN and ResNet-101 as the backbone implemented in Detectron is trained on the MS-COCO dataset. To learn the visually distinguishing representations between vehicles, a simple network using DCNN was trained through the CompCars dataset consisting of 136,726 images with 163 car brands and 1,716 different car models. The experiments resulted in a detection ratio of 1.00 using the SORT and Deep-SORT methods, and respectively had RMSE 9.54 mph and 10.10 mph in the NVIDIA AI City Challenge 2018 dataset. Identity switches are found to be more frequent in SORT-based tracker.

III. Design and Implementation

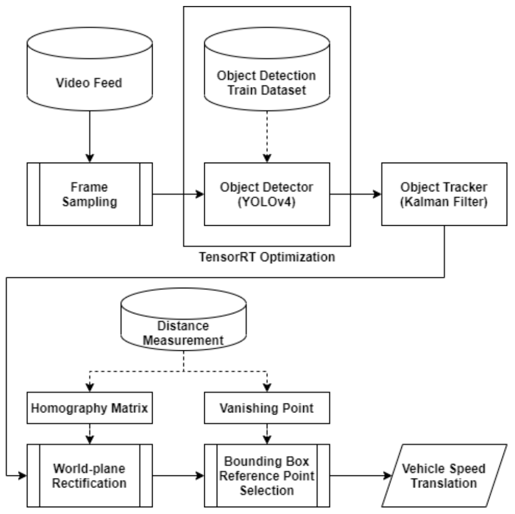
Video Feed
Object Detection Train Dataset
Frame Sampling
Object Detector (YOLOv4)
Object Tracker (Kalman Filter)
TensorRT Optimization
Distance Measurement
Homography Matrix
Vanishing Point
World-plane Rectification
Bounding Box Reference Point Selection
Vehicle Speed Translation

Fig. 1. Proposed architectural pipeline scheme

The architecture of the vehicle speed estimation system is designed following the architectural pipeline scheme in Fig. 1. In summary, an input in the form of a video file will be processed in the pipeline with the output being the bounding box of each vehicle detected in the video frame with its identity obtained from the tracking process and vehicle speed estimation in the real world plane resulting from the image projection to the real dimension.

A. Frame Sampling

Frame sampling module asks input in the form of video and returns the image sequence representation. Processing is done using OpenCV VideoCapture which will return a matrix list of size h × w × 3 where h, w, and 3 each represent the image height in pixels, the image width in pixels, and the three color intensity values of red, green, and blue (red, green, blue or RGB) with values ranging from 0 to 255 for each frame of the video. After one image matrix in the video frame is processed into the object detector model, several video frames with a ratio of *N_JUMP* - 1 will be skipped so as not to pass through the detection process. In the normal case (*N_JUMP* = 1), no video frame is skipped. In the optimized case (*N_JUMP* > 1), several video frames are skipped after sending one frame, hence the total load for processing one video is 1/*N* of the original load.

B. Object Detector

You Only Look Once version 4 (YOLOv4), the state-of-the-art object detector based on Darknet framework [7] is trained using filtered MS-COCO [8] dataset of 18,199 images and 772 test images with 4 classes: car, motorbike, bus, and truck. Multiple configurations of YOLOv4 will be tested: YOLOv4 (default), YOLOv4-CSP, YOLOv4x-Mish, YOLOv4-Tiny, and YOLOv4-Tiny-3L. These models are then optimized using TensorRT by converting them into ONNX models and then TensorRT models using steps described in the J. K. Jung's repository "tensorrt_demos" [9]. Experiments of this module will be presented in the next section, with evaluation metrics of mean average precision (mAP) and inference time in frames per second (FPS).

C. Object Tracker

In the context of simple, online, and real-time object tracking [6], there are two concepts: (1) estimation model; and (2) data association. Estimation model is used to estimate the next bounding box of a detected bounding box, whereas data association is used to associate predicted next bounding boxes with actual or detected bounding boxes to associate them into a unique identification number. Estimation model uses Kalman filter, with each state is described as

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T$$

where u and v are the horizontal and vertical pixel coordinate of the bounding box center points, s and r represent scale and aspect ratio of the bounding box, and $\dot{u}$, $\dot{v}$, $\dot{s}$ representing the velocity or change ratio for each parameter u, v, s. Data association, on the other hand, utilizes Hungarian algorithm where each pair of predicted and detected bounding boxes has a cost function of their intersect-over-union value. Experiments of this module use evaluation metrics of multi-object tracking accuracy (MOTA), multi-object tracking precision (MOTP), mostly tracked (MT), mostly loss (ML), identity switch (IDSw), and inference time in frames per second (FPS).

D. *World-plane Rectification*

Point projection from image to world-plane coordinate is done via homography matrix multiplication. Parameters from homography matrix is calculated by processing once in the beginning of the pipeline, using four reference points in image and their respective points in world-plane in meter unit. By stacking the sample points in 2×9 matrices into 8×9 matrix P with every parameters of homography matrix flattened into a 9×1 matrix H where $PH = 0$, the values of H can be calculated by finding the singular value decomposition of P. Rectification is then done by multiplying the 3×3 unflattened matrix from H with 3×1 image point matrix in (x, y, 1) form.

E. *Bounding Box Reference Point Selection*

In a bounding box of a vehicle, the reference point that will be used for displacement calculation must be selected. In the baseline solution, reference point is selected by using the bottommost centre point of the bounding box. In the proposed system, a line will be drawn from the vanishing point of the road parallel lines to the centre of the bounding box, until it reaches the edge of the box. This point in the edge of the box will be used as the selected reference point.

F. *Vehicle Speed Translation*

Vehicle speed is calculated using the simple formula, $v = | x_2 - x_1 | / t$, where x_1 is the reference point in frame $i - N\_BEFORE$ after rectification, x_2 is the reference point i in frame i after rectification, and t is the distance between frames. In case of no frame sampling, then the t will be the second per frame of the video. Experiments of this module, which means by using the entire pipeline, use evaluation metrics of mean absolute error (MAE), accepted error percentages (AEP) which lies between [-3, 2] km/h, maximum error, minimum error, and inference time in frames per second (FPS).

IV. EXPERIMENT AND ANALYSIS

Due to limitations in dataset i.e. there are too few motorcycles, trucks, and buses in tracking and speed estimation dataset and there are only small number of cars in the speed estimation dataset, experiments will be conducted on both module-level and architecture-level. Experiment environment uses a system with Intel® Core™ i5-9600K CPU, NVIDIA GeForce RTX 2060 Super, and 16 GB of RAM.

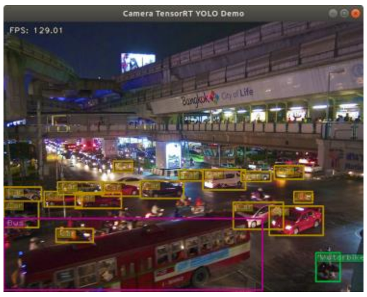

Fig. 2. Sample image and detection result of MS-COCO, YOLOv4-416-TRT

A. *Object Detector Experiment*

There are three models to be tested: (1) Pre-trained Mask-RCNN using MS-COCO [10], (2) YOLOv4 before TensorRT, and (3) YOLOv4 after TensorRT optimization. Those models are tested on MS-COCO filtered test dataset which consists of 772 images with 4 classes: 1,991 cars, 440 motorcycles, 290 buses, and 415 trucks. Sample image can be seen in Fig. 2.

TABLE I. EXPERIMENT RESULT OF DETECTOR MODEL

No	**Framework**	**Model**	**AP car**	**AP motor-cycle**	**AP bus**	**AP truck**	**mAP**	**FPS**
1	Tensor-flow	Mask-RCNN	0.684	0.715	0.819	0.526	0.686	3.59
2	YOLOv4-416 Darknet	v4	0.729	0.754	0.866	0.611	0.740	45.41
3		v4x-mish	0.684	0.681	0.800	0.508	0.668	36.76
4		v4-csp	0.686	0.711	0.827	0.527	0.689	59.38
5		v4-tiny	0.522	0.527	0.732	0.414	0.549	257.33
6		v4-tiny-3l	0.582	0.577	0.758	0.430	0.587	257.33
7	YOLOv4-416 TensorRT	**v4**	**0.711**	**0.764**	**0.852**	**0.552**	**0.720**	**196.16**
8		v4x-mish	0.673	0.685	0.785	0.433	0.644	146.84
9		v4-csp	0.682	0.715	0.805	0.453	0.664	209.51
10		v4-tiny	0.524	0.546	0.724	0.370	0.541	452.08
11		v4-tiny-3l	0.576	0.584	0.756	0.379	0.574	417.476

In Table I, it is shown that both the mAP and FPS of YOLOv4 models with default configuration outperform pretrained Mask-RCNN. YOLOv4 has 12 times FPS of Mask-RCNN with mAP of 0.054 (5.4%) higher, and TensorRT further improves the base model by 4.3 times FPS with only a minimal degradation of mAP by 0.02 (2%), mainly due to the high degradation of truck AP (0.06 or 6%). Due to the speed estimation dataset of BrnoCompSpeed later having 50 FPS, we can assume that the best configuration with real-time inference is YOLOv4-416 TensorRT with 0.720 mAP and 196.16 FPS. This configuration will be tested as the control for the speed estimation experiments.

B. *Object Tracker Experiment*

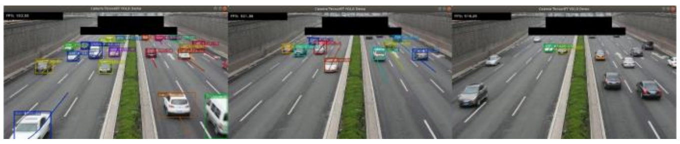

Fig. 3. Object tracker experiment using YOLO-Kalman with *N_JUMP* = 2, 5, and 10 respectively

There are two models to be tested: (1) Kalman filter with Hungarian algorithm; and (2) Deep-SORT using pre-trained models from N. Wojke's repository "deep_sort" [11]. In this experiment, UA-DETRAC [12] vehicle tracking dataset will be used as test dataset, mainly on videos: MVI_20011 (normal road, 53 vehicle ID, 664 frames) and MVI_40191 (toll road, 340 vehicle ID, 2495 frames) each having 25 FPS frame rate. Sample image from MVI_40191 video can be seen in Fig. 3, each having custom configuration in their *N_JUMP* parameter.

TABLE II. EXPERIMENT RESULT OF TRACKER CONFIGURATIONS

No	Detector	Tracker	MOTA	MOTP	MT	ML	IDSw	FPS
1	Mask-RCNN	Deep-SORT	74.546	78.434	328	8	**105**	3
2	YOLOv4-416 TRT	Deep-SORT	63.19	78.768	311	7	117	15
3	YOLOv4-416 TRT	Kalman	**77.675**	**80.108**	**334**	**5**	140	62
4	YOLOv4x-mish-416 TRT	Kalman	75.204	78.625	282	6	379	70
5	YOLOv4-csp-416 TRT	Kalman	72.219	78.424	227	14	285	78
6	YOLOv4-tiny-416 TRT	Kalman	72.282	77.078	264	9	246	83
7	YOLOv4-tiny-3l-416 TRT	Kalman	74.8	77.509	260	7	138	**90**

In Table II, it is shown that Deep-SORT, paired with either Mask-RCNN or YOLOv4-416-TRT, performs not in real-time. This shows that data association part from Deep-SORT is indeed taxing for large number of object tracking. In this result, it is found that Mask-RCNN has higher MOTA compared than YOLOv4-416-TRT if the tracker used is Deep-SORT. The best configuration is YOLOv4-416-TRT paired with Kalman-based tracker as proposed, where the MOTA (77.675), MOTP (80.108), MT (334), and ML (5) ranks the highest. On the other hand, Deep-SORT, when paired with either Mask-RCNN or YOLO, has fewer number of identity switches (105 and 117 respectively) as opposed to using Kalman filter (140 in the best configuration). This is due to Deep-SORT using deep association metric therefore some lost tracks will not be reinitialized instead reassigned to existing tracks.

TABLE III. TRACKER WITH VARIED N_JUMP EXPERIMENT

No	*N JUMP*	MOTA	MOTP	MT	ML	ID Sw	Data-set IDs	Video FPS	FPS (Imply)
1	1	77.675	80.108	334	5	140	393	25	62 (62)
2	2	76.502	78.709	300	6	127	393	12.5	61 (122)
3	5	62.71	78.285	56	11	141	393	5	54.2 (271)
4	10	21.926	79.816	4	187	31	393	2.5	47.2 (472)
5	20	-12.83	77.539	3	361	36	389	1.25	35.1 (702)

In Table III, tracker with varied *N_JUMP* will be tested. It is shown that on sample video of 25 FPS, using *N_JUMP* of 1 and 2 will still result a high MOTA and MOTP (above 70). This means that if we downsample a 25 FPS video to 12.5 FPS, tracking with YOLOv4-416-TRT and Kalman filter will still evaluate in good results with an implied increase of FPS which is twice as fast. On the other hand, using *N_JUMP* of 5 (means downsample to 5 FPS) will lower the MOTA to 62.71, with MT or mostly tracked IDs numbering 56 (a sixth of before).

C. *Vehicle Speed Estimation Experiment*

Integration of all modules ultimately results in the estimated speed of vehicles in a video. This result will be evaluated with BrnoCompSpeed dataset [13]. As the detected bounding box of a single unique vehicle spans for frames, hence it has more than one estimated speed, the median of all estimated speeds will be used as detected value which will be compared with the ground truth for each vehicle to remove outliers. First experiment of speed estimation, shown in Table IV, uses combinations of detector and tracker with *N_BEFORE* = 10 and *N_JUMP* = 1. Sample image on the dataset and how the system shows the result can be seen in Fig. 4.

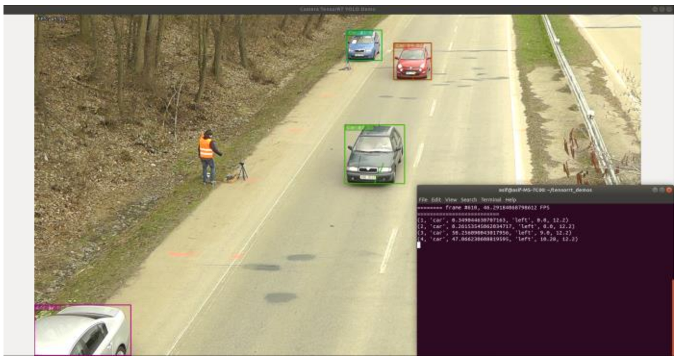
```
======== frame #810, 40.29184066798612 FPS
==========================
(1, 'car', 0.34984463078716З, 'left', 0.0, 12.2)
(2, 'car', 0.2615354506203471Ꞁ, 'left', 0.0, 12.2)
(3, 'car', 50.25609804381795б, 'left', 9.0, 12.2)
(4, 'car', 47.06623860881959б, 'left', 10.20, 12.2)
```
Fig. 4. Sample image and estimated result of BrnoCompSpeed dataset

TABLE IV. EXPERIMENT RESULT OF SPEED ESTIMATION

No	Detector	Tracker	MAE (km/h)	AEP (%)	*Max Error* (km/h)	*Min Error* (km/h)	FPS
1	Mask-RCNN	Deep-SORT	1.12	90.21	27.28	-7.11	3
2	YOLOv4-416 TRT	Deep-SORT	**0.86**	**93.81**	**3.28**	**-3.24**	49
3	YOLOv4-416 TRT	Kalman	1.08	88.14	6.01	-5.62	**116**

From Table IV, it is observed that YOLO & Deep-SORT combination has the best MAE, AEP, max error, and min error. On the other hand, from Table II it is known that that combination could not perform in real-time in extreme case where there are more vehicles on each frame (15 FPS), whereas YOLO & Kalman will still perform at 62 FPS with MAE, AEP, max error, and min error outperform Mask-RCNN and Deep-SORT combination.

TABLE V. SPEED ESTIMATION WITH VARIED N_BEFORE, YOLO+KALMAN

No	*N JUMP*	*N BEFORE*	MAE (km/h)	AEP (%)	*Max Error* (km/h)	*Min Error* (km/h)	FPS
1	1	1	1.24	90.21	4.28	**-4.72**	**121**
2	1	2	**0.96**	**93.81**	**3.73**	-5.14	118
3	1	5	1.04	91.75	**3.73**	-5.33	106
4	1	10	1.08	88.14	6.01	-5.62	116
5	1	20	1.25	85.57	5.12	-20.41	117

In Table V, it is found that *N_BEFORE* = 2 outperform *N_BEFORE* = 1, having 0.96 km/h MAE (0.28 km/h lower) and 93.81% detection within accepted error [-3, 2] with little to no impact in the FPS. *N_BEFORE* = 20 has the worst MAE at 1.25, whereas the FPS for all combination ranges not much different at 106–121 FPS. Still, this combination does not outperform the YOLO & Deep-SORT combination (0.1 km/h higher MAE).

TABLE VI. SPEED ESTIMATION WITH VARIED N_N_BEFORE AND N_JUMP, YOLO+KALMAN

No	*N JUMP*	*N_BEFORE*	MAE (km/h)	AEP (%)	*Max Error* (km/h)	*Min Error* (km/h)	Video FPS	Implied FPS
1	1	2	**0.96**	**93.81**	3.73	-5.14	50	118
2	2	1	1.09	90.72	11.4	-5.28	25	154
3	1	5	1.04	91.75	3.73	-5.33	50	106
4	5	1	1.07	86.08	**3.31**	-4.44	10	177
5	1	10	1.08	88.14	6.01	-5.62	50	116
6	2	5	1.08	89.18	4.57	-18.63	25	133
7	5	2	1.14	83.51	3.72	**-3.28**	10	190
8	10	1	1.56	72.16	4.94	-7.1	5	**206**

From Table VI, usage of *N_JUMP* with values under or equal to 5 will still retain an acceptable MAE at 0.96–1.14 km/h. This is in accordance to result in Table III, where tracking on video with 25 and 12.5 FPS (in this case, 10 FPS) perform on similar accuracy. Also, surprisingly, the change of *N_JUMP* from 1 to 5 does not straightforwardly improve the FPS by 5 times, instead it is only 50 to 70%. This is probably caused by a bottleneck in the video reading process, where using OpenCV in Python it can only run at 250 FPS even without any other process.

TABLE VII. SPEED ESTIMATION WITH VARIED BOUNDING BOX REFERENCE POINT SELECTION

No	Ref. Point Selection	*N BEFORE*	MAE (km/h)	AEP (%)	*Max Error* (km/h)	*Min Error* (km/h)	FPS
1	Baseline, bottommost centre	5	1.04	91.75	3.73	-5.33	106
2		10	1.08	88.14	6.01	-5.62	116
3		20	1.25	85.57	5.12	-20.41	117
4	Proposed, vanishing point line	5	1.06	89.18	4.4	-5.27	114
5		10	1.12	87.11	6.25	-5.53	121
6		20	1.29	85.57	5.42	-20.36	118

In Table VII, it is observed that for any *N_BEFORE* configuration, using the proposed reference point selection results in inferior values for all evaluation metrics other than FPS. This means that the initial hypothesis that using the edge point from vanishing point line has better information is denied. In other words, the baseline method which is choosing the bottommost center point of the bounding box will result in better MAE and AEP.

TABLE VIII. SPEED ESTIMATION WITH VARIED NETWORK SIZE

No	Detector Model	Network Size	MAE (km/h)	AEP (%)	*Max Error* (km/h)	*Min Error* (km/h)	Video FPS	FPS
1	YOLOv4 TRT	416	**0.96**	**93.81**	**3.73**	**-5.14**	50	**118**
2		640	1.21	88.66	7.81	-6.96	50	70

In Table VIII, the trained weights from YOLOv4 with network size of 416 are used to produce another model with network size of 640. Unfortunately, this results in worse MAE, AEP, max error, min error, and FPS. This is most probably caused by rounding error of straightforwardly use the 416 weights in 640 networks. As pixel coordinate is discrete, when an object with discrete size is detected in that model, it will be detected as a real number and instead must be rounded. This means that using weights from lower network size models into higher network size models is not guaranteed to perform better.

TABLE IX. SPEED ESTIMATION WITH CPU COMPUTATIONAL CAPACITY

No	Detector Model	*N JUMP*	*N_BEFORE*	MAE (km/h)	AEP (%)	*Max Error* (km/h)	*Min Error* (km/h)	Implied FPS
1	YOLOv4 416x416	1	2	2.09	83.51	4.62	-63.12	1.6
2		2	1	2.26	84.02	9.73	-65.71	3.2
3		1	5	2.17	83.51	9.18	-63.97	1.6
4		5	1	**1.39**	**85.57**	**3.06**	**-47.10**	3.3
5	YOLOv4-tiny 416x416	1	2	2.98	67.53	31.42	-74.25	8.4
6		2	1	2.22	75.26	17.63	-74.25	16
7		1	5	3.95	52.58	51.90	-74.42	8.4
8		5	1	2.40	70.62	6.13	-75.11	38
9		8	1	2.37	70.10	5.85	-71.16	**62**

In the last experiment, which is shown in Table IX, CPU-only computation results are shown. As TensorRT framework is only able to be run in Nvidia GPUs, this experiment runs the original Darknet framework YOLOv4 without TensorRT optimization. It is observed that the results from identical configurations of *N_JUMP* and *N_BEFORE* are different from that of using TensorRT framework. The best configuration is when YOLOv4, *N_JUMP* = 5, and *N_BEFORE* = 1 with 1.39 km/h MAE. However, this configuration could only run at 3.3 FPS on 50 FPS video, which is not real-time. For real-time configuration, it is found that using YOLOv4-tiny, *N_JUMP* = 8, and *N_BEFORE* = 1, we can achieve worse result at 2.37 km/h MAE but with real-time inference at 62 FPS, almost 8 times faster than its *N_JUMP* = 1 counterpart. Surprisingly, this configuration even performs better than YOLOv4-tiny with *N_JUMP* = 2 and *N_BEFORE* = 1 with its 2.98 km/h MAE.

V. CONCLUSION

Architectural pipeline that is designed and implemented using TensorRT-optimized YOLOv4, object tracking based on Kalman filter and Hungarian algorithm, and image rectification to world-plane coordinates can be used to estimate vehicle speed with arbitrary camera placement as long as there is a length and width information of the road segment. The best combination has a mean absolute error (MAE) of 0.96 km/h, with 93.81% of its detected speeds still fall within the acceptable speed estimation error interval by US standards ([-3, 2]). The test is done by comparing BrnoCompSpeed ground truth car speeds with median of detected speeds by the system for each car. Median is used to remove outliers from the detection results.

TensorRT can be used to optimize YOLOv4 by 3.7 times faster (45 to 196 FPS) with only 0.02 mAP disadvantage from the original model with Darknet framework. To further improve the FPS of the system, frame sampling can also be used to improve the FPS with only little MAE increase, where a 1/5 sampling can improve the FPS by 50% with only 0.11 km/h worse MAE tradeoff to 1.07 km/h on GPU (YOLOv4-TRT), or 700% FPS increase with 0.61 km/h better MAE on CPU (YOLOv4-tiny).

References

[1] World Health Organization. (2008). *Speed management: a road safety manual for decision-makers and practitioners*. Geneva: Global Road Safety Partnership. ISBN 978-2-940395-04-0.

[2] Maerivoet, Sven. Moor, Bart De. (2005). *Traffic Flow Theory*. arXiv:physics/0507126.

[3] Luvizon, D. C.. Nassu, B. T.. Minetto, R.. (2017) "*A Video-Based System for Vehicle Speed Measurement in Urban Roadways*," in IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 6, pp. 1393-1404, doi: 10.1109/TITS.2016.2606369.

[4] Sunkara, J. K.. Santhosh, M.. Cherukuri, S. B.. Krishna, L. G.. (2017). "*Object tracking techniques and performance measures — A conceptual survey*," IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Chennai, 2017, pp. 2297-2305, doi: 10.1109/ICPCSI.2017.8392127.

[5] Kumar, Amit. Khorramshahi, Pirazh. Dhar, Prithviraj. Chellappa, Rama. Lin, Wei-An. Chen, Jun-Cheng. (2018). *"A Semi-Automatic 2D Solution for Vehicle Speed Estimation from Monocular Videos,"* IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), doi: 10.1109/CVPRW.2018.00026.

[6] Bewley, Alex. Ge, Zongyuan. Ott, Lionel. Ramos, Fabio. Upcroft, Ben. (2016). *Simple Online and Realtime Tracking*. arXiv:1602.00763.

[7] Bochkovskiy, Alexey. Wang, Chien-Yao. Liao, Hong-Yuan Mark. (2016). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934.

[8] Lin, T.. Maire, M.. Belongie, S.. Bourdev, L.. Girshick, R.. Hays, J.. Perona, P.. Ramanan, D.. Zitnick, C. L.. Dollár, P.. (2014). *Microsoft COCO: Common Objects in Context*. arXiv:1405.0312.

[9] J. K. Jung. *TensorRT Demos.* GitHub repository: https://github.com/jkjung-avt/tensorrt_demos

[10] Abdulla, Waleed. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. GitHub repository: https://github.com/matterport/Mask_RCNN

[11] Wojke, Nicolai. Bewley, Alex. Paulus, Dietrich. (2017). *Simple Online and Realtime Tracking with A Deep Association Metric*. arXiv:1703.07402.

[12] Wen, Longyin. Du, Dawei. Cai, Zhaowei. Lei, Zhen. Chang, Ming-Ching. Qi, Honggang. Lim, Jongwoo. Yang, Ming-Hsuan. Lyu, Siwei. *UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking*. arXiv:1511.04136.