Robust image processing in MAVs: YOLO implementation in MATLAB

Alireza Mikaeili
Department of Electrical Engineering
Iran University of Science and Technology
Tehran, Iran
alireza.mikaeili1380@yahoo.com

Mobina Lashgari
Department of Electrical Engineering
Iran University of Science and Technology
Tehran, Iran
mobina.lashgary@yahoo.com

Hadi Firouzi
Department of Electrical Engineering
Iran University of Science and Technology
Tehran, Iran
hadi.frz.2001@gmail.com

Alireza Abbasi
Department of Electrical Engineering
Iran University of Science and Technology
Tehran, Iran
alireza.abbassi0@gmail.com

***Abstract*— Nowadays, image processing plays a significant role in the autonomous use of MAVs. Over time, various algorithms have been introduced to improve the speed and accuracy of this process, gradually expanding based on deep learning —now a fundamental element in many industries, such as automotive and healthcare. Currently, the YOLO algorithm is considered one of the best among them. On the other hand, MATLAB is recognized as one of the most widely used software due to its unique features in programming and simulation, as well as its ability to directly interface with industrial tools and devices used across a vast range of scientific fields. Some of the key advantages of this software including its high speed in data processing, a very user-friendly interface and the ability to simulate and analyze many systems before practical implementation under real-world conditions. Despite these benefits, in the field of deep learning there are still certain limitations. One of the most significant of these is the lack of support for the newer versions of YOLO (You Only Look Once). The aim of this research is to address MATLAB's limitations in utilizing all versions of YOLO. This study offers practical solutions that not only enable the use of the latest YOLO versions in MATLAB but also ensure they are implemented without delay, enhancing the efficiency of projects based on this software.**

Keywords*— *image processing, MAV, YOLO algorithm, MATLAB, object detection

I. Introduction

In MAVs, image processing plays a vital role for autonomous driving. Traditional image processing techniques have been replaced by modern neural networks due to their limitations, such as not being able to detect objects in various environmental lighting conditions, drop in frame rate and adding up noises which can make old-fashioned algorithms in computer vision, totally useless. So the modern algorithms emerged such as two-stage algorithms like RCNN and SPPNet, Fast RCNN and Faster RCNN, as well as single-stage algorithms such as YOLO, SSD and RetinaNet [1].

By comparing accuracy and inference time of these methods, we concluded that YOLO has the best overall performance and is being enhanced by each version.

In this paper, we will first review the principles and techniques of image processing and the implementation of recent YOLO versions in the Python environment. This section will also include performance charts for these versions, highlighting their strengths and weaknesses.

MATLAB offers a vast toolbox used in different applications including MAV simulations in realistic or experimental environments. In addition, MATLAB’s usage by a huge community of engineers has caused it to be implemented in many automation applications. Currently MATLAB only supports YOLO versions from YOLOv2 to YOLOv4 and YOLOX [2] and recently, YOLOv8 [3] and YOLOv9 [4]. However, they still do not perform as well in terms of accuracy and speed compared to the implementations of the algorithm in Python. These limitations have challenged researchers and engineers to find a way to utilize these newer versions within MATLAB.

So as the second step, we will focus on implementing and analyzing the YOLO versions supported in MATLAB by presenting their performance charts. For these implementations, MATLAB's available tools and functions will be utilized.

Subsequently, we will compare the performance of YOLO models in both Python and MATLAB environments in terms of accuracy, speed and efficiency to determine the necessity of employing the proposed methods for using these models in MATLAB. Finally, a web server-based method will be proposed to enable the use of newer YOLO versions in MATLAB.

This solution aims to provide advanced capabilities of the latest YOLO versions within the MATLAB environment and allow researchers to leverage the benefits of both platforms. The goal of this paper is to offer practical solutions for utilizing the latest YOLO versions in MATLAB and enhancing the efficiency of projects based on this software. It is hoped that this research will contribute to improving the performance and accuracy of image processing projects within the MATLAB environment.

II. Used MAV

For the implementation of this project, we used the DJI Tello drone and its educational version, the DJI Tello EDU. This drone is small, lightweight and relatively affordable given its capabilities and it is designed for educational purposes. As mentioned in [5], some of its features are a flight range of 100 meters, a flight time of 13 minutes, a 5-megapixel camera capable of recording 720p HD videos, programmability through an SDK and support for various programming languages. In this project, we chose to use MATLAB and the Ryze library. This library allows us to easily control the drone and perform the necessary processing quickly and efficiently without the drone overheating. Ultimately, this enabled us to apply modern object detection methods on aerial drones.

III. USING MATLAB INSTEAD OF PYTHON

Despite Python's wide access to more extensive and practical libraries, its support for DJI's official SDK, being free, not requiring powerful hardware and supporting various up-to-date machine learning algorithms, certain capabilities of MATLAB address many of the shortcomings of the Tello drone. These advantages have led us to replace Python with MATLAB. Fortunately, MATLAB offers a dedicated support package for Ryze Tello drones, enabling the use of this drone with enhanced functionality.

By using this package, issues like delays in transmitting images and receiving commands, sudden connection losses, overheating and high battery consumption — problems often encountered when using Python to communicate with the Tello drone — are resolved. These significant advantages of MATLAB convinced us to adopt it for our project.

IV. IMAGE PROCESSING

Object detection can be achieved using either traditional image processing methods or deep learning networks. As compared in [6], traditional methods, such as those using OpenCV, do not require training and operate in an unsupervised manner. These methods do not need labeled images but face limitations in complex lighting conditions, multi-object detection and processing large images quickly. On the other hand, deep learning methods, which are predominantly supervised, perform better in complex scenes and challenging lighting conditions but require large amounts of training data and time-consuming labeling. Today, deep learning methods are widely used by researchers and companies due to their higher accuracy and efficiency.

A. *Object Detection Methods*

Before the introduction of deep learning, object detection was traditionally performed using algorithms such as Viola-Jones Detector, HOG Detector and DPM. After the introduction of deep learning, object detection shifted to two-stage algorithms like versions of RCNN and SPPNet, as well as one-stage algorithms such as YOLO and SSD [1].

B. *Evaluation and Comparison of Object Detection Methods*

To evaluate and compare various object detection algorithms, a consistent and reliable dataset is required. One of the most prominent datasets currently used for evaluating new object detection algorithms is the Microsoft Common Objects in Context (MS COCO) dataset [7].

Typically, different algorithms are compared using a metric called Average Precision. AP indicates the accuracy of object detection on a dataset and serves as a general metric for assessing a model's performance in object detection. In this case, we used AP, AP50 and AP75 metrics to compare models. AP50 refers to the model's accuracy at a threshold of 50, which evaluates the model's performance under less strict conditions. AP75 applies to stricter scenarios and provides a more precise prediction. Finally, AP represents the average accuracy of the model across thresholds from 50% to 95%, indicating the range from the lowest to the highest accuracy. The higher these three values are in a model, the more accurate the model is, making it a better choice for selection.

As inferred from Table 1 [6][8][9][10][11][12], generally, the accuracy of two-stage detectors is lower compared to other methods. Among one-stage detectors, the algorithms yolov9, yolov7, EfficientDet-D5 and EfficientDet-D7x demonstrate higher accuracy compared to other algorithms. Finally, in the category of DETR detectors, which offer good accuracy, DINO and Sparse DETR exhibit better accuracy. Therefore, to choose a suitable model for object detection, we must also consider other parameters that are important in this field and compare them across these models.

Another important metric for evaluating the performance of algorithms is the time taken to process images. This can be measured in two ways: inference time (where shorter is better) or frames per second (FPS), where higher is preferable.

To obtain the inference time, simply divide 1 by the frames per second. As the model's accuracy increases, due to the need for more processing time, the inference time also increases. To make the best decision regarding the selection of the appropriate model, we choose the most accurate version of each model and compare them. Additionally, to determine the inference time for different models and compare them, it is necessary not only to use the same dataset but also to use the same GPU. This is because the GPU's power plays a significant role in determining the inference time.

Also as seen in Table 2 [8][9][10][11][12], with the introduction of the YOLO algorithm, FPS was significantly increased, marking a major breakthrough in computer vision technology, such that the next best performance after YOLOv9-e is approximately 80% slower. This difference is significant, especially for MAVs, which require quick decision-making in complex situations. In this table, a Tesla A100 GPU was used to compare DINO, while Tesla V100 was used for the other algorithms. Despite the A100 model being faster than the V100, the DINO model is still slower compared to the YOLO algorithms.

TABLE I. THE DETECTION RESULTS OF COCO DATASETS.

Method		AP	AP_{50}	AP_{75}
one-stage detectors	YOLOv9e	55.6	72.8	60.6
	YOLOv7x	53.1	71.2	57.8
	ExtremeNet	43.7	60.5	47
	CornerNet	42.1	57.8	45.3
	RefineDet	41.8	62.9	45.7
	EfficientDet-D5	51.5	70.5	56.1
	EfficientDet-D7x	55.1	74.3	59.9
two-stage detectors	Cascade R-CNN	42.8	62.1	46.3
	Grid R-CNN	43.2	63	46.6
	Faster R-CNN+++	34.9	55.7	37.4
DETR detectors	DINO	49.4	66.9	53.8
	Sparse DETR	49.3	69.5	53.3
	DETR-R101	43.5	63.8	46.4
	Deformable DETR	48.7	68.1	52.9

TABLE II. FPS COMPARISON OF MOST ACCURATE METHODS

Algorithm	**FPS**
YOLOv7x	114
YOLOv9e	87
Sparse DETR	17.2
EfficientDet-D7x	13.8
DINO	10
EfficientDet-D5	6.5

Given the strong performance of the YOLO algorithm, we will now explore the different versions of this algorithm in more detail and their implementation using MATLAB and Python.

V. YOLO ALGORITHM IMPLEMENTATION METHODS

The YOLO algorithm has been implemented in various programming languages and platforms, such as Python and MATLAB. These implementations typically use one of the neural frameworks, like TensorFlow or PyTorch, for training and running YOLO models. In this part of the research, we will focus on implementing versions 7 to 11 using Python and versions 4 and X using MATLAB.

A. *YOLO Implementation with Python*

1) Dataset Preparation: This step involves loading and preprocessing selected images, including resizing the images, creating labels and splitting the dataset into training and testing sets. Our chosen dataset consists of 995 car images, which was labeled using MATLAB's Image Labeler tool. Then, using the writetable command, we export the bounding box table to a text file and, with the help of a script, convert it to YOLO format for implementation in Python. Finally, the label coordinates are converted from [x, y, width, height], where the first two elements represent the coordinates of the top-left corner, to [label, x_center, y_center, width, height].

2) Model Training: In this step, we train the YOLO model using the prepared dataset. It's important to note that the main difference between version 7 and later versions is that version 7 needs to be cloned from GitHub before training, while later versions can be executed directly using the Ultralytics library [13][14][15][16]. In this model, the number of epochs is set to 100.

3) Model Evaluation: After training, we evaluate the model on the test data to measure its performance.

According to the mAP50 metric, except YOLOv7 all models which were trained with python performed very similarly; however, versions 7 appeared to perform not as well. The other metric used for comparing our models is F1 score. The higher this metric, the more it assures us that there is a better balance between the two. This is why it is referred to as the harmonic mean of precision and recall. This metric is another factor that can indicate the model's accuracy, with values ranging from 0 to 1. According to the obtained curves, versions 8 and 9 are very close to each other; however, they have performed significantly better compared to version 7. The results of comparing mAP50 and F1 score of trained models are shown in fig. 1 and fig. 2.

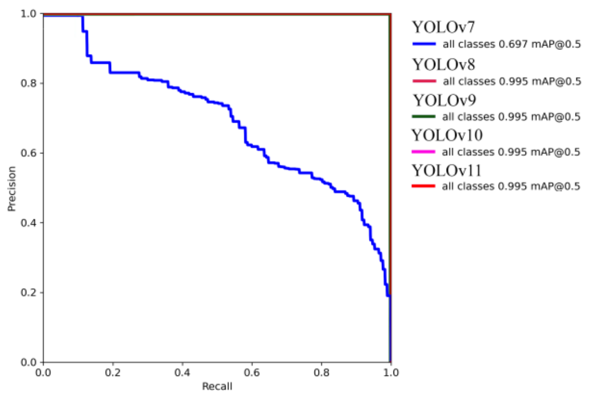

Fig. 1. Comparison of the mAP50 for five versions: YOLOv7 to YOLOv11

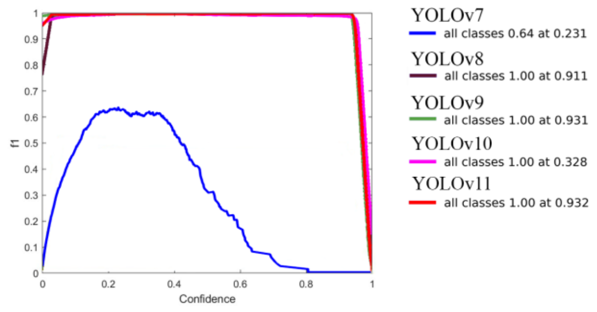

Fig. 2. Comparison of the F1 score for five versions: YOLOv7 to YOLOv11

B. *YOLO Implementation with MATLAB*

Since early 2024, MATLAB Deep Learning team integrated YOLOv8 and 9 directly into the MATLAB software, similar to YOLOv2 through YOLOv4 and YOLOX. In this unofficial method, pre-trained models are available for download and can be used directly in MATLAB. However, for training new models using this method, Python libraries must first be imported into MATLAB to train the new model. Currently, this capability is only available for YOLOv8 and yet it is not possible to train model with MATLAB implemented YOLOv9 [3][4].

One of the advantages of YOLOv8 in MATLAB is the ability to train models, with the main difference from previous versions being the ability to use the final model in an environment outside of MATLAB. However, for this, in addition to cloning the GitHub repository in [3], a Python environment must be created, so that Python libraries can be used within MATLAB. The resulted model will be the same as the model trained with python. But for training older versions of YOLO on MATLAB, we follow the steps below:

1) Dataset Preparation: To implement different versions of YOLO on MATLAB, the first step is to select the dataset, which is the same as in the previous section, consisting of 995 car images. This allows us to compare the results of the implementation in both Python and MATLAB. The image labeling process is the same as in the previous step, except that we no longer convert the gTruth format and instead, we use the output file directly from MATLAB's Image Labeler. The ROI format for the labels is as [x, y, width, height], where x and y represent the coordinates of the top-left corner of the bounding rectangle and the next two elements represent the width and height, just like in YOLO for Python. Then, we train the YOLOv4 and YOLOX models using the prepared dataset.

2) Model Training: When training the models, it should be considered that the first difference between YOLOv4 and YOLOX lies in the anchor box [17][18].

In some older object detection methods like YOLO, the use of anchor boxes is a key approach that directs the model's attention to specific points in the image, rather than scanning every point. Instead of gathering information from every possible position, anchor boxes of various sizes and aspect ratios are used. By utilizing these anchor boxes, the model searches only within specified regions of the image, which results in faster and more accurate detection, allowing the model to detect the desired objects more effectively [19]. Therefore, in YOLOv4, the number of anchors is set to 10.

In YOLOX, instead of using pre-defined anchor boxes, which consume significant memory, a center-based approach is used. In this approach, each pixel in the image predicts the presence of objects at the center of that pixel. This eliminates the need for anchor boxes and offers greater flexibility and efficiency in object detection. To achieve this, the network divides the input image into a three-scale grid and uses the grid points as offsets for the top-left corners of the bounding boxes [20].

In the detector stage, we need to choose the desired YOLO version, all labels, images, anchor boxes (for version 4) and the input image size. In MATLAB, for YOLOv4, only two models are available: csp-darknet53-coco and tiny-yolov4-coco. For YOLOX, only two models, small-coco and tiny-coco, are provided. To maintain consistency in the trained models, we select the "tiny" versions for both.

In the next section, we proceed to training settings. At this stage, the number of epochs is set to 100. The remaining features are left unchanged with their default values and each training stage is saved to a specific location so that, in case of an issue, the training can be resumed from that stage. It is worth noting that most of MATLAB's settings are similar to Python's, but perhaps the most significant difference lies in the optimization algorithms. MATLAB uses the SGDM algorithm by default, while in Python, the optimization algorithm is automatically adjusted based on the volume and type of data. In the previous section, the ADAMW algorithm was used. While MATLAB has the ADAM algorithm, it was not selected due to the need for specific adjustments.

Finally, we begin model training and once completed, we save the trained model in the specified location. The output will be saved as .mat file and can be used only by Matlab and also can’t be converted to .pt or .onnx file.

3) Model Training: After training, the model is evaluated on the test data to measure its accuracy and performance. In YOLOX, it is possible to directly obtain the mAP50 output, but this is not available in YOLOv4, so we configure the YOLOX settings accordingly.

According to the fig. 3 and fig. 4, the accuracy of the YOLOX model is significantly better than the other model in terms of mAP50 and F1 score at a threshold of 0.5. Unfortunately, in the F1 score section, due to limitations in the MATLAB library and the inability to set measurement intervals, the F1 score was only evaluated at a single threshold, making it difficult to comment on other criteria.

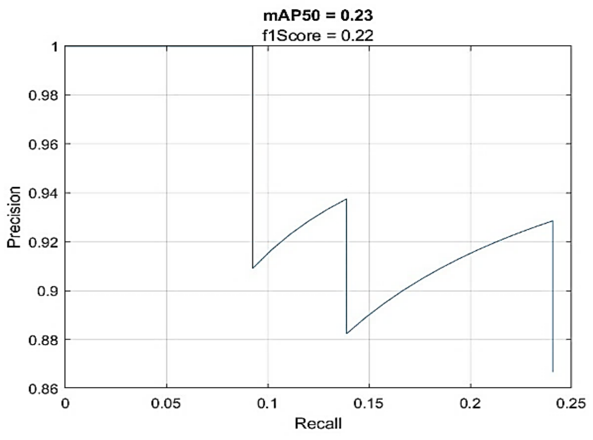

Fig. 3. mAP50 and F1 score for YOLOv4 model

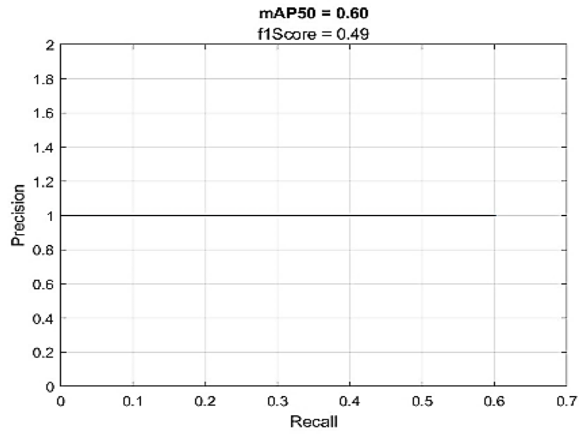

Fig. 4. mAP50 and F1 score for YOLOX model

C. *Comparison of the results of YOLO versions*

For conducting this comparison, efforts were made to use the simplest model in each version, so the comparison could be effectively carried out. Therefore, the Nano model was used in versions 8, 10 and 11 and the Tiny model was employed in the other versions.

In Python, after training the model, all results are automatically provided to the user as output. However, in MATLAB, these results need to be extracted manually. To do this, we first load the trained model in the software. Then, we load the data table and define the test data, which is shared across all models and assign the desired class.

As resulted in fig. 5 and Fig. 6, YOLOv8 and later have performed significantly better in terms of the accuracy metrics we used, namely F1 and mAP and are very close to each other given the dataset we utilized. They achieve a perfect F1 score of 1.00, indicating a balance between precision and recall at specific confidence thresholds and also have the highest mAP50, indicating the best overall performance in terms of average precision. Also, despite the better performance of YOLOv7 compared to the models used in MATLAB, this model has lower accuracy than the newer versions of YOLO. Of course, it's better to point out that YOLOX has performed much better than YOLOv4, but it doesn’t reach the near-perfect performance of the other YOLO models. The YOLOv4 curve shows a significant drop compared to the other models, indicating less reliable detection performance.

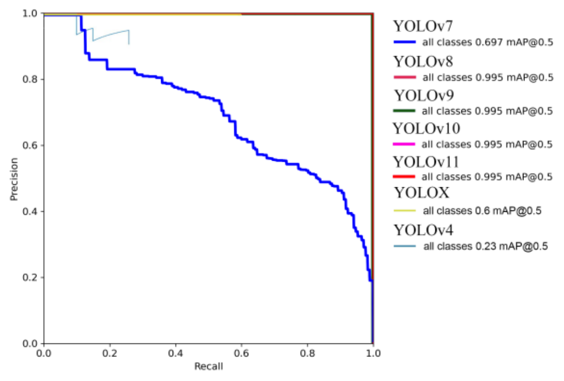

Fig. 5. Comparison of mAP50 in the reviewed YOLO versions

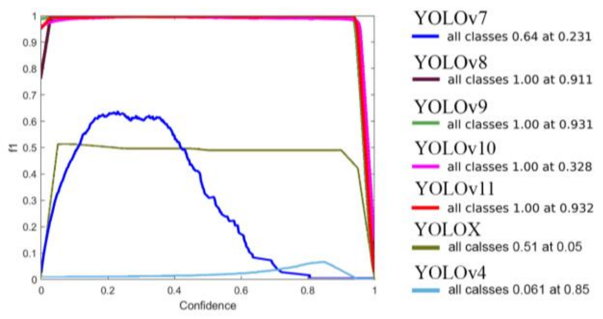

Fig. 6. Comparison of F1 score in the reviewed YOLO versions

On the whole it can be concluded that YOLO versions implemented on python have higher accuracy compared to the one implemented on MATLAB. These reasons convinced us to find some new ways to implement newer versions of YOLO on MATLAB to use either performance of our MAV, DJI Tello and accuracy and speed of newest versions of YOLO.

VI. Introducing a Solution to Overcome the Limitations of MATLAB

As can be seen, although there are differences in criteria such as speed and quality in the new YOLO models trained in Python software, the parameters to be investigated are simpler in the older models trained in MATLAB and there are significant differences even within the same class. Despite the fact that YOLOX is much better than YOLOv4, it has performed poorly against Python-trained models. These factors compel us to find a way to simultaneously use the extensive capabilities of MATLAB and precise Python models.

Two proposed methods for running new YOLO versions in the MATLAB environment using Python are examined in this article.

A. Using Python libraries in MATLAB

This method is common, as it allows for creating a communication environment [22] between these two platforms. At first glance, this method may seem very simple and logical, but due to the differences in the structures of MATLAB and Python, constant data format changes and even writing conventions should be considered. For example, in Python, it is enough to write a few short lines of code to track objects with YOLO models, but in MATLAB, many transformations must be performed to achieve the desired result. For example, image formats must be converted to uint8 to use them and then reversed to the previous format for use in Python libraries. Therefore, as mentioned, traditional methods cannot fully utilize Python's language and capabilities in MATLAB.

B. Using web server

A web server can be used simultaneously and without any delays between these two platforms. To do this, a web server is set up to receive requests from MATLAB and perform the necessary processing in Python. This method allows us to take advantage of advanced Python capabilities in MATLAB.

To implement this method, all information from MATLAB is placed on a server using the TCP protocol by setting up a host and a custom port. Subsequently, using Python, all necessary processing is done on the data and the results are sent back to the software. This method eliminates the need to create a new environment and reinstall Python libraries and sending and receiving can be done from two different devices. However, here we are using a local host, which first sets up a TCP server to receive images from clients, processes them and returns the result, which we elaborate on accordingly.

Initially, a TCP server function is set up that waits for client connections and manages each connection. It also manages keyboard interruptions and socket timeouts. Then, three functions are respectively written to receive data from the socket until reaching the desired length, convert a 1D array into a 3D matrix with specified dimensions, receive the length and dimensions of the image data from the client and process and send the results back to the client. Finally, the main code function for performing all processing on the sent data is written, which first converts them to a NumPy array, performs the required tracking operation and displays the result.

After creating the host, it is MATLAB software's turn to send the information. To do this, we first connect to the host. Then, the software reads frames of the video in order, sends them to the server, receives the processed results and assigns the result received from the server to the result.

VII. Comparison of Available Solutions

Finally, in order to find the fastest method for model training and image processing, we compare the two solutions presented in the previous section with the method provided in MATLAB software. For this purpose, we use YOLO version 8, which has been fully implemented in MATLAB.

Due to the interface created between Python and MATLAB, the speed of model training in MATLAB using Python libraries is much slower than training data directly in Python. This method is also used in YOLOv8, which was recently added to MATLAB. In the web server method, model training is done directly in Python, and it is faster compared to the other two methods. The accuracy of the output model in all three methods is equal, due to the use of Python, whether directly or indirectly, and the models are saved with the .pt extension, with the option to export outputs as .onnx as well.

Another criterion for comparing the three methods is the inference time, measured in frames per second. In this comparison, all pre-trained YOLOv8 models on the COCO dataset were tested under the same conditions using the RTX 4070 Ti SUPER graphics card. According to Fig. 7, using the web server method results in significantly less delay compared to the other two methods, and this performance advantage increases as the model complexity grows. Additionally, the method provided by MATLAB itself performs better and faster than the method of using Python libraries within this software.

VIII. Conclusion

The official YOLO versions implemented in MATLAB not only require a significant amount of time for tuning but also exhibit lower accuracy and speed compared to the newer models. For instance, the latest YOLO version, YOLOX, achieves only 51% of the accuracy of the most recent complete YOLO version, YOLOv11. With the increasing adoption of artificial intelligence across various industries and the continuous release of newer YOLO versions, we can expect this gap to widen. Therefore, there is a need to establish a connection with the newer versions, for which we proposed two methods: direct use of Python libraries and a web server. Also recently, MATLAB Deep Learning team has introduced unofficial implementation of YOLOv8 and YOLOv9 on MATLAB.

By comparing YOLOv8 version on both MATLAB and Python, the first drawback is the longer training time in MATLAB compared to Python, which could be due to the use of various Python libraries in MATLAB. This disadvantage becomes more noticeable with larger datasets. Another point in comparing YOLOv8 in MATLAB and Python is that Python processes faster.

To compare the three methods, we utilized all pre-trained YOLOv8 models on identical hardware and data, and it was observed that the web server method has a higher detection speed compared to the other methods, as illustrated by the chart in Fig. 7. Therefore, with this method, we can simultaneously benefit from the good connectivity capabilities between MATLAB and the UAV, as well as the fast and accurate detection capabilities of all the new versions of YOLO for image processing.

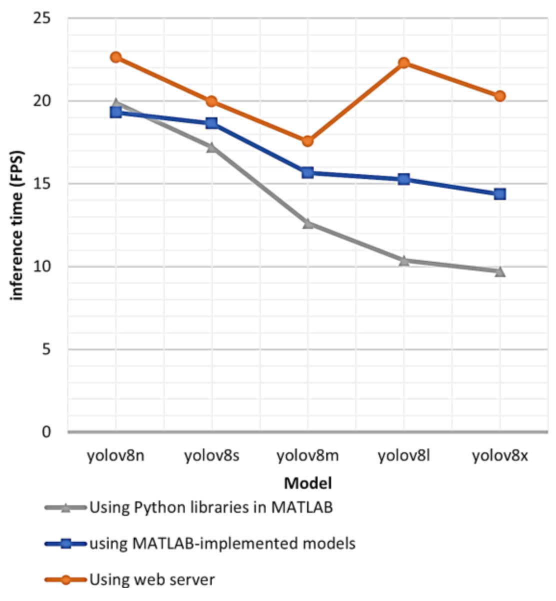

Fig. 7. Inference time comparison of discussed solutions

References

[1] I. S. Gillani *et al.*, “Yolov5, Yolo-x, Yolo-r, Yolov7 Comparison: A Survey,” no. Figure 1, pp. 17–28, 2022, doi: 10.5121/csit.2022.121602

[2] “Get Started with Object Detection Using Deep Learning,” MathWorks, https://www.mathworks.com/help/vision/ug/getting-started-with-object-detection-using-deep-learning.html

[3] Matlab-Deep-Learning, “Matlab-deep-learning/pretrained-yolov8-network-for-object-detection: Yolo V8 training and inference in MATLAB for object detection with YOLOV8N, yolov8s, YOLOV8M, YOLOV8L, yolov8x, networks,” GitHub, https://github.com/matlab-deep-learning/Pretrained-YOLOv8-Network-For-Object-Detection

[4] Matlab-Deep-Learning, “Matlab-deep-learning/pretrained-YOLOV9-network-for-object-detection: Yolo V9 inference in MATLAB for object detection with YOLOV9T, Yolov9s, yolov9m, yolov9c and Yolov9e Networks,” GitHub, https://github.com/matlab-deep-learning/Pretrained-Yolov9-Network-For-Object-Detection

[5] Fahmizal, D. Afidah, S. Istiqphara and N. S. Abu, “Interface design of DJI Tello Quadcopter Flight Control,” *Journal of Fuzzy Systems and Control*, vol. 1, no. 2, pp. 49–54, Aug. 2023. doi:10.59247/jfsc.v1i2.35

[6] Y. Sun, Z. Sun and W. Chen, “The evolution of object detection methods,” *Engineering Applications of Artificial Intelligence*, vol. 133, p. 108458, Jul. 2024, doi: https://doi.org/10.1016/j.engappai.2024.108458.

[7] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in context,” in *Lecture notes in computer science*, 2014, pp. 740–755. doi: 10.1007/978-3-319-10602-1_48.

[8] M. Yaseen, What is YOLOv9: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector, Sep. 2024. doi: https://doi.org/10.48550/arXiv.2409.07813

[9] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” *arXiv (Cornell University)*, Jul. 2022, doi: https://doi.org/10.48550/arxiv.2207.02696.

[10] M. Tan, R. Pang, and Q. V. Le, “EfficientDet: Scalable and Efficient Object Detection,” Nov. 2019, doi: https://doi.org/10.48550/arxiv.1911.09070.

[11] H. Zhang *et al.*, “DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection,” *arXiv (Cornell University)*, Jan. 2022, doi: https://doi.org/10.48550/arxiv.2203.03605.

[12] B. Roh, J. Shin, W. Shin, and S. Kim, “Sparse DETR: Efficient End-to-End Object Detection with Learnable Sparsity,” *arXiv (Cornell University)*, Jan. 2021, doi: https://doi.org/10.48550/arxiv.2111.14330.

[13] WongKinYiu, “GitHub - WongKinYiu/yolov7: Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” GitHub. https://github.com/WongKinYiu/yolov7

[14] Ultralytics, “GitHub - ultralytics/ultralytics: Ultralytics YOLO11 🚀,” GitHub. https://github.com/ultralytics/ultralytics

[15] WongKinYiu, “GitHub - WongKinYiu/yolov9: Implementation of paper - YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information,” GitHub. https://github.com/WongKinYiu/yolov9

[16] THU-MIG, “GitHub - THU-MIG/yolov10: YOLOv10: Real-Time End-to-End Object Detection [NeurIPS 2024],” *GitHub*, May 26, 2024. https://github.com/THU-MIG/yolov10?tab=readme-ov-file

[17] “Object Detection Using YOLO v4 Deep Learning - MATLAB & Simulink.” https://www.mathworks.com/help/vision/ug/object-detection-using-yolov4-deep-learning.html

[18] “Detect defects on printed circuit boards using YOLOX Network- MATLAB & Simulink.” https://www.mathworks.com/help/vision/ug/detect-pcb-defects-using-yolox-deep-learning.html

[19] “Anchor Boxes for Object Detection - MATLAB & Simulink,” *www.mathworks.com*. https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html

[20] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “YOLOX: Exceeding YOLO Series in 2021,” arXiv (Cornell University), Jan. 2021, doi: https://doi.org/10.48550/arxiv.2107.08430.

[21] “PythonEnvironment,” Python environment information - MATLAB, https://www.mathworks.com/help/matlab/ref/matlab.pyclient.pythonenvironment.html