Szymon CHERUBIN[1], Wojciech KACZMAREK[1], Michał SIWEK[1]

[1]Faculty of Mechatronics, Armament and Aerospace, Military University of Technology, Kaliskiego 2 Street, Warsaw, 00-908, Poland

doi:10.15199/48.2024.09.04

YOLO object detection and classification using low-cost mobile robot

Abstract. *The article presents a study of object detection and classification methods based on deep learning YOLO (You Only Look Once) implemented on the on-board computer (Raspberry Pi 4B) of a mobile robot. The research was carried out using a mobile robot working with the ROS Noetic platform, equipped with LiDAR, Kinect and odometric sensors. During the operation of deep neural networks, the average precision of mAP detection and the computing power consumption of the central unit without a graphics processor were determined. The HTOP package was used to observe computational processes and the level of CPU computing power consumption in real time.*

Streszczenie*. W artykule przedstawiono badanie metod wykrywania i klasyfikacji obiektów opartych na głębokim uczeniu YOLO (You Only Look Once) zaimplementowanych na komputerze pokładowym (Raspberry Pi 4B) zbudowanego robota mobilnego. Badania przeprowadzono z wykorzystaniem robota mobilnego współpracującego z platformą ROS Noetic, wyposażonego w czujniki: LiDAR, Kinect i odometryczne. Podczas działania głębokich sieci neuronowych określono średnią precyzję wykrywania mAP oraz zużycie mocy obliczeniowej jednostki centralnej pozbawionej procesora graficznego. Do obserwacji procesów obliczeniowych i poziomu zużycia mocy obliczeniowej procesora w czasie rzeczywistym wykorzystano pakiet HTOP. (***Wykrywanie i klasyfikacja obiektów YOLO przy użyciu niedrogiego robota mobilnego**)

Keywords: mobile robot, deep neural network, YOLO, object detection and tracking, Raspberry Pi
Słowa kluczowe: robot mobilny, głęboka sieć neuronowa, YOLO, detekcja i klasyfikacja obiektów, Raspberry Pi

Introduction

Advanced object detection and classificat ion algorithms are increasingly often implemented in mobile robots. Detection and classification algorithms are one of the sub-fields of AI (artificial intelligence), which has been dominating technology environments with remarkable speed in recent years and is being rapidly developed by the technology hegemons - Microsoft, Tesla etc. These types of algorithms play an extremely important role in commercial environments, as well as in the military. In both cases, they allow the detection of objects of interest. In the case of military applications, the lion's share is the use of these types of algorithms on platforms that carry out reconnaissance of the environment and enemy activities without risking human life and health. For the commercial environment, a great example is the implementation of these types of algorithms in autonomous vehicles, whereby the vehicle can pick out objects of interest from the environment and use them appropriately for its operations. An example of this is the detection of traffic signs, pedestrians or other objects that threaten safe movement. Another particularly important use for public safety is the implementation of such algorithms in CCTV cameras in public places. This makes it possible to detect and identify vehicle drivers breaking traffic regulations (registration number classification), or to detect offenders (face detection and classification).

Background

Artificial neural networks provide enormous human support in all fields [1]. Nevertheless, their implementation requires large decks of computing power. Nor is it different for detection and classification algorithms. Given the wide spectrum of applications for this type of algorithms, it is often impossible to use computing units equipped with advanced computing graphics processors. Therefore, we have conducted studies to examine and determine the detection accuracy, processing speed and the amount of computing power required to perform the task by a computing unit (CPU only) of mobile robot. A important aspect is the fact that we use the Raspberry Pi 4B single-plate computer, which can be successfully mounted on unmanned vehicles of small size, as well as on which construction is a small budget (fig.1).

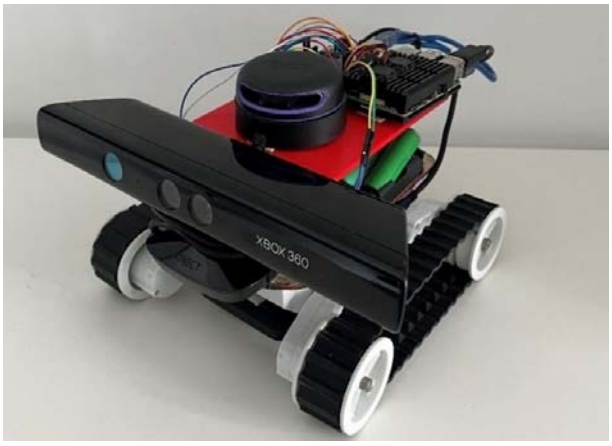

Fig.1. Low-cost mobile robot

A mobile robot requires an appropriate hardware structure to function properly and efficiently in the real environment. Implemented sensory systems allow the accumulation of relevant data from the environment [2-4]. The computing units, in turn, are responsible for the processing and transmitting them to the user's computing unit using IEEE 802.11ac remote data transmission. The hardware structure of the robot is shown below in block form in Figure 2.

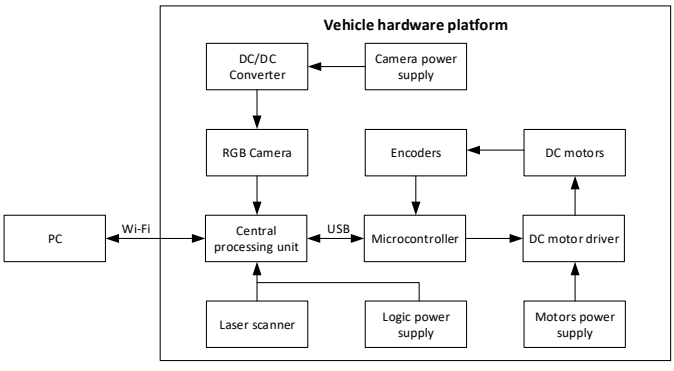

Fig.2. Hardware structure of the autonomous vehicle

The mobile robot with the structure presented in Figure 2, cooperates with the ROS Noetic platform, equipped with LiDAR, Kinect and odometry sensors. ROS is currently the most popular platform used in mobile robots in the world [5-7].

YOLO Algorithm

The YOLO algorithm was developed by Joseph Redmon [8]. Its innovation over other algorithms used for detection and classification is the use of functions learned by a deep convolutional neural network and from the fact that detection is treated as a regression problem [8-10]. This allows real-time processing of video streams with a delay of less than 25 ms [8]. This approach makes it possible to obtain a vector (bounding box) in the output containing the coordinates of the object's position together with its class probability. In addition, it has the highest speed and accuracy. The figure below shows the average detection precision along with the speed of the YOLOv3 algorithm compared to other commonly used algorithms.

Operating principle

The YOLO algorithm superimposes n grids of equal size S x S on the image. Each of these grids is responsible for detecting and locating the object contained within it. This is done by predicting the coordinates of the boundary frame B relative to the coordinates of their cell, as well as the label of the object and the probability of its presence in the cell. The prediction process of one image is made by multiple cells with different probability envelopes, leading to duplicate predictions. This is offset by the use of a non-maximal suppression function (NMS) [10]. It works by analysing the probability scores and eliminating frames with lower probability values. This procedure is carried out in a loop until the final B-frame is obtained.

Each boundary frame consists of 5 prediction values: *x, y, w, h* and confidence. The parameters *(x,y)* represent the centre of the frame relative to the grid cell boundaries. The others (*w,h*) define its width and height relative to the whole image. The confidence prediction represents the IoU between the predicted field and any actual field. Each grid cell also predicts a conditional class probability $\mathrm{P}(Class_i|Object)$. These probabilities are conditional on the grid cell containing the object. Only one set of class probabilities per grid cell is predicted, regardless of the number of B-frames. During testing, the conditional class probabilities and confidence predictions of the individual fields are multiplied as shown in the equation below (1). Figure 3. shows the principle of the YOLO algorithm together with the NMS function.

$$(1)\quad lass_i|Obiect) \cdot P(Obiect) \cdot IoU\left(\frac{Truth}{Prediction}\right) = \mathrm{P}(Klass_i) \cdot IoU\left(\frac{Truth}{Predicton}\right)$$

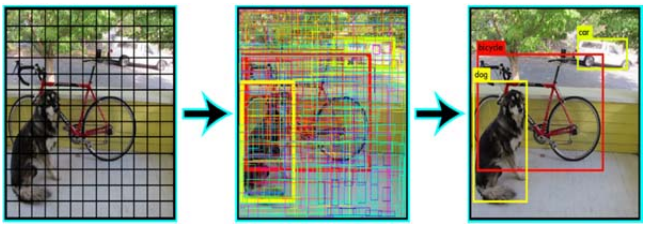

Fig.3. Operating procedure of the YOLO algorithm with NMS function [8]

Architecture

The YOLO algorithm network is a deep convolutional neural network. The architecture of the YOLO network is modelled on the GoogLeNet model [9-11], however, it uses reduction layers instead of inceptive modules. The YOLO network is built with convolutional layers and fully connected layers. Convolutional layers of 3x3 are used, which are responsible for extracting image features. These layers are alternated with 1x1 convolutional layers responsible for reducing the feature space from the previous layer. The convolutional layers are in turn responsible for predicting the probability and coordinates of objects in the image. Figure 4 shows the architecture of a YOLO network with 24 convolutional layers and 2 connected layers [8].

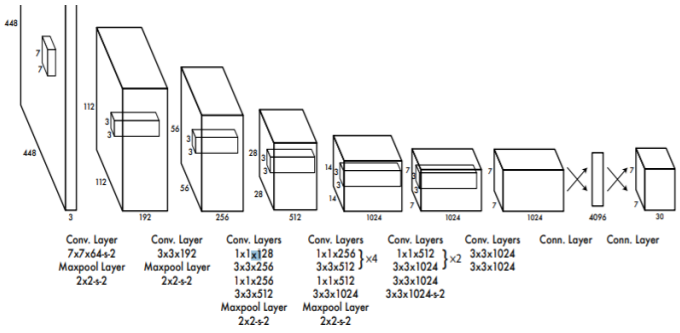

Fig.4. Architecture of a YOLO network [8]

Accuracy of YOLO models

YOLO networks were tested for detection and classification accuracy. The aim of the study was to identify the model in the YOLO family with the highest precision in object detection and classification and the shortest out-of-task time. The parameter determining the performance of the network was the average precision - mAP. The research methodology presented in this chapter consisted of running the YOLO algorithms on a test set of 20 randomly selected images from the VOC2007 database. The results are presented as graphs indicating the average detection precision in the mAP unit commonly used in work on detection and classification algorithms. The graphs also show the detection and classification precision of individual elements found in the images. The study mainly consisted of comparing the veracity of the objects and their locations in the images with the data calculated by the algorithms. The results were then compiled into a mAP unit.

Initially, detection results are treated as true matches when they have the same label and an intersection over union (IoU = 0.5) greater than 50%. A version of the measured precision curve with monotonically decreasing precision is then calculated. The average precision (AP) is calculated as the area under this curve using numerical integration. The average of all APs is then calculated, resulting in a mAP value from 0 to 100%.

- YOLOv2 based on the Darknet-19 architecture [12],
- YOLOv2-Tiny based on the Darknet-19 architecture [12],
- YOLOv3 based on Darknet-53 architecture [13],
- YOLOv4 based on CSPDarknet-53 architecture [14],
- YOLOv5 based on CSPDarknet-53 architecture [14],
- YOLOv7 based on CSPVoVNet architecture [15].

YOLOv2

YOLOv2 consists of a darknet-19 network containing 19 convolutional layers and 5 max-pooling layers. Convolution layers act as adaptive filters thatallow the image to be processed by various methods to detail features. The

remaining layers serve as linking layers and are responsible for reducing the dimension and number of parameters and computational effort. The YOLOv2 algorithm was trained and tested on the ImageNet 1000 dataset. The YOLOv2 network was trained and validated on images from the PASCAL VOC2007 data set. With that said, 80% of all images were used for training. This is the standard value for dividing the collection into training and validation parts. For validation, the network achieved an accuracy of 19 mAP [12]. Figure 5 shows graph of the average precision of detected objects of YOLOv2 [16].

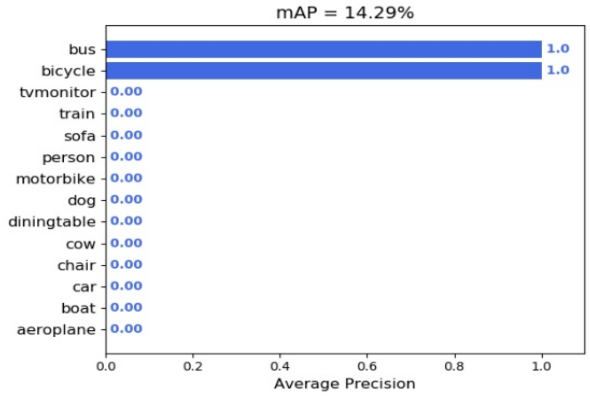

Fig.5. Graph of the average precision of detected objects of YOLOv2

YOLOv2-Tiny

YOLOv2-Tiny consists of 9 convolution layers with ReLU based activation functions and batch normalization. These layers are interleaved with 6 max-pooling layers. The YOLOv2-tiny network was trained on 80% of the images of the PASCAL VOC2007 data set. The validation accuracy was 35 mAP. Figure 6 shows graph of the average precision of detected objects of YOLOv2-Tiny. Figure 7 shows CPU computing power consumption of YOLOv2-Tiny.

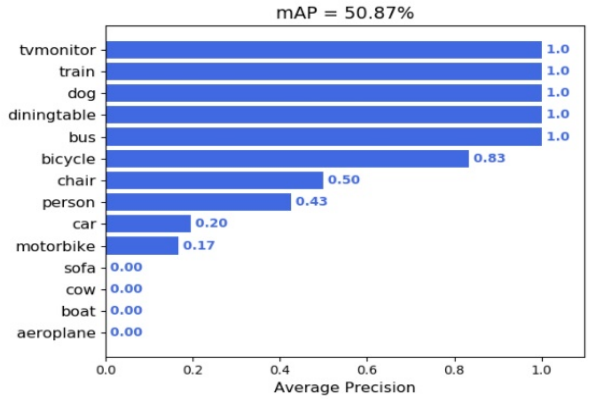

Fig.6. Graph of the average precision of detected objects of YOLOv2-Tiny

Fig.7. CPU computing power consumption of YOLOv2-Tiny

Figure 7 shows the consumption of each core of the Raspberry Pi 4B processor of the YOLOv2-Tiny algorithm. For the YOLOv2-Tiny algorithm the average CPU usage is 26.6%.

YOLOv3

YOLOv3 consists of a darkent-53 network containing 53 convolutional layers, each with batch normalization and Leaky ReLU activation. The algorithm was trained on the MS COCO dataset. The max-pooling layers were replaced by strided convolutions layers. The MS COCO dataset was used to train and validate the YOLOv3 network. As in the previous case, 80% of the dataset was used for training. The accuracy on the validation dataset was 28 mAP for 320 x 320 images [13]. Figure 8 shows graph of the average precision of detected objects of YOLOv3.

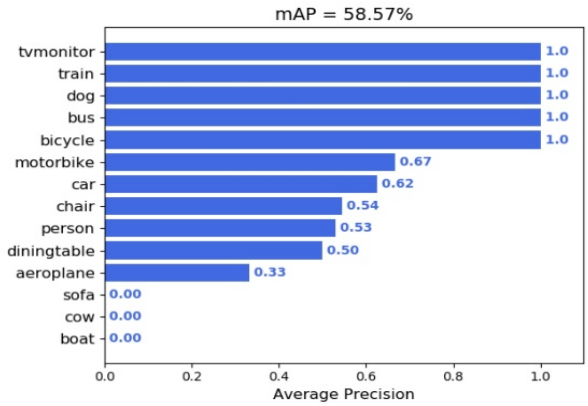

Fig.8. Graph of the average precision of detected objects of YOLOv3

YOLOv4

YOLOv4 is an advanced algorithm of which the ”backbone” of this network is darknet-53 containing 53 convolution layers and the ”neck” part, for which the convolutional neural architecture SPP-net was chosen. It uses spatial pyramid linking to remove the limitation of fixed network size. The ”head” part is still a YOLO network. The YOLOv4 network used, like the YOLOv3 network, the MS COCO dataset, with 80% devoted to the training dataset. For this algorithm, the accuracy on the validation set was 43 mAP [15]. Figure 9 shows graph of the average precision of detected objects of YOLOv4. Figure 10 shows CPU computing power consumption of YOLOv4.

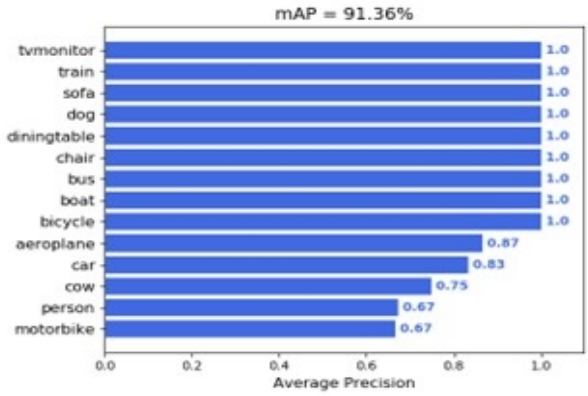

Fig.9. Graph of the average precision of detected objects of YOLOv4

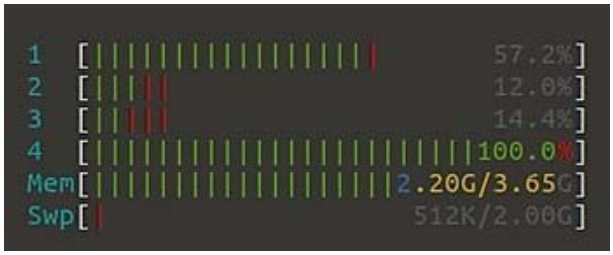

Fig.10. CPU computing power consumption of YOLOv4

Figure 10 shows the consumption of each core of the Raspberry Pi 4B processor of the YOLOv4 algorithm. For the YOLOv4 algorithm the average CPU usage is 46%.

YOLOv5

The YOLOv5 algorithm is a network built with 290 convolution layers. The core of the network is CSPDarknet53, which consists of multiple CBS modules (Convolutional Layers, Batch Normalized and SiLU), C3 modules and SPPF module. The CBS and C3 modules are used for feature extraction, while SPPF enhances feature expression capability. The YOLOv5 algorithm was trained on the MS COCO dataset, which allows the detection of a much larger number of objects than are contained in the PASCAL VOC test set [17]. Figure 11 shows graph of the average precision of detected objects of YOLOv5. Figure 12 shows CPU computing power consumption of YOLOv5.

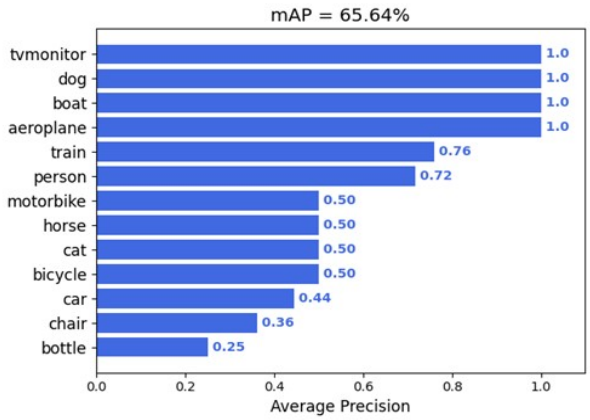

Fig.11. Graph of the average precision of detected objects of YOLOv5

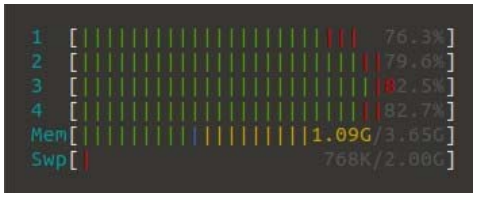

Fig.12. CPU computing power consumption of YOLOv5

Figure 12 shows the consumption of each core of the Raspberry Pi 4B processor of the YOLOv5 algorithm. For the YOLOv5 algorithm the average CPU usage is 80.3%.

YOLOv7

The YOLOv7 algorithm is a network built with 306 convolution layers. YOLOv7 uses the E-ELAN computing block, which stands for extended efficient layer aggregation network. The E-ELAN architecture in YOLOv7 enables the model to learn better by using ”expand, shuffle, merge cardinality” to achieve the ability to continuously improve the network’s learning capability without destroying the original gradient path [15]. Figure 13 shows graph of the average precision of detected objects of YOLOv7. Figure 14 shows CPU computing power consumption of YOLOv7.

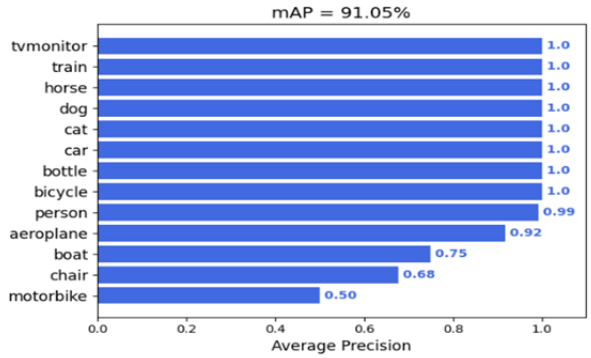

Fig.13. Graph of the average precision of detected objects of YOLOv7

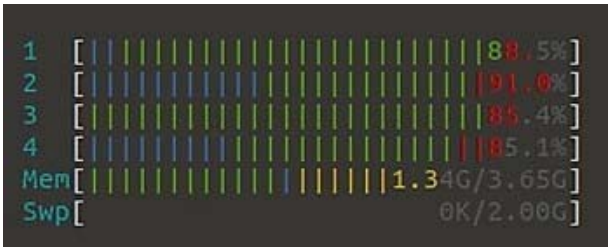

Fig.14. CPU computing power consumption of YOLOv7

Figure 14 shows the consumption of each core of the Raspberry Pi 4B processor of the YOLOv7 algorithm. For the YOLOv7 algorithm the average CPU usage is 87.5%.

Results

During the study, the average detection precision (figures 5-6,8-9,11,13) was determined as follows: 14.29% for YOLOv2, 50.87% for YOLOv2-Tiny, 58.57% for YOLOv3, 91.36% for YOLOv4, 65.64% for YOLOv5 and 91.05% for YOLOv7. In case of YOLOv5, simultaneous use of the deep network algorithm and the operation of other robot nodes was not possible, due to the high computing power consumption of the on-board computer. Nevertheless, the YOLOv5 algorithm also has errors in proper classification e.g. the cat in the image by the algorithm was classified as a dog.

In addition, the authors extended the research by measuring the CPU computing power consumption of the aforementioned algorithms. CPU consumption during the operation of deep neural networks was measured using the HTOP package that allows observation of computational processes and CPU consumption in real time. Figure 12 shows the consumption of each core of the Raspberry Pi 4B processor by the YOLOv5 and figure 7 of the YOLOv2-Tiny algorithm. YOLOv4 (figure 10), like YOLOv3, performed the set task in about 84s, which is too long to wait for the detection and classification result for mobile platforms. In addition, the YOLOv4 algorithm used 46% of the CPU’s computing resources. YOLOv7, as shown in figure 14, uses the most, about 88% of the CPU’s computing resources, with a task execution time of more than 14s.

Table 1. Average CPU consumption and detection execution time for YOLO algorithms

Algorithm	mAP [%]	Total CPU usage [%]	Detection execution time [s]
YOLOv2	14.29	28.1	39.7
YOLOv2-Tiny	50.87	26.6	3.40
YOLOv3	58.57	26.5	85.4
YOLOv4	91.36	46.0	84.0
YOLOv5	65.64	80.3	3.94
YOLOv7	91.05	87.5	14.4

Figure 15 shows the detection accuracy and task execution time of the YOLOv2-Tiny algorithms. In the case of the YOLOv2-Tiny algorithm, the task execution time was 3.4 seconds and the classification accuracy of the objects that were successfully detected in the test image was approximately 92%.

Analogous tests were conducted for the YOLOv3, YOLOv4, YOLOv5, YOLOv7 algorithms. For the YOLOv3 algorithm, the number of convolutional layers was 106, the task execution time was 85.4s, and the precision of object detection in the test image was about 100%. For the YOLOv4 algorithm, the number of convolution layers was 161, the task execution time was 83.1s, and the precision of object detection in the test image was about 95%. For the YOLOv5 algorithm, the number of convolution layers was 290, the task execution time was 3.9s, and the precision of object detection in the test image was about 98%. For the YOLOv7 algorithm, the number of convolution layers was 306, the task execution time was 14.3s, and the precision of object detection in the test image was about 99%. Average CPU consumption and detection execution time for YOLO algorithms are summarized in the table 1. Analyzing the consumption of computing resources of CPU cores during the operation of detection and classification algorithms, the YOLOv5 and YOLOv7 algorithms showed the highest consumption of over 80%. The smallest, on the other hand, the YOLOv3 algorithms together with YOLOv2Tiny amounted to about 26.5%. The YOLOv2 version of the algorithm had slightly higher CPU consumption at 28%.

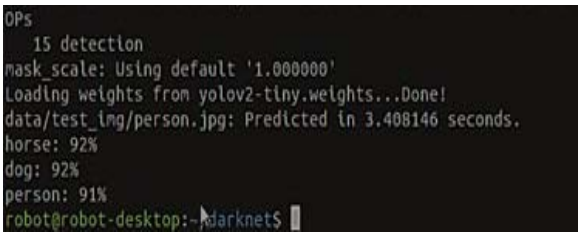

Fig.15. Object detection precision and task execution time by YOLOv2-Tiny algorithm

Conclusion

Considering the speed of the task at hand (detection and classification time) and the amount of resource consumption, the authors selected the YOLOv2-Tiny algorithm as the optimal one for use on unmanned platforms using CPUs without GPUs. The study showed that it is an algorithm that performs the task in about 3s, while consuming only about 27% of the computational resources of a 4-core CPU. It is also worth mentioning that the YOLOv5 algorithm, which has higher precision detection and classification, performs the task just as fast. The task execution time is approximately 4s. However, the disadvantage of using this algorithm on a mobile robot platform with limited computing resources (RPi 4B) is that it consumes about 80% of the CPU resources. The YOLOv3 algorithm had the lowest CPU consumption, but the execution time of the set task was as high as 85s, which disqualifies it for use on an unmanned platform.

Given the current direction of technological development, artificial intelligence is an inevitable "must-have" in the military as well as commercial sphere. As has already been mentioned at the beginning, they allow inspections to be carried out and gain an information advantage over the enemy without risking human life and with a higher reliability factor than in the case of human resources. However, without adequate artificial intelligence, the one responsible for the detection and classification of objects, it is not possible to an information advantage. In the case of commercial applications, algorithms of this type allow for increased public safety and will also facilitate everyday operations in the future.

This work was co-financed by Military University of Technology under research project UGB 826/2023.

REFERENCES

[1] Zainudin, M. N., et al., A Framework for Chili Fruits Maturity Estimation using Deep Convolutional Neural Network, *Przegląd Elektrotechniczny*, 12 (2021), 77-81, doi:10.15199/48.2021.12.13.

[2] Khamil, K. N., Adnan, M. A. A., & Annuar, M. A. K., Design and Development of a Sanitization Robot (ROBOSAN V2), *Przegląd Elektrotechniczny*, 10 (2023), 82-87, doi:10.15199/48.2023.10.16.

[3] Panasiuk J., Siwek M., Kaczmarek W., Borys S., Prusaczyk P., The concept of using the mobile robot for telemechanical wires installation in pipelines, p. 020054. doi:10.1063/1.5066516

[4] Zuniga W. P. C., Navarro J. J. Q., Diaz J. D. P., J. Tavera S., A. Montoya A., Design of a Terrain Mapping System for Low-cost Exploration Robots based on Stereo Vision, *Przegląd Elektrotechniczny*, 5 (2023), 270-275, doi:10.15199/48.2023.05.46.

[5] Rezoug, N., Zerikat, M., & Chekroun, S., An efficient fuzzy pi approach to real-time control of a ros-based mobile robot. *Przegląd Elektrotechniczny*, 2 (2022), 1-5, doi:10.15199/48.2022.02.01.

[6] Siwek M., Panasiuk J., Baranowski L., Kaczmarek W., Prusaczyk P., Borys S., Identification of differential drive robot dynamic model parameters, *Materials* 16 683. doi:10.3390/ma16020683.

[7] Siwek M., Baranowski L., Panasiuk J., Kaczmarek W., Modeling and simulation of movement of dispersed group of mobile robots using simscape multibody software, p. 020045. doi:10.1063/1.5092048.

[8] Redmon J., Divvala S., Girshick R., Farhadi A., You only look once: Unified, real-time object detection.

[9] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A., Going deeper with convolutions, IEEE, pp. 1–9. doi:10.1109/CVPR.2015.7298594.

[10] Hosang J., Benenson R., Schiele B., Learning non-maximum suppression, *Proceedings of the IEEE conference on computer vision and pattern recognition,* 2017, 4507-4515..

[11] L. L. Yin, M. N. Shah Zainudin, W. H. Mohd Saad, N. A. Sulaiman, M. I. Idris, M. R. Kamarudin, R. Mohamed, M. S. J. A Razak. Analysis Recognition of Ghost Pepper and Cili-Padi using MaskRCNN and YOLO, *Przegląd Elektrotechniczny*, 08 (2023), 92-97, doi:10.15199/48.2023.08.15.

[12] Redmon J., Farhadi A.. 2016. Yolo9000: Better, faster, stronger, *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, 7263-7271.

[13] Redmon J., Farhadi A., YOLOv3: An Incremental Improvement, *arXiv:1804.02767, University of Washington 2018*.

[14] Bochkovskiy A., Wang C.-Y., Liao H.-Y. M., Yolov4: Optimal speed and accuracy of object detection, *arXiv preprint arXiv:2004.10934* (2020).

[15] Wang C.-Y., Bochkovskiy A., Liao H.-Y. M., YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, *arXiv:2207.02696* (2022).

[16] Raji F., Miao L., Optimal wireless rate and power control in the presence of jammers using reinforcement learning. *ITU Journal on Future and Evolving Technologies. Volume 3, Issue 2, 508-522 (2022),* doi:10.52953/ANSC4385.

[17] Jocher G., ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, *Zenodo, lis. 22*, 2022. doi: 10.5281/zenodo.7347926.