Computers, Materials & Continua
DOI: 10.32604/cmc.2022.030016
Article

Tech Science Press

Methods and Means for Small Dynamic Objects Recognition and Tracking

Dmytro Kushnir*

Department of Computer Engineering, Lviv Polytechnic National University, Lviv, 79013, Ukraine
*Corresponding Author: Dmytro Kushnir. Email: dmytro.o.kushnir@lpnu.ua
Received: 16 March 2022; Accepted: 12 May 2022

Abstract: A literature analysis has shown that object search, recognition, and tracking systems are becoming increasingly popular. However, such systems do not achieve high practical results in analyzing small moving living objects ranging from 8 to 14 mm. This article examines methods and tools for recognizing and tracking the class of small moving objects, such as ants. To fulfill those aims, a customized You Only Look Once Ants Recognition (YOLO_AR) Convolutional Neural Network (CNN) has been trained to recognize Messor Structor ants in the laboratory using the LabelImg object marker tool. The proposed model is an extension of the You Only Look Once v4 (Yolov4) 512 × 512 model with an additional Self Regularized Non–Monotonic (Mish) activation function. Additionally, the scalable solution for continuous object recognizing and tracking was implemented. This solution is based on the OpenDatacam system, with extended Object Tracking modules that allow for tracking and counting objects that have crossed the custom boundary line. During the study, the methods of the alignment algorithm for finding the trajectory of moving objects were modified. I discovered that the Hungarian algorithm showed better results in tracking small objects than the K–D dimensional tree (k-d tree) matching algorithm used in OpenDataCam. Remarkably, such an algorithm showed better results with the implemented YOLO_AR model due to the lack of False Positives (FP). Therefore, I provided a new tracker module with a Hungarian matching algorithm verified on the Multiple Object Tracking (MOT) benchmark. Furthermore, additional customization parameters for object recognition and tracking results parsing and filtering were added, like boundary angle threshold (BAT) and past frames trajectory prediction (PFTP). Experimental tests confirmed the results of the study on a mobile device. During the experiment, parameters such as the quality of recognition and tracking of moving objects, the PFTP and BAT, and the configuration parameters of the neural network and boundary line model were analyzed. The results showed an increased tracking accuracy with the proposed methods by 50%. The study results confirmed the relevance of the topic and the effectiveness of the implemented methods and tools.

Keywords: Object detection; artificial intelligence; object tracking; object counting; small movable objects; ants tracking; ants recognition;

YOLO_AR; Yolov4; Hungarian algorithm; k-d tree algorithm; MOT benchmark; image labeling; movement prediction

1 Introduction

The increased need for autonomous systems for object finding, tracking, and monitoring [1–4] pushes for the development and applying new Machine Learning techniques. The most applicable task such systems may be applied to is car traffic monitoring [2–6]. However, at the same time, such tasks open doors for analyzing specific small objects, like ants [7,8]. Such systems analyze ants' movement in specific isolated environments, but they do not provide a scalable solution for counting and analyzing ants' movement patterns. Moreover, recognizing small movable objects requires great recognition and tracking algorithms since such objects occupy small space on the screen and may not move in strict, predictable directions.

Therefore, the development of methods and means that would allow to continuously train and process custom CNN models with specific classes such as ants species, and analysis of the advantages and disadvantages of using the developed methods for specific conditions, is an actual scientific task.

1.1 Purpose of the Study

In this study, the proposed system aims to continuously train and process custom CNN models, like a model for recognizing ant species. Afterward, I perform recognition and tracking for small movable objects in real-time. Finally, I propose the set of filters and modified tracking algorithm to enchant output results.

1.2 Contributions of the Study

The main contribution consists of the following:

- the current study designs a new CNN YOLO_AR model with Mish activation function and 516 × 516 dimensions;
- The ant's dataset in the indoor environment, which was used for labeling and training the CNN model;
- the scalable docker environment is employed as a storage for all system modules, like training, processing, and tracking nodes;
- the OpenDataCam object tracker module was enchanted with the usage of the Hungarian algorithm instead of the K-d tree algorithm, which showed a 50% accuracy increase for tracking small objects;
- the additional tracker filters were used to smooth the trajectory prediction like *BAT* and *PFTP;*
- finally, all modules were combined in the OpenDataCam system [1], which serves as an analytics tool for showing the results of the predictions and tracking. A wide range of simulations was carried out to highlight the YOLO_AR CNN model and modified tracking algorithm accuracy increase.

1.3 Organization of the Study

This paper consists of several sections. Section 2 briefs the recently-developed models for object Recognition and Tracking and scalable systems for showing output results related to the study domain. Section 3 introduces the proposed system architecture. Section 4 showed the training labeling and training process using the proposed dataset. Also, it introduces a set of filters for Object Recognition

output. In Section 5, a new tracking algorithm with matching and smoothing parameters is proposed to obtain better accuracy during small object counting. Section 6 showed the test execution process. Afterward, Section 7 showed the test results calculation and obtained data interpretation. Lastly, Section 8 draws the conclusion.

2 Related Works

There are plenty of means for continuous monitoring and tracking of objects embedded on mobile devices. The most popular systems are Opendatacam [1,2] and DeepStream Software Development Kit (SDK) [3].

Opendatacam is an open-source system for continuously monitoring objects using a custom Machine Learning model. The system can be integrated on mobile devices like Jetson Nano, Raspberry Pi, or Android operating system or executed on Linux-based machines via a web server. It reads data from the camera or through the Camera Serial Interface (CSI) bus.

DeepStream SDK allows executing programs directly on Nvidia mobile devices, like Jetson Nano or Jetson Xavier NX. In addition, it provides more quick data transfer between the visual sensor (Camera) via the CSI bus. However, it does not support many customization options for the Machine Learning model and Tracking parameters.

The Opendatacam system was chosen for further investigation as it provides better scalability and can be easily customized.

There are several approaches to formalizing the task of detecting small movable objects [5,6].

In particular, the Re-Identification (Re–ID) model [5] is based on optimized DenseNet121 with joint loss. This model applies the Squeeze–and–Excitation (SE) block to automatically obtain the importance of each channel feature and assign the corresponding weight to it. Features are transferred to the deep layer by adjusting the corresponding weights, which reduces the transmission of redundant information in the process of feature reuse in DenseNet121. The proposed model leverages the complementary expression advantages of the middle features of the CNN to enhance the feature expression ability [5].

At the same time, Real–the time Small Object Detection Algorithm (RSOD) [6] algorithm improves the small object detection accuracy by

- using feature maps of a shallower layer containing more fine-grained information for location prediction;
- fusing local and global features of shallow and deep feature maps in Feature Pyramid Network (FPN) to enhance the ability to extract more representative features;
- assigning weights to output features of FPN and fusing them adaptively;
- improving the excitation layer in the Squeeze–and–Excitation (SE) attention mechanism to adjust the feature responses of each channel more precisely [6].

Such approaches may be considered a good start for implementing our Tracking algorithm methods and means. However, such systems were applied only for the car–traffic tasks, where objects usually move in strict patterns. Therefore, I may consider ants tracking systems [7,8]. For example, in [7], the authors present their detection framework for ant's movement tracking. They propose:

- adopting a two-stage object detection framework using (Residual Network with 50 Layers) ResNet–50 as the backbone and coding the position of regions of interest to locate ants accurately;

– using the ResNet–50 model to develop the appearance descriptors of ants;
– constructing long–term appearance sequences and combining them with motion information to achieve online tracking.

At the same time, the following article [8] proposes an online MOT framework to track ant individuals. This framework combines both motion and appearance matching, effectively preventing trajectory fragments and ID (True Positive Id's) switches from long–term occlusion caused by frequent interactions of ants, achieving efficient and high–precision tracking [8].

Assuming the results of the authors, the MOT benchmark could be used to test the Tracker performance. However, the ResNet–50 model looks not so efficient for ant's recognition. Therefore, finding the most effective model for small movable object recognition is valuable.

Currently, many families of neural network models provide the ability to search and recognize objects. It is vital to highlight the family of You Only Look Once (Yolo) models, which use the division of the input video stream into cells and calculate the recognition probabilities for each of them [9].

In general, the following models of the Yolo family can be distinguished:

- You Only Look Once v3 (Yolov3) [10]—based on previous Yolo models with objectivity assessment of the regions. As a backbone, it uses the Darknet–53 framework instead of ResNet–152 as in previous versions. Additionally, Rectified Linear Activation Unit (Relu) [11] is used as an activation function. A three-level probability estimate has also been added to the model to improve the recognition rate of small objects.
- Yolov4 [12]—An updated version of Yolo [10], which shows an improvement of 10% mean Average Precision (mAP) compared to the previous model. As a backbone uses a modified version of Darknet CSPDarknet53. Mish [13] is used as an activation function. The SPP (Spatial Pyramid Pooling) unit is also used to increase the efficiency of the receptive field. At the same time, the PAN (Path Aggregation Network) unit is used for more efficient aggregation of parameters between different levels of the backbone. Additionally, Yolov4 offers Mosaic data augmentation methods and Self–Adversarial Training (SAT) to improve recognition.
- You Only Look Once v4 Scaled (Yolov4 Scaled) [14]—a modified version of yolov4 [12]. It has additional backbone layers: ResNet and ResNeXt and the main CSPDarknet53. The authors also added some functionality to scale the power of the model.
- You Only Look Once v5 (Yolov5) [15]—A completely new Yolo [10] model implementation on the PyTorch framework. However, identical architecture with Yolov4 [12] significantly reduces image recognition speed and quality.
- You Only Learn One Representation (YoloR) [16]—Continuation of research to improve the effectiveness of the yolov4 model [11]. The idea is to add to the learning mechanism conscious cognition (prepared data for learning) and unconscious cognition (by analogy with the human subconscious).

The Yolov4 algorithm was chosen for further research because it showed the highest efficiency among all models of neural networks for tracking small objects. Furthermore, the chosen model was modified with minimizing and smoothing filters [9] to increase recognition output.

For image labeling purposes in Yolov4 format, the LabelImg [17] tool was chosen, as it is free, open-source, and can spread object labeled rectangles to the next frame. Therefore, it is useful when there is a need to annotate the video stream, where objects are not moving much compared to the next frame.

The tracking algorithms may provide different results with different output parameters [18,19]. The MOT (Multiple Object Tracking Benchmark) [20,21] evaluation tool may be used to standardize observation results. It provides several measures, from recall to precision to running time.

The matching algorithms can be applied for multiple tasks, but the most efficient object trajectory tracking methods are Hungarian [22] k-d tree [23] algorithms.

The Docker system [24] was used to increase the reliability and scalability of the recognition and tracking model. Docker tasks manager proved to be the most efficient for this class. The system does not require large capacities and calculations but configures the environment and performs the necessary operations.

In conclusion, the reviewed tracking systems do not provide efficient ways to analyze small movable objects, as they are usually applied to car traffic tasks. Therefore, it is advisable to develop and research methods and tools to recognize and track small movable objects in real-time.

3 Means for Objects Recognizing and Tracking

According to the results of literary works research, I propose a system for small movable objects recognition and tracking. Furthermore, the system is fully scalable, which means any custom objects may be trained via the proposed train service and applied to the system.

The proposed system consists of an OpenDataCam tool extension with extra integrated modules for achieving the research goal. The schematic diagram of the recognition and tracking system is shown in Fig. 1.

The system is fully dockerized, which means it can be deployed on an external cyber-physical system based on Jetson Nano or Raspberry PI. The current implementation was handled on an Ubuntu machine with Ray Tracing Texel eXtreme (RTX) 2070 graphics card.

The system consists of the following components:

- Model train service. The extended service for the autonomous deployment of new weights of the model to the Neural Network Execution Environment;
- Neural Network Execution Environment (NNEE). This docker image stores the YOLO_AR pre-trained object recognition model based on the YOLOv4 Darknet CNN model. A Compute Unified Device Architecture (CUDA) processor processes all parallel operations. Additionally, this container listens to the input video stream from the camera. It bypasses the data via the Open Source Computer Vision Library (OpenCV) module to the Node Js application backend service.
- Node Js Application.

It consists of two server apps: OpenDataCam Server and Hypertext Transfer Protocol (HTTP) Proxy.

- OpenDataCam HTTP Proxy. This app is the starting point that receives model configurations and executes the process of execution of the NNEE docker image. In addition, this component starts web UI and passes input data by Motion Joint Photographic Experts Group (MJPEG) stream via port 8090 to the Frontend Application.
- OpenDataCam Server. Backend Service continuously receives recognition data from NNEE JsonStream via exposed port 8070. This object recognition data is processed and stored in MongoDB Database via port 27017. In the next step, the obtained video frames with recognition results are bypassed to the Tracker Module to track and count recognized objects.

This data will be stored and remain in the database volume memory even if the system is offline. After each frame execution, the processed data is sent to the Frontend Application via public port 8080.

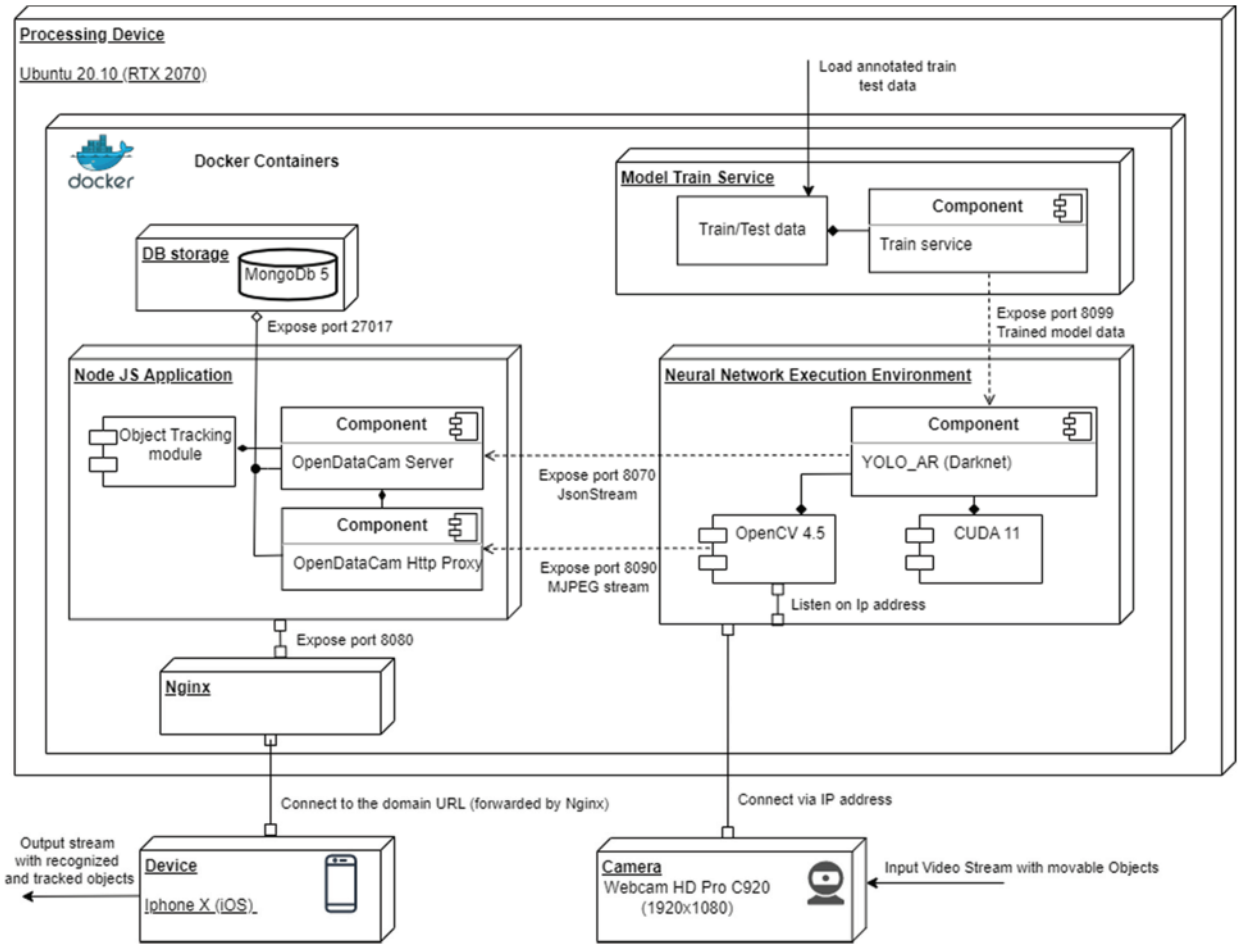

Figure 1: The schematic diagram of the proposed system for continuously recognizing and tracking movable objects

- Database storage. Storage for user history of tracking information per some recognized classes.
- Engine–X (Nginx). Container for forwarding Application Stream to the Frontend App. Uses reverse–proxy technique to direct requests to a particular client domain.
- Device. User device to see and manipulate Application results.
- Camera. Sensor for recording Object Recognition and Tracking stream.

4 Training and Evaluating the Proposed Model Based on the Ants Dataset

For study purposes, I choose the YoloV4 model architecture. The resulted YOLO_AR consists of 3 layers with dimensions of 512 × 512 pixels. The additional Mish layer was added to increase the model recognition accuracy. Such YOLO architecture overview with image processing techniques as

described in the previous paper [9]. The resulted model was adjusted with smoothing and minimization filters.

4.1 Preparing the Dataset for the Model and Marking Process

The target object is ants ranging in size from 8 to 14 mm. I created the corresponding image dataset [25] to fulfill research goals.

The dataset was created using Fast Forward MPEG (*FFmpeg*) Linux tool by obtaining an image sequence from the video stream. I have used two types of videos: big and small size of ants. Each frame from the dataset has 2160 × 3840 dimensions.

The dataset consists of 500 hundred images that capture ant movements in the indoor environment. The important thing here is that ants should be located on the visible surface. Otherwise, the recognition results will not be enough as objects may merge with the surrounding landscape or nest surface. For test evaluation, Camera HD Pro C920 was used. For labeling output image results, the LabelImg tool was used. The labelimg process is depicted in Fig. 2.

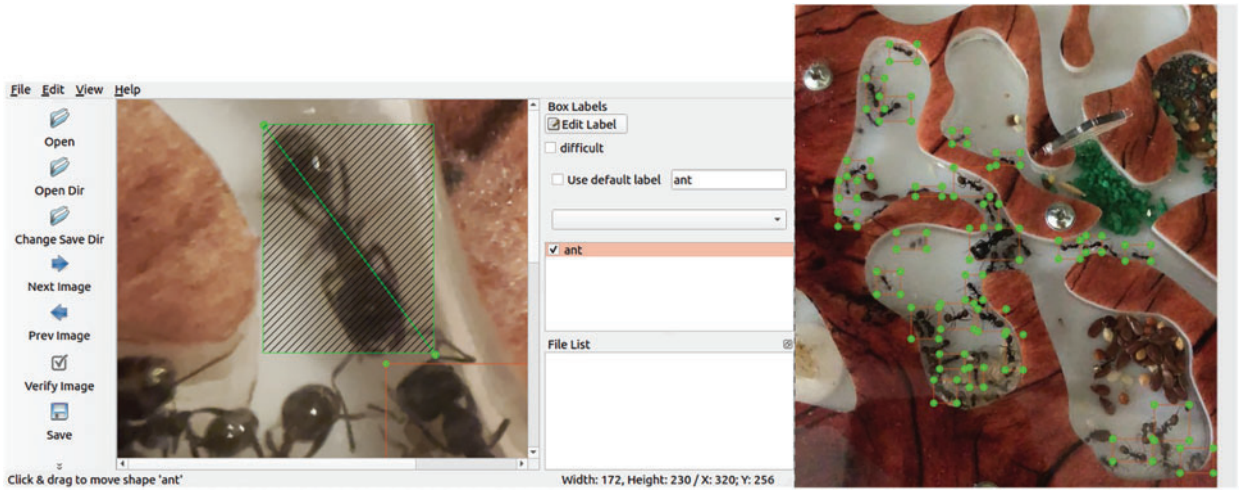

Figure 2: Labeling process of marking messor structor ants with different objects size. the boundary limit for ants is set to 14 mm

4.2 Training the YOLO_AR Model

We used a darknet framework [13] to train the YOLO_AR model on GeForce RTX 2070 with CUDA 11. The results are the follows (Fig. 3).

The blue curve indicates how many errors occur when learning and demonstrates whether the network is learning (graph is down) or degrading (graph is up). The y–axis represents the value of the loss ratio, while the x–axis represents the number of performed iterations. Also, the red graphic shows mAP (mean over precision) per iteration. The training was finished at 4000 iterations as it was noticed that model was not improved much after that point. After the training, the resulting model weights and configuration were loaded to the NNEE docker image module, from where it was accessed by the recognition and tracker modules.

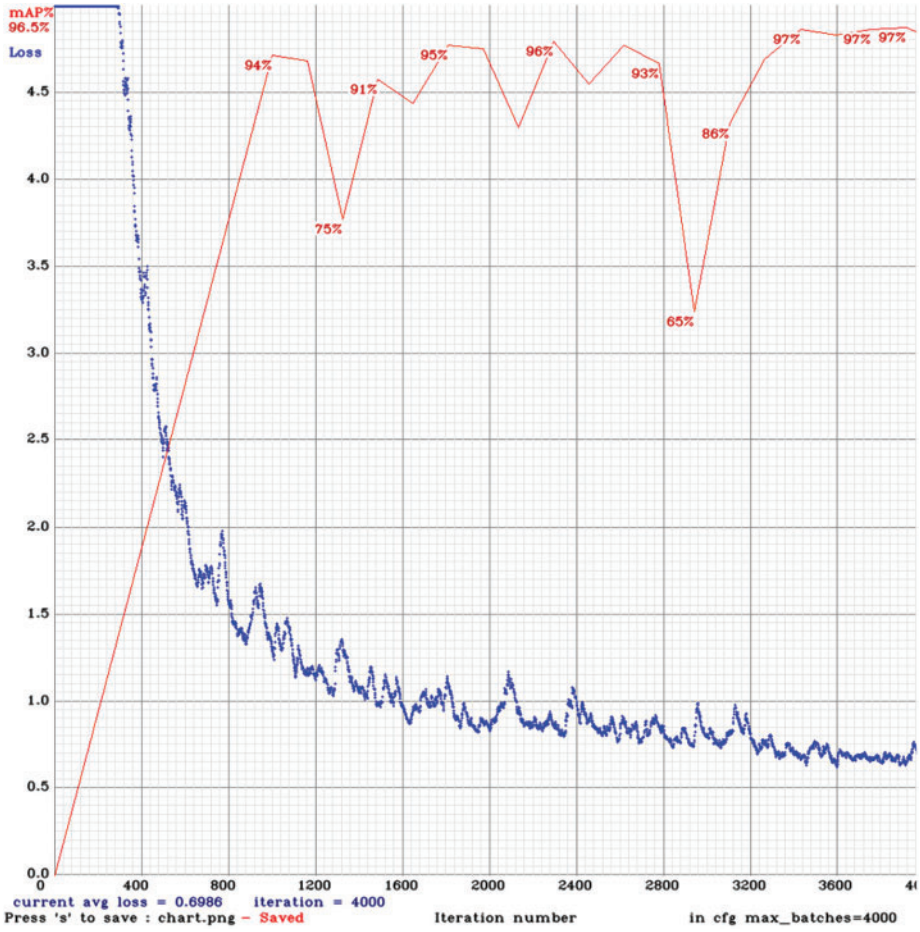

Figure 3: The training process of the proposed YOLO_AR neural network

4.3 Model Output Results Dynamic Customization

We applied an additional set of filters to increase Object Recognition accuracy and remove objects overlapping. Such filters are crucial for YOLO models since such architecture divides the frame into regions with a probability of recognition. Afterward, the regions may overlap each other. The following set of filters should help omit such collisions:

– ***recognized frame size filter (RFSF)***: Filters out objects whose area is higher than a certain percentage of the total frame area calculated by the following formula:

$$(detection_w \times detection_h) \leq \left(\frac{frame_w \times frame_h}{area\_koef}\right) \times 100 \quad (1)$$

Where:

$detection_{wh}$—input frame resolution (height and width) of the recognized bounding box;

$frame_{wh}$—input frame resolution (height and width) from the optical sensor;

area_koef—recognized frame size coefficient determines how many recognized objects should be skipped.

– ***confidence threshold minimizing filter (CTMF):*** minimization filter for removing the recognized object with the confidence value more minor than in the provided filter value;
– ***IOU smoothing filter limit:*** filter checks the border values of input object recognition IOU results per frame. I expect less overlapping with the small value of this filter.

5 Methods for Object Tracking and Counting

5.1 Applying Tracker Algorithm based on V–IOU

Based on Visual Intersection over Union (V–IOU) [18], the algorithm compares the overlapping areas between two recognized objects. This method allows checking if the object is the same during recognition. Under the hood, it computes the IOU (Intersection over Union) value by following the formula.

$$IOU = Si/Su \tag{2}$$

Where Si means interception area and Su means union area.

The graphical representation of this formula is the follows (Fig. 4).

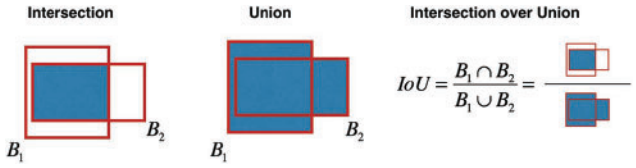

Figure 4: The graphical representation of the IOU formula

If gaps are in the prediction array, the algorithm skips those predictions and restores them after the wrong path is restored. The schematic diagram of the Tracking algorithm can be seen in Fig. 5.

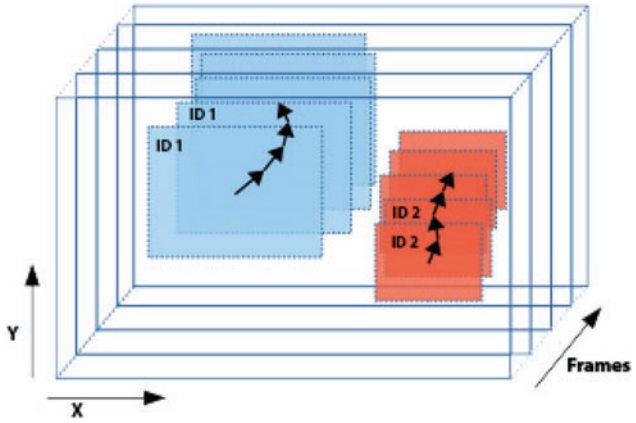

Figure 5: The YOLO objects tracker algorithm. It assigns a unique identifier for each object and tracks it over the frame's stream

Additionally, the tracking algorithm matches predictions for the next frame based on the velocity and acceleration vector to avoid ID (True Positive Id) reassignment when the object is missed only for a few frames. The existent OpenDataCam matching algorithm is based on the k-d tree algorithm.

5.2 Customization Options for Tracking Results Enchantment

We added several sets of parameters to adjust tracking results. The primary enchantments are the follows:

- ***frame pending limit (FPL)*:** the number of the frame to keep predicting the object trajectory if the next frame does not match it. Setting this higher will cause fewer ID switches but more potential false positives with an ID going to another object.
- ***boundary angle threshold (BAT)*:** Count items crossing the counting line only if the angle between their trajectory and the counting line is superior to this angle (in degree). Ninety degrees would count only perfectly perpendicular objects, whereas 0 degrees will count every recognized object. The graphic representation of such a method is depicted in Fig. 6.

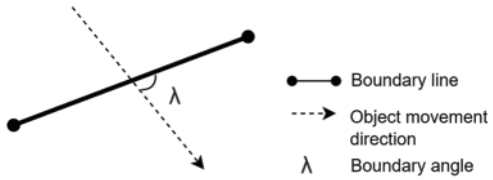

Figure 6: The boundary angle threshold parameter for changing objects' accessibility to the boundary line

- ***past frames trajectory prediction (PFTP)*:** parameter computes the trajectory to determine if an object crosses the line based on this number of the past frames. In most cases, the trajectory of the center of the bounding box given by YOLO_AR changes per frame, so the smoothing filtering will help verify object line crossing time and the angle of crossing. For example, if the object ID were lost during tracking, the algorithm would choose the last frame assigned to the ID from memory (Fig. 7).

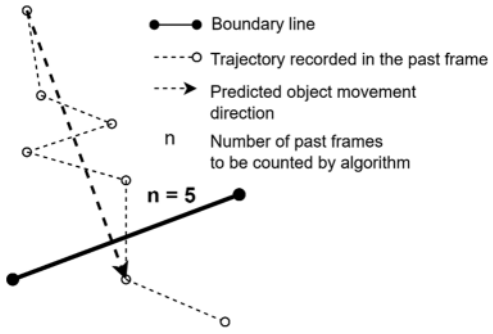

Figure 7: The past frames trajectory prediction parameter normalizes the object trajectory to the boundary line

5.3 Hungarian Algorithm Integration in Comparison to K-D Tree Algorithm

I discovered that an alternative matching algorithm using the Hungarian algorithm (also called the Munkres assignment algorithm) instead of the existing k-d tree algorithm used in OpenDataCam could improve efficiency with YOLO models, especially for small tracked objects. Furthermore, that algorithm should show better accuracy because it finds the minimum cost matching while the k-d tree algorithm only approximates a positive solution.

The tests were executed on the MOT17 benchmark by applying Tracker logic with the k-d tree matching algorithm and the proposed Hungarian algorithm. The results are the follows (Figs. 8 and 9).

	IDF1	IDP	IDR	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP	IDt	IDa	IDm
MOT17-09-SDP	48.8%	64.2%	39.3%	57.3%	93.5%	26	7	17	2	211	2274	45	78	52.5%	0.149	34	18	7
MOT17-05-FRCNN	44.7%	61.6%	35.1%	51.4%	89.9%	133	24	56	53	397	3365	62	72	44.7%	0.172	82	20	40
MOT17-04-SDP	61.9%	74.5%	52.9%	69.0%	97.1%	83	32	39	12	981	14763	119	367	66.6%	0.151	82	46	13
MOT17-09-FRCNN	43.5%	61.1%	33.7%	53.3%	96.6%	26	6	16	4	99	2485	35	38	50.8%	0.096	25	15	5
MOT17-02-SDP	34.7%	47.2%	27.4%	44.7%	77.2%	62	11	32	19	2458	10276	168	341	30.6%	0.199	110	69	14
MOT17-11-DPM	50.5%	72.2%	38.8%	50.0%	93.1%	75	8	27	40	351	4720	50	55	45.7%	0.219	33	27	11
MOT17-02-FRCNN	31.5%	52.7%	22.5%	34.6%	81.1%	62	6	26	30	1494	12152	91	124	26.1%	0.126	53	49	11
MOT17-11-SDP	53.6%	63.3%	46.5%	66.6%	90.6%	75	21	37	17	648	3154	54	100	59.1%	0.149	39	27	14
MOT17-09-DPM	40.5%	50.4%	33.8%	53.4%	79.7%	26	2	19	5	726	2479	70	164	38.5%	0.273	49	23	6
MOT17-13-FRCNN	50.9%	61.7%	43.4%	55.8%	79.3%	110	32	48	30	1698	5150	192	257	39.5%	0.168	131	84	32
MOT17-10-SDP	49.0%	56.9%	42.9%	67.4%	89.5%	57	23	30	4	1018	4180	166	292	58.2%	0.205	111	61	10
MOT17-13-DPM	24.9%	69.3%	15.2%	19.2%	87.6%	110	8	25	77	317	9412	40	106	16.1%	0.272	30	26	16
MOT17-11-FRCNN	52.4%	70.3%	41.7%	55.2%	93.0%	75	15	32	28	393	4232	42	48	50.5%	0.095	31	23	12
MOT17-10-DPM	29.5%	50.2%	20.9%	36.2%	87.2%	57	7	18	32	682	8190	80	142	30.3%	0.252	38	48	7
MOT17-04-FRCNN	49.7%	68.0%	39.1%	53.2%	92.4%	83	17	43	23	2072	22239	91	88	48.7%	0.108	41	55	5
MOT17-13-SDP	59.8%	77.2%	48.8%	54.8%	86.6%	110	44	23	43	987	5266	74	204	45.7%	0.213	72	25	27
MOT17-10-FRCNN	39.5%	46.3%	34.4%	57.8%	77.8%	57	15	35	7	2117	5422	223	306	39.5%	0.162	143	88	16
MOT17-04-DPM	33.4%	53.9%	24.2%	38.8%	86.3%	83	4	45	34	2934	29093	235	389	32.2%	0.217	104	139	10
MOT17-05-SDP	43.3%	54.1%	36.1%	58.0%	86.8%	133	32	63	38	611	2907	88	125	47.9%	0.165	113	22	47
MOT17-02-DPM	21.9%	61.0%	13.4%	18.9%	86.6%	62	4	14	44	546	15061	48	102	15.7%	0.246	23	30	5
MOT17-05-DPM	36.3%	62.1%	25.6%	35.9%	87.0%	133	11	52	70	370	4433	65	118	29.6%	0.248	71	22	29
OVERALL	44.5%	62.4%	34.6%	49.2%	88.7%	1638	329	697	612	21110	171253	2038	3516	42.3%	0.170	1415	917	337

Figure 8: The k-d tree matching algorithm tests results with mocked data on the MOT17 benchmark

	IDF1	IDP	IDR	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP	IDt	IDa	IDm
MOT17-09-SDP	42.7%	53.1%	35.7%	64.2%	95.5%	26	9	16	1	161	1905	72	130	59.8%	0.158	69	11	11
MOT17-05-FRCNN	48.9%	66.4%	38.7%	52.9%	90.7%	133	24	58	51	375	3258	58	79	46.6%	0.171	78	23	43
MOT17-04-SDP	66.2%	77.2%	58.0%	73.2%	97.5%	83	38	33	12	910	12728	121	338	71.1%	0.153	93	28	14
MOT17-09-FRCNN	50.2%	68.2%	39.8%	57.4%	98.6%	26	5	19	2	43	2266	35	49	56.0%	0.109	29	13	7
MOT17-02-SDP	28.7%	37.4%	23.3%	47.7%	76.7%	62	12	36	14	2692	9725	273	406	31.7%	0.200	202	72	21
MOT17-11-DPM	39.2%	54.2%	30.7%	51.9%	91.7%	75	9	26	40	444	4538	59	69	46.6%	0.219	49	25	16
MOT17-02-FRCNN	33.1%	54.8%	23.7%	35.0%	80.8%	62	6	27	29	1546	12085	96	138	26.1%	0.125	68	42	14
MOT17-11-SDP	50.9%	57.9%	45.4%	70.8%	90.4%	75	26	33	16	709	2755	82	109	62.4%	0.151	64	26	20
MOT17-09-DPM	38.3%	45.7%	33.0%	56.9%	78.8%	26	2	19	5	815	2296	81	165	40.1%	0.270	68	18	7
MOT17-13-FRCNN	47.0%	53.7%	41.8%	59.3%	76.1%	110	37	49	24	2163	4737	278	326	38.3%	0.178	199	90	39
MOT17-10-SDP	49.5%	55.3%	44.7%	72.1%	89.2%	57	29	24	4	1124	3582	186	305	61.9%	0.206	126	49	11
MOT17-13-DPM	26.5%	71.9%	16.2%	19.8%	87.7%	110	9	26	75	323	9337	29	92	16.8%	0.271	18	24	13
MOT17-11-FRCNN	47.4%	63.1%	37.9%	55.6%	92.4%	75	16	32	27	430	4192	48	50	50.5%	0.095	38	22	14
MOT17-10-DPM	34.9%	58.1%	24.9%	37.4%	87.2%	57	7	18	32	707	8042	57	137	31.4%	0.251	23	41	7
MOT17-04-FRCNN	52.0%	70.5%	41.2%	54.2%	92.9%	83	18	42	23	1969	21769	86	95	49.9%	0.108	48	42	4
MOT17-13-SDP	60.0%	75.5%	49.8%	57.2%	86.7%	110	47	25	38	1017	4987	83	224	47.7%	0.214	85	19	26
MOT17-10-FRCNN	39.8%	45.7%	35.3%	59.8%	77.4%	57	14	38	5	2243	5163	235	329	40.5%	0.165	150	84	18
MOT17-04-DPM	35.9%	54.9%	26.7%	41.9%	86.4%	83	7	43	33	3144	27627	273	428	34.7%	0.218	173	98	12
MOT17-05-SDP	46.9%	56.1%	40.2%	63.4%	88.4%	133	32	70	31	578	2533	96	131	53.6%	0.165	132	22	59
MOT17-02-DPM	20.3%	55.2%	12.4%	19.5%	86.4%	62	4	14	44	568	14965	58	113	16.1%	0.248	36	25	3
MOT17-05-DPM	34.5%	56.3%	24.9%	37.4%	84.4%	133	10	55	68	477	4332	68	123	29.5%	0.247	78	22	34
OVERALL	45.2%	61.4%	35.8%	51.7%	88.6%	1638	361	703	574	22438	162822	2374	3836	44.3%	0.171	1826	796	393

Figure 9: The Hungarian(munkers) matching algorithm tests results with mocked data on the MOT17 benchmark

Where:

- **IDF1**: The ratio of correctly identified detections over the average number of ground-truth and computed detections;
- **IDP**: global min–cost precision;
- **IDR**: global min–cost recall;
- **GT**: number of ground–truth objects;
- **RCLL**: Ratio of correct detections to the total number of GT boxes;
- **PRCN**: Ratio of TP/(TP+FP). Where TP–True Positive, FP–False Positive;
- **MT**: number of mostly tracked trajectories. I.e., the target has the same label for at least 80% of its life span;
- **PT**: partially tracked ground–truth trajectory;
- **ML**: number of mostly lost trajectories. i.e., the target is not tracked for at least 20% of its life span;
- **FP**: number of false detections;
- **FN**: number of missed detections;
- **IDs**: number of times an ID switches to a different previously tracked object;
- **Frag**: number of fragmentations where a track is interrupted by miss detection;
- **FM**: the number of track fragmentations. Counts how many times a ground–truth trajectory is interrupted;
- **MOTA**: Multi–object tracking accuracy. Summarize three errors sources (FN, FP, IDS) with a single performance measure (GT);
- **MOTP**: Multi–object tracking precision. The average dissimilarity between all true positives and their corresponding ground–truth targets;
- **iDt, iDa, iDm**: True positive ids.

Since the proposed tracking algorithm finds optimal matching, the Hungarian algorithm shows better results in object tracking, with more false positives (FP) and fewer false negatives (FN). Additionally, the Mostly tracked parameter (MT) is higher (361 *vs*. 329).

In general, the tracker using the Hungarian algorithm performs slightly worse when the detector has a lot of false positives and slightly better when the detector does not have that many false positives. However, the Hungarian algorithm should be slightly better when using YOLOv4 detections, as the number of false positives tends to be relatively small.

Additionally, the IOU smoothing filter limit was decreased from default 0.05 to 0.2 to increase the overlap and decrease the number of double detections.

6 Results for Ants Trail Recognition and Tracking

The set of tests was executed on the iOS mobile device. The tests include: ants tracking and unique identifier assignment; Ants object path creation; Ant's entities custom boundary lines crossing. The results are the follows (Figs. 10–14).

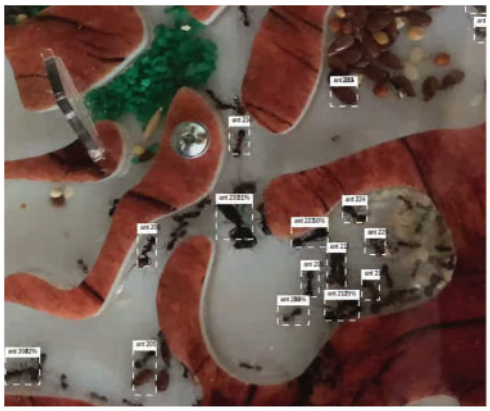

Figure 10: The Recognition and Tracking results of ants in formicarium and unique object assignment

Figure 11: The Recognition and Pathfinding for each ant. On the left–with Hungarian matching algorithm, on the right–with k-d tree matching algorithm

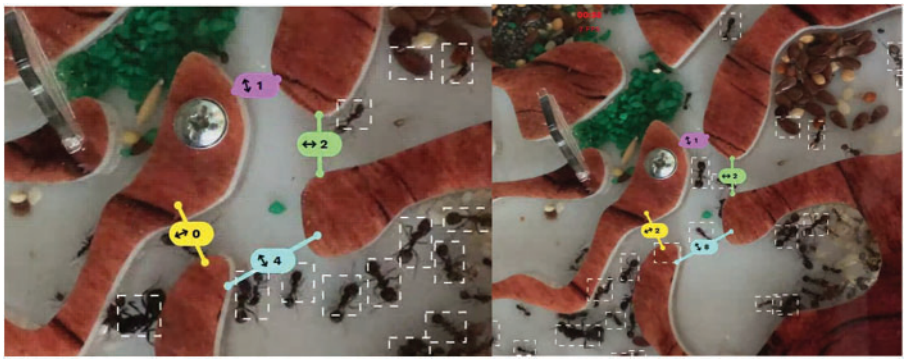

Figure 12: The Recognition and Counting ant in different nest directions. On the left is applying the Hungarian matching algorithm, and on the right is applying the k-d tree matching algorithm. The BAT parameter is set to 45 for both scenarios

Figure 13: The recognition and counting ant in different nest directions. On the left is applying the Hungarian matching algorithm, and on the right is applying the k-d tree matching algorithm. The BAT parameter is set to 0 for both scenarios

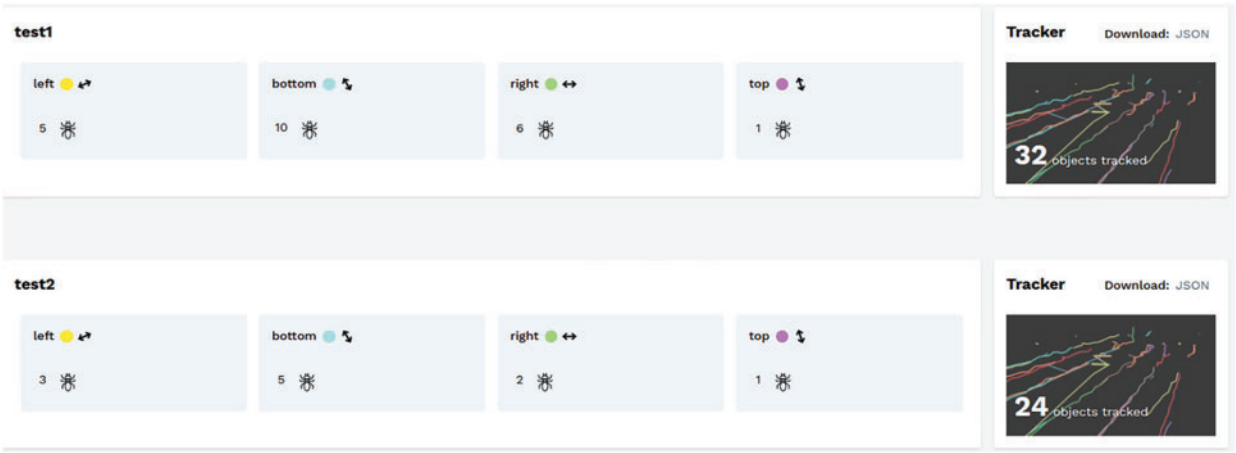

Figure 14: The results are stored in the database for each test for each boundary line

7 Results Analysis

The results of the test execution are provided in Tab. 1.

Table 1: Test execution with created YOLO_AR model for checking the performance of Hungarian/k-d tree matching algorithms

No. of runs	Matching algorithm	BAT	PFTP	Objects crossed boundary lines	Recognized objects
Run–1	Hungarian	1	1	22	32
Run–2	k-d tree	1	1	11	24
Run–3	k-d tree	45	45	0	27
Run–4	Hungarian	45	45	4	23

(Continued)

Table 1: Continued

No. of runs	Matching algorithm	BAT	PFTP	Objects crossed boundary lines	Recognized objects
Run–5	k-d tree	20	60	5	28
Run–6	Hungarian	20	60	9	29
Run–7	k-d tree	0	0	0	21
Run–8	Hungarian	0	0	7	24

Some parameters were stable during the test:

- *Model name*: YOLO_AR 512x512;
- *Test duration*: 1 min. The video fragment is the same for each test;
- *Approximate FPS (Frames Per Second)*: 12–16. FPL was set to 18 to keep high number of True Positives (TP);
- *Ant's average size*: 8mm;
- *IOU smoothing filter*: 0.1;
- *RFSF:* such formula was applied for input screen resolution 2532 × 1170;
- *CTMF*: 90.

The tests were executed on one video recording where ants with an average size equal to 8 mm move through the nest. The overall recognized objects count is the total number of unique IDs assigned to the ants. Some ids were reassigned during the test since the ant trajectory was lost when it did not move on the white surface. The results also show a more extensive trajectory loss (this can be observed in Fig. 11) in the case of the K–d tree algorithm. In addition, the count of tracked ants that crossed the boundary line is also lower compared to Hungarian matching algorithm integration. The tracking results for the Hungarian algorithm shown on Tab. 1 depict an accuracy increase of 50%. If BAT and PFTP filters are disabled, the trajectory loss will increase by 40%. This is observable in the case of high FPS; however, the negative results may be adjusted by the FPL filter. The ant's recognition results are worse in case the ant is located on the nest wall. Also, ants can hardly be recognized if they are not on a white surface. Such results are expected, as the current train data contain only ants filmed in the laboratory environment. Overall, real-time tests showed promising results in recognizing, tracking, and counting ants with the size up to 14 mm. The proposed CNN model YOLO_AR showed efficient results in recognizing such ant species.

8 Conclusions

The paper introduced a scalable system for recognizing and tracking small movable objects in sizes from 8 to 14 mm.

Analyzing the results of the research, I can say that:

- The Hungarian algorithm is overcoming k-d tree algorithm in matching small movable objects with the YOLO_AR model because of the lack of false positive values. The actual results were confirmed on MOT benchmark mock data. The obtained results on the real environment showed a 50% accuracy increase comparting to the existing method;

– The implemented BAT and PFTP filters showed an increase in tracking accuracy in real-time, mainly when the video–stream contains high FPS. Without the implemented filters, the loss for tracked paths is higher by 40%;
– The OpenDataCam system can be easily extended to fulfill research needs. Also, additional system modules may be added or enchanted. For example, I modified the Tracker module in our particular case and added an autonomous Model Train Service.
– The proposed system can be used for ants in sizes from 8 to 14 mm behavioral investigation. In addition, it can be scaled to recognize other classes of ant species or breeds.

The additional results of the research are the follows:

– The Messor Structor ant's dataset was created from 2 video streams via the FFmpeg tool. Afterward, the LabelImg image framework was used to annotate the obtained dataset. Finally, the received data was used to train the YOLO_AR 512 × 512 model with the Mish activation function;
– Additional smoothing and minimization filters were used to remove image overlapping and increase object recognition results;
– The train service was created to train the CNN model autonomously and forward the training output to the subsequent system nodes.

The obtained results can be used to design and implement scalable systems for continuously recognizing, tracking, and counting custom counts of object classes.

9 Prospects for Future Research

The key results of the research could be extended in two main directions. The first is a complete iOS mobile platform integration without using an external server. To achieve this Core Machine Learning (CoreML) framework [26] may be used to recognize and track a small movable object. The second one is to adjust the Neural Network model to recognize different ant species castes. So, the system may track and count different types of ants of different sizes and behaviors. The sizes of ants now will be crucial. For example, the size of solder is up to 10mm, the size of workers is from 4 to 9.5 mm, and the size of the queen is up to 14 mm.

Funding Statement: The author received no specific funding for this study.

Conflicts of Interest: The author declares that they have has no conflicts of interest to report regarding the present study.

References

[1] B. Groß, M. Kreutzer and T. Durand, "OpenDataCam 3.0.2: An open source tool to quantify the world," *Move Lab*, 2021, https://www.move-lab.com/project/opendatacam.
[2] S. Valladares, M. Toscano, R. Tufiño, P. Morillo and D. Vallejo-Huanga, "Performance evaluation of the Nvidia Jetson Nano through a real-time machine learning application," in *Advances in Intelligent Systems and Computing*, Palermo, Italy, pp. 343–349, 2021.
[3] H. N. Abdulghafoor and H. N. Abdullah, "Real-time moving objects detection and tracking using Deep-Stream technology," *Journal of Engineering Science and Technology*, vol. 16, no. 1, pp. 533–545, 2021.
[4] N. Pavych and V. Zahurskii, "Software architecture for analyzing the impact of news on the stock market," in *2021 11th Int. Conf. on Advanced Computer Information Technologies (ACIT)*, Ternopil, Ukraine, pp. 613–617, 2021.

[5] X. R. Zhang, X. Chen, W. Sun and X. Z. He, "Vehicle re-identification model based on optimized DenseNet121 with joint loss," *Computers, Materials & Continua*, vol. 67, no. 3, pp. 3933–3948, 2021.
[6] W. Sun, L. Dai, X. R. Zhang, P. S. Chang and X. Z. He, "Real-time small object detection algorithm in UAV-based traffic monitoring," *Applied Intelligence*, vol. 92, no. 6, pp. 1–16, 2021.
[7] M. Wu, X. Cao and S. Guo, "Accurate detection and tracking of ants in indoor and outdoor environments," *BioRxiv*, vol. 2, no. 1, pp. 1–26, 2020.
[8] X. Cao, S. Guo, J. Lin, W. Zhang and M. Liao, "Online tracking of ants based on deep association metrics: Method, dataset and evaluation," *Pattern Recognition*, vol. 103, no. 11, pp. 1–25, 2020.
[9] D. Kushnir and Y. Paramud, "Model for real-time object searching and recognizing on mobile platform," in *2020 IEEE 15th Int. Conf. on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, Slavsk, Ukraine, pp. 127–130, 2020.
[10] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," ArXiv, 2018, https://arxiv.org/abs/1804.02767.
[11] C. Banerjee, T. Mukherjee and E. Pasiliao, "The multi-phase ReLU activation function," in *Proc. of the 2020 ACM Southeast Conf. (ACM SE '20)*, New York, NY, USA, Association for Computing Machinery, pp. 239–242, 2020.
[12] D. Misra, "Mish: A self-regularized non-monotonic activation function," ArXiv, 2019, https://arxiv.org/abs/1908.08681.
[13] A. Bochkovskiy, J. Redmon, S. Sinigardi, T. Hager, J. JaledMC *et al.,* "AlexeyAB/darknet:YOLOv4 (version yolov4)," *Zenodo*, 2021, https://doi.org/10.5281/zenodo.562267.
[14] W. Kin-Yiu, "Implementation of Scaled-YOLOv4 using PyTorch framework (v1.0.0)," *Zenodo*, 2021, https://doi.org/10.5281/zenodo.5534091.
[15] G. Jocher, A. Stoken, A. Chaurasia, J. Borovec, T. Xie *et al.,* "Ultralytics/YOLOv5:v6.0–YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support (v6.0)," *Zenodo*, 2021, https://doi.org/10.5281/zenodo.5563715.
[16] C. Wang, I. Yeh and H. M. Liao, "You only learn one representation: Unified network for multiple tasks," ArXiv, 2021, https://arxiv.org/abs/2105.04206.
[17] D. T. Zutalin, "LabelImg," *GitHub*, 2021, https://github.com/tzutalin/labelImg.
[18] E. Bochinski, T. Senst and T. Sikora, "Extending IOU based multi-object tracking by visual information," in *2018 15th IEEE Int. Conf. on Advanced Video and Signal Based Surveillance (AVSS)*, Auckland, New Zealand, pp. 1–6, 2018.
[19] A. Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft, "Simple online and real-time tracking," in *2016 IEEE Int. Conf. on Image Processing (ICIP)*, Phoenix, Arizona, pp. 3464–3468, 2016.
[20] A. Milan, L. Leal-Taixe, I. Reid, S. Roth and K. Schindler, "MOT16: A benchmark for multi-object tracking," ArXiv, 2016, https://arxiv.org/abs/1603.00831.
[21] P. Dendorfer, A. Ošep and A. Milan, "MOTChallenge: A benchmark for single-camera multiple target tracking," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 845–881, 2021.
[22] H. Kuhn, "The hungarian method for the assignment problem," *50 Years of Integer Programming*, 2010, https://www.semanticscholar.org/paper/The-Hungarian-Method-for-the-Assignment-Problem-Kuhn/b6a0f30260302a2001da9999096cfdd89bc1f7fb.
[23] M. Skrodzki, "The k-d tree data structure and a proof for neighborhood computation in expected logarithmic time," ArXiv, 2019, https://arxiv.org/abs/1603.00831.
[24] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, no. 239, pp. 1–2, 2015, https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.
[25] D. Kushnir, "Ants dataset (indoor/outdoor Messor Structor) + trained YOLOv4 weights," *Mendeley Data*, 2022, https://doi.org/10.17632/zprk7wkf9j.1.
[26] A. Mishra, Machine learning for iOS developers. New York, NY, USA: John Wiley & Sons, 2020. [Online]. Available: https://www.wiley.com/en-ie/Machine+Learning+for+iOS+Developers-p-9781119602903.