Embedded Self Organizing Systems (Vol 9. No 3. 2022) (pp.73-81)

Embedded Selforganizing Systems

Special Issue Topic: "International Symposium on Computer Science and Educational Technology"

Tree Detection and Localization Approach for UAV-based Forest Inspection

Batbayar Battseren
Chemnitz University of Technology
Chemnitz, Germany
batbayar.battseren@informatik.tu-chemnitz.de

Mohamed Salim Harras
Chemnitz University of Technology
Chemnitz, Germany
mohamed-salim.harras@informatik.tu-chemnitz.de

Diego Alejandro Orjuela Aguirre
Chemnitz University of Technology
Chemnitz, Germany
batbayar.battseren@informatik.tu-chemnitz.de

Shadi Saleh
Chemnitz University of Technology
Chemnitz, Germany
shadi.saleh@informatik.tu-chemnitz.de

Wolfram Hardt
Chemnitz University of Technology
Chemnitz, Germany
wolfram.hardt@informatik.tu-chemnitz.de

***Abstract*—In the last years, the insect infection is causing a massive impact on deforestation in Germany. The authorities attempt to fight back this rapidly spreading problem throughout the usage of conventional methods. These methods require forest patrolling routines in order to identify the infected trees. Due to the size and density of the forest, it is a time-consuming and labor-intensive process that may result in an uninspected area. On the other hand, UAV-based inspections can cover large areas in a relatively short amount of time, and infected trees can be identified using aerial imagery. This paper addresses a remote sensing-based forest inspection solution for detecting and localizing the infected trees. The proposed approach is divided into two main phases: tree detection and tree localization. A deep learning-based approach is used for the tree detection. Due to the data scarcity, an active learning method is employed to train the AI model. The localization step maps the infected trees from the video frame to the real-world using the detection result and the UAV geodata.**

Keywords—remote sensing, forest inspection, bark beetle infection, object detection, geo-localization, GPS coordinate, UAV-based inspection

I. INTRODUCTION

Machine learning-based technologies have recently started to be used in agriculture to detect diseases and pests. Solutions like the one presented by Bagheri et al. opt to detect fire blight in infected pear trees [1], or even plant diseases identification using the trees leaves [2]. Plenty of these solutions have focused on solving agriculture problems [3] and multiple have made use of Convolutional Neural Networks (CNNs) as a base of their architecture.

According to the Federal Statistics Office of Germany, the amount of wood that has been damaged by insects, namely the bark beetle, is rapidly increasing (Fig. 1). It can be noticed that bark beetles are responsible of 60.1 million cubic meters of damaged wood. In 2020, this accounts to 72% of the total amount of damaged wood in Germany [4]. Forestry services have encountered difficulties in finding dying and dead trees due to the large forest areas and limited accessibility. Normally, this task requires human intervention and is carried out manually. Foresters walk through the forests, inspect the trees, note the geolocation, and state of the suspicious trees. This allows them later they to perform the appropriate actions based on the health of the suspected trees. In case of a bark beetle infestation, the infected trees have to be cut down immediately in order to stop the contamination. This is why it is crucial to detect the infected trees at early stage of infection.

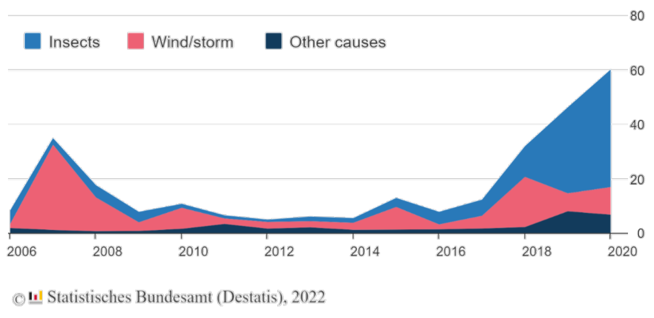


Fig. 1 Timber logged because of damage, by cause [4]

Forest inspection can be carried out with help of Unmanned Aerial Vehicles (UAV). However, the collection of the aerial image is only one part of the task. The challenge remains in the inspection of this data and the accurate localization of the detected trees.

It is possible to develop UAV-based fully autonomous systems, capable of performing the autonomous flight, data collection, and data processing for inspection routines [5]. This paper focuses on the aspect of data processing for the forest inspection routine, namely the detection and the geo-localization of the infected trees.

II. CONCEPT

A. Research questions

To tackle this inspection routine, few points have to be considered. First, a forest inspection routine must be achieved for the data collection. Second, an optimal image processing algorithm must be chosen for the tree detection. Finally, a geo localization algorithm should take care of finding the GPS coordinates of the detected objects. Therefore, these aspects have been formulated into research questions (RQ).

RQ1: What data is needed for detecting the infected trees? Distinguishing between the dead and the dying trees? And finally localizing them?

RQ2: What technology can satisfy the detection of bark beetle infection from aerial imagery?

RQ3: How to map the detected object frame coordinates to their relevant geolocation?

As a requirement from the forest services, the proposed approach is expected to provide an accurate GPS location for the suspected trees. The latter have to be classified based on the level of infection (dead or dying).

B. Proposed solution

Following are three steps that address the above-mentioned research questions:

First, as questioned in the RQ1, several data are required to achieve all subtasks of this system. The drone flying over the forest records the forest with an onboard downward-pointing camera as shown in Fig. 2. This data is used for the detection and classification between dead and dying trees. During this flight, the drone also logs its position data in a fixed frequency, allowing to the track the movement of the drone and to localize the trees.

Second, several approaches can tackle the RQ2. It is proven that Deep Learning (DL) approaches are more robust, when dealing with detecting objects under changing environments. Therefore, a CNN-based solution is used to provide the detection and classification of dying and dead trees in a video [6]. Furthermore, the dataset will be created entirely from scratch using videos from inspections of various forests in East-Germany.

Third, as asked in the RQ3, the geolocation (longitude, latitude) will be calculated based on the detected object coordinates on the video frame and drone flight coordinate information. Aside from those, only the technical specifications of the camera are required for the calculation. Finally, all calculated tree coordinates will be clustered to obtain the definitive coordinates of each detected tree.

In general, the proposed system takes aerial video and drone flight coordinates as input and returns the detected tree geo-coordinates (GPS coordinates) as output. In order to keep the system architecture as modular as possible, all intermediate outputs will be stored as files.

C. Inspection routine

The UAV-based forest inspection process consists of two main steps: data collection and data processing. The data collection sub-process starts with a flight planning. Inspection coverage is determined by the copter flight time and range. In Central Europe, the average tree height is around 30 m [7] the accepted maximum drone flight altitude and in Germany is 100 m. Therefore, the flight is performed across the selected area at 50 to 100 m above ground level altitude, with 70% overlapping view. An appropriate flight path has to be used in order to scan the area entirely. In this study, "Survey" is used as a flight mode, where the paths are parallel to each other across the selected field and the distance will be defined by the overlapping and flight altitude. Fig. 4 illustrates a basic flight path with key waypoints.

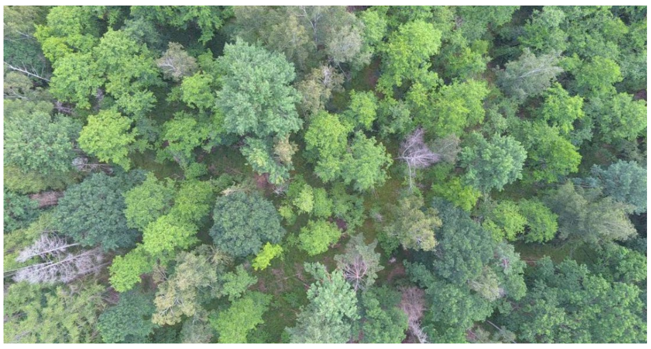


Fig. 2 Frame taken from a video used as data input.

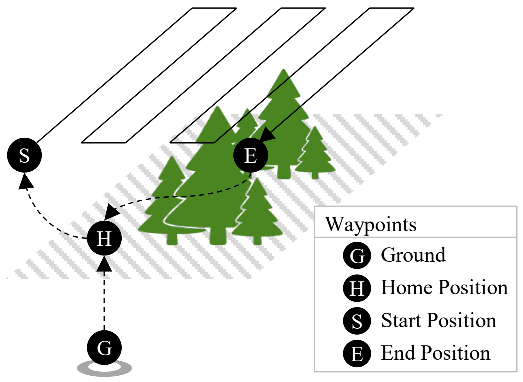


Fig. 4 Forest inspection flight, key waypoints and path

Due to the fact that forested areas are commonly not flat, a "Terrain Follow" flight mode is used to avoid the potential crashes, caused by changing ground level altitudes. This flight mode maintains the drone altitude relative to the ground level. Another benefit of this mode is the recorded video will have a consistent field of view. Yuneec H520 drone is used in the forest inspection. This drone has a built-in "Survey" flight mode and "Terrain Follow" functionality. For the data processing step, video recording from waypoint *S* to waypoint *E* will be used (Fig. 4).

III. METHODOLOGY

The system architecture is mainly divided into two sections: Tree Detection and Tree Localization (Fig. 3). Tree Detection part is responsible for detecting the dead and dying trees. It takes a video as input and saves the detection result as an output file. The Tree Localization part receives the detection file as input, as well as other required information to calculate the geolocation of the detected trees. These two parts will be explained in detail in following sub-chapters.

A. Tree Detection

1) Dataset preparation

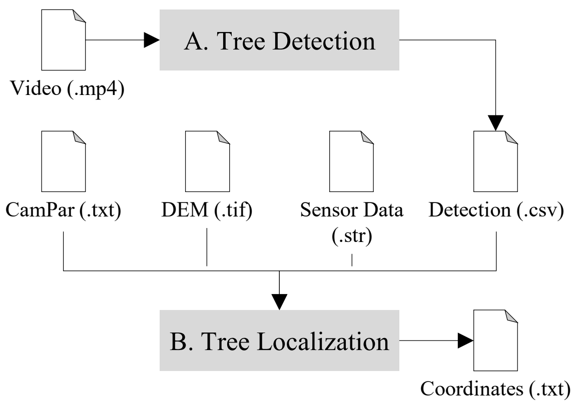


Fig. 3. Context diagram of the developed system.

In the last years, the computer vision field saw the emergence of a new research trend focusing on optimizing the quality of datasets for the model training. In fact, it is proven that the dataset quality reflects on the model performance. This is why, special attention has been given in this paper to the dataset preparation.

The selected DL model was trained with a generated and labelled dataset. A total of 401 frames, with a 4K resolution of 3840×2160, were taken from forest inspection videos captured during the UAV flights. These frames were subject to an annotation in order to label the different trees into two main object classes: dying and dead. Since the chosen model is based on the YOLO architecture, it requires a normalized annotation for the different bounding boxes.

In order to distinguish the level of infection in the trees, the color feature is considered such as that the dying trees are characterized with a white crown top with yellowish leaves. On the other hand, the dead trees are also characterized by a white crown top with no leaves at all. Fig. 5 showcases a sample from the dataset, where a dying tree (yellow) and few dead trees (purple) are visualized within the bounding boxes.

To increase the number of images within the dataset and also diversify the features representing the dying trees, the current approach makes usage of data augmentation techniques.

Since the YOLO models accept frames with a resolution of multiples of 32, a Python script was developed to resize the images from their original size to a lower resolution of 448×448, adding black padding on top and under the image to keep the original image ratio. Additionally, the frames are flipped horizontally and vertically, resulting in an augmented dataset of 1552 labelled images, as depicted in Fig. 6(b).

As a final step, the dataset was then divided into a training dataset, with 1076 images (70% of the whole dataset); validation dataset, with 164 images (10% of the entire dataset); and a test dataset, with 312 images (20% of the entire dataset). In addition, the training dataset is further ordered in terms of quality into negative, unsure, and positive. To achieve this, an expert manual evaluation is done to distribute the frames into the three pools. This is primordial for the active learning approach used during the training. The details surrounding the latter distribution are discussed in the next section.

2) *Active learning approach*

Since most DL solutions are based on a passive approach for the training of the model, it becomes important to invest a considerable amount of time on preparing large datasets. In the case of small datasets, it is difficult to achieve satisfactory performances. However, active learning techniques, or also called query learning techniques, stand out to solve the issue of data scarcity [8]. These methods are pool-based approaches, where the dataset is organized in terms of pools and, iteratively, samples of the pools are chosen following a query strategy.

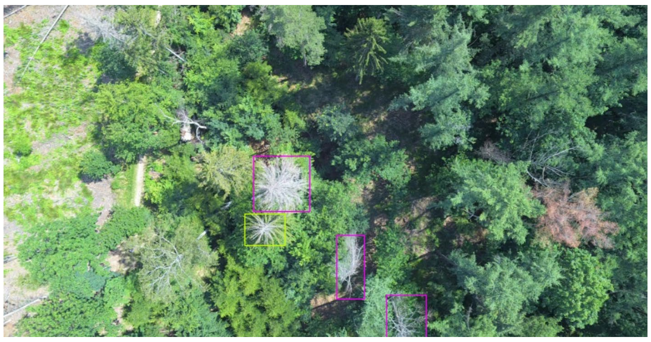


Fig. 5. Example of labelled dying and dead trees

In the scope of this paper, a query pool-based batch (QPBB) learning method was used to compare the model's performance against passive learning. The model was trained with a number of images that were increased iteratively. This means that the model was trained during every iteration with a controlled number of frames (batches). After every iteration, the Mean Average Precision (mAP) of the model is evaluated again to check if any optimization has taken place. The selection of the frames happens based on a query strategy, which defines which instances are most informative for the training [9].

The main purpose of the query approach is to test the minimum number of images necessary to achieve a similar precision result compared to using the entire training dataset. Therefore, the query strategy adopted was based on the identifiability of the objects. The training dataset samples (Fig. 7(a)) were augmented differently, depending on how easily recognizable the objects should be. The resulting training dataset was divided randomly into three separate pools. 35% of the training samples were placed in a pool called "Positive", 45% called "Unsure" and 20% called "Negative". The idea was to have clearly identifiable objects in the images for the positive pool (Fig. 7(b)) and hardly identifiable objects in the images for the negative pool (Fig. 7(d)). The frames in the unsure pool are characterized by having identifiable objects but were not as easily identifiable as the ones in the positive pool (Fig. 7 (c)).

To force a high, medium and low score from the "Positive", "Unsure" and "Negative" evaluation of the images respectively when feeding the prediction model, different filters were applied to each folder using a developed Python script using the library "imgaug" [10]. The following filters were applied:

- Gamma Contrast and Multiply Saturation to the images of the Positive folder.
- Laplacian Noise with a value of 0.06 to images in the Unsure folder.

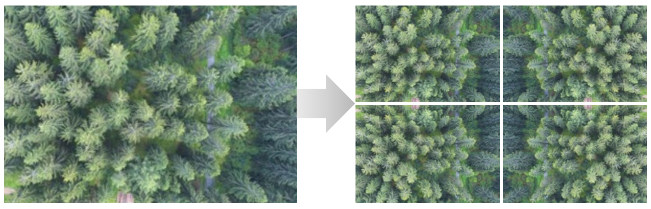

(a) Original (b) Augmented

Fig. 6. Data set augmentation. (a) Original image. (b) Augmented images.

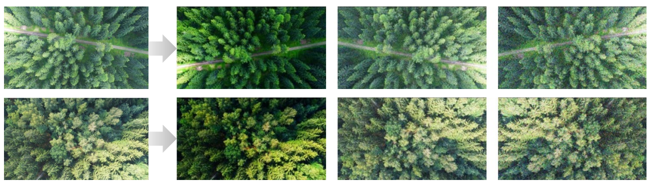

(a) Original (b) Positive (c) Unsure (d) Negative

Fig. 7. Dataset for Query pool-based batch learning

- Laplacian Noise with a value of 0.12 to images in the Negative folder.

The application of the Gamma Contrast filter brightens the image highlights whilst darkening shadows by scaling the image pixel values. Meanwhile, Multiply Saturation increases the difference between colors by transforming the image to HSV color space and increasing the H channel value, to later transforming it back to RGB. This makes it easier to perceive object boundaries, increasing the chances of detecting the white tree tops. Laplacian Noise add very high and low values sampled from Laplace distributions to each pixel, making it harder for the detector to determine the object's presence in a region of the image. The more outliers, the more difficult it is to find it.

Additionally, the adopted approach was developed to facilitate for each iteration the moving of images between the pools and the final training dataset. It makes use of a JSON file which acts as a log file or catalogue, allowing to track for each image its initial folder, if it has been moved or not, and the location of its labels file. This solution was based on three function calls:

- "Replicate()": Move images from the Pool folders to the Training folder as specified in the catalogue.
- "Increment(int n)": Move *n*% of images from the Pool folders to the training folder. The images to be relocated will be the first images found in the catalogue that have not yet been moved. The percentage is calculated from the total of the remaining images in the Pool folders.
- "Summary()": Print catalogue counters as summary. This means, how many images are left in each of the Pool folders and how many images from which Pool folders have already been moved to the Training dataset folder.

In what concerns the *n* percentage value for the increment function, we opted for 15% in the first iteration. This means, moving 15% of the images from each of the folders in the pool dataset to the training folder. Afterwards, increments of 4% of the pool dataset were used on each iteration. In every iteration, the training of the model takes place and an evaluation of the precision and loss function against the previous iteration is performed. It is important to note that the 4% of the images to be added to the training dataset, in each iteration, is calculated from the remaining images in the Pool folders, and not from the totality of the initial training dataset images. These results are also compared to the results of the passive learning approach. Fig. 8 illustrates the active learning approach used in the scope of this thesis. The training continues until the results stop changing between consecutive iterations.

3) *Deep Learning Model*

The selection of the right DL model for the detection of a specific object is dependent on several factors. As a matter of fact, it is necessary to take into account the quality-efficiency tradeoff [11]. Because the presented solution was targeted for running on an embedded device, the adopted approach opted for prioritizing the efficiency factor, or in other terms the speed of the model. Since most of two stage detectors are known for their slow speeds [12], we decided to choose the single stage detector from the YOLO family.

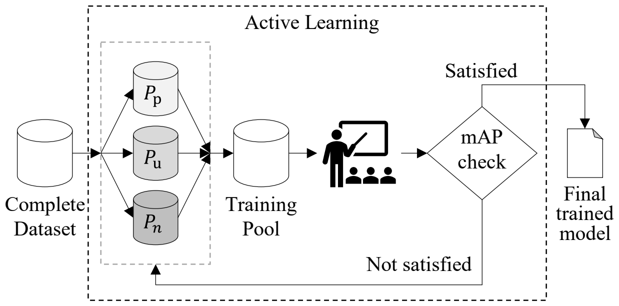


Fig. 8. Query pool-based batch learning method

According to Fig. 9, which was published by Wang et al. [6], the YOLO model in its fourth version demonstrated top performance compared to other two-stage detectors. The main difference between the normal YOLOv4 version and its derivative Scaled YOLOv4 consists in the size of the model. The second is a scaled version that maintains the higher speed of detection while providing higher accuracy than the first. Based on the aforementioned points, the decision to choose Scaled YOLOv4 over YOLOv4 was supported by the desire to have a lightweight model capable of running on the embedded platform. As of the time of this research, Scaled YOLOv4 [6] and YOLOv5 [13] were the latest models, but new versions of the YOLO family have emerged this year (YOLOv6 [14] and YOLOv7 [15]) with the ability to reach higher accuracies and faster time performances.

Scaled YOLOv4 [6] comes in three different branches which are: CSP-ized, tiny, and large. In fact, CSP stands for Cross-Stage-Partial and refers to the CSPNet strategy, which is used in the backbone of Scaled-YOLOv4 [6]. This strategy aims at splitting the feature map of the base layer and merging the two parts through cross-stage hierarchy. The Path Aggregation Network (PAN) [17] makes use of the CSPNet strategy [6] to reduce up to 40% of the calculations performed at the level of the neck in YOLOv4 [16]. In addition, this version of YOLO has CSPDarknet53 as the backbone with bottleneck ratio 1, width growth ratio 2 and depth of 65. While the large version is dedicated to cloud solutions, the tiny version and the CSP-ized version are both suitable for low-end GPU devices, making it suitable for the future scope of this project.

Scaled-YOLOv4 takes an image as an input and uses the backbone CNN to compress its features, then to draw the bounding boxes. These features are subject to image classification, where the feature layers from the backbone are

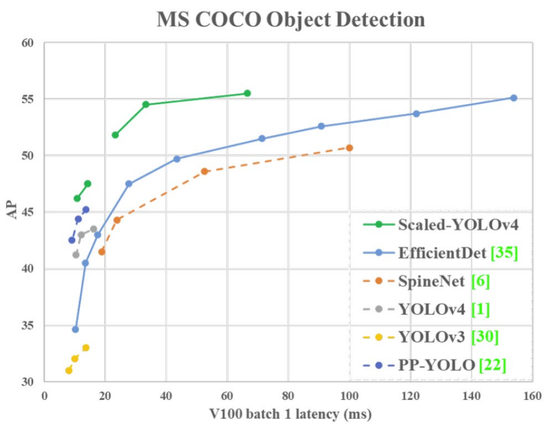
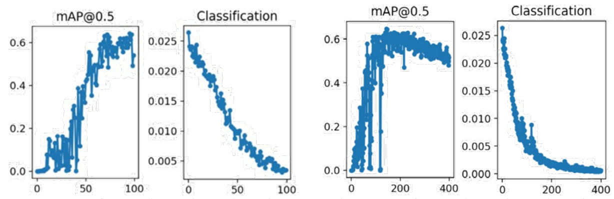


Fig. 9. Comparison of object detectors by Wang et al. [6]

connected to each other in the Model's Neck. Feature layers are mixed and visible to other layers to deal with the vanishing gradient problem, boost feature reuse in the network, diminish the network number of parameters, and to help feature propagation. To combine the features obtained from the backbone, Path Aggregation network (PAN) is used as a Neck in addition to CSP, which later is used by the Head to detect the objects. The Head is in reality three different heads based on the architecture of Feature Pyramid Network (FPN), that grant the detection of objects in different dimensions. This is due to the difference in context size and resolution of the input from the three detection blocks [18].

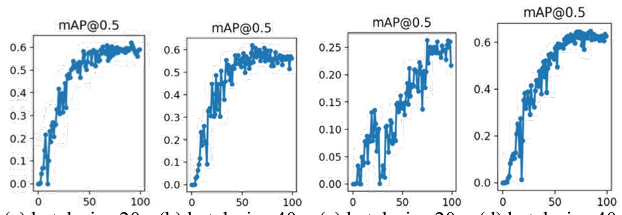
The hardware used for training was a Tesla T4 GPU. To find the optimal hyper parameter configuration, the model was trained using passive or conventional learning. Initially, the model was trained for 100 and 400 epochs. These values were chosen arbitrarily. The mAP at 0.5 (mAP@0.5) curve shown in the Fig. 10(a) and (b) that training takes only 100 epochs to reach its mAP@0.5 maxima. Furthermore, it can be seen that the training with longer epochs reflects in a quick drop of the classification error. This means that the use of 100 epochs for the training provides the similar results to using 400 epochs.

The model training was, as well, tested using a batch size of 20 and 40 with two datasets. On one hand the first dataset version comes with only padding and flipping as augmentation techniques. On the other hand, the second dataset version uses the query pool-based distribution for active learning. As seen in Fig. 11, although no difference is to be noticed for the training with the first dataset version. The results of the training showed a substantial drop in mAP@0.5 when using a batch size of 20 (Fig. 11(c)), with QPBB strategy, compared to using a batch size of 40 with the same dataset (Fig. 11(d)). Therefore, a batch size of 40 was used for training the model. A higher number of batches was not tested due to hardware restrictions, where the system was already using more than 15GB of RAM when training with a batch number of 40 whilst only having 16GB of RAM available for computations. Any other hyper parameter was left as default by the developers of Scaled YOLOv4 [6] CSP branch.



(a) mAP@0.5, loss function for 100 epochs (b) mAP@0.5, loss function for 400 epochs.

Fig. 10. Comparison between training the model with 100 (a) and 400 epochs (b)



(a) batch size 20 Passive Learning (b) batch size 40 Passive Learning (c) batch size 20 Active Learning (d) batch size 40 Active Learning

Fig. 11. Comparison of passive and active learning with respect to the batch size

To identify the impact of the number of batches and filters used in the training dataset, the model was trained with all variations and the results compared (TABLE I). In the table, Filter 1 refers to using contrast and Saturation filters (35% of the Dataset), Filter 2 refers to the Laplacian Noise (0.06) (45% of the Dataset) and Filter 3 refers to the Laplacian Noise (0.12) (20% of the Dataset). The obtained results demonstrated that the system performed its best at 40 batches with all image filters applied (all pools used for active learning).

TABLE I. INFLUENCE OF FILTERS AND BATCH SIZE IN TRAINING

Filter 1	Filter 2	Filter 3	Batches	Precision	Recall	mAP @0.5	mAP @0.5-0.95
Yes	Yes	Yes	20	21.0%	23.7%	21.7%	8.57%
Yes	Yes	No	20	42.9%	71.9%	61.8%	26.0%
Yes	No	No	20	43.7%	70.1%	59.4%	23.4%
No	No	No	20	43.3%	68.6%	59.0%	22.8%
Yes	Yes	Yes	40	44.7%	73.7%	62.7%	24.4%
Yes	Yes	No	40	45.2%	69.6%	61.7%	22.9%
Yes	No	No	40	44.5%	68.1%	57.7%	21.2%
No	No	No	40	45.2%	68.3%	58.4%	20.9%

B. *Tree Localization*

The main concept of tree localization is based on current position of the drone (Fig. 12(b), *O´*), which is also the center of the frame (Fig. 12(a), *O*). The tree detection is stored as bounding box (*x, y, width, height*) in a CSV file. Each line represents single frame of a video. For the localization, the center of the bounding box (Fig. 12(a), *T*) is used.

It is represented in Cartesian coordinates (x_T, y_T), that must be converted first into an angle α and a distance d (Fig. 12(a)). The angle α refers to the angular difference between the frame vertical axis and the ray, from the frame center to the bounding box center. The distance d is the distance from the frame center to the bounding box center in pixels. Equation (1) showcases how the angle α is calculated.

$$\alpha = \tan^{-1} {x_T}/{y_T} \tag{1}$$

A distance between two points (the drone, and the detected tree) can be calculated with the trigonometric function (2). Here, the height h is the drone height above the ground level, and the angle β is the angular difference between the rays from the camera position to the frame center, and to the detected tree (Fig. 12(a)).

$$d' = \tan\beta \cdot h \tag{2}$$

Angle β can be found based on the T coordinate, video frame width (v_{width}), height (v_{height}), and camera diagonal field of view (*dFoV*). The video width and height are in pixel values, and the diagonal field of view is in degrees.

$$\beta = \sqrt{x'^2 + y'^2}\left(\frac{dFoV}{\sqrt{v_{\text{width}}^2 + v_{\text{height}}^2}}\right) \tag{3}$$

After the angle α and the distance d are found, the only parameter left is the angle between the North and the tree position from the drone position. This can be calculated using the following equation (4) based on the drone yaw angle γ and the angle α in the frame.

$$\delta = \alpha - \gamma \tag{4}$$

A new GPS coordinate (latitude φ_2, longitude λ_2) can be defined by the following equations (5) and (6), which are based on the distance d' and the angle δ from the starting point (latitude φ_1, longitude λ_1).

$$\varphi_2 = \arcsin\left(\sin\varphi_1 \cos\frac{d'}{r} + \cos\varphi_1 \sin\frac{d'}{r}\cos\delta\right) \tag{5}$$

$$\lambda_2 = \lambda_1 + \operatorname{atan2}\left(\sin\delta \sin\frac{d'}{r}\cos\varphi_1, \cos\frac{d'}{r} - \sin\varphi_1 \sin\varphi_2\right) \tag{6}$$

The starting point refers to the drone position (φ_1, λ_1) and the calculated coordinate is the tree position (φ_2, λ_2).

As described above, the tree localization process aims at mapping the detected trees from the pixel context to the real world. In general, the pixel coordinate of the detected object on the frame and the GPS coordinate of the referred frame are required for the geo-localization process. However, other parameters are also required to achieve this.

The implementation consists of 4 main steps. All steps were developed as independent components. Each of them reads an input file and generates an output file with processed data. Finally, it returns a list of detections with their geographical coordinates as an output. Fig. 13 illustrates the overall dataflow diagram of tree localization process.

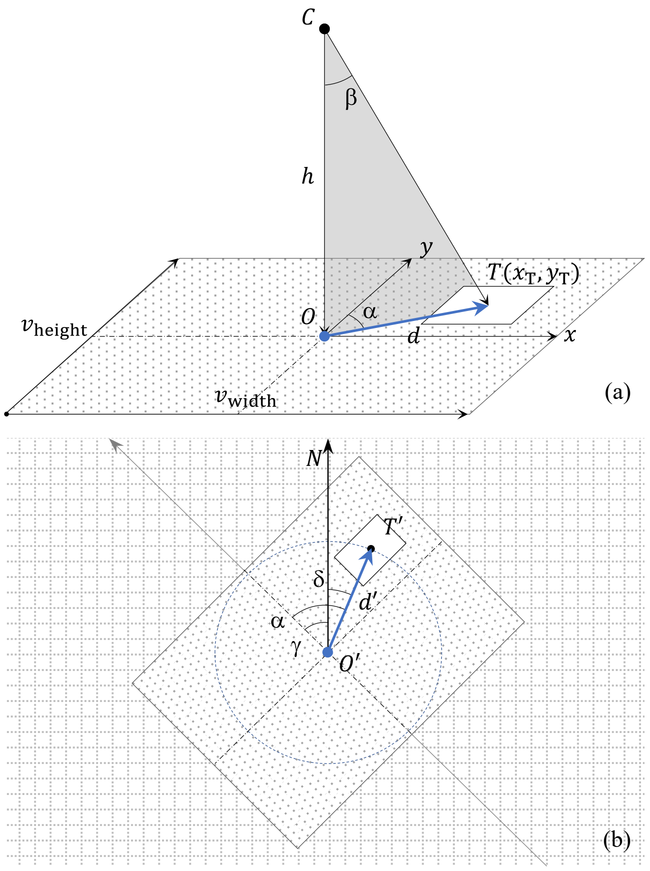


Fig. 12 Tree geo-localization calculation. (a) Frame level calculation, (b) Video frame mapping to real-world

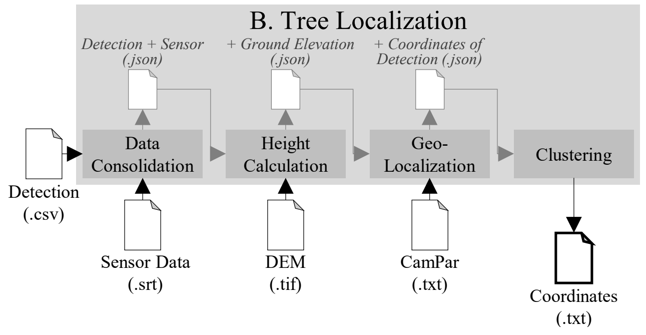


Fig. 13. Data flow diagram for the Object Localization step

1) *Data Consolidation*

Yuneec E90 camera is used for the data collection (for object detector training dataset) and forest inspection. The inspection video is recorded in 4K resolution with 30 fps. The recorded video also contains the geolocation of the flight path. However, this information is updated every 200 milliseconds. The first step, Data Consolidation addresses this aspect by linking the sensor data (GPS coordinates) to the detection data.

This step loads a subtitle SRT file containing the GPS coordinates, and an object detection CSV file containing the bounding boxes of the detected objects. Afterwards, it assigns the corresponding coordinates to the respected frames, according to the timestamps. As a result, a JSON file is generated.

2) *Height Calculation*

According to Equation (2), the drone height above the ground level is required to find the distance d' between the drone and the detected tree. However, the height information that have been collected from the drone correspond to the height above mean sea level. In order to find the height above ground level, the Digital Elevation Model (DEM) is used. The DEM is a graphical representation of terrain surface elevation, often rendered as an image file. The dataset used in our study is encoded as GeoTIFF format and represented in terms of raster tiles. Each pixel value of the TIFF image represents a ground height of 25 × 25 m area.

The second component loads the TIFF files in a given folder, and expects them as DEM raster tiles. It also loads the JSON file, given as an output of the previous step, and loops through all its objects. For every object in the file, it reads the ground altitude from the corresponding pixel in the corresponding tile, based on the GPS coordinate of the frame. In case of not finding a corresponding tile, or not finding the frame coordinates, the frame is then ignored. The obtained height values are appended to the JSON file.

3) *Geo-localization*

To geo-locate the bounding boxes, this step performs the proposed localization calculations (Equation (1) - (6)). It loads all the data structures containing the bounding boxes, sensor data and height above ground level. In addition, some camera parameters are required in Equation (3). These parameters are the frame width, height and camera diagonal field of view.

The implemented process loops over each bounding box and calculates the geo-location following the above steps:

1. Calculate the drone height with respect to the coordinate for each frame

2. Reverse the normalization of the bounding box centroid using the height and width of the image
3. Get the *x* and *y* Cartesian's coordinates of the centroid of the bounding box in the image with respect to the calculated Cartesian's image center
4. Calculate the angle (β) between the ray to center of the frame and ray to bounding box center using Equation (3)
5. Calculate the distance d' between drone and bounding box center in meters using Equation (2). The average height of the tree is given as an input argument to calculate the position correctly, because the DEM's height value includes the height of the tree
6. Calculate the angle α using Equation (1)
7. Calculate the angle δ to obtain the direction of the bounding box centroid with respect to the north
8. Calculate the bounding box GPS coordinates using Equation (5) and (6)

In order to guarantee accurate geo-localization, the camera needs to be pointing downwards. This was achieved by setting a threshold value on the Gimbal Pitch angle (-90±1 degrees of offset). The frames that are out of bounds are ignored.

4) Clustering

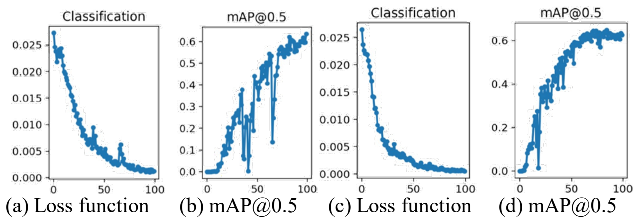
The last step reads the data structure from the latest outputted JSON file. Due to the sequential capture of frames, multiple detections of the same tree exist within the file. In order to generalize these detections, DBSCAN clustering algorithm is applied on the all calculated coordinates [19]. DBSCAN was selected as the clustering approach since it performs well for density group points.

IV. EVALUATION

A. Tree Detection

While the active learning strategy aims at reaching similar accuracies to the conventional learning strategies, it tries to achieve this by the usage of a minimal set of training data. Therefore, the proposed model was evaluated in relation to a conventional model. The evaluation was based on the mAP metric and the classification loss score. In fact, the mAP was calculated taking into account an Intersection over Union (IoU) threshold of 0.5. As shown in Fig. 14, the test scenarios differ in the number of training data used for the training.

While the conventional model used the full training dataset, the active learning strategy opted at using initially 15% of images from the pool dataset, and then incrementing it by 4% with every iteration. The following table (TABLE II) shows the results of training the model with a Tesla T4 GPU.



(a) Loss function with 56% of dataset (b) mAP@0.5 with 56% of dataset (c) Loss function with 100% of dataset (d) mAP@0.5 with 100% of dataset

Fig. 14. mAP@0.5 and Loss function when using 56% and 100% of the Pool dataset for training

TABLE II. COMPARISON OF TRAINING THE MODEL USING ACTIVE LEARNING

Used Dataset	Used Dataset [%]	Precision	Recall	mAP @0.5	mAP @0.5-0.95
161	15%	0.317	0.324	0.201	0.0633
197	18%	0.140	0.603	0.332	0.107
232	22%	0.120	0.730	0.363	0.112
266	25%	0.179	0.688	0.421	0.146
298	28%	0.232	0.622	0.421	0.138
329	31%	0.370	0.547	0.444	0.148
358	33%	0.34	0.536	0.430	0.164
387	36%	0.226	0.752	0.515	0.197
415	39%	0.251	0.679	0.448	0.147
441	41%	0.367	0.613	0.490	0.179
466	43%	0.353	0.659	0.537	0.202
491	46%	0.381	0.654	0.561	0.206
515	48%	0.298	0.730	0.553	0.197
537	50%	0.412	0.676	0.574	0.220
559	52%	0.468	0.623	0.555	0.199
579	54%	0.400	0.686	0.565	0.189
599	56%	0.293	0.790	0.635	0.231
619	58%	0.369	0.692	0.566	0.198
637	59%	0.428	0.639	0.571	0.213
1076	100%	0.440	0.737	0.627	0.244

The results from TABLE II show that after 17 iterations, using 56% of the original training dataset (599 images), the model can already show a mAP score of 63.5, compared to 62.7 from the conventional model (TABLE I). Furthermore, the loss curve from the active learning strategy shows similar results to the loss curve of the passive approach, concluding that the active learning approach is optimal for the scope of this project.

B. Tree Localization

According to the carried-out tests, the found coordinate accuracy is depending on the tree height and the drone height above ground level. From an aerial view, trees can be projected with different length to the side of the frame (shown as black points in Fig. 15). This means that the localized coordinates do not represent the tree root (Fig. 15, diamond marks). Instead, they represent the ground hidden behind the tree top (Fig. 15, round marks). Furthermore, the system is affected by different height of trees. In addition, the DEM tile represents a certain area with one average value; however, the actual ground level can vary a lot in that area. As a result, these aforementioned issues cause inaccurate localization.

For test purpose, different offset values were used for representing the tree height. Fig. 16 illustrates the object localization results with three different offset values. From these offsets, the value that caused the lowest object drifting was -15 (Fig. 16(b)). After comparing offset 0 with -30

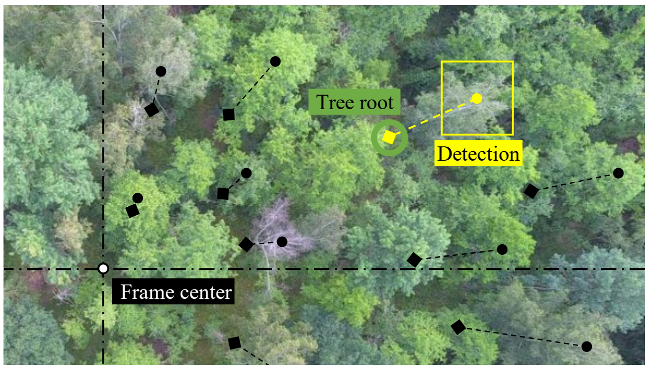


Fig. 15. Height error source of the tree detection

(Fig. 16(a) and (c)), it becomes clear that the bigger the offset is, the detections will drift further.

Finally, the calculated coordinates were clustered with the DBSCAN. It requires two values to cluster the given input, which are the epsilon and the minimum number of items. The minimum number of the items was arbitrarily set as 3, due to the nature of the problem, where if an object was detected in more than 3 frames, then will be considered as a detected dying or dead tree.

On the other hand, the epsilon (ε) value defines the distance between the neighbor items. To find the correct value, the found coordinates were clustered with different epsilon values and compared (TABLE III) against the ground truth. The test video had 44 trees to detect (ground truth), and in total 153 516 objects were detected. The epsilon values 0.1, 0.06 and 0.07 had the closest resulting group, while the value 0.7 returned only 3 clusters (too low) and the value 0.01 returned 319 clusters (too high). Based on the minimum number of items and distance values, some items were considered as outliers and discarded. Fig. 17 illustrates the clustering result in different colors under three different epsilon value. Each color represents individual clusters.

TABLE III. CLUSTERING WITH DIFFERENT EPSILON VALUES

Epsilon (ε)	Discarded	Clusters
0.70	0	3
0.50	1	7
0.30	3	13
0.10	24	45
0.07	39	66
0.06	53	84
0.05	71	133
0.02	287	218
0.01	859	319

Finally, the median values of each cluster's longitude and latitude are calculated to obtain a single coordinate representing each cluster. The result is illustrated in the Fig. 18.

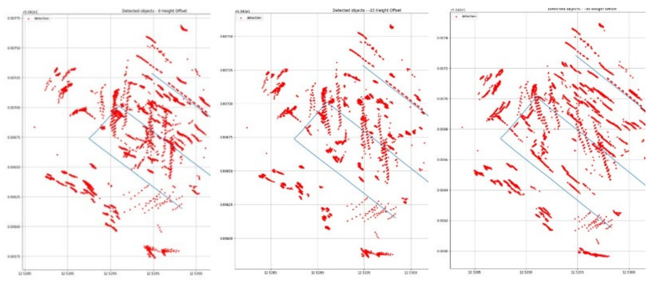

(a) Offset = 0 (b) Offset = -15 (c) Offset = -30

Fig. 16. Object geo-localization using different offset tree height

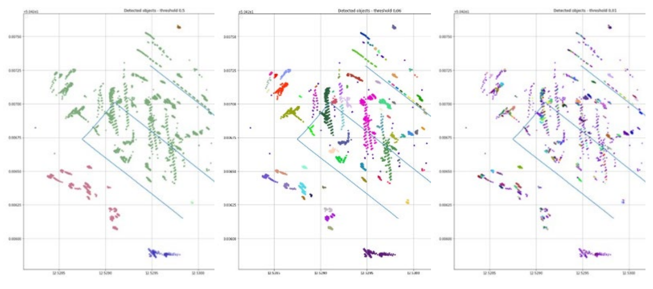

(a) $\varepsilon = 0.50$ (b) $\varepsilon = 0.06$ (c) $\varepsilon = 0.01$

Fig. 17. Clustered objects with different epsilon values

Based on the clustering comparison, the results from epsilon value 0.1, 0.07 and 0.06 were evaluated in TABLE IV. Result of $\varepsilon = 0.1$ had 44 clusters, but only 27 (61.36%) of them were correct. In other hand result from $\varepsilon = 0.07$ has 66 clusters, but 38 (86.36%) of them were correct. Finally, the result obtained from $\varepsilon = 0.06$ had 84 clusters, and 41 (93.18%) of them were correct.

The geo-localization average error rate varies slightly depending on the clustering. The minimum localization error was 0.93 meters, while maximum error was 13.89 meters. However, the overall average localization error was 4.59 meters.

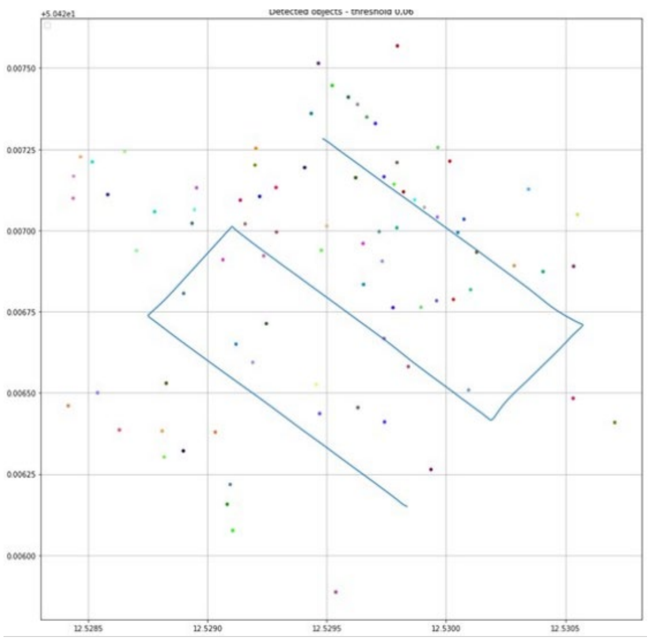


Fig. 18 Localization result, final coordinates (points) and the flight path (lines) at $\varepsilon = 0.06$

TABLE IV DETECTION AND LOCALIZATION RESULT COMPARISON BASED ON EPSILON VALUE

Epsilon (ε)	Clusters	True-Positive	True-Negative	False-Positive	Avr. Err. [m]	Det. Rate [%]
0.06	84	41	3	43	4.90	93.18
0.07	66	38	6	28	4.52	86.36
0.10	45	27	17	21	4.35	61.36

V. CONCLUSION AND FUTURE SCOPE

In order to decrease the insect caused deforestation, foresters must identify the bark beetle-infected trees immediately and cut them down before further contamination. However, it is a challenging task to identify them without technical support. UAV-based solution can ease the inspection process. It saves time, increases the coverage and could return accurate GPS coordinates of the suspicious trees.

The proposed solution uses DL-based object detector and geo-localization algorithm to find the GPS coordinates of the dying and deed trees from aerial video. This paper covers the data processing step of the UAV-based forest inspection, which consist of two sub-steps. Implemented system uses separate stand-alone components to process the data to keep the system modularity. Each script reads the input as a file and outputs the result as an output file.

Tree detection step applied the Scaled-YOLOv4 object detector. Training dataset has been prepared from real forest inspection which is carried out in East-Germany. Due to the limitations encountered during the collection of the dataset, the training strategy was based on an active learning method, which solved the data scarcity problem. The study achieved promising results in detecting infected trees with a small image dataset. In fact, the developed tree detector is a tentative approach and is still subject to optimization. As a result of the detection, the location of the detected trees in the frame are stored in a file and passed to the coming step.

The tree geo-localization step reads the detection output and returns the GPS coordinate list of all detected suspicious trees. This phase consists of 4 main steps, namely data consolidation, height calculation, geo-localization, and clustering.

To conclude the study, the proposed system is able to detect the dying and dead trees from aerial inspection and it is capable of geo-localizing them with an average error rate of 4.59 m. Forestry services would greatly benefit from this system since it is inspecting the forest from above with UAV, instead of directly patrolling through the forest by foot.

As it is presented in the Evaluation chapter, the localization error source must be addressed within the further study. Also, the dying tree detection model has to be trained further with different datasets from different place under different weather and light conditions.

REFERENCES

[1] N. Bagheri, "*Application of aerial remote sensing technology for detection of fire blight infected pear trees*" Computers and Electronics in Agriculture, vol. 168, p. 105147, 01 2020

[2] S. Sladojevic, M. Arsenovic, A. Anderla, and D. Stefanovi ́c, "*Deep neural networks based recognition of plant diseases by leaf image classification,*" Computational Intelligence and Neuroscience, vol. 2016, pp. 1–11, 06 2016

[3] A. Kamilaris and F. X. Prenafeta-Bold ́u, "*Deep learning in agriculture: A survey,*" Computers and Electronics in Agriculture, vol. 147, pp. 70-90,2018.[Online].Available:https://www.sciencedirect.com/science/article/pii/S0168169917308803

[4] Destatis, "*Forest damage: logging of timber damaged by insect infestation grew more than tenfold within five years*," Federal Statistical Office. https://www.destatis.de/EN/Press/2021/08/PE21_N050_41.html (accessed Nov. 22, 2022).

[5] U. Tudevdagva, B. Batbayar, W. Hardt, S. Blokzyl and M. Lippmann, "*UAV-based fully automated inspection system for high voltage transmission lines*," in Proc. 12th IFOST Conf., Ulsan, South Korea, Jun. 2017.

[6] C. Wang, A. Bochkovskiy, and H. M. Liao, "*Scaled-yolov4: Scaling cross stage partial network*," CoRR, vol. abs/2011.08036, 2020. [Online]. Available: https://arxiv.org/abs/2011.08036

[7] "*Bäume*," ForstBW. [Online]. Available: https://www.forstbw.de/wald-im-land/lebensraum/pflanzen/baeume/. [Accessed: 15-Nov-2022].

[8] S. Zhang, J. Yin, and W. Guo, "*Pool-based active learning with query construction*," in Foundations of Intelligent Systems, Y. Wang and T. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 13–22.

[9] B. Settles, "*Active learning: Active learning Survey*," San Rafael, UNITED STATES: Morgan & Claypool Publishers, 2012

[10] A. B. Jung, "*imgaug: Image augmentation for machine learning experiments*." GITHUB. https://github.com/aleju/imgaug [accessed 17-Oct-2021]

[11] R. Baeza-Yates and Z. Liaghat, "*Quality-efficiency trade-offs in machine learning for text processing*," in 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 897–904

[12] P. Soviany and R. T. Ionescu, "*Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction,*" 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2018, pp. 209-214, doi: 10.1109/SYNASC.2018.00041

[13] G. Jocher, A. Stoken, A. Chaurasia, J. Borovec, NanoCode012, TaoXie, Y. Kwon, K. Michael, L. Changyu, J. Fang, A. V, Laughing, tkianai, yxNONG, P. Skalski, A. Hogan, J. Nadar, imyhxy, L. Mammana, AlexWang1900, C. Fati, D. Montes, J. Hajek, L. Diaconu, M. T. Minh, Marc, albinxavi, fatih, oleg, and wanghaoyang0106, "*ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support*," Oct. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5563715

[14] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, "*Yolov6: A single-stage object detection framework for industrial applications*," 2022. [Online]. Available: https://arxiv.org/abs/2209.02976

[15] 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*," 2022. [Online]. Available: https://arxiv.org/abs/2207.02696

[16] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "*Yolov4: Optimal speed and accuracy of object detection,*" 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[17] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "*Path aggregation network for instance segmentation*," 2018. [Online]. Available: https://arxiv.org/abs/1803.01534

[18] J. Redmon and A. Farhadi, "Yolov3: *An incremental improvement,*" 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "*A density-based algorithm for discovering clusters in large spatial databases with noise*", in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, ser. KDD'96. AAAI Press, 1996, p. 226–2