electronics

Article

Real-Time Object Detection and Tracking for Unmanned Aerial Vehicles Based on Convolutional Neural Networks

Shao-Yu Yang [1], Hsu-Yung Cheng [1,*] and Chih-Chang Yu [2]

[1] Department of Computer Science and Information Engineering, National Central University, Taoyuan 32001, Taiwan
[2] Department of Information and Computer Engineering, Chun-Yuan Christian University, Taoyuan 320, Taiwan
* Correspondence: chengsy@csie.ncu.edu.tw

Abstract: This paper presents a system applied to unmanned aerial vehicles based on Robot Operating Systems (ROSs). The study addresses the challenges of efficient object detection and real-time target tracking for unmanned aerial vehicles. The system utilizes a pruned YOLOv4 architecture for fast object detection and the SiamMask model for continuous target tracking. A Proportional Integral Derivative (PID) module adjusts the flight attitude, enabling stable target tracking automatically in indoor and outdoor environments. The contributions of this work include exploring the feasibility of pruning existing models systematically to construct a real-time detection and tracking system for drone control with very limited computational resources. Experiments validate the system's feasibility, demonstrating efficient object detection, accurate target tracking, and effective attitude control. This ROS-based system contributes to advancing UAV technology in real-world environments.

Keywords: UAV; deep learning; ROS; convolutional neural network; pruned network; target tracking network; PID control

check for updates

Citation: Yang, S.-Y.; Cheng, H.-Y.; Yu, C.-C. Real-Time Object Detection and Tracking for Unmanned Aerial Vehicles Based on Convolutional Neural Networks. *Electronics* **2023**, *12*, 4928. https://doi.org/10.3390/electronics12244928

Academic Editors: Fan Yang, Slaviša Jovanović, Zongwei Wu, Virginie Fresse and Chao Li

Received: 1 November 2023
Revised: 6 December 2023
Accepted: 6 December 2023
Published: 7 December 2023

1. Introduction

In recent years, the robotics industry has experienced significant growth. Among different applications, unmanned aerial vehicles (UAVs), also known as drones, have emerged as a popular application area. A UAV is an autonomous or remotely controlled flying vehicle that does not require a human pilot onboard and has the capability to carry payloads. At first, UAVs found their primary use within the military, where they were employed for high-risk tasks like reconnaissance, attack operations, and supply missions, with the goal of minimizing the risk to human personnel. Nevertheless, progress in aerospace materials, inertial sensors, navigation technology, image processing, and data transmission has broadened the range of UAV applications, now extending into civilian fields.

UAVs find wide-ranging applications in civilian life. For example, remote-controlled aircraft are widely used for entertainment and recreational activities, and the emergence of first-person view (FPV) racing drones [1] has created a popular sport. Furthermore, drones are utilized for aerial photography and videography, offering unique perspectives and camera angles from the sky. Additionally, some companies have begun utilizing drones for delivery services [2], enabling faster and more efficient logistics. However, with the widespread adoption of drones, regulatory and safety concerns have arisen. To ensure safe operations, the Federal Aviation Administration (FAA) in the United States has established the "Operation and Certification of Small Unmanned Aircraft Systems" [3], which categorizes quadcopter drones and sets standards and requirements for their operation and certification. During the development of UAVs, compatibility issues between different systems often arise. To address these challenges, the Robot Operating System (ROS), an open-source tool for robot operations, has emerged. The ROS provides a framework and tools for handling inter-system connections, enabling developers to establish, control, and monitor UAV systems more easily. It also offers numerous software modules and libraries for various functionalities, such as sensor data processing, motion control, and mission planning. The goal of this paper is to apply ROS technology to UAVs, allowing for a modular system design and simplifying the overall development process, making UAV development more accessible and efficient.

Many newly introduced drones on the market are equipped with tracking and following capabilities. Typically, this functionality is achieved through electronic devices worn by the target, utilizing GPS coordinates to track and follow the target. However, in environments where GPS signals are unavailable, such as tunnels or basements, GPS positioning becomes ineffective or even impossible. Therefore, image tracking serves as an auxiliary method to realize target tracking. One significant aspect of this research is to explore the feasibility of controlling UAV systems via detection and tracking techniques based on images. By utilizing detection and tracking techniques, drones can capture target images through cameras and perform real-time analysis to achieve precise detection and tracking of the position and orientation of the target. With the rapid development of deep learning, many deep learning-based models have been proposed to address the problem of object detection based on images. R-CNN [4] applies high-capacity convolutional neural networks to bottom-up region proposals in order to localize and segment objects. SPP-Net [5] utilizes spatial pyramid pooling to eliminate the requirement of fixed size input images. Faster R-CNN [6] improves R-CNN and SPP-Net to reduce the training and testing speed while also increasing the detection accuracy. The Single Shot MultiBox Detector (SSD) [7] utilizes multi-scale convolutional bounding box outputs attached to multiple feature maps at the top of the network to detect objects in images using a single deep neural network. Unlike prior works that treat detection as a classification problem, the work named You Only Look Once (YOLO) [8] considers object detection as a regression problem to spatially separated bounding boxes and the associated class probabilities. A single neural network that can be optimized end-to-end is used to predict bounding boxes and class probabilities directly from full images in one evaluation. Therefore, YOLO has achieved great success. YOLO9000 and YOLOv2 [9] improve the original YOLO by introducing the concepts of batch normalization [10], high resolution classifier, convolution with anchor boxes [6], dimension clusters, direct location prediction, fine-grained features, and multi-scale training. YOLOv3 [11] made some little changes to update YOLO and to make it better. YOLOv4 [12] performs extensive experiments on the techniques of weighted residual connections, cross-stage partial connections, cross mini-batch normalization, self-adversarial training, mish-activation, mosaic data augmentation, drop-block regularization, and CIoU loss, and it combines a subset of these techniques to achieve state-of-the-art results. Person detection is a specialized form of object detection designed to identify the specific class "person" within images or video frames. Therefore, we utilize YOLOv4 to perform the detection task for the drone. To further reduce the computation complexity of YOLOv4 so that it can be applied in the environment with very limited computational resources, we perform pruning on the original YOLOv4 model.

Pruning methods have been proposed to reduce the complexity of CNN models [13–17]. Channel pruning intends to exploit the redundancy of feature maps between channels and remove channels with the minimal performance loss [13]. Li et al. [14] proposed pruning deep learning models using both channel-level and layer-level compression techniques. Liu et al. [16] designed a pruning method that can be directly applied to existing modern CNN architectures by enforcing channel-level sparsity in the network to reduce the model size, decrease the run-time memory footprint and lower the number of computing operations while maintaining the accuracy of the model. In [17], the authors demonstrate how to prune YOLOv3 and YOLOv4 models and then deployed them on OpenVINO with an increased frame rate and little accuracy loss. Since we utilize YOLOv4 for detection in the framework, we refer to the pruning methods described in [16,17].

Tracking algorithms based on Siamese Networks have become mainstream for visual tracking recently [18]. Bertinetto et al. [19] utilized a fully convolutional Siamese network that can be trained end-to-end for tracking applications. Zhu et al. [20] designed a distractor-aware module to perform incremental learning, which is able to transfer the general embedding to the current video domain effectively. Li et al. [21] proposed a tracker based on a Siamese region proposal network that is trained offline with large-scale image pairs. A ResNet-driven Siamese tracker is trained in [22]. SiamMask [23] improves the offline training procedure of popular fully convolutional Siamese methods for visual tracking by augmenting their loss with a binary segmentation task.

In this work, we utilize the ROS [24] (Robot Operating System) to implement image detection and tracking for controlling UAVs. Due to hardware constraints on the laptop, lightweight models are required. Therefore, for the object detector, we train a convolutional neural network based on the YOLOv4 architecture and prune it accordingly. In this work, the target object for detection is a person. We employ the pruned version of the YOLOv4 object detector and the SiamMask [23] monocular object tracker to detect and track the target person captured by the camera of the drone. Our system consists of four main components: (1) object detection, (2) target tracking, (3) Proportional Integral Derivative (PID) control, and (4) the UAV driver package. We utilize the Tello drone for implementing the object detection and tracking system. During the tracking process, the UAV control parameters include the roll, pitch, yaw, and altitude, all of which are controlled using PID controllers. These PID controllers take the position and distance of the target object as inputs. The position and distance are calculated using the monocular front-facing camera of the UAV.

2. Approach

The details of the methods used in the proposed framework, including object detection, model pruning, and visual tracking, are elaborated in this section. Figure 1 illustrates the system framework. A laptop computer (PC) is connected to the Tello drone via Wi-Fi for communication. The drone transmits images at a constant frequency of 30 Hz, which is preconfigured in the drone's driver software. These images are processed on the PC using a pruned version of the YOLOv4 algorithm for object detection. Users have the ability to select bounding boxes based on their requirements. The system utilizes the Siamese network, called SiamMask, for object tracking. Based on the tracked object's position and distance, a tracking algorithm based on a PID controller is employed to calculate estimates of the roll, pitch, yaw, and altitude. These estimated values for the roll, pitch, yaw, and altitude are then sent back to the drone to initiate the tracking process and to utilize the texture information of the background to enhance the final results.

The overall flowchart of the system architecture is shown in Figure 1. The drone sends the image feed to the PC, where the received images are processed using Pruned-YOLOv4 for person detection. If a person is detected in the image, their bounding box is displayed on the screen. The green boxes in Figure 1 represent the human detection results. If the user selects a specific object of interest by clicking on its bounding box, the system extracts the person within that bounding box as a template frame for the SiamMask network, enabling subsequent tracking. The tracking algorithm calculates the error between the target and the center of the frame. This error serves as the input for the PID controller, which generates flight commands for the yaw, roll, and altitude. As for the fourth flight command, pitch, it is calculated based on the relative distance of the tracked object using its position data. If no target is detected in the image, the drone maintains its position until a target appears.

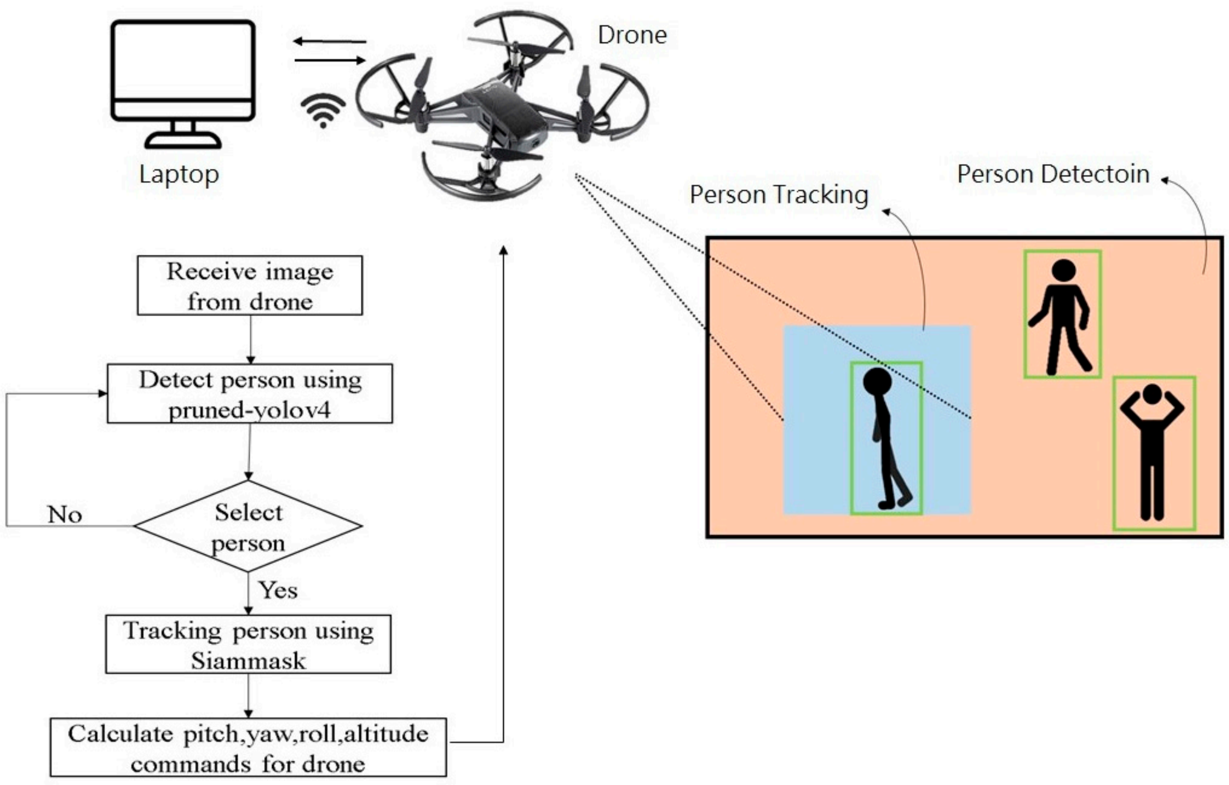

Laptop
Drone
Person Tracking
Person Detectoin
Receive image from drone
Detect person using pruned-yolov4
Select person
No
Yes
Tracking person using Siammask
Calculate pitch,yaw,roll,altitude commands for drone

Figure 1. System framework.

2.1. Hardware Specifications

The DJI Tello [25] drone is a small and easy-to-control consumer-grade drone. It has dimensions of approximately 98 × 92 cm and weighs around 80 g. The design of this drone allows for its usage in both indoor and outdoor environments. In terms of its features, it is equipped with a front-facing camera, a 3-axis gyroscope, a 3-axis accelerometer, a 3-axis magnetometer, a pressure sensor, and an ultrasonic altitude sensor. The resolution of the front-facing camera is 1280 × 720, capturing video at 30 frames per second. The Tello drone can communicate with other devices, such as smartphones or laptops, through a Wi-Fi network. In this particular study, a PC was used for communication with the drone.

2.2. Detection Model Pruning and Object Detection

During the object detection process, we first need to train the model. Our training process is illustrated in Figure 2. The yellow part in Figure 2 represents the modules for the Darknet framework. The light blue part in Figure 2 represents the modules for the pruning stage. Firstly, we employ the Darknet framework to train the YOLOv4 base model. Then, during the pruning stage, we perform sparse training, channel pruning, layer pruning, and fine-tuning on the base model using the Darknet framework. Once the pruning stage is complete, the model undergoes fine-tuning training on the Darknet framework. Finally, we deploy the model on a laptop for detection.

A. Darknet Training

We use the Darknet framework to train the YOLOv4 model [26] and adjust several hyperparameters during the initial training phase to improve the accuracy and performance of the model. One of the first hyperparameters to adjust is the input size of the network. Increasing the input size helps in detecting small objects, although it may also slow down the model's inference speed and consume more GPU memory. It is important to note that the YOLOv4 network downsamples the input size by a factor of 32 in both the vertical and horizontal directions, so the input width and height must be multiples of 32. To achieve this, we decided to use 416 × 416 as the input size.

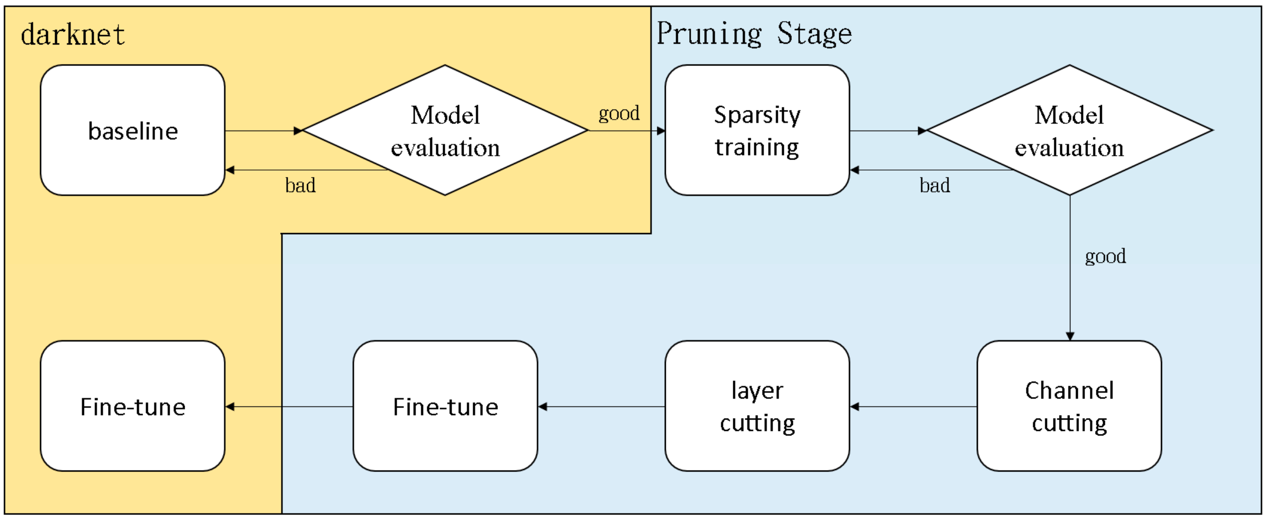


Figure 2. Training steps for the detection model.

The second and third hyperparameters to adjust are the batch size and subdivisions. These settings are adjusted based on the GPU's performance. The batch size hyperparameter represents the number of images to load during training, with a default value of 64. However, if the GPU's memory size is insufficient, it will not be able to load 64 images at once. To address this issue, each batch is further subdivided into multiple sub-batches. Each sub-batch is fed into the GPU one by one until the batch is completed. In this study, we set the batch size and subdivisions to 64 and 8, respectively. The fourth hyperparameter to adjust is the number of iterations (note that in the Darknet framework, training is measured in iterations, not epochs). According to the Darknet framework's guidelines, each object class should have at least 2000 iterations. Since we have only one class, the number of iterations should exceed 2000. We set the number of iterations to 2200 to achieve higher accuracy.

B. Pruning Stage

Due to the hardware limitations of the laptop, a lightweight model needs to be used. Therefore, after training the model using the Darknet framework, it needs to be pruned to achieve the goal of lightweighting. We use the metrics of accuracy (mAP@0.5) and inference speed (BFLOPs) to evaluate the pruned model. However, it is important to note that there is a trade-off between accuracy and inference speed. Assuming the hardware configuration is fixed, when the model is pruned to a very small size, its inference speed may increase but its accuracy is typically reduced.

Before pruning, the weights from the Darknet framework undergo a basic training process. Once the basic training is completed, the obtained model is pruned using the pruning strategy from [27]. This strategy involves first conducting sparse training on the model, where the channel sparsity in deep models helps with channel pruning. To facilitate channel pruning, each channel in the convolutional layers is associated with a scaling factor. During training, L1 regularization is applied to these scaling factors to automatically identify unimportant channels. Channels with smaller scaling factor values (orange color) are pruned (left side). After pruning, we obtain a compact model (right side), which is then fine-tuned to achieve comparable (or even higher) accuracy with the fully trained network. The pruning process is illustrated in Figure 3.

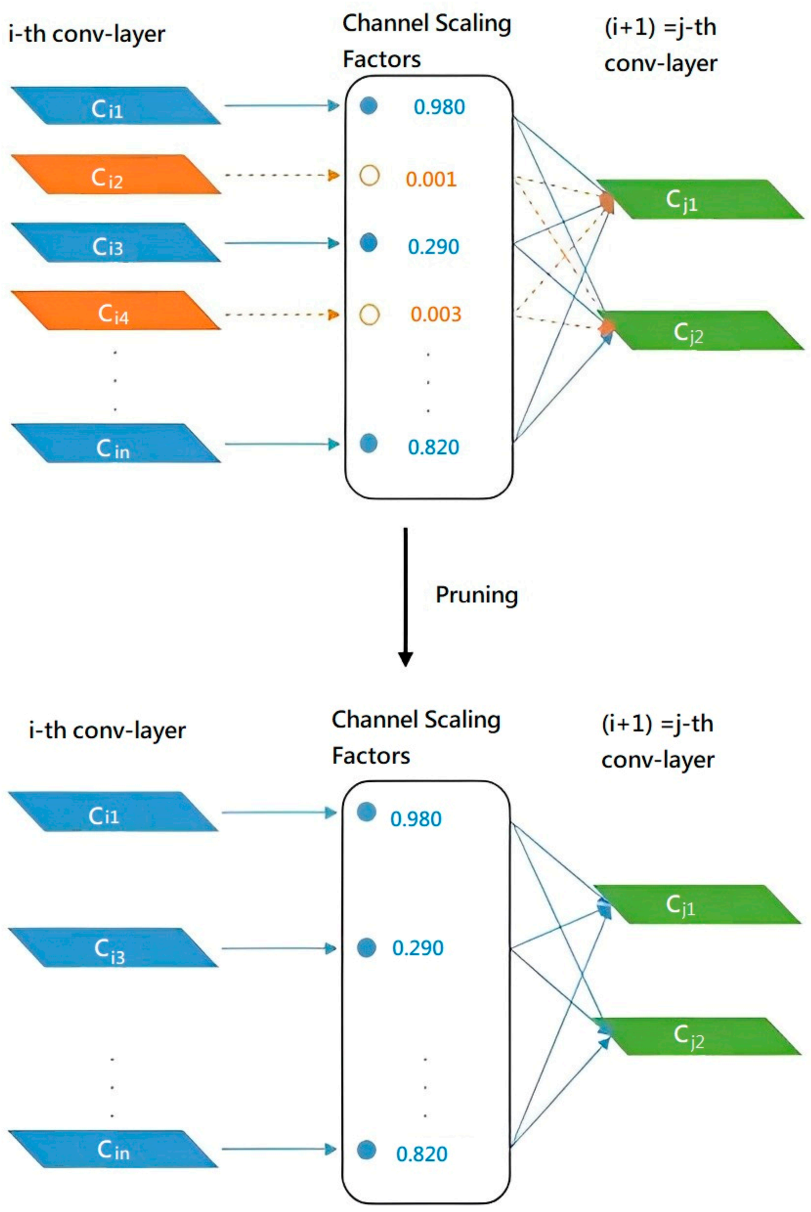


Figure 3. Pruning method.

C. Sparsity Training

We add a Batch Normalization (BN) layer after each convolutional layer in YOLOv4 to accelerate convergence and improve generalization. The BN layer utilizes batch statistics to normalize the convolutional features as:

$$y = \gamma \times \frac{x - \overline{x}}{\sqrt{\sigma^2 + \varepsilon}} + \beta \tag{1}$$

Here, $\overline{x}$ and σ^2 represent the mean and variance of the input features in the mini-batch, respectively. γ and β represent the trainable scale factor and bias in the BN layer. In this study, we directly use the scale factor in the BN layer as an indicator of channel importance. To effectively distinguish between important and unimportant channels, we apply L1 regularization to γ, enabling channel-level sparse training. The loss function for sparse training is shown as:

$$L = loss_{yolo} + \alpha \sum_{\gamma \epsilon \Gamma} f(\gamma) \tag{2}$$

The function $f(\gamma)$ represents the L1 norm applied to γ, which is widely used in the sparsification step. α represents the penalty factor that balances the two loss terms.

The effectiveness of pruning depends on the sparsity of the model. Prior to sparse training, the distribution of γ in the BN layer of YOLOv4 is expected to be uniform. After sparse training, most of the γ values in the BN layer are compressed toward zero. This brings two benefits:

(1) Achieving network pruning and compression to improve model efficiency: The weights in the BN layer are typically used for standardizing and scaling each input sample in the network. When the weights are close to zero, the corresponding standardization and scaling operations are reduced, thereby reducing the computational complexity.

(2) By sparsifying the weights of the BN layer close to zero, it becomes possible to identify parameters that have minimal impact on network performance and prune them.

D. Channel cutting

Once sparse training is completed, channel cutting can be performed. Here is an explanation of how to proceed with channel cutting. First, the total number of channels in the backbone is computed. Once the number of channels is determined, the corresponding γ values are stored in a variable and sorted in ascending order. The next step is to decide which channels to keep and which ones to prune. This can be achieved by setting a pruning rate, which represents the proportion of channels to be pruned. The pruning rate is typically a value between 0 and 1, where a higher value indicates a greater degree of pruning. By following these steps, the channel-cutting process can be carried out to selectively retain or remove channels based on the specified pruning rate.

E. Layer cutting

Within the YOLOv4 backbone, there are multiple CSPX modules, where each CSPX module consists of three CBL layers and X ResUnit modules. The resulting features of these modules are concatenated together, as depicted in Figure 4a. For layer cutting, we mainly prune the ResUnit within YOLOv4. The architecture of the ResUnit is illustrated in Figure 4b, which consists of two CBL layers and a shortcut connection. The CBL layer comprises a Conv layer, a BN layer, and a Leaky ReLU activation function, as shown in Figure 4c. In layer cutting, the mean values of γ for each layer are first sorted, and by evaluating the previous CBL layer of each shortcut, the minimum value can be selected for layer pruning. To ensure the structural integrity of YOLOv4, when pruning one ResUnit, both the shortcut layer and the preceding CBL layer are simultaneously pruned, resulting in the pruning of three layers in total.

F. Fine-tuning

Different pruning strategies and threshold settings yield different effects on the pruned model. Sometimes, the accuracy of the pruned model may even increase, although in most cases, pruning can have a negative impact on model accuracy. In such cases, it is necessary to perform fine-tuning on the pruned model to compensate for the accuracy loss caused by pruning. Fine-tuning is crucial for restoring the accuracy of the pruned model. In our experiments, we directly retrained the Pruned-YOLOv4 using the same training hyperparameters as the normal training process for YOLOv4.

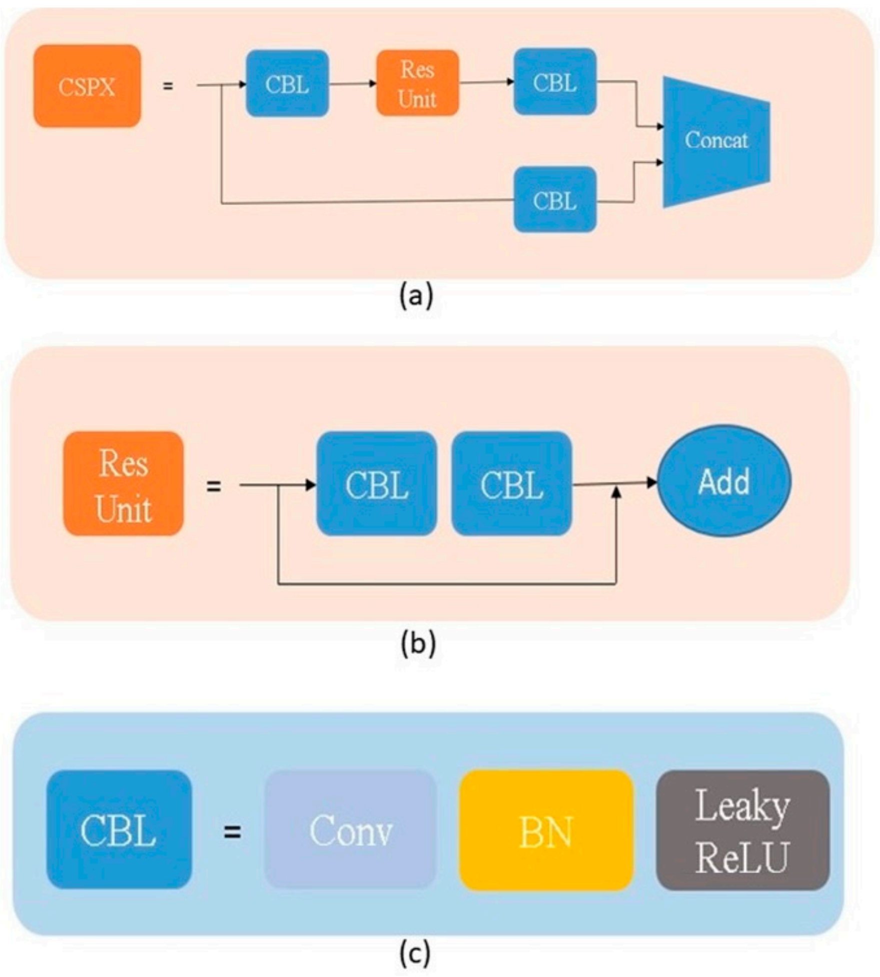


Figure 4. (**a**) CSPX, (**b**) ResUnit, and (**c**) CBL layer in YOLOv4.

2.3. Object Tracking and Drone Control

The laptop performs the real-time detection of drones, allowing users to track the detected targets until the tracking is completed. The objects detected using Pruned-YOLOv4 are represented by bounding boxes, each containing four coordinates (x, y, w, h). Here, x and y represent the coordinates of the top left corner of the bounding box, while w and h represent the width and height of the bounding box, respectively. Once the coordinates are obtained, we continuously detect the user's mouse position. If the mouse click falls within a bounding box, the four coordinates of the bounding box are passed to the object tracking module, which utilizes SiamMask [23]. SiamMask is a target-tracking algorithm based on Siamese Neural Networks [28]. Siamese Neural Networks were initially proposed by Bromley and LeCun to address signature verification problems [29] and have since been widely applied in various fields, such as image matching and target tracking.

In the task of target tracking, Siamese Neural Networks employ two identical subnetworks with shared parameters and weights. The tracking template is fed into the network, and the output weights are obtained. These weights are then matched with the output weights of the search region to calculate the similarity score. The target's location to be tracked is determined by computing the response score map. Building upon the traditional Siamese network, SiamMask incorporates target segmentation computation, which allows for the extraction of the target's contour. This helps mitigate the effects of target feature variations caused by rotation and deformation.

While performing tracking, SiamMask simultaneously returns an image with the bounding box of the tracked object. This bounding box contains information about the object's position in the image. The bounding box of the tracked object consists of four points: (x_{min}, y_{min}), (x_{max}, y_{min}), (x_{min}, y_{max}), and (x_{max}, y_{max}).

We can use these four points to calculate the center of the object's position, which is computed as:

$$(x_{center}, y_{center}) = \frac{x_{min} + x_{max}}{2}, \frac{y_{min} + y_{max}}{2} \tag{3}$$

In order to track an object accurately, it is necessary to know the exact center position of the drone's screen. This is because the detected object's center should always align with the center of the drone's screen for proper tracking. The calculation of the disparity between the center of the drone's screen and the object's center is performed as:

$$e_x = imgx_{center} - x_{center} \tag{4}$$

$$e_y = imgy_{center} - y_{center} \tag{5}$$

e_x and e_y should always be equal to or close to zero to achieve effective tracking.

The drone has a total of four control parameters: roll, yaw, altitude, and pitch. Roll controls the drone's lateral movement, yaw controls the drone's clockwise or counterclockwise rotation, altitude controls the drone's vertical movement, and pitch controls the drone's forward or backward movement. Figure 5 illustrates the basic flight maneuvers of the drone.

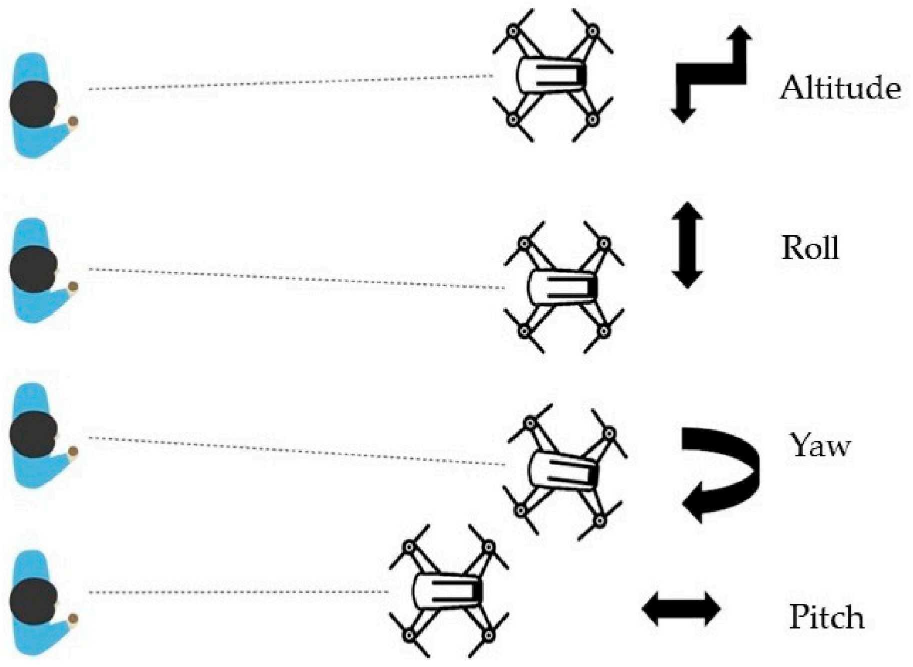


Figure 5. Fundamental maneuvers of a drone.

Next, we will explain how PID controls drone flight. It is evident that by using the center point of the tracked object and the center point of the screen, we can obtain the error in the *X*-axis. This error is related to the drone's roll for lateral movement and yaw for clockwise or counterclockwise rotation. If the drone detects that the object is moving left or right, we can choose to adjust the drone's heading to face the object or perform lateral movements to keep up with it. Additionally, there is the pitch axis, which involves forward and backward movements. By subtracting the distance between the drone and the real object from the desired ideal distance, we can calculate the distance error and control the drone's forward or backward movements accordingly. Finally, regarding altitude, by subtracting the Y-coordinate of the tracked object from the Y-coordinate of the screen center, we can obtain the error in the *Y*-axis. This allows us to calculate the necessary altitude for the drone's vertical ascent or descent. The specific control methods are illustrated in Figure 6.

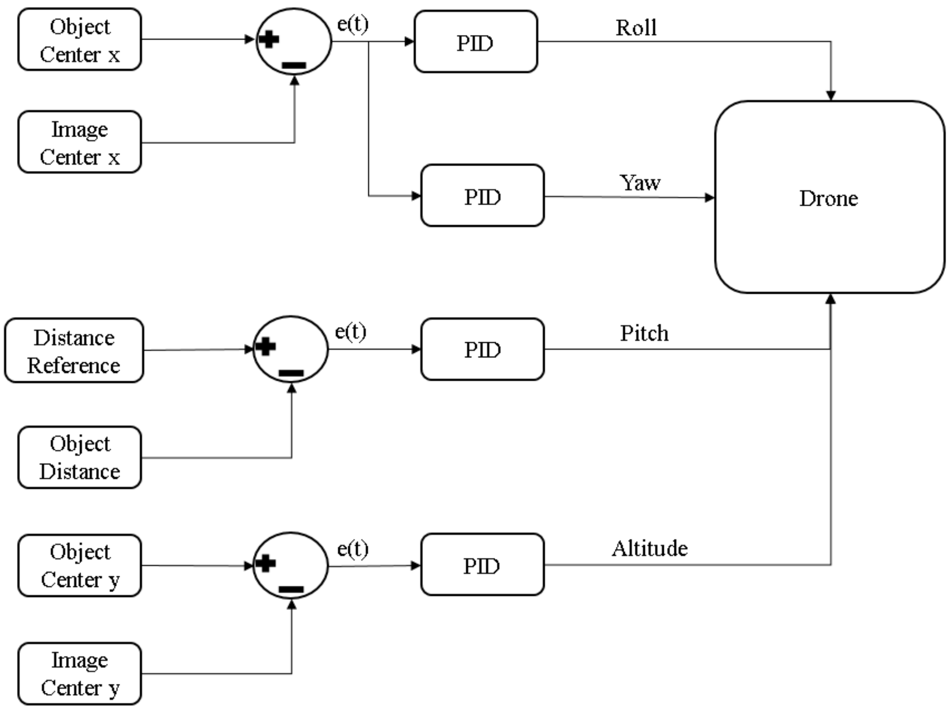


Figure 6. The control scheme for the drone.

When it comes to the error in the *X*-axis, the choice between roll and yaw is designed to be dependent on the derivative term (D) of the PID controller. The D term represents the rate of change of the error. If the error changes rapidly, the drone needs to increase its power to keep up with the object's movements. However, if the object continues to move along the *X*-axis, as shown in Figure 7a, a stronger control is required to track the target quickly. The red rectangle represents the bounding box of the subject. Therefore, we choose the roll option, which means performing lateral movements to the right in order to follow the object, represented by the green arrow illustrated in Figure 7a. On the other hand, if the tracked object does not exhibit significant movement, the yaw option is selected, which only requires adjusting the drone's heading to follow the target, represented by the green arrow illustrated in Figure 7b.

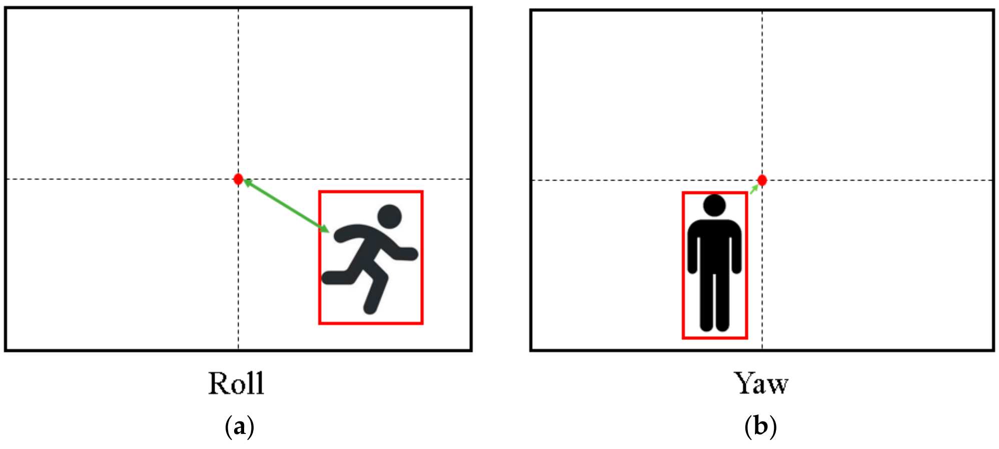


Figure 7. Examples of object movement (**a**) Roll (**b**) Yaw.

3. Experimental Results

In this section, we demonstrate and analyze the performance of the detection model pruning, object detection, tracking and drone control.

3.1. Detection Model Pruning and Object Detection

We trained the baseline YOLOv4 model using the coco2014 [30] dataset. By applying pruning techniques to the baseline YOLOv4, we obtained a lightweight model known as Pruned-YOLOv4. In our comparison, we considered not only the baseline YOLOv4 but also Tiny-YOLOv4. The Tiny version of YOLOv4 is specifically designed as a lightweight variant for devices with lower computational resources.

To evaluate the performance of the object detector, we applied the following four metrics:

(1) Precision: Precision is a widely used metric in the object detection community. It measures the proportion of true positives among all the detections made by the system. A higher precision indicates that the system can accurately identify target objects, reducing the likelihood of false alarms.
(2) Recall: Recall is another commonly used metric in object detection. It measures the proportion of true positives among all the actual target objects. A higher recall indicates that the system can successfully detect a larger portion of the target objects, reducing the risk of missed detections.
(3) BFLOPs: BFLOPs is a metric used to measure the computational efficiency of a computer system or machine-learning model. BFLOPs represents the number of billion floating-point operations required for a specific convolution operation. By summing up the BFLOPs consumed by multiple convolution and other operations, the complexity of an algorithmic model can be quantified. It is a commonly used metric for evaluating the computational efficiency and speed of systems or models.
(4) mAP@0.5 (mean Average Precision at IoU 0.5): mAP@0.5 is a commonly used evaluation metric in object detection. It measures the average precision at an Intersection over Union (IoU) threshold of 0.5 across different classes.

A. Basic Training in Darknet

The performance results of YOLOv4 and Tiny-YOLOv4 after basic training using Darknet are shown in Tables 1 and 2. From Table 1, it is evident that there is a significant difference in the model accuracy between the two models. YOLOv4 achieves an mAP@0.5 of 0.749, while Tiny-YOLOv4 achieves 0.55. However, Table 2 reveals that while YOLOv4 achieves higher accuracy, it also requires a larger number of parameters for the model to learn. The trained weight size of YOLOv4 is 10.6 times larger than that of YOLOv4-Tiny, and the BFLOPs is 7.64 times larger. The large size of the model also results in a longer inference time, highlighting the importance of pruning YOLOv4 in this study.

Table 1. Evaluation of Darknet basic training.

Model	Precision	Recall	mAP@0.5
YOLOv4	0.818	0.605	0.749
Tiny-YOLOv4	0.65	0.41	0.55

Table 2. Parameter size of Darknet basic training.

Model	Params	Size of .Weights	BFLOPs
YOLOv4	63.9 M	250.2 MB	59.563
Tiny-YOLOv4	5.87 M	23.6 MB	7.79

B. Sparsity Training Results

During the sparse training, we adjusted the balance between the two losses, denoted as α. As the training progressed, the number of smaller scaling factors increased, while the number of larger scaling factors decreased. Sparse training effectively reduces the scaling factors, leading to channel sparsity in the convolutional layers of YOLOv4. This enables the identification and pruning of parameters with less impact on network performance. However, when using a larger penalty factor (i.e., $\alpha = 0.01$) for sparse training, the scaling factors decay too aggressively, resulting in underfitting and failure of the model. This can be observed from the significant drop in the Recall and mAP@0.5 in Table 3, indicating that this model is not suitable for pruning. On the other hand, using a very small penalty factor (i.e., $\alpha = 0.0001$) leads to insufficient sparsity in the scaling factors, resulting in poor pruning performance. Although this model demonstrates excellent performance in Table 3, it becomes challenging to identify parameters with less impact on network performance during subsequent pruning operations. In our experiments, we trained the YOLOv4 model with a penalty factor $\alpha = 0.001$ for channel pruning. The scaling factors were compressed to near zero, and the performance remained relatively stable, as shown in Table 3.

Table 3. Evaluation of the penalty factor.

Model	Precision	Recall	mAP@0.5
$\alpha = 0.0001$	0.822	0.664	0.75
$\alpha = 0.001$	0.851	0.626	0.741
$\alpha = 0.01$	0.981	0.059	0.26

C. Channel Cutting Results

During the channel-pruning process, we can adjust the pruning rate to perform pruning, which ranges between 0 and 1. A higher pruning rate indicates a greater level of pruning. From Table 4, it can be observed that when the pruning rate is set to 0.74, there is a significant drop in the model's mAP@0.5 and Recall. However, when the pruning rate is set to 0.73, the decrease in the mAP@0.5 and Recall is not as significant as with a pruning rate of 0.74. Therefore, we should explore a finer range below the pruning rate of 0.73 to find the optimal value. From Table 5, it can be inferred that a higher pruning rate theoretically leads to a reduction in the model parameters and an increase in the inference speed. If the mAP@0.5 values are similar for different pruning rates, it is preferable to select the pruning model with a higher pruning rate to achieve faster speed. Hence, we ultimately chose a pruning rate of 0.735.

Table 4. Precision, Recall, and mAP under different pruning rates.

Pruning Rate	Precision	Recall	mAP@0.5
0.5	0.86	0.56	0.7423
0.6	0.865	0.559	0.741
0.7	0.865	0.559	0.741
0.71	0.87	0.55	0.742
0.72	0.883	0.515	0.728
0.73	0.935	0.266	0.649
0.735	0.94	0.21	0.51
0.74	0.92	0.00013	0.38

Table 5. Parameter size and BFLOPs under different pruning rates.

Pruning Rate	Params	Size of .Weights	BFLOPs
0.5	15.03 M	58.8 MB	28.564
0.6	9.134 M	35.7 MB	24.886
0.7	6.42 M	25.2 MB	21.922
0.71	6.25 M	24.5 MB	21.592
0.72	6.02 M	23.6 MB	21.270
0.73	5.7 M	22.4 MB	20.999
0.735	5.57 M	21.8 MB	20.858
0.74	5.4 M	21.2 MB	20.703

D. Layer Cutting Results

During the layer-cutting process, we can determine the number of ResUnits to be pruned. From Table 6, it can be observed that when the number of ResUnits is set to 15, there is a significant drop in the model's mAP@0.5 and Recall. However, when the number of ResUnits is set to 14, the model's mAP@0.5 and Recall do not decrease as dramatically as with 15 ResUnits. Additionally, from Table 7, we can infer that the more units pruned, the fewer model parameters theoretically and the faster the inference speed. If the mAP@0.5 values are similar for different numbers of ResUnits, it is preferable to choose the pruning model with a higher number of units to achieve a faster speed. Therefore, we ultimately decided to prune 14 ResUnits.

Table 6. Evaluation of detection accuracy for pruning the number of ResUnits.

Cut ResUnit Numbers	Precision	Recall	mAP@0.5
11	0.914	0.175	0.5336
12	0.91	0.132	0.494
13	0.91	0.131	0.483
14	0.912	0.116	0.463
15	0.929	0.0188	0.335

Table 7. Parameter Sizes of pruning the number of ResUnits.

Cut ResUnit Numbers	Params	Size of .Weights	BFLOPs
11	4.6 M	18 M	19.490
12	4.4 M	17.2 M	19.220
13	4.3 M	17.1 M	18.991
14	4.2 M	16.8 M	18.610
15	4.15 M	16.2 M	17.869

E. Final Model Tuning

After pruning the model, it is common for pruning to have a negative impact on model accuracy. In such cases, fine-tuning the pruned model becomes crucial to recover the lost accuracy. In our experiment, we directly used the same training hyperparameters and dataset as in the normal training of YOLOv4 to retrain the Pruned-YOLOv4 model. The performance after fine-tuning is shown in Table 8. Compared to the baseline YOLOv4, there is a slight sacrifice of 0.013 in the mAP@0.5. However, from Table 9, we can observe that Pruned-YOLOv4 achieves a faster inference speed and significantly reduces the model's parameter count and the size of the .weights file. This allows the model to be deployed on embedded devices with limited memory space. Compared to Tiny-YOLOv4, there is a significant improvement in the mAP@0.5, as shown in Table 8. Although there is a slight sacrifice in the BFLOPs, as seen in Table 9, this trade-off is worthwhile as it results in a more accurate model.

Table 8. Evaluation of various detection models.

Model	Precision	Recall	mAP@0.5
SSD	0.607	0.398	0.504
YOLOv3	0.661	0.436	0.579
YOLOv4	0.818	0.605	0.749
Tiny-YOLOv4	0.652	0.411	0.551
Pruned-YOLOv4	0.784	0.619	0.736

Table 9. Parameter size of various detection models.

Model	Params	Size of .Weights	BFLOPs
SSD	34.3 M	135.3 MB	70.40
YOLOv3	61.6 M	242.9 MB	65.86
YOLOv4	63.9 M	250.2 MB	59.563
Tiny-YOLOv4	5.87 M	23.6 MB	7.79
Pruned-YOLOv4	4.28 M	16.8 MB	18.61

3.2. Subject Tracking and Drone Control

The aim of this study is to explore the feasibility and effectiveness of automated control in real-world applications. To achieve this goal, we design a series of experiments to simulate the exploration needs of drones in real environments and require the drones to successfully track target objects automatically. To ensure the reliability of the experimental results, we perform experiments in both indoor and outdoor environments. By conducting experiments in these locations, we are able to better assess the adaptability and performance of the automated control in various real-world scenarios. Figures 8 and 9 list the selected outdoor scenes and indoor scenes in the experimental videos, respectively. The subjects being tracked include ten different people. Each person is tracked for 50 to 90 s in outdoor and indoor environments five times. During the tracking process, a random number of 0 to 7 other people would appear as passersby in the scene.

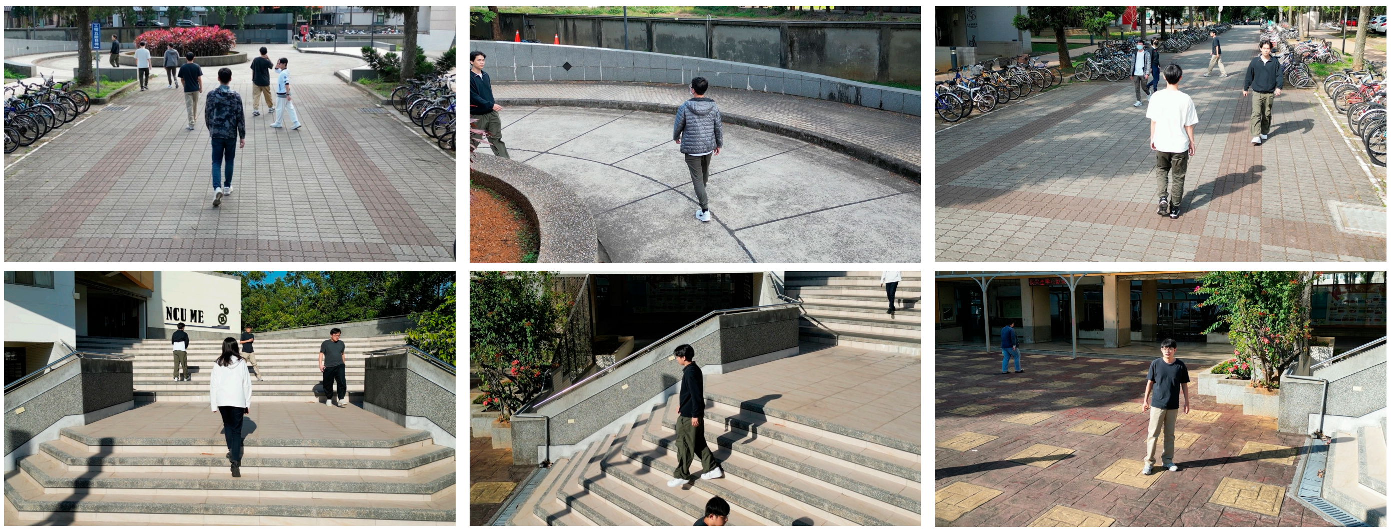


Figure 8. Selected scenes in outdoor environments.

Figure 9. Selected scenes in indoor environments.

A. Parameters for PID

After obtaining the error values of the actual drone state, the flight maneuvers of the drone can be determined to enable it to track the target. To address this issue, we employ a PID control system. It is important to note that the PID system generates outputs for all three axes, the x, y, and z axes, represented by the green, red, and blue arrows illustrated in Figure 10. Therefore, there are nine parameters in total. Setting higher Kp values allows the controller to respond faster to control errors. Consequently, the Kp parameters for all three axes are set to the highest value. Conversely, the Ki parameters are set to the minimum value to reduce the impact of the accumulated errors, thereby avoiding excessive system tuning or instability. As for the configuration of the Kd parameters, an appropriate setup provides a response to the rate of error variation, thereby enhancing the system's response speed and stability. The initial values for parameter settings are selected according to [31], followed by a grid search to determine the optimal values.

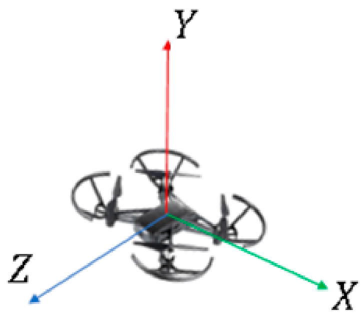
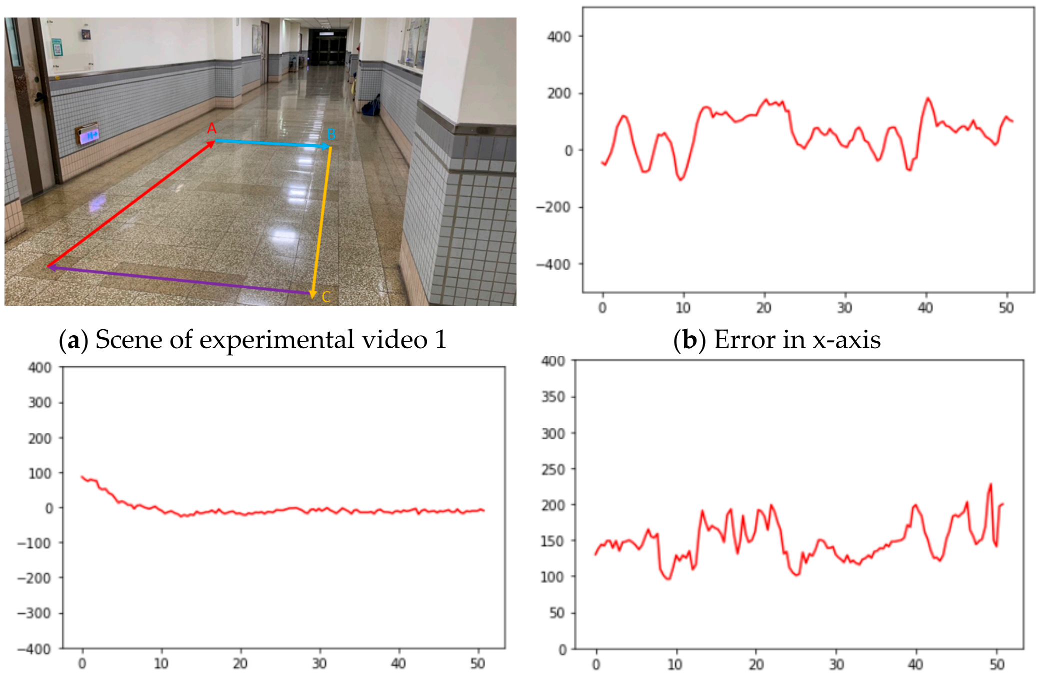


Figure 10. Drone three-axis orientation.

B. Analysis of PID control for drones

In this sub-setcion, we demonstrate how the drone continuously tracks the target object in flight and adjusts its flight actions as the target object moves. Two experimental videos are selected to demonstrate the tracking and control processes. In video 1, the tracked subject walks on a flat surface, as shown in Figure 11a. The four directions that the subject moves are represented as the red, blue, yellow, and purple arrows in Figure 11a. The response of the PID control to the error in the x-axis position of the tracked object in video 1 is plotted in Figure 11b. Figure 11b shows that as the target object moves, the error increases, and the PID control quickly corrects the error to minimize it toward zero. The drone continuously tracks the target object, while the PID control attempts to reduce the x-axis error of the target object. For the error in *y*-axis, since the subject in video 1 does not undergo significant changes in height, we can observe from Figure 11c that there is not a significant variation in the y-axis error. As for the distance in the *z*-axis position of the tracked object, the distance between the drone and the target object is fixed at a reference value of 150, which corresponds to a distance of 1.5 m on the ground between the target object and the drone. When the target object moves forward and backward, the drone has to continuously track the target object. Figure 11d plots the response of the PID control to the error in the *z*-axis position of the tracked object in the selected video. It demonstrates that the error varies with the distance between the target object and the drone, and the PID control attempts to reduce the z-axis error of the target object.



(**a**) Scene of experimental video 1 (**b**) Error in x-axis

(**d**) Error in y-axis (**c**) Distance in z-axis

Figure 11. Response of the PID control to the errors in different directions for video 1.

In video 2, the tracked subject moves upstairs and downstairs, as shown in Figure 12a. The red arrow and blue arrow represent the directions moving up and down the stairs, respectively. In video 2, the drone needs to follow the subject and fly straight up or down along the stairs. The main focus is to test whether the drone can adjust the *y*-axis error in real time. Figure 12a–c show the response of the PID control to the error in the *x*-axis and *y*-axis positions and the distance in the *z*-axis position of the tracked object in video 2, respectively. Figure 12 demonstrates that the PID control continuously adjusts the *x*-axis and *y*-axis errors to approach zero as the subject moves forward and backward while ascending or descending the stairs.

C. Tracking Accuracy Evaluation

The mean absolute errors between the target and the center of the frame for tracking are listed in Table 10. The errors in the *x*-axis are slightly higher than the errors in the *y*-axis because the subjects change their moving directions in most experimental videos. When the targets being tracked are moving on the ground plane without ascending stairs or descending stairs, the errors in the *y*-axis are close to zero. The tracking errors in the outdoor environments are higher than the errors in indoor environments due to the influence of wind.

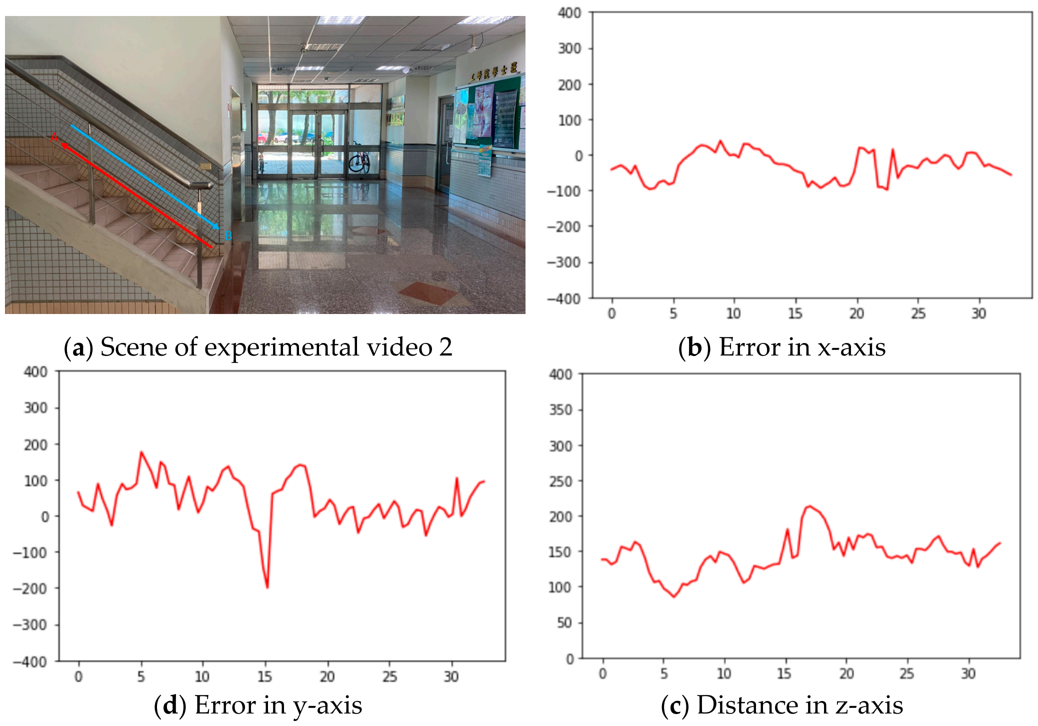


Figure 12. Response of the PID control to the errors in different directions for video 2.

Table 10. Mean absolute error of tracking.

	Indoor	**Outdoor**
x-axis	128.95	137.62
y-axis	98.73	116.98

3.3. Discussion

It is a challenging issue to balance the size of the model parameters and the accuracy of the model in the process of pruning. Through analyzing the Precision, Recall, and mAP as well as the parameter sizes under various pruning rates, we are able to determine a suitable pruning rate to balance the trade-offs. Also, fine-tuning the pruned model is helpful to recover and increase the model accuracy. The PID control process, which continuously minimizes the error between the subject being tracked and the center of the frame, can complete the task of automatic drone control and maintain a stable flight path in real time. This is attributed to the ability of the automatic control system to promptly adjust to the position and movement of the target object to minimize the error between the reference position and the actual position. This allows the drone to track target objects accurately and respond quickly to changes. Automated control is of great significance to human–machine collaboration. It extends the high cognitive capabilities of human operators. At the same time, automated control ensures good execution efficiency and stability. Based on the above observations and analysis, automatic drone control has obvious advantages. It can provide accurate and stable tracking capabilities while taking into account the execution efficiency of automated control. This collaborative model allows researchers and operators to participate in drone missions and leverage their respective expertise while achieving higher efficiency with the help of automatic control systems.

4. Conclusions

In this paper, we propose an implementation method for an object detection and target tracking system based on the Robot Operating System (ROS) and apply it to the Tello drone. The system achieves efficient object detection and target-tracking capabilities in real-time environments. We utilize the pruned YOLOv4 architecture as the detection model and select SiamMask as the tracking model. Additionally, we introduce a PID module to calculate the errors and determine the flight attitude and action. For the detection module, we choose the pruned YOLOv4 architecture, which provides a faster execution speed while maintaining the detection accuracy. By reducing the redundant parameters and computations in the model, we achieve lightweight and accelerated performance. This allows our system to efficiently perform object detection tasks in real-time environments. For the tracking module, we adopt the SiamMask model. SiamMask is a single-object tracking method capable of real-time target tracking. In our system, SiamMask is used to track the objects detected by YOLOv4, enabling continuous object tracking and positioning. Furthermore, we introduce the PID module to calculate the errors and adjust the flight attitudes. PID is a classical control algorithm that computes control signals based on the current error, accumulated error, and rate of error change, aiming to bring the system output closer to the desired value. In our system, the PID module calculates errors based on the target object's position and the drone's current state, and adjusts the drone's attitude control signals to stably track the target object. Through flight experiments, we validate the feasibility of applying this system in everyday environments. The pruned YOLOv4 model provides efficient object detection capabilities, enabling fast target detection in real-time environments. SiamMask is used for tracking the target object, and the PID module accurately calculates the errors and adapts to different flight situations, allowing the drone to stably track the target object.

Author Contributions: Conceptualization, S.-Y.Y.; Methodology, H.-Y.C.; Software, S.-Y.Y.; Validation, C.-C.Y.; Investigation, H.-Y.C.; Resources, H.-Y.C.; Writing—original draft, S.-Y.Y.; Writing—review & editing, H.-Y.C.; Visualization, C.-C.Y.; Supervision, C.-C.Y.; Project administration, C.-C.Y.; Funding acquisition, H.-Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Science and Technology Council NSTC, Taiwan under grant number 112-2221-E-008-069-MY3.

Data Availability Statement: The datasets generated and analyzed during the current study are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Attention Drone Geeks! Here's Some Answers You've Been Looking for. The Local Brand. Available online: https://thelocalbrand.com/attention-drone-geeks-some-answers/ (accessed on 18 November 2023).
2. Amazon Plans to Start Drone Deliveries in the UK and Italy Next Year. Engadget. Available online: https://www.engadget.com/amazon-plans-to-start-drone-deliveries-in-the-uk-and-italy-next-year-185027120.html (accessed on 18 November 2023).
3. Operation and Certification of Small Unmanned Aircraft. Federal Aviation Administration. Available online: https://www.federalregister.gov/documents/2016/06/28/2016-15079/operation-and-certification-of-small-unmanned-aircraft-systems#h-33 (accessed on 18 November 2023).
4. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
5. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In Proceedings of the 13th European Conference on Computer Vision (ECCV 2014), Zurich, Switzerland, 6–12 September 2014; pp. 346–361.
6. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015; pp. 91–99.
7. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.; Berg, A.C. SSD: Single Shot Multibox Detector. In Proceedings of the 14th European Conference on Computer Vision (ECCV 2016), Amsterdam, The Netherlands, 11–14 October 2016; pp. 21–37.
8. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 779–788.
9. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. In Proceedings of the 30th IEEE International Conference on Computer Vision (CVPR 2017), Venice, Italy, 22–29 October 2017; pp. 6517–6525.

10. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML 2015), Lille, France, 6–11 July 2015; pp. 448–456.
11. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767. [CrossRef]
12. Bochkovskiy, A.; Wang, C.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv* **2020**, arXiv:2004.10934. [CrossRef]
13. He, Y.; Zhang, X.; Sun, J. Channel Pruning for Accelerating Very Deep Neural Networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 1398–1406.
14. Li, Q.; Li, H.; Meng, L. Deep Learning Architecture Improvement Based on Dynamic Pruning and Layer Fusion. *Electronics* **2023**, *12*, 1208. [CrossRef]
15. Liu, X.; Li, C.; Jiang, Z.; Han, L. Low-Complexity Pruned Convolutional Neural Network Based Nonlinear Equalizer in Coherent Optical Communication Systems. *Electronics* **2023**, *12*, 3120. [CrossRef]
16. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning Efficient Convolutional Networks Through Network Slimming. In Proceedings of the 30th IEEE International Conference on Computer Vision (CVPR 2017), Venice, Italy, 22–29 October 2017; pp. 2736–2744.
17. Pruned-OpenVINO-YOLO. TNTWEN. Available online: https://github.com/TNTWEN/Pruned-OpenVINO-YOLO (accessed on 10 May 2023).
18. Li, J.; Zhang, K.; Gao, Z.; Yang, L.; Zhuo, L. SiamPRA: An Effective Network for UAV Visual Tracking. *Electronics* **2023**, *12*, 2374. [CrossRef]
19. Bertinetto, L.; Valmadre, J.; Henriques, J.F.; Vedaldi, A.; Torr, P.H.S. Fully-Convolutional Siamese Networks for Object Tracking. In Proceedings of the European Conference on Computer Vision (ECCV 2016), Amsterdam, The Netherlands, 11–14 October 2016; pp. 850–865.
20. Zhu, Z.; Wang, Q.; Li, B.; Wu, W.; Yan, J.; Hu, W. Distractor-aware Siamese Networks for Visual Object Tracking. In Proceedings of the European Conference on Computer Vision (ECCV 2018), Munich, Germany, 8–14 September 2018; pp. 101–117.
21. Li, B.; Yan, J.; Wu, W.; Zhu, Z.; Hu, X. High Performance Visual Tracking with Siamese Region Proposal Network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018), Salt Lake City, UT, USA, 18–23 June 2018; pp. 8971–8980.
22. Li, B.; Wu, W.; Wang, Q.; Zhang, F.; Xing, J.; Yan, J. SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 4282–4291.
23. Wang, Q.; Zhang, L.; Bertinetto, L.; Hu, W.; Torr, P. Fast Online Object Tracking and Segmentation: A Unifying Approach. In Proceedings of the 32nd IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 15–20 June 2019; pp. 1328–1338.
24. Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; Ng, A. ROS: An Open-Source Robot Operating System. In Proceedings of the IEEE International Conference on Robotics and Automation, Workshop on Open Source Software (ICRA 2009), Kobe, Japan, 12–17 May 2009; pp. 1–6.
25. Tello Edu. Ryze Robotics. Available online: https://www.ryzerobotics.com/zh-tw/tello-edu?site=brandsite&amp;from=landing_page (accessed on 18 November 2023).
26. YOLOv4 Baseline Training. Available online: https://github.com/AlexeyAB/Darknet (accessed on 1 June 2023).
27. Zhang, P.; Zhong, Y.; Li, X. SlimYolov3: Narrower, Faster, and Better for Real-Time UAV Applications. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshop (ICCVW 2019), Seoul, Republic of Korea, 27–28 October 2019; pp. 37–45.
28. Koch, G.; Zemel, R.; Salakhutdinov, R. Siamese Neural Networks for One-Shot Image Recognition. In Proceedings of the International Conference on Machine Learning Deep Learning Workshop (ICML 2015), Lille, France, 6–11 July 2015; pp. 1–8.
29. Bromley, J.; LeCun, Y. Signature Verification Using a "Siamese" Time Delay Neural Network. In Proceedings of the Advances in the 6th Neural Information Processing Systems, Denver, CO, USA, November 1993; pp. 737–744.
30. Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. MicROSoft COCO: Common Objects in Context. In Proceedings of the 13th European Conference on Computer Vision (ECCV 2014), Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
31. Drone-Face-Tracking. Available online: https://github.com/murtazahassan/Drone-Face-Tracking (accessed on 12 March 2023).