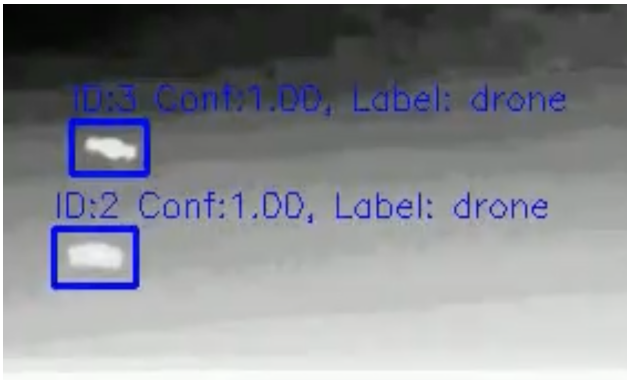

Eye in the Sky

Vision System for Automatic Detection and Tracking of Flying Objects

Bachelor's thesis in Computer Science and Engineering

Edvin Karlsson
Eric Sångberg Karlsson
Simon Wiman
Alice Olson
Ida Kjellerstedt
Jonathan Alm

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Bachelor's thesis 2026

Eye in the Sky

Vision System for Automatic Detection and Tracking of Flying Objects

Edvin Karlsson
Eric Sångberg Karlsson
Simon Wiman
Alice Olson
Ida Kjellerstedt
Jonathan Alm

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Eye in the Sky
Vision System for Automatic Detection and Tracking of Flying Objects
Edvin Karlsson Eric Sångberg Karlsson Simon Wiman Alice Olson Ida Kjellerstedt
Jonathan Alm

Supervisor (Handledare): Arne Linde, Department of Computer Science and Engineering
Examiners: Arne Linde, Department of Computer Science and Engineering, Patrik Jansson, Department of Computer Science and Engineering
Graded by teacher (Rättande lärare): Örjan Askerdal, Department of Computer Science and Engineering

Bachelor's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image showing two detected drones in gazebo simulation of system.

Typeset in LaTeX
Gothenburg, Sweden 2026

Eye in the Sky
Vision System for Automatic Detection and Tracking of Flying Objects
Edvin Karlsson, Eric Sångberg Karlsson, Simon Wiman, Alice Olson, Ida Kjellerstedt, Jonathan Alm

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The widespread deployment of Unmanned Aerial Vehicles (UAVs) in both civilian airspace and military operations has made onboard perception an increasingly critical capability. Detecting other airborne objects from a moving platform is particularly challenging: targets are often small, the camera itself is in motion, and IR imagery introduces additional complexity due to scarce training data. This thesis presents a modular system designed to handle these constraints in real-time, using both RGB and IR cameras mounted on a UAV.

Object detection is handled by RF-DETR, a transformer-based detector suitable for a real-time application. For tracking, we extend ByteTrack with optical-flow-based ego-motion compensation, which proves important when the camera platform itself is moving. Classification relies on a CLIP-based model that we adapt to IR through cross-modal knowledge distillation combined with Low-Rank Adaptation (LoRA) fine-tuning, allowing us to transfer visual representations from RGB to IR without requiring pretraining on IR imagery.

Experiments on simulated UAV imagery show that the RGB detector achieves an F1 score of 0.924 and the IR detector 0.824, despite the latter having access to substantially less training data. The classifier adaptation has a large effect on drone recall, which increases from 0.252 to 0.654 in RGB mode and from 0.049 to 0.617 in IR mode. Tracking performance, measured with the HOTA metric, reaches around 20 in RGB mode and 14 in IR mode, with most errors caused by long occlusions and very distant targets.

The complete pipeline runs at approximately 21 fps on a standard laptop GPU, suggesting that it is capable of real-world deployment. The initial results are promising and indicate that the proposed system can handle several real-world constraints. However, two key limitations remain: performance drops on far-range targets, and all evaluation is performed in simulation with pseudo-IR imagery, leaving a gap to real-world deployment.

Sammandrag

Den omfattande användningen av obemannade luftfarkoster (UAV:er) inom både civilt luftrum och militära operationer har gjort förmågan att uppfatta och tolka omgivningen allt mer avgörande. Att detektera luftburna objekt från en rörlig plattform är en utmanande uppgift, eftersom målen ofta är små och kameran är i rörelse. Detta kandidatarbete presenterar ett modulärt visionsystem utformat för att hantera dessa begränsningar i realtid med hjälp av både RGB- och IR-kameror monterade på en UAV. Användningen av IR-bilder introducerar ytterligare komplexitet på grund av begränsad träningsdata.

Objektdetektion utförs med RF-DETR, en transformerbaserad modell som är lämplig för realtidssystem. För spårning används en utökad version av ByteTrack, där egorörelse kompenseras med hjälp av optiskt flöde. Klassificeringen baseras på en CLIP-modell som anpassas till IR-domänen genom korsmodal kunskapsdestillering i kombination med Low-Rank Adaptation (LoRA)-finjustering. Detta möjliggör överföring av visuella representationer från RGB till IR utan att modellen behöver vara förtränad på IR-data.

Utvärdering på simulerad UAV-data visar att RGB-modellen uppnår ett F1-värde på 0,924 och IR-modellen 0,824, trots att den senare har tillgång till betydligt mindre träningsdata. Anpassningen av klassificeringsmodellen har en stor påverkan på återkallningen för drönarklassen, som ökar från 0,252 till 0,654 i RGB-läge och från 0,049 till 0,617 i IR-läge. Spårningsprestanda, mätt med HOTA-måttet, uppgår till omkring 20 i RGB-läge och 14 i IR-läge, där de flesta fel orsakas av långvariga ocklusioner och mål på långa avstånd.

Det kompletta systemet uppnår cirka 21 FPS på en normal GPU-utrustad laptop, vilket indikerar potential för realtidsanvändning i praktiska tillämpningar. De initiala resultaten är lovande och visar att det föreslagna systemet kan hantera flera verkliga begränsningar. Två centrala begränsningar kvarstår dock: prestandan försämras för mål på långt avstånd, och all utvärdering har genomförts i en simulerad miljö med pseudo-IR-data, vilket innebär att det fortfarande finns ett gap till verklig driftsättning.

Keywords: Multi-object Tracking, Airborne Object Detection, Zero-shot Classification, Cross-modal Knowledge Distillation, Computer Vision, RF-DETR, CLIP, ByteTrack, Airspace monitoring

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Arne Linde, for his continuous support and the many valuable discussions throughout this project. We also wish to thank Mikael Skillt at Evolved Aerospace for his guidance and for providing access to GPU resources.

Edvin Karlsson, Eric Sångberg Karlsson, Simon Wiman, Alice Olson, Ida Kjellerstedt, Jonathan Alm, Gothenburg, June 2026

Contents

List of Figures

List of Tables

Glossary

Bounding box A rectangular region marking the location and spatial extent of a detected object.

ByteTrack A multi-object tracker following the tracking-by-detection paradigm that retains low-confidence detections by matching them to unmatched tracklets.

Cosine similarity A measure of similarity between two vectors based on the angle between them.

Cross Modal Knowledge Distillation (CMKD) Cross-modal Knowledge Distillation; knowledge distillation where teacher and student operate on different sensory modalities, such as RGB to IR.

Ego-motion Motion of the scene caused by movement of the camera platform itself, rather than by the objects observed.

Hungarian Algorithm An algorithm which solves the assignment problem in polynomial time.

IoU Intersection over Union, a metric that measures the overlap between two boxes.

IR Infrared (thermal) image format capturing radiation beyond the visible red spectrum, often used ot sense heat.

Kalman filter A recursive estimation method used to predict the state of a system from uncertain observations.

Knowledge Distillation (KD) A technique in which a student model is trained to replicate the behaviour of a teacher model.

LoRA Low-Rank Adaptation; a parameter-efficient fine-tuning method that injects small trainable low-rank matrices while keeping the original weights frozen.

RANSAC Random Sample Consensus, an iterative algorithm used to estimate model parameters while minimizing the influence of outliers and noise.

RF-DETR RoboFlow Detection Transformer, a real-time transformer detector .

RGB A standard colour camera or image format capturing visible light across red, green, and blue channels.

SORT Simple online and realtime tracking, a lightweight multi object tracking algorithm.

UAV Unmanned Aerial Vehicle; an aircraft operated without an onboard human pilot.

YOLO You Only Look Once, a real time object detection algorithm that locates multiple objects in an image using a single pass through a neural network.

Zero-shot classification Classifying categories never seen during training by mapping inputs into a shared semantic space.

1
Introduction

Unmanned Aerial Vehicles (UAVs), commonly referred to as drones, were introduced in the early 20th century [1]. Since then, through breakthroughs in material and electronics, they have become one of the most significant technologies of the modern era.

The modern UAV industry is constantly evolving, currently undergoing a significant transformation across military, commercial and consumer sectors [2, 3, 4]. This transformation is driven by several factors, including advancements in large language models and other foundation models that enable enhanced autonomous intelligence. Machine learning model families like YOLO [5] and Grounding DINO [6] provide high-speed localization and language-to-vision grounding, allowing UAVs to interact with their surroundings by identifying and responding to specific environmental cues. Parallel to these software advancements, the mass production of smartphones has created a massive market for micro-electro-mechanical systems [7]. The resulting availability of low-cost, off-the-shelf sensors has drastically reduced the cost of producing small UAVs, further accelerating their widespread adoption.

While these technological advancements can be utilized in various industries such as healthcare [3] or agriculture [4], they can also be used for malicious purposes. The dual-use of UAVs has enabled applications ranging from tactical deployments as seen in Ukraine [8] to espionage and smuggling [9]. This rapid development and dual-use of UAVs has sparked a corresponding acceleration in development of anti-UAV technologies [10].

Current anti-UAV capabilities, which typically rely on a combination of detection, classification and tracking, often struggle in complex or cluttered environments [10]. To address these challenges, several strategies have been proposed, for instance some advocate for RGB-infrared (RGB-IR) sensor fusion [11] or RGB-LiDAR fusion [12], while others explore transformer-based architectures [13]. However, the current approaches often require significant hardware optimization or extensive dataset curation to achieve high real-world performance, resulting in high costs. Recent attempts at lightweight Mobile Vision Transformers have emerged to address this [14], yet existing approaches often suffer from high computational latency [10], making them difficult to deploy on resource-constrained real-world systems.

Furthermore, many state-of-the-art models fail to maintain robust performance in scenarios where detection quality degrades due to extreme conditions [15], such as

fog and rain that affects visible texture and clarity. Furthermore, they are generally trained on large-scale RGB datasets [16], limiting cross-modal applications. Consequently, a gap remains in creating multi-modal systems that utilizes a modular detection-classification-tracking pipeline for real-time onboard processing on resource-constrained drone platforms, particularly for complex environmental conditions and multi-object tracking.

1.1 Purpose

The purpose of this thesis is to develop a modular, real-time vision system for automatic detection, classification, and multi-object tracking of airborne targets from a moving UAV platform, using both RGB and IR sensors. A central aim is to bridge the domain gap between RGB-pretrained models and IR deployment, while maintaining sufficient computational efficiency for onboard real-time processing. The system is designed to remain robust under ego-motion, partial occlusions, and varying target distances.

1.2 Delimitations

The scope of this project is defined by several important delimitations that reflect both practical constraints and deliberate design choices. Firstly, the system is designed with the understanding that it will not attempt to maintain persistent identification of objects that move outside the camera frame. In other words, if an unidentified flying object, such as a drone or bird, leaves the field of view, the system is not expected to reacquire or track its identity upon re-entry. This limitation is due to the complexity of object re-identification.

Secondly, the system is constrained by its reliance on optical sensors alone. Neither Radar nor LiDAR technologies will be incorporated for the purposes of detection, classification, or tracking. This reflects a conscious choice to focus on vision-based approaches, leveraging RGB and IR data exclusively. The exclusion of radar and LiDAR is based on resource and technical considerations.

The system is required to operate in an online manner, meaning that all processing is performed in real-time using only current and past frames from the input stream. No future-frame information is permitted. This online constraint is fundamental to the intended application of the system in the real world.

Finally, subclassification of aerial objects is outside the scope of this work. The project focuses on broad object categories rather than identification of specific drone or aircraft models.

1.3 System Requirements

The system requirements define the essential functional and performance capabilities of the proposed UAV vision system. The requirements were derived from a list of criteria specified by Evolved Aerospace. These requirements are divided into functional and non-functional categories to distinguish what the system shall do from how it shall perform.

1.3.1 Functional requirements

The functional requirements specify the tasks and operations that the system is expected to fulfill. The system shall be capable of processing both RGB and IR video streams. It shall detect airborne objects within the camera field of view and perform classification of detected objects into predefined categories, including drones, birds, airplanes, and helicopters. Although multiple categories are considered, the main focus of the classification component is accurate drone detection. The system shall support multi-object tracking, enabling multiple airborne objects to be detected, classified, and tracked within the same frame while maintaining individual tracking identities throughout their visibility in the scene.

The system shall output detection and tracking data for each detected object, including bounding box coordinates, predicted class labels, tracking IDs, and associated confidence values. The system shall also generate an annotated video output containing visualized bounding boxes, predicted class labels, confidence scores, and tracking IDs for all tracked objects.

1.3.2 Non-functional requirements

The non-functional requirements specify how the system is expected to perform during operation. The system shall process video in real time, maintaining a sufficient inference speed to ensure that incoming frames are processed without delay or backlog, thereby ensuring low latency. The system shall maintain stable performance under varying environmental conditions, including changes in illumination, camera motion, object distance, and partial occlusions. Furthermore, the system shall be robust to false positives and missed detections, ensuring consistent tracking performance over time. Finally, the system shall be computationally efficient, enabling deployment on hardware with limited resources while maintaining acceptable detection accuracy.

1.4 Method

To satisfy the purpose and system requirements, the work was divided into concrete steps and sub-steps. First, an extensive literature review was conducted to obtain a broad overview of the subject and the tasks to be performed. The initial search was intentionally wide in scope to build a high-level understanding before focusing on specific details. This broad review provided the necessary background to make

informed critical design choices.

Based on the synthesized information, the system was decomposed into three subcomponents: detection, tracking, and classification. This modular structure was chosen to improve development efficiency and to simplify testing and debugging. Each subcomponent was tested continuously throughout development to ensure that the project progressed according to the planned timeline.

Once all subcomponents were implemented, they underwent extensive individual testing before being integrated into the complete system. Finally, comprehensive end-to-end tests were performed on the full pipeline to assess whether it fulfilled the stated purpose and system requirements.

This thesis adheres to Chalmers' guiding principles regarding academic integrity and the use of AI tools [17]. In addition to being integrated into the project pipeline, AI tools were used as an aid for literature searches, for guidance on grammar, and for assistance in debugging code. Since AI-generated content can include incorrect or biased information, all outputs were critically assessed by cross-referencing against original sources, relevant literature, and manual testing before being incorporated.

2
Theory

This chapter provides the technical theory necessary to understand the individual components of the system. The three main components, detection, classification, and tracking, are each treated in their own section, beginning with a brief introduction to the subproblem, followed by a review of how prior models have approached it. Any additional background theory required to understand the remaining report is covered.

2.1 Detection

Object detection aims to identify objects within an image by predicting bounding boxes and class labels [18], a task that is particularly challenging when detecting UAVs due to their size, high speed, and long-range observation conditions [19]. At extended distances, UAVs typically occupy only a few pixels in an image, placing the problem within the domain of small object detection. Although there is no strict definition, small objects are commonly considered to be those with spatial dimensions smaller than 32×32 pixels [20]. As a result, they contain limited spatial and contextual information, making accurate detection significantly more difficult compared to larger objects.

UAV detection is further complicated by variations in appearance and environmental conditions [19]. UAVs vary widely in shape, size, color, material, and orientation, ranging from micro-drones to larger platforms. Environmental factors such as illumination changes, shadows and weather variations can further affect visibility. For example, bright sunlight may introduce strong contrasts, while overcast conditions or adverse weather such as fog, rain, or snow can obscure objects and reduce distinguishability. Although aerial scenes often contain relatively simple backgrounds, such as open sky, UAV detection remains challenging due to rapid motion, scale changes, and unpredictable trajectories. These factors can introduce motion blur and temporal inconsistencies, while partial occlusions from clouds or other objects can further degrade detection performance.

Traditional computer vision methods based on hand-crafted features were the primary approach to object detection and performed well on smaller, more constrained datasets, but struggled to generalize to complex real-world scenarios [21]. With the introduction of deep learning in computer vision, significant improvements in detection performance were achieved. Deep learning models learn feature repre-

sentations directly from data, enabling them to better handle large-scale datasets and complex visual variations. As a result, modern object detection systems have largely shifted toward deep learning-based approaches due to their improved accuracy and robustness. In particular, convolutional neural networks (CNNs) have become widely adopted for object detection and image recognition tasks due to their ability to learn hierarchical feature representations from visual data. Despite their strong performance, CNN-based detectors are limited in capturing global context and long-range relationships, which has led to an increasing interest in transformer-based models that leverage self-attention to model global reasoning across image regions [22].

Among these transformer-based models, Detection Transformers (DETR) formulate object detection as a set prediction problem [18], where a fixed set of learned object queries directly predict bounding boxes and class labels. This removes the need for hand-crafted components such as anchor boxes and non-maximum suppression, which are typically required in CNN-based detectors like YOLO. However, limitations of the original DETR model, including slow convergence and poor small-object performance, have motivated several improved variants[22].

2.1.1 RF-DETR

RoboFlow Detection Transformer (RF-DETR) is a real-time object detection model developed by Roboflow, based on the DETR family of transformer-based detectors [23]. It builds upon Lightweight Detection Transformer (LW-DETR) and consists of a Vision Transformer (ViT) encoder, a projector, and a DETR decoder. RF-DETR is designed to improve both accuracy and efficiency of DETR-based detectors through architectural simplifications and the use of neural architecture search (NAS).

RF-DETR simplifies LW-DETR's architecture and training process to improve generalization across diverse target domains [23]. The most notable change is the replacement of the original CAEv2 backbone with DINOv2, a self-supervised ViT pretrained on a large quantity of curated data, whose rich feature representations improve detection accuracy [24]. The ViT backbone extracts multi-scale features from the input image [23], producing feature maps that are passed through a projector before being processed by the decoder. Each decoder layer consists of self-attention, deformable cross attention, and a feed-forward network. The deformable attention mechanism focuses computation on relevant image regions instead of processing the entire image uniformly [25]. This reduces computational cost while still preserving both local detail and global context. Unlike LW-DETR that uses three decoder layers, RF-DETR treats the number of decoder layers as a tunable parameter, enabling different accuracy-latency trade-offs [23]. An overview of the RF-DETR architecture is shown in Figure 2.1.

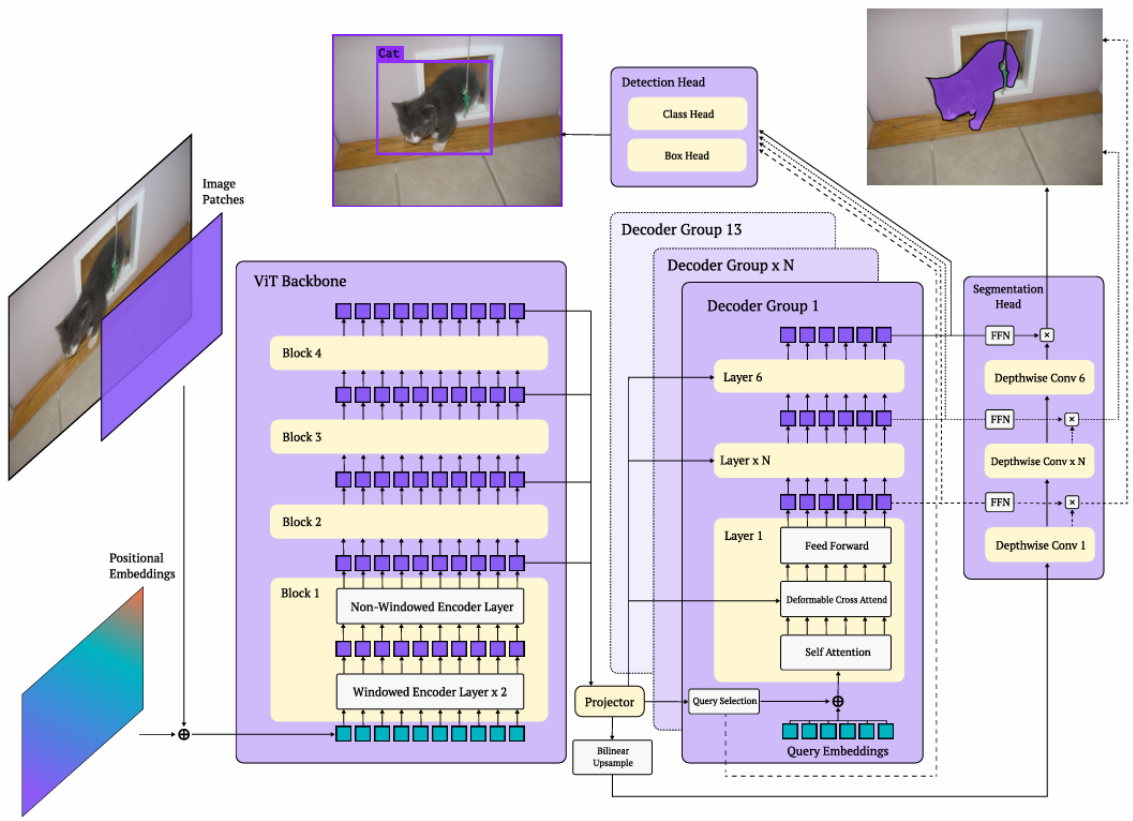

Figure 2.1: Overview of RF-DETR architecture. [23], CC-BY-4.0

A key component of RF-DETR is its use of NAS to identify Pareto-optimal model configurations [23]. NAS explores variations in architectural parameters such as image resolution, patch size, and number of decoder layers, selecting configurations that achieve an optimal trade-off between accuracy and latency. The model supports inference across multiple input resolutions by adjusting its positional embeddings at runtime, allowing lower resolution for faster inference and higher resolutions for improved accuracy without retraining [23].

RF-DETR's transformer-based architecture captures long-range dependencies and global context, enabling robust detection in complex scenes even when objects are small or partially occluded [23],[26]. The model is also designed for efficient fine-tuning on custom datasets [27]. Its DINOv2 backbone benefits strongly from large-scale pretraining, allowing effective adaptation even with limited domain-specific data. Empirical results show that RF-DETR maintains high performance when fine-tuned on small or specialized datasets, making it well suited for domains such as aerial imagery or small object detection [25]. RF-DETR is released in multiple model sizes including nano, small and large [23], enabling different trade-offs between speed and accuracy depending on the application.

2.2 Classification

Object classification is the task of assigning a semantic label to a detected object. Before deep learning became dominant, this problem was typically approached using traditional computer vision algorithms such as SIFT [28], which rely on hand-engineered features. While effective on small, controlled datasets, such methods struggle to scale and generalize to real-world imagery. CNNs address this limitation by learning relevant feature representations directly from raw pixel data [29].
Modern image classification is typically framed as a supervised learning problem: given an input image x, the goal is to predict a class label y from a predefined set of K categories. A neural classifier f_θ maps the image to a vector of scores (logits), which are converted into class probabilities via a softmax layer; the predicted class is then chosen as the one with highest probability. The parameters θ are learned from a dataset of image-label pairs by minimizing a loss function, such as cross-entropy, so that the network's predictions become increasingly consistent with the ground-truth annotations.

2.2.1 Zero-shot Classification

Traditional classification methods require labeled training examples for every category the model is expected to recognize [30]. This closed-world assumption becomes a significant limitation in practice due to the extensive need to collect data for every possible class, or the model will fail on any category not seen during training. Zero-shot classification addresses this by learning a mapping of inputs to a semantic space, rather than directly to class labels [31]. The idea was further developed by Lampert et al. [32], who demonstrated that classes could be described through visual attributes such as 'has wings' or 'has rotors', enabling transfer of knowledge from seen to unseen categories. Given a semantic description of an unseen class (e.g. a natural language prompt), a zero-shot classification model can classify inputs into that class without having observed any examples during training. Formally the model learns a function $f : \mathcal{X} \to \mathcal{S}$, where $\mathcal{S}$ is a shared semantic space, and classification is performed by

$$\hat{y} = \arg\max_{c \in \mathcal{C}} \text{sim}(f(x), s_c) \tag{2.1}$$

where s_c is the semantic representation of class c and sim denotes a similarity measure such as cosine similarity. In other words, instead of labels, the model learns semantic attributes which can be transferred across classes. This is what makes it possible to classify classes the model has not seen during training.

2.2.2 CLIP

Contrastive Language-Image Pre-training (CLIP), introduced by Radford et al. [33], is a multimodal framework that enables zero-shot classification by aligning images and natural language in a shared embedding space. Instead of training on a fixed set of labels, CLIP is trained on large-scale, noisy image-text pairs collected from the web, enabling classification of arbitrary categories through natural language

prompts without task-specific supervised training.

The architecture consists of two separate encoders: a text encoder and an image encoder. The text encoder is a Transformer-based model (similar to GPT-2) that maps an input prompt to a fixed-dimensional embedding. The image encoder is a ViT or a ResNet that maps an input image to an embedding of the same dimensionality. Both encoders are trained jointly so that corresponding text-image pairs are mapped to nearby points on the shared embedding space, while mismatched pairs are pushed apart.

Training uses a contrastive objective. For each minibatch of N images and N text descriptions, the image and text embeddings are compared using a cosine similarity to form an $N \times N$ similarity matrix. The model is optimized so that the similarity between each image and its correct caption is higher than the similarity between that image and all other captions in the batch, and vice versa. This is implemented using a symmetric cross-entropy loss over these similarities [33].

Once trained, CLIP can perform zero-shot classification by converting class names into natural language prompts and comparing them with the image embedding. For a given set of class labels, each label is transformed into one or several natural language prompts (e.g., "a photo of a {label}"). These are encoded by the text encoder, and the resulting text embeddings act as classifiers in the joint embedding space. The image is encoded by the image encoder, and the predicted class is chosen as the label whose text embedding has the highest cosine similarity with the image embedding. This allows CLIP to adapt to new classification tasks simply by changing the text prompts, without retraining the underlying model.

2.2.3 Image Encoder

The image encoder in CLIP maps an input image to a fixed-dimensional vector in the same joint embedding space as the text encoder. In the original CLIP work, several architectures were considered, including ResNet-style convolutional network and Vision Transformers (ViT) [34]. In this work, we use the ViT-B/32 variant of CLIP.

Input representation and patch embedding

The image encoder operates on fixed-size RGB images. Let the input be

$$x \in \mathbb{R}^{H \times W \times C}, \tag{2.2}$$

where H and W denote the spatial height and width, and $C = 3$ is the number of channels. The image is partitioned into a grid of non-overlapping patches of size $p \times p$. Each patch is flattened and projected to a d_{model}-dimensional embedding using a learned linear projection (often implemented as a convolution with kernel size $p \times p$ and stride p).

This yields a sequence

$$(z_1^0, z_2^0, \ldots, z_N^0), \quad N = \frac{H}{P} \cdot \frac{W}{P}, \tag{2.3}$$

where $z_i^0 \in \mathbb{R}^{d_{\text{model}}}$ is the embedding of the i-th patch. Following ViT [34], a learnable class token embedding $z_{\text{CLS}}^0 \in \mathbb{R}^{d_{\text{model}}}$ is prepended to this sequence. The resulting token sequence

$$Z^0 = (z_{\text{CLS}}^0, z_1^0, \ldots, z_N^0) \in \mathbb{R}^{(N+1)\times d_{\text{model}}} \tag{2.4}$$

is augmented with a learned positional embedding

$$E_{\text{pos}} \in \mathbb{R}^{(N+1)\times d_{\text{model}}}, \tag{2.5}$$

which is added element-wise:

$$H^{(0)} = Z^0 + E_{\text{pos}}, \tag{2.6}$$

The positions are fixed with respect to the input grid, and the class token aggregates a global representation of the image.

Transformer encoder structure

The image encoder consists of a stack of L Transformer encoder blocks. Each block applies two sub-layers, each wrapped in residual connections and layer normalization:

1. Multi-head self-attention (MHSA) over all tokens (class token + patches.)
2. A position-wise feed-forward network (MLP).

Let $X \in \mathbb{R}^{(N+1)\times d_{\text{model}}}$ denote the token representations entering a self-attention sub-layer. For each attention head, the model computes queries (Q), keys (K), and values (V) via learned linear projections,

$$Q = XW^Q. \quad K = XW^K, \quad V = XW^V, \tag{2.7}$$

where $(W^Q, W^K, W^V) \in \mathbb{R}^{d_{\text{model}}\times d_k}$, and d_k is the per-head dimension. Scaled dot-product attention [35] is then

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{2.8}$$

Multiple heads are computed in parallel, concatenated along the feature dimension, and projected back to d_{model} with an *attention output projection* W^0:

$$\text{MHSA}(X) = \Big[\text{head}_1, \ldots, \text{head}_h\Big] W^O. \tag{2.9}$$

The MLP sub-layers is a two-layer feed-forward network with a non-linearity, typically GELU [36]:

$$\text{MLP}(h) = \sigma(hW_1 + b_1)W_2 + b_2, \tag{2.10}$$

where $W_1 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ff}}}$ expands the representation to a higher-dimensional hidden space, and $W_2 \in \mathbb{R}^{d_{\text{ff}}\times d_{\text{model}}}$ projects it back down. We refer to W_1 as the MLP *up-projection* and W_2 as the MLP *down-projection*.

Global image representation

After the L Transformer blocks, we obtain the final token sequence

$$H^{(L)} = (h_{\text{CLS}}^{(L)}, h_1^{(L)}, \ldots, h_N^{(L)}). \tag{2.11}$$

As in ViT and CLIP, the hidden state of the class token serves as a global image descriptor:

$$h_{\text{CLS}}^{(L)} \in \mathbb{R}^{d_{\text{model}}}. \tag{2.12}$$

This vector is mapped into the joint image-text embedding space via a learned linear projection

$$z_{\text{image}} = W_{\text{image}}\, h_{\text{CLS}}^{(L)}, \tag{2.13}$$

where $W_{\text{image}} \in \mathbb{R}^{d \times d_{\text{model}}}$ and d is the shared embedding dimension used by both encoders. Both image and text embeddings are ℓ_2-normalized prior to computing cosine similarities during contrastive pretraining and downstream classification. An overview of the ViT architecture is shown in Figure 2.2.

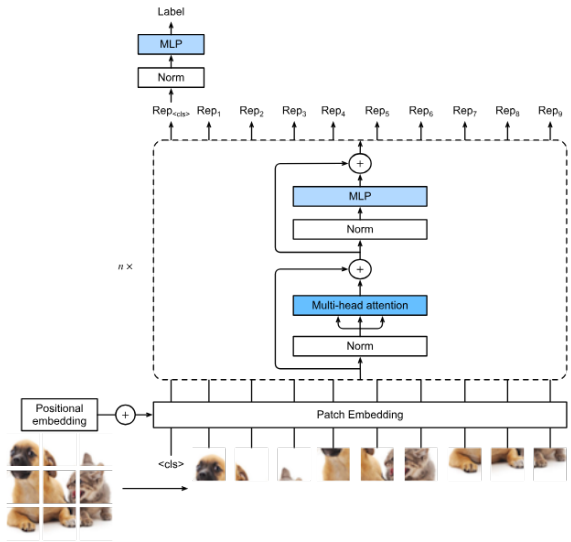

Figure 2.2: Overview of ViT architecture. Source: [37]. CC-BY-SA-4.0

2.2.4 Cross-modal Knowledge Distillation

Knowledge distillation (KD) is a machine learning technique in which a smaller student model is trained to replicate the behavior of a larger teacher model [38]. It is commonly used for model compression, enabling complex capabilities to run on resource-constrained devices [39]. The core idea was introduced by Bucilia et al. in the paper *Model Compression* [40], while the method and term "Knowledge Distillation" were popularized by Geoffrey Hinton, Oriol Vinyals and Jeff Dean in the paper *Distilling the Knowledge in a Neural Network* [41]. Typically, a high-capacity teacher is first trained on a dataset until it reaches peak performance.

There are several ways to transfer knowledge to the student [38]. In response-based distillation, the student learns to match the teacher's softened output probabilities, the soft targets that encode the teacher's internal relational structure between classes. In intermediate distillation, the student is trained to mimic the teacher's internal feature representations at intermediate layers, providing a richer supervision signal than output probabilities alone. In relation-based distillation, the student learns to reproduce the pairwise structural relationships between samples in the teacher's representation space, preserving notions of which samples are more similar to each other.

Cross-modal knowledge distillation (CMKD) [42] extends KD to the case where teacher and student operate on different sensory modalities. In standard KD, teacher and student process the same modality, so knowledge transfer can rely directly on response-based or intermediate distillation. In CMKD, the modality gap fundamentally breaks this assumption. The student receives different sensory evidence than the teacher, making direct feature matching ineffective since the same object can appear radically different across modalities.

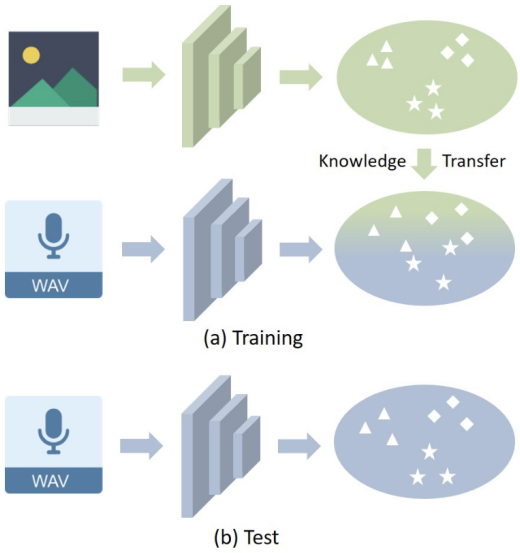

Figure 2.3: Cross-modal knowledge, image-to-audio example. During training (a), an image-based teacher model transfers knowledge to a speech-based student model, aligning their feature spaces. At test time (b), only the student receives speech input, yet benefits from the structured representations learned via distillation. Source: [43], CC-BY-4.0.

Common approaches to bridge this gap include feature alignment losses that minimize the distance between modality-specific embeddings [42, 44], and adversarial losses that learn domain-invariant representations [45]. A further distinction arises between closed-set CMKD, where training and test classes are the same, and zero-shot CMKD, where the student must generalize to unseen classes at test time. In

closed-set CMKD, the objective is simply to align the student's features with the teacher's so that task-specific classifiers transfer correctly. Zero-shot CMKD makes this more complex, the student must also preserve the semantic structure of a joint vision-language embedding space in order to classify unseen categories through natural language prompts.

2.2.5 Low-Rank Adaptation

Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning technique for large neural networks, originally introduced in the context of adapting large language models [46]. Instead of updating all parameters of a pretrained model, LoRA keeps the original weights frozen and introduces small trainable low-rank matrices into selected layers. This drastically reduces the number of trainable parameters and the memory footprint of fine-tuning, which is particularly useful when only limited computational resources or task-specific data are available.

Formulation

Consider a linear layer with weight matrix

$$W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}} \tag{2.14}$$

mapping an input $x \in \mathbb{R}^{d_{\text{in}}}$ to an output

$$h = W_0 x. \tag{2.15}$$

In conventional fine-tuning the entries of W_0 are updated during training. In LoRA, W_0 is kept fixed and a trainable low-rank update ΔW is added:

$$h = (W_0 + \Delta W)x. \tag{2.16}$$

The update is parametrized as

$$\Delta W = BA, \tag{2.17}$$

where

$$A \in \mathbb{R}^{r \times d_{\text{in}}} \quad B \in \mathbb{R}^{d_{\text{out}} \times r} \tag{2.18}$$

and $r \ll \min(d_{\text{in}}, d_{\text{out}})$ is a low-rank hyperparameter. During fine-tuning, only A and B are optimized, while the pretrained weights W_0 remains unchanged. The forward computation in a LoRA-augmented layer can therefore be written as

$$h = W_0 x + BAx. \tag{2.19}$$

A scaling factor is often introduced to control the magnitude of the low-rank update:

$$h = W_0 x + \frac{\alpha}{r} BAx, \tag{2.20}$$

where α is a scalar hyperparameter. A common initialization sets $B = 0$, so that at the start of fine-tuning the network behaves identically to the original pretrained model and the effect of the low-rank adaptation is introduced gradually.

Application in Transformer Architectures

Transformer-based architectures, such as those used in CLIP, contain many linear projection layers. In multi-head self-attention, for example, the query, key, and value projections and the output projection are implemented as linear maps. LoRA can be applied to any subset of these projection matrices by adding low-rank updates to the corresponding weight matrices. Because only the low-rank factors are trainable, the number of additional parameters grows with the chosen rank r rather than with the full dimensionality of the underlying weight matrices. This makes it feasible to adapt large vision or vision-language models, such as CLIP with a ViT backbone, without the computational cost of full-model fine-tuning.

Advantages for Adapting Pretrained Vision Models

LoRA can be interpreted as constraining the space of permissible weight updates to a low-dimensional subspace. This constraint tends to regularize fine-tuning, which is beneficial when the downstream dataset is small or specialized, for example when focusing on a particular object categories or sensor modalities. At the same time, leaving the pretrained weights W_0 frozen preserves the broad representations acquired during large-scale pretraining, while the low-rank factors encode task- or domain-specific corrections. Since the base model remains unchanged and only the low-rank adapters are modified, different adapters can be trained for different domains or tasks and combined with the same underlying backbone without interference. In this way, Low-Rank Adaptation provides a mechanism for tailoring pretrained Transformer models to new visual domains and classification tasks, while keeping computational and data requirements manageable.

2.3 Tracking

In computer vision, the problem of object tracking can be defined as either single-object tracking or multi-object tracking (MOT). For the vision system, MOT was mainly considered because the capability to track multiple objects was desired. MOT is the problem of detecting and tracking objects of a given category in a video [47]. While the problem of detection is exclusively about locating objects in a video, tracking additionally maintains a consistent identification of objects. The output of tracking results in a set of tracks, where each track is a set of detections for a particular object. There are many frameworks for MOT, one among them being tracking-by-detection.

Tracking-by-detection is an efficient framework, yielding high accuracy and speed [47]. In the framework, an object detector first detects objects within a frame. These detections, in the form of bounding boxes, are then associated across frames, producing trajectories over time. The paradigm started with the introduction of SORT [47, 48]. The main idea behind SORT was to use a strong detector, which at the time was Faster R-CNN, along with a Kalman filter [49]. The Kalman filter is used to predict the location of the bounding box of the objects in the next frame [48].

With the input of the detections from the next frame, the intersection-over-union (IoU) is calculated for every prediction matched with every detection from the detector. The assignment of track identities is completed using the Hungarian algorithm based on the results from the IoU. With further development of stronger detectors, in particular deep learning based detectors, the tracking-by-detection paradigm has benefited further from this baseline setup [47].

2.3.1 ByteTrack

ByteTrack is a multi-object tracker, which follows the tracking-by-detection paradigm [50]. Like SORT, it employs a Kalman filter for predicting the location of tracked objects in the next frame, as well as its use of IoU and the Hungarian algorithm for the assignment task. As ByteTrack uses the tracking-by-detection framework, it is composed of two parts; its detector and its association method. The detector it uses is YOLOX and the association method is called BYTE. This separation allows BYTE to be used with other detectors. While ByteTrack has been surpassed by more recent approaches, it still provides competitive performance [47].

A key difference of BYTE is how it handles the input of detections, by handling low-confidence detections differently from high-confidence detections. Many MOT trackers in the tracking-by-detection paradigm before ByteTrack simply used a confidence threshold, where any detection below the threshold is thrown away [50]. The core idea behind BYTE is to instead utilize these low-confidence bounding-boxes by matching them with unmatched tracklets. Tracklets refer to smaller building blocks of a track. In this context, they are the existing trajectories from the previous frame that are being built. If these tracklets are not matched with high-confidence bounding-boxes, then it might be matched with a low confidence one. The confidence drop of the bounding box might be due to, e.g. occlusion or motion blur. The way the association between the low-confidence bounding-boxes and the unmatched tracklets is made is different from the high-confidence ones. Handling them differently, yet not discarding them, is the key idea behind BYTE.

An artificial situation illustrating how this works in practice is shown in Figure 2.4. Object 2 is eventually occluded by object 1, hence the confidence of the box around object 2 drops dramatically. The method which discards the low-confidence boxes has no chance to recapture the tracklet, as shown in (b). However, by associating every detection as in (c), it is able to associate the low-confidence box with the tracklet that is left from the previous frame. In fact, the situation in (b) is the result after the first stage of BYTE, where it only takes into account the detections above the threshold. The situation in (c) shows the result after also handling the detections below the threshold. At the same time, it will not make new tracklets out of the low-confidence bounding boxes, the problem that previous state-of-the-art trackers have to deal with. This makes BYTE less sensitive to change in the threshold hyperparameter, as detections below the threshold will be filtered out if they do not match a tracklet [50].

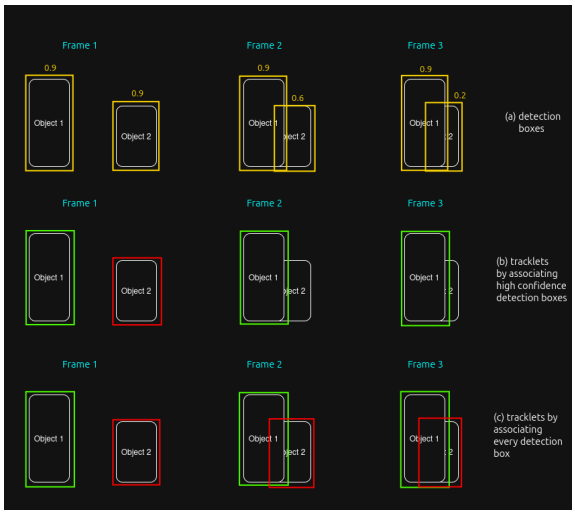

Figure 2.4: Adapted example from [50]. It illustrates the method which associates every detection box. In (a) all detection boxes are shown, as well as their confidence. (b) shows the resulting tracklets from association done on detection boxes with higher confidence only. (c) shows the resulting tracklets from association done on all detection boxes, through the method used in ByteTrack.

An issue with tracking-by-detection methods is their reliance on the predicted bounding box of the tracklets [51]. It is the overlap between the predicted bounding box and the detected bounding box that determines whether a tracklet is maintained. Since a Kalman filter is used to predict the bounding box in the subsequent frame, its performance directly influences the overall tracking quality. Consequently, inaccuracies in the Kalman filter predictions can lead to tracking failures.

The Kalman filter operates under the assumption of linear movement [52]. This results in worse performance for tracking objects of non-linear movement. As the proposed vision system is supposed to track flying objects that may exhibit non-linear movement, it is noted that performance degradation is expected in these scenarios. Moreover, the Kalman filter does not account for camera motion. The subsequent prediction can thus be inaccurate since the perspective has shifted, causing false negatives.

2.3.2 Optical Flow

When the camera is in motion, which is inherent to UAV platforms, the resulting ego-motion displaces objects in the image between frames, regardless of their own movement. Tracking models that rely on Kalman filters to predict object positions in the next frame are particularly affected by ego-motion. This displacement can cause the predicted boxes to drift away from their detected positions [53]. Since IoU between predicted and detected boxes determines whether they are associated, it can potentially cause the tracker to lose tracks or misassociate detections with tracks. One approach to address this issue is to incorporate optical flow [54].

Optical flow approximates the 2D displacement vector for pixels between two frames, computing where each pixel in frame_k corresponds in frame_{k+1} by measuring displacement of brightness in the image. It works on the assumption that pixel intensities stay the same between frames even if they move [55]. There are two main techniques for estimating optical flow: sparse optical flow and dense optical flow. Sparse optical flow computes the two-dimensional motion vectors for a selected set of pixels, typically feature points extracted from the image, such as object edges [56]. Dense optical flow calculates the vector for all pixels in the image, requiring more computation but returns a more accurate approximation of the whole frame making it more eligible for ego-motion estimation.

Estimating dense optical flow is computationally expensive, which has driven the development of lightweight deep learning models such as FastFlowNet, designed to enable real time dense optical flow estimation on resource constrained devices [57].

One way to mitigate the effect of ego-motion on tracking, is to model the background motion and use it to correct the trackers predictions. The background motion can be estimated as a transformation matrix that describes how the entire scene has shifted between frames [58]. By applying this matrix to the Kalman filters predicted boxes, the predictions are warped into the same spatial coordinates as the current frame before matching is performed.

2.3.3 Evaluation Metrics

Evaluating MOT-systems remains a challenging problem [59]. Consequently, a variety of performance metrics have been proposed to provide insight into different aspects of tracker behavior. Since MOT inherently combines object detection and data association, many metrics tend to emphasize one component at the expense of the other. For example, Multiple Object Tracking Accuracy (MOTA) is largely driven by detection performance, whereas IDF1 places greater emphasis on identity preservation and association quality.

To address this imbalance, Higher Order Tracking Accuracy (HOTA) has been introduced as a metric that explicitly balances detection and association performance [59]. HOTA jointly evaluates these aspects by integrating them into a single unified score while maintaining a more balanced contribution from each component. Additionally, HOTA decomposes into interpretable sub-metrics, enabling a more detailed analysis of tracker performance.

The output of the task of MOT is composed of three items: the objects present in each frame, the location of said objects and the association between objects across frames [59]. Correspondingly, tracking errors can be categorized into detection, localization, and association errors. Detection errors occur when the tracker produces false positives or misses ground-truth objects. Localization errors arise when predicted object positions are misaligned with the corresponding ground-truth annotations. Association errors occur when the tracker incorrectly maintains object

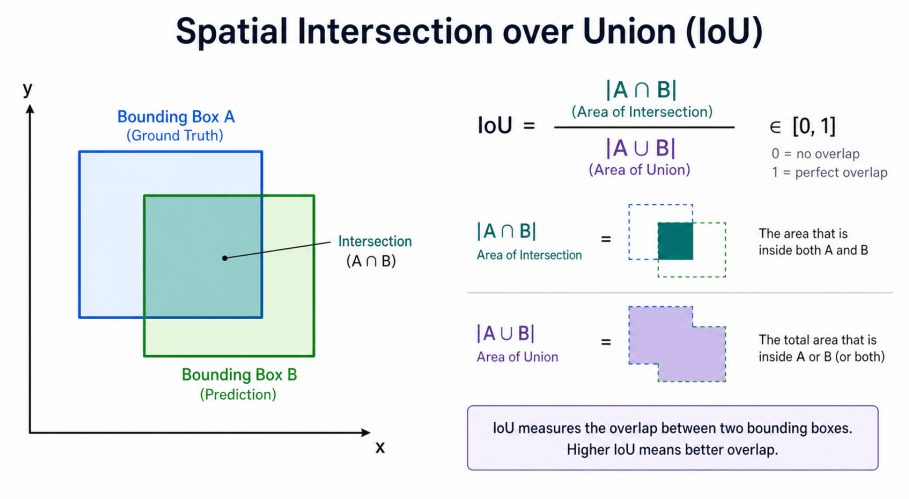

Figure 2.5: The spatial IoU between two bounding boxes (generated using GPT-5.3, 2026)

identities over time, by assigning the same identity to multiple distinct objects or by assigning multiple identities to one ground-truth identity. These three types of errors form the basis of the HOTA metric.

To compute the HOTA score, both ground-truth annotations and tracker outputs are required. Each entry must have an ID, indicating which object it is across frames. Additionally, it must have a bounding box, containing information about the objects size and position. First, a matching between prDet (predicted detections) and gtDet (ground-truth detections) are made [59]. It is performed using the localization similarity score $\mathcal{S}$, where a match occurs when $\mathcal{S} \geq \alpha$, and $\alpha \in [0, 1]$ is some threshold. A higher value of α means a more restrictive matching, with $\alpha = 1$ meaning that only perfect matches are accepted. The score $\mathcal{S}$ is calculated with the spatial IoU, using the 2D bounding box of the ground-truth data and the prediction. A visual representation can be seen in Figure 2.5. The matching selected is the one that maximizes the final HOTA score, creating a bijective matching between prDet and gtDet based on $\mathcal{S}$ and α. This matching is completed with the Hungarian algorithm. Furthermore, a pair of gtDet and prDet generated from this stage is considered a true positive (TP), the gtDet that are not matched are false negatives (FN), and all prDet not matched are false positives (FP).

To measure association, TPA (True Positive Association), FNA (False Negative Association) and FPA (False Positive Association) are introduced [59]. Given a TP detection, each of TPA, FNA and FPA are defined. Let c be a TP detection, then TPA is computed as the set of TPs that have the same gtID and prID as c.

$$\mathrm{TPA}(c) = \left\{ k \in \mathrm{TP} \;\middle|\; \mathrm{prID}(k) = \mathrm{prID}(c) \;\wedge\; \mathrm{gtID}(k) = \mathrm{gtID}(c) \right\}$$

FNA is the set of gtDets that has the same gtID as c, but different or missing prID.

$$\mathrm{FNA}(c) = \left\{ k \in \mathrm{TP} \;\middle|\; \mathrm{prID}(k) \neq \mathrm{prID}(c) \;\wedge\; \mathrm{gtID}(k) = \mathrm{gtID}(c) \right\} \cup \left\{ k \in \mathrm{FN} \;\middle|\; \mathrm{gtID}(k) = \mathrm{gtID}(c) \right\}$$

FPA is then the set of prDets that has the same prID as c, but different or missing gtID.

$$\mathrm{FPA}(c) = \left\{ k \in \mathrm{TP} \;\middle|\; \mathrm{prID}(k) = \mathrm{prID}(c) \;\wedge\; \mathrm{gtID}(k) \neq \mathrm{gtID}(c) \right\} \cup \left\{ k \in \mathrm{FP} \;\middle|\; \mathrm{prID}(k) = \mathrm{prID}(c) \right\}$$

Finally, the HOTA score for a certain α is given by:

$$\mathrm{HOTA}_\alpha = \sqrt{\frac{\sum_{c \in \mathrm{TP}} \mathcal{A}(c)}{|\mathrm{TP}| + |\mathrm{FN}| + |\mathrm{FP}|}}$$

$$\mathcal{A}(c) = \frac{|\mathrm{TPA}(c)|}{|\mathrm{TPA}(c)| + |\mathrm{FNA}(c)| + |\mathrm{FPA}(c)|}$$

The above formula accounts for how good the association is by including the association score $\mathcal{A}(c)$, while also accounting for the detection accuracy since the matching is done with respect to detections. To also account for the localization accuracy, the final score is calculated by integrating over the values of α:

$$\mathrm{HOTA} = \int_0^1 \mathrm{HOTA}_\alpha \, d\alpha \approx \frac{1}{19} \sum_{\alpha \in \{0.05, \ldots, 0.95\}} \mathrm{HOTA}_\alpha$$

Furthermore, two out of the three sub-metrics AssA_α (association accuracy), DetA_α (detection accuracy) are defined in the following way:

$$\mathrm{DetA}_\alpha = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FN}| + |\mathrm{FP}|}$$

$$\mathrm{AssA}_\alpha = \frac{1}{|\mathrm{TP}|} \sum_{c \in \mathrm{TP}} A(c)$$

Another aspect of evaluation is how well a tracker maintains consistent trajectories without interruptions. One metric capturing this property is the number of track fragmentations (Frag) [60], which counts how often a previously active track is interrupted and later resumed. Finally, identity consistency can be assessed using the number of identity switches (IDSW) [60]. This metric measures how many times a tracked trajectory changes its assigned identity from one ground-truth object to another over time. The assignment between predicted and ground-truth boxes for Frag and IDSW follows a similar procedure as in HOTA, using the Hungarian algorithm with IoU as the matching criterion. A threshold of 0.5 is applied, such that any pair with IoU below this value is not considered a valid match.

3

Approach

This chapter describes the implementation of each stage in the proposed pipeline, as well as the methodology used for system evaluation. The complete pipeline consists of four sequential stages: ego-motion estimation, object detection, tracking, and classification, as illustrated in Figure 3.1. The system operates in two modes: one for RGB frames and one for IR frames. While the overall pipeline remains the same for both modes, certain components are replaced with their respective RGB- or IR-compatible counterparts. These modes enable the system to function under varying lighting conditions, independent of external illumination. For instance, the IR mode can be used in low-light or nighttime scenarios, where capturing sufficient RGB data is challenging.

For the detection stage, RF-DETR is employed, and the fine-tuning procedure is described. The classification stage utilizes CLIP, for which the training procedure is presented. In the tracking stage, ByteTrack is integrated with the detector, and the modifications made to ByteTrack are outlined. Additionally, ego-motion compensation using FastFlowNet is incorporated to improve tracking during camera motion. The chapter then describes the overall inference pipeline and how the individual components interact during runtime. Finally, the simulated dataset used for evaluation is presented.

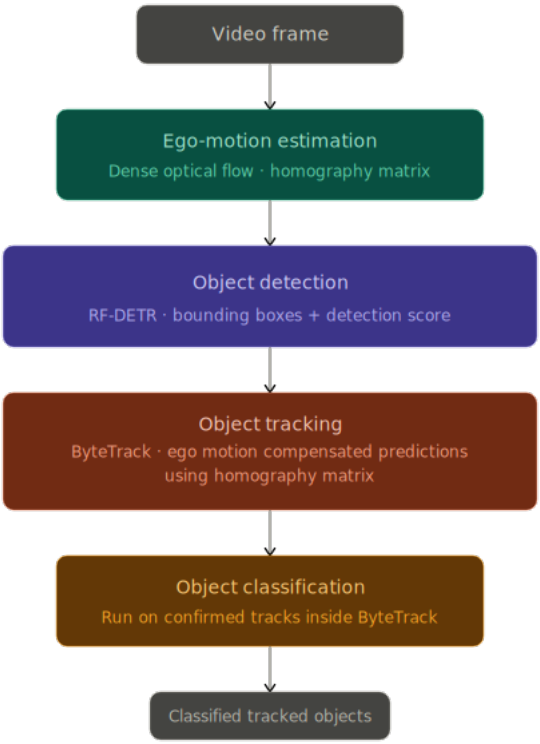

Figure 3.1: Pipeline overview - (generated using Claude Sonnet 4.6, 2026)

3.1 Detection Pipeline and Fine-Tuning

Object detection is performed using the RF-DETR Small model operating in inference mode. RF-DETR was selected because it provides strong performance within transformer-based detection models, while still being suitable for real-time applications. In this system, reliable detection of small and fast-moving airborne objects is critical, making strong feature representation at small scales particularly important. The Small variant was chosen to balance detection accuracy and computational efficiency. Since the system is intended for real-time applications, the model must maintain low latency while still providing sufficiently accurate detections to support downstream processing. For each video frame, the input image is converted to the required format and passed to the detector, which predicts bounding boxes and corresponding confidence scores for objects of interest. The resulting detections are returned as image-coordinate bounding boxes with associated confidence values.

To further reduce computational cost, a two-mode detection approach was explored during development, in which a motion-based module using frame differencing and optical flow first identified regions of interest before passing them to the detector. In practice this did not yield a net improvement, as the transformer-based architecture of RF-DETR does not scale efficiently with reduced input regions, and the overhead of the additional processing steps outweighed any computational savings from cropping. Detection was therefore run on full frames throughout.

RF-DETR provides pretrained weights trained on the Microsoft COCO dataset. To further improve detection performance for the target application, the pretrained RF-DETR Small model was fine-tuned separately for the IR and RGB pipelines using the training framework provided by Roboflow [61], in a cloud environment equipped with dual NVIDIA T4 GPUs.

For the IR pipeline, the model was fine-tuned on the Birds-IR dataset by Mohammad Bilal available on Roboflow Universe [62] to improve object detection performance on IR images. Birds-IR contains more than 2700 images consisting primarily of single-object drone scenes, along with a limited number of background-only images. To improve robustness against false positives and enhance the detection of multiple objects within the same frame, the dataset was extended in two ways. First, an additional 500 background images generated from the simulations were added to the dataset. Second, approximately 2000 synthetic images were generated using copy-paste augmentation, where objects of interest were duplicated and inserted into additional locations within the same image. These modifications increased scene complexity and improved the model's ability to detect multiple targets simultaneously.

For the RGB pipeline, the model was fine-tuned using the UAV Computer Vision Model dataset from Roboflow Universe [63]. This dataset contains more than 8000 images of drones captured in a variety of environments, scales, and distances, including both close-range and long-range views. Some images also contained multiple drones within the same frame, enabling learning under multi-object conditions. Fine-tuning on this dataset was used to reduce instances where the detector produced separate bounding boxes for individual drone components, such as propellers and the main body, instead of a single bounding box around the entire drone.

3.2 Classification training

The system relies on two complementary vision modalities: an RGB camera and an IR camera. To enable classification in the IR pipeline, we introduce an IR-specific preprocessing stage and adapt CLIP to IR imagery.

To systematically investigate how CLIP can be leveraged for IR data, we develop four model variants, ranging from straightforward fine-tuning with a single target class to more advanced cross-modal knowledge distillation configurations. The following sections describe this adaptation process in detail.

The training pipeline is implemented in Python using PyTorch. The dataset is split into 80% training and 20% testing. All images are preprocessed and converted to tensors prior to training to avoid expensive on-the-fly transformations. Label embeddings are precomputed using CLIP's text encoder so that the same text prompts do not need to be re-encoded at every epoch. For the CMKD models, a custom dataset class constructs paired RGB-IR samples. To improve numerical stability,

the logit scale is clamped to avoid excessively large values.

Model parameters are optimized using the AdamW optimizer, following the original CLIP configuration, but with one modification: the weight decay has been reduced from 0.2 to 0.01. On a smaller dataset than CLIP's original training data, this reduction prevents the optimizer from over-shrinking the pretrained weights while still providing regularization. The learning rate follows a cosine annealing schedule, decaying from the initial to one tenth of it over the course of training, which reduces oscillations near convergence without manual step tuning.

For all CMKD model variants, rather than fine-tuning the full image encoder, only the last six of the twelve Transformer blocks are set as trainable, together with the final layer normalization and the projection layer. The first six blocks are kept frozen to preserve low-level visual structure learned during CLIP pretraining, while allowing the later blocks to adapt to the IR domain. This also reduces the risk of catastrophic forgetting, which is important when fine-tuning on a dataset orders of magnitude smaller than the original CLIP training data.

Hyperparameter optimization is performed using the Weights&Biases framework to manage, track, and compare different configurations. By specifying both the set of hyperparameters and their ranges, Weights&Biases automatically explores combinations to identify well-performing settings. Early stopping is employed to terminate underperforming runs and avoid unnecessary computation. All experiments were conducted on a single NVIDIA RTX 5070 GPU with 12 GB of memory.

3.2.1 Preprocessing

Since CLIP is originally designed for RGB images, its default preprocessing pipeline assumes three-channel RGB inputs. In this work, we follow the preprocessing procedure from the original paper [33] with one minor modification. Instead of converting every image to RGB, IR images are converted to grayscale and the single channel is replicated across three channels. This allows IR images to be processed by CLIP without any changes to the underlying architectures.

3.2.2 Fine-tuning with one class

The first approach evaluated in this work is a straightforward fine-tuning of the pretrained RGB CLIP image encoder on IR data. The model is initialized with the original CLIP weights and trained only on IR images containing drones, all annotated with the single class label `"drone"`. The text encoder is kept fixed throughout training; only the image encoder is updated.

If the loss is minimized only on `"drone"` examples, the model quickly collapses to predicting `"drone"` for every input: without negative evidence, it can arbitrarily increase the `"drone"` logit while ignoring the structure of the embedding space. This yields high apparent training accuracy but does not produce a useful representation.

To reduce this effect, we introduce negative labels in the loss and optimize a standard cross-entropy objective over a small set of text prompts. For each IR drone image we compute its image embedding and the cosine similarities to one positive prompt (`"drone"`) and a set of negative prompts (e.g., `"helicopter"`, `"airplane"`, `"bird"`). These similarities form a vector of logits over the prompt set $\mathcal{C}$, which is passed through a softmax, and the cross-entropy loss is minimized with respect to the positive class.

This objective encourages high similarity between IR drone images and the `"drone"` text embedding, while simultaneously reducing the similarity to the negative labels. Since the visual training data still consist only of drones, the model does not see true non-drone examples and therefore has limited ability to learn full discriminative features. By choosing negative prompts that are semantically and morphologically close to drones, we nevertheless encourage separation between drones and other aerial objects in the embedding space.

3.2.3 Loss function

This loss function was designed around requirements specific to the cross-modal distillation: Aligning the IR student's embedding space with the RGB teacher's, preserving the relational structure between classes within that space, and optimizing for the aerial classification task. Together these are captured by a joint function:

$$\mathcal{L} = \lambda_{CLIP} \cdot \mathcal{L}_{CLIP} + \lambda_{kl} \cdot \mathcal{L}_{kl} \tag{3.1}$$

The first objective is a contrastive CLIP loss that, unlike standard CLIP pretraining, aligns the student's IR embeddings with the teachers RGB embeddings in the shared feature space rather than learning the space from scratch. Given a batch of paired samples, with normalized embeddings $\hat{z}_t, \hat{z}_s \in \mathbb{R}^{B\times D}$, where D is the embedding dimension and a learned logit scalar τ, the similarity matrix is calculated as

$$S = \tau \cdot \hat{z}_t \hat{z}_s^\top \in \mathbb{R}^{B\times B}. \tag{3.2}$$

The diagonal S_{ii} is the similarity between the matched IR-RGB pairs, everything off-diagonal is a mismatch within the batch. The loss is calculated as

$$\mathcal{L}_{CLIP} = \mathcal{L}_{row} + \mathcal{L}_{col}, \tag{3.3}$$

where $L_{row}, \mathcal{L}_{col}$ is calculated as

$$\mathcal{L}_{row} = \frac{1}{B} \sum_i -log(\frac{\exp{(S_{ii})}}{\sum_j \exp{S_{ij}}}) \tag{3.4}$$

$$\mathcal{L}_{col} = \frac{1}{B} \sum_j -log(\frac{\exp{(S_{jj})}}{\sum_i \exp{S_{ij}}}). \tag{3.5}$$

CLIP contrastive loss aligns each IR embedding with its corresponding RGB embedding, but it does not by itself preserve the full relational structure of the teacher's

embedding space. To capture this richer information, we add a distillation term that matches the teacher's and the student's similarity within each batch.

For each sample i, these similarity scores from the cross-similarity matrices are converted into softened probability distributions using a temperature parameter T as

$$P_{ij}^{(T)} = \frac{\exp\left((\hat{z}_t \hat{z}_t^\top)_{ij}/T\right)}{\sum_{k=1}^{B} \exp\left((\hat{z}_t \hat{z}_t^\top)_{ik}/T\right)} \tag{3.6}$$

$$Q_{ij}^{(T)} = \frac{\exp\left((\hat{z}_s \hat{z}_t^\top)_{ij}/T\right)}{\sum_{k=1}^{B} \exp\left((\hat{z}_s \hat{z}_t^\top)_{ik}/T\right)}. \tag{3.7}$$

$P_i^{(T)}$ describes how the teacher relates sample i to the other samples in the batch, while $Q_i^{(T)}$ describes the structure for the student. The $\mathcal{L}_{kl}$ is then defined as the Kullback-Leibler divergence between these similarity distributions,

$$\mathcal{L}_{kl} = T^2 \cdot \frac{1}{B} \sum KL(P_i^{(T)} || Q_i^{(T)}). \tag{3.8}$$

This can be interpret as a generalization of classical knowledge distillation introduced by Hinton et al. [41], instead of matching softened class-probability, the student matches softened similarity distributions over the samples in the batch.

The established loss function (3.1) does not yet address which class an image belong to. It simply pushes the students embedding space to have the same shape as the teacher's. When multiple classes were introduced in the dataset, additional terms was required. Due to the focus of classifying aerial objects, we can improve system performance by further separating class embeddings from each other, therefore we add $\mathcal{L}_{focal}$ [64], defined as

$$\mathcal{L}_{focal} = \frac{1}{N} \sum_{i=1}^{N} (1 - p_{t,i})^\gamma \mathcal{L}_{CE,i}, \quad p_{t,i} = e^{-\mathcal{L}_{CE,i}}, \tag{3.9}$$

if C is the set of class prompts and y is the correct class label, then,

$$\mathcal{L}_{CE} = -w_i \log \frac{\exp\left(a_y\right)}{\sum_{c \in C} \exp(\tau \cdot \hat{z}_s \hat{z}_{text,c}^\top)}, \tag{3.10}$$

where w_{y_i} denotes the class weight for the ground-truth class of sample i. The weights was chosen as proportional to the class frequency in the dataset. Compared to using a standard cross-entropy loss, the focal loss addressed the unbalanced dataset. If each sample contributes equally, the model will be dominated by the gradient signal of the majority class. In other words, if the dataset contains 90% drones, predicting drone for every frame will yield a 90% accuracy. Focal loss also handles sample difficulty imbalance. It downweights easy examples through the parameters γ and focuses on harder misclassified ones. In other words, if a sample is already classified correctly with high confidence, it contributes less and vice versa.

Even though the class embeddings are separated, the same class embeddings might be scattered and the focal loss will still be satisfied. This means that the model might have learned the right answer but not a coherent representation of what the specific class looks like which result in poor class margins [65], which was observed during training. Therefore, we introduced a supervised contrastive loss (SupCon) [66]. This loss pulls same class embeddings together, while simultaneously pushing apart clusters of different classes. SupCon is defined as:

$$\mathcal{L}_{SupCon} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(z_i \cdot z_p / \tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a / \tau)}, \tag{3.11}$$

where I is the set or all samples indices in the current batch. $A(i)$ is all samples except sample i, $P(i)$ is all samples in $A(i)$ that share the same class label as i.

Then, the final joint loss function for CMKD with multiple classes is defined as:

$$\mathcal{L} = \lambda_{CLIP} \cdot \mathcal{L}_{CLIP} + \lambda_{kl} \cdot \mathcal{L}_{kl} + \lambda_{focal} \cdot \mathcal{L}_{focal} + \lambda_{SupCon} \cdot \mathcal{L}_{SupCon} \tag{3.12}$$

where λ denotes hyperparameter weights coefficients that balance the loss function.

3.2.4 Cross-modal knowledge distillation with one class

To address the limitation that simple fine-tuning on a single positive class does not lead to a rich representation of the image distribution, we employ cross-modal knowledge distillation (CMKD). In other words, the IR encoder is encouraged to reproduce the teacher's representation of each sample, rather than only predicting a single class label, which provides a much richer training signal.

The RGB CLIP is used as a frozen teacher, with paired RGB and IR images fed into teacher and student respectively, optimized using Equation 3.1. Since we only use images that visually contain drones, this setup does not completely resolve the lack of negative visual evidence. In preliminary experiments, such a cross-entropy objective caused the student to artificially increase the similarity between all IR embeddings and the `"drone"` text embedding, in order to maximize positive-class accuracy. To avoid the student collapsing to a trivial solution, we deliberately do not include a standard cross-entropy term with the hard `"drone"` label. A visual outline of the training procedure can be seen in Figure 3.2.

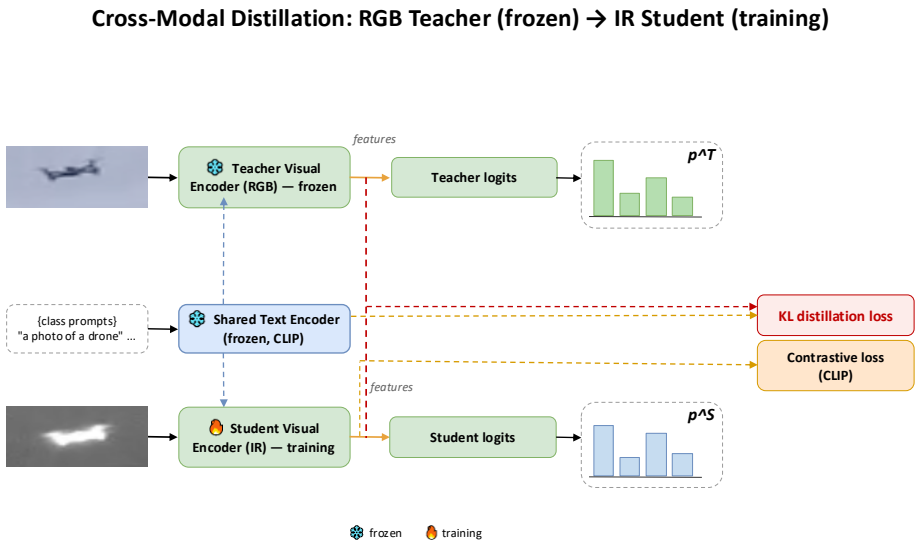

Figure 3.2: Cross-modal knowledge distillation (one class)

3.2.5 Cross-modal knowledge distillation with multiple classes

In the final configuration, cross-modal knowledge distillation is extended from a single target class to a multi-class setting. Instead of treating all IR samples as belonging to the single label `"drone"`, the student is trained to distinguish between morphologically related aerial categories, such as `"bird"`, `"airplane"`, `"drone"`, and `"helicopter"`. This allows the IR encoder to learn a richer, more discriminative embedding space while still leveraging the pretrained CLIP model as teacher.

As in the single-class setup, the RGB CLIP image encoder acts as a frozen teacher. In addition, we exploit the fact that the dataset contains multiple labeled classes: each image is associated with a ground-truth category $c \in \mathcal{C}$ where, $\mathcal{C}$ denotes the set of aerial object classes. For every class a set of text prompts is constructed (e.g. "a photo of a drone"), and their embeddings are averaged to obtain a single prototype text embedding per class. These class-specific text embeddings remain fixed during training. The student is then optimized using the full joint loss function Equation 3.12.

Combined with the embedding-level distillation, this setup encourages the IR encoder to simultaneously reproduce the teacher's fine-grained embeddings and separate classes that are morphologically and semantically similar, such as drones and birds. As a result, the student learns IR representations that are well aligned with the RGB CLIP space and better suited for downstream tasks. A visual outline of the training procedure can bee seen in Figure 3.3.

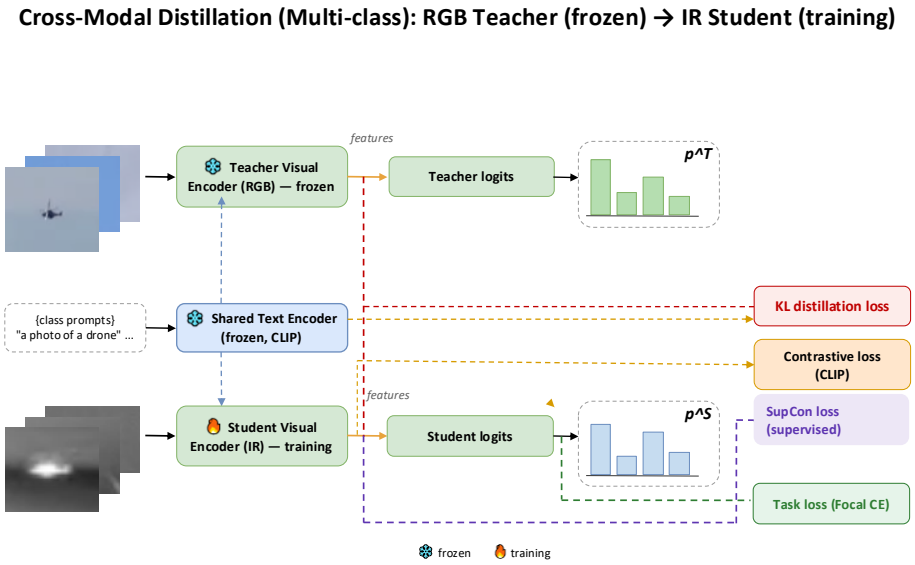

Figure 3.3: Cross-modal knowledge distillation (multi-class).

During training, a scale augmentation is applied stochastically to drone crops in each batch. With probability p, a drone image is downscaled to a random fraction of its original size (around 5-25% of the 224×224 input resolution), pasted onto a randomly selected background image from the same batch, and then recropped with 1.5× padding around the paste location before being upscaled back to 224×224. This mirrors the inference pipeline, where bounding box crops are padded and resized, and is intended to teach the model to recognize drones at far distances where they appear as small thermal blobs. The augmentation is only applied to the student's IR input; the teacher receives the original unaugmented RGB crop.

3.2.6 Pretraining teacher with LoRA

In the early stages of CMKD, we observed that the student struggled to inherit the teacher's embedding structure: IR images of drones were frequently confused with birds, airplanes, and helicopters. This motivated a closer analysis of CLIP representations and our training data. Our initial hypothesis was that different aerial objects, which tend to be small at long range, are mapped to overlapping regions in the embedding space.

To investigate this, we used Uniform Manifold Approximation and Projection (UMAP), a dimensionality reduction technique for visualizing high-dimensional data [67]. As shown in Figure 3.4, most aerial classes occupy a similar region in the original CLIP embedding space, which explains the poor separability and, consequently, the suboptimal classification performance. At the same time, different drone datasets exhibit

some degrees of separation, as expected due to differences in their underlying visual distributions. These observations confirmed that the teacher representations themselves needed to be adapted for our aerial domain.

To address this issue, we apply LoRA-based adaptation to the CLIP visual encoder before performing CMKD to the IR student. The underlying theory and method of LoRA are described in detail in Section 2.2.5. Here, we focus on how it is instantiated for the CLIP teacher, and why specific parts of the architecture are targeted.

Concretely, we insert LoRA modules into two linear layers inside each of the 12 Transformer blocks of the CLIP image encoder:

- `attn.out_proj`: the output projection of the multi-head self-attention module, which writes the attention-computed features back into the residual stream.
- `mlp.c_proj`: the second linear layer in the MLP block, which projects the expanded hidden dimension back down to the model width.

These layers control how newly computed features are mixed into the token representations and how non-linear transformations are compressed back into the shared embedding space. By restricting LoRA to these write and projection layers, we allow the model to adjust how information is integrated and shaped, while keeping the core feature extraction and attention pattern intact.

We deliberately do not apply LoRA to:

- `att.in_proj` (the $Q/K/V$ projections): we keep how the model reads and attends to tokens frozen, in order to preserve the robust attention structure during large-scale pretraining.
- `mlp.c_fc` (the MLP up-projection): the expansion from model width to the higher-dimensional hidden space remains unchanged, so that the expressive capacity of the MLP is preserved while only its final shaping back into the embedding space is adapted.

This design constrains the adaptation to a low-rank subspace that primarily reshapes the visual embedding space for aerial objects, instead of globally rewriting the CLIP feature extractor. In practice, this has two advantages. First, it makes the adaptation data-efficient and stable, as the majority of the pretrained weights remain fixed. Second, it encourages the LoRA modules to specialize in better separating morphologically similar aerial classes while retaining compatibility with the original CLIP text embeddings.

A concern when adapting the teacher with LoRA is that the resulting embedding space may drift so far from the original CLIP space that an IR student initialized from the base CLIP model would struggle to distill effectively. The teacher's adapted embeddings should therefore remain reasonably close to the original CLIP space to provide a meaningful distillation target. To monitor this, a cosine floor penalty was introduced during teacher pretraining. A base IR CLIP model, with

no task specific training, was used as a proxy to monitor drift. For each batch, the mean cosine similarity $\bar{c}$ between the LoRA-adapted teacher RGB embeddings and the corresponding base IR CLIP embeddings on the IR crops. If this similarity falls below a target threshold δ, a squared penalty term is added to the classification loss:

$$\mathcal{L}_{\text{penalty}} = \lambda \text{p} \cdot \max(0,\ \delta - \bar{c})^2, \tag{3.13}$$

where λ_{p} is a scalar weight. The intention is to constrain the teacher so that it improves class separability without drifting so far from the original CLIP space that the student cannot follow.

After LoRA pretraining on our aerial RGB dataset, the resulting teacher exhibits a more structured and better separated embedding space, as illustrated in Figure 3.5. Compared to the original CLIP teacher (Figure 3.4), the different aerial categories form more distinct clusters, and drones in particular are less entangled with birds and airplanes. This improves separability translates directly into more informative targets for CMKD: when the IR student is trained to match the LoRA-adapted teacher embeddings, it benefits from a teacher whose representations already reflects the fine-grained differences required by our downstream task.

The LoRA adapted RGB encoder serves a dual role in the overall system. First, it acts as the teacher in the LoRA + CMKD training configuration. Second, the same model is deployed directly as the classifier in the RGB pipeline, without any additional training.

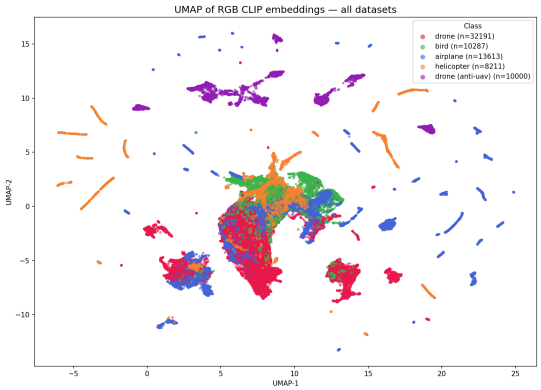

Figure 3.4: UMAP of embedding space for CLIP teacher

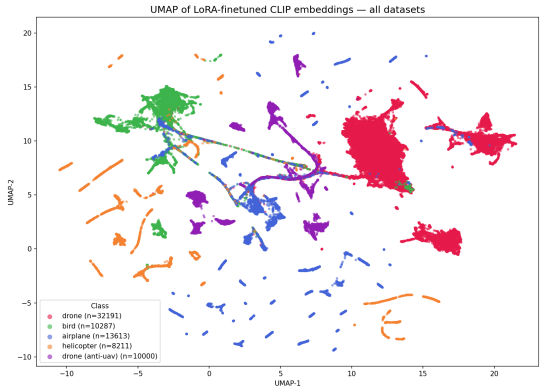

Figure 3.5: UMAP of embedding space for CLIP teacher with LoRA

3.2.7 Dataset

In order to train the model, a dataset that is representative of the real-world environment and includes paired IR and RGB images is required. It is essential that the dataset contains drones, as they are the primary objects to be classified. It is also beneficial to include additional objects, preferably objects that are morphologically similar to drones, in order to achieve higher accuracy.

The dataset that fits these criteria is from a master's thesis by Fredrik Svanström at Halmstad University [68]. The dataset comprises a total of 650 videos, of which 365 are IR videos and 285 are RGB videos. There are three different types of drones included, all of them are standard commercially available models. In addition to drone videos, the dataset also contains videos of airplanes, birds, and helicopters. After extracting individual frames from the videos, approximately 150,000 images were obtained. The dataset has three different distance ranges for each class which can be seen in Table 3.1.

Table 3.1: The distance for the different target classes.

Bin	**Class**			
	AIRPLANE	**BIRD**	**DRONE**	**HELICOPTER**
Close	< 1000 m	< 40 m	< 20 m	< 500 m
Medium	1000–3000 m	40–120 m	20–60 m	500–1500 m
Distant	> 3000 m	> 120 m	> 60 m	> 1500 m

To improve drone classification performance in cluttered backgrounds and scenarios with varying thermal signatures, we diluted the dataset with images from the Anti-UAV300 dataset [69]. This dataset contains approximately 300 IR-RGB video pairs, resulting in 580,000 annotated frame pairs. From this, we randomly sampled approximately 12,000 frames containing a drone and approximately 2000 frames without a drone present. The non-drone frames were included to provide negative examples and reduce false positives.

3.2.8 Hyperparameter selection

Hyperparameters were selected through a systematic search using Weights and Biases (W&B) sweep [70]. Sweeping is a method to automate hyperparameter optimization by evaluating model performance across different ranges of possible values. The sweep explored combinations of the following parameters and values using a W&B sweep with a "Bayesian search´´ algorithm. This algorithm uses Bayesian optimization to intelligently explore the hyperparameter space, with each combination evaluated using $F1_{macro}$, defined as the mean F1 over all C classes.

$$F1_{macro} = \frac{1}{C} \sum_{c} F1_c, \tag{3.14}$$

where,

$$F1_c = \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}. \tag{3.15}$$

The W&B sweeps resulted in the parameters for the different models presented in Table 3.2.

Table 3.2: Hyperparameter configuration for the fine-tuned classification model variants. A dash (—) indicates the parameter is not applicable for that variant. IR-CLIP Base uses the pre-trained CLIP model without fine-tuning and is therefore excluded from this table.

Hyperparameter	**CMKD 1-class**	**CMKD multi-class**	**LoRA + CMKD (IR)**	**LoRA (RGB)**
Optimiser				
Learning rate	1.00×10^{-5}	7.01×10^{-6}	8.32×10^{-6}	3.31×10^{-5}
Weight decay[†]	0.01 (all variants)			
Batch size	128 (all variants)			
Training epochs	100	15	63	37
Loss				
Focal loss γ	—	2	0	2
λ_{CLIP}	1	3	0.5	—
λ_{kl}	2	1	1	—
λ_{cross}	—	0.2	0.3	—
λ_{supcon}	—	—	0	—
Supcon temperature T	—	—	0.2	—
KD temperature T	6	2	4	—
Architecture				
Frozen encoder blocks	—	6	6	—
LoRA rank r[‡]	—	—	8	8
LoRA alpha α[‡]	—	—	32	32
LoRA dropout	—	—	0	0

[†] Weight decay is fixed at 0.01 across all variants (reduced from the CLIP default of 0.2 to prevent over-regularisation of fine-tuned layers).
[‡] LoRA (RGB) serves as both the RGB pipeline classifier and the CMKD teacher; its LoRA configuration is identical to that of LoRA + CMKD (IR).

3.3 Tracking

The tracking pipeline employs BYTE from ByteTrack, using the proposed detection pipeline as its detector in place of YOLOX. The implementation is based on [71], a fork of the original ByteTrack repository that reduces external dependencies while preserving the core functionality of BYTE. Two extensions were introduced: integration of classification on tracks and ego-motion compensation.

As BYTE is dependent on a Kalman filter to make predictions of the position of objects in the next frame, it may suffer during ego-motion, as it does not account for it. In order to mitigate this problem, ego-motion compensation is integrated into the pipeline. This is done by modeling the background motion between consecutive frames and then transforming the predicted points accordingly. Background motion is modeled using dense optical flow. The dense optical flow between two consecutive frames is computed using FastFlowNet [57]. A homography matrix is then estimated by using pair points from points in the frame before and displacing them using their optical flow vector to get their position in the current frame. RANSAC is used during homography estimation to reliably output a matrix with the most inliers. Under the assumption that more than half of the frame is background, the homography matrix with the most inliers will then mostly describe the background motion [54]. This homography matrix will be used to reproject the predictions into

the coordinate space of frame_k, allowing tracks to persist across camera motion.

The classification is integrated into BYTE, enabling classification inference only on confirmed tracks. This approach aims to achieve a computationally lighter pipeline. An alternative approach is to perform classification prior to the tracking stage. This would allow the system to filter detections before they are passed to BYTE, thereby limiting tracking to relevant objects only. In this configuration, the classifier effectively acts as a filter, preventing unnecessary tracking of undesired objects. The former approach was chosen as optimizing the performance was prioritized.

For good persistent labels and confidence scores, every active track is classified every third frame. A limitation of this approach is that if the video footage contains, for example, two objects that remain visible throughout the entire sequence, each object will be classified approximately $\frac{number\ of\ frames}{3}$ times. This can be computationally expensive and avoidable. To address this, a confidence threshold was introduced so that once the system is sufficiently certain of an object's class, further classification of that object is discontinued. When a tracks' label has been consistent for 5 consecutive classifications with a high confidence score, classification inference will no longer be performed on that specific track. The confidence score will be set to 1.0 to indicate that the track has been classified.

ByteTrack was selected for two primary reasons. First, it offers decent performance while maintaining a relatively low computational cost [50]. When combined with YOLOX as the detection component, the system is capable of achieving real-time performance of approximately 30 frames per second on a V100 GPU. Second, ByteTrack provides a high degree of modularity in its design, allowing the detection module to be replaced with alternative detectors without requiring modifications to the tracking framework.

3.4 Inference in system

At inference time, each video frame passes through a sequential pipeline that integrates motion analysis, object detection, multi-object tracking, and classification. In IR mode, each incoming frame is converted into a three-channel representation by replicating the grayscale thermal channel since both the detector and the classifier expect RGB-format input. The frame is then passed, along with the previous frame, to FastFlowNet, which together with RANSAC estimates the camera motion in a homography matrix that is later used to compensate for camera movement in the tracking stage.

Detection runs on every frame using RF-DETR, which produces bounding boxes with associated confidence scores. These are passed to ByteTrack together with the full frame and the homography matrix. Inside ByteTrack, Kalman filter predictions are corrected using the homography before Hungarian matching associates detections to tracks. Unmatched tracks are kept alive for a short buffer period to handle brief detection failures.

When a track is initialized or updated, the annotated frame is drawn with a bounding box expanded by a display padding factor around the detection center. This ensures that small detections remain visible, and the crop passed to the classifier contains enough surrounding context for the ViT to separate classes. Before online inference, visual prototypes [72] were computed using all training images through the IR-CLIP encoder, averaging and normalizing the resulting embeddings per class to produce a visual prototype that summarizes the locations in the embedding space. At inference, classification is performed by computing the cosine similarity between the encoded crop and each prototype, assigning the class whose prototype is most similar to the crop embedding.

Finally, the output for each iteration is a set of tracks, where for each track its ID, bounding box and class is given. Over time, the track will contain more bounding boxes, displaying the objects trajectory. The full pipeline is shown in Figure 3.6.

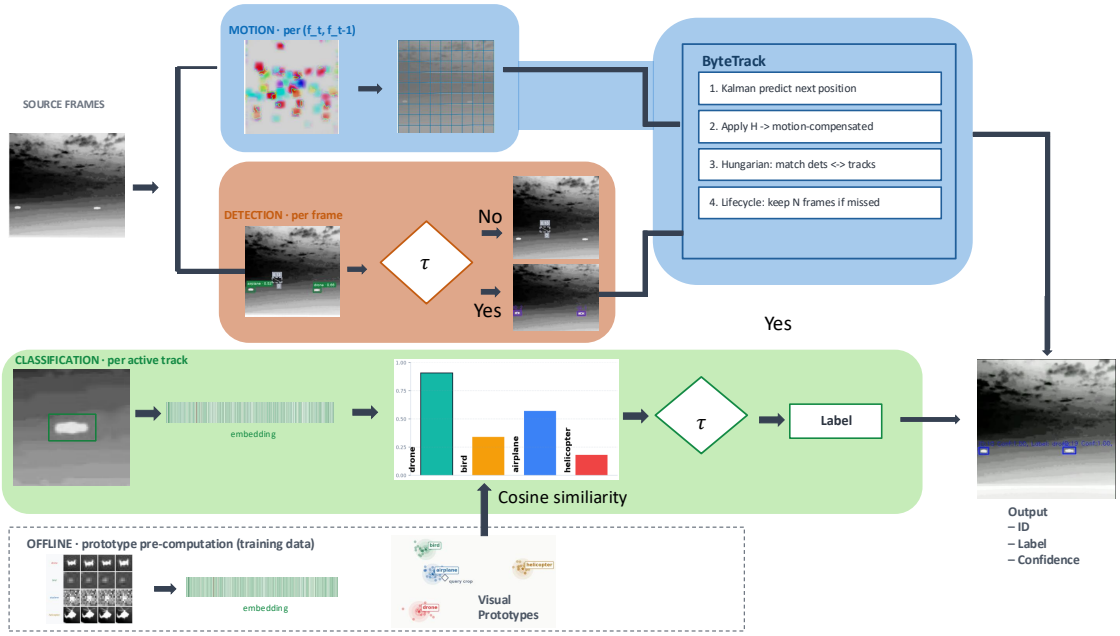

Figure 3.6: Overview full system pipeline, input mode: IR

3.5 Acquisition of Evaluation Data

Evaluation was performed on videos generated from a simulation. With control of the simulation, specific scenarios could be constructed and recorded in the simulation, allowing for targeted testing. The environment simulates flying objects, which are recorded by a drone equipped with a camera sensor. The simulation framework primarily consists of Gazebo, ArduPilot, and QGroundControl. A detailed description of the configuration of these components, along with their interactions, is provided in section A.1.

The sensor data from the simulations are captured through a Python script. More details about the script is provided in section A.2. The data acquired are RGB-frames from the camera sensor and ground-truth bounding boxes from the drones being recorded. A key consideration is how boxes are handled during object occlusion. One configuration produces bounding boxes that encompass the entire object, including occluded regions, while another restricts the box to only the visible portions. The latter is selected, as the detection pipeline does not predict bounding boxes for occluded regions but is capable of detecting partially occluded objects. Figure 3.7 shows how bounding boxes are computed in occluded scenarios. However, this configuration can produce ground-truth bounding boxes that are particularly challenging to predict in complex occlusion cases. An example of such a case is shown in Figure 3.8. Although it may be unreasonable to expect a model to correctly predict this specific bounding box as a drone, such scenarios occur infrequently and are therefore considered acceptable.

Acquisition of test data for IR footage is performed through post-processing of RGB

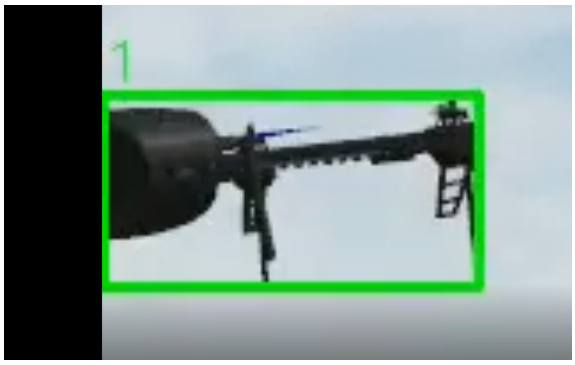

Figure 3.7: Example of a simple occlusion scenario, where a drone is partially out-of-view from the frame.

Figure 3.8: Example of a complex occlusion scenario, where a drone occludes another drone. A very small box can be seen labeled as 1.

images using OpenCV, generating images with visual characteristics resembling IR imagery. First, the RGB image is converted to grayscale to represent intensity values independently of color information. Next, a Gaussian blur is applied to produce a smoother appearance, similar to that of thermal sensors. The image is then normalized to utilize the full intensity range, followed by histogram equalization to enhance contrast between regions representing higher and lower temperatures. Finally, a bitwise inversion is applied so that darker objects, such as the drone, appear brighter. This further mimics the typical representation of warmer objects in IR imagery. This process results in images that approximate the visual properties of IR footage, with particular emphasis on making drones appear as distinct hot targets.

3.5.1 Evaluation Dataset

The dataset consists of two sets of videos. In the first set, the camera remains stationary, while in the second set, the camera is in motion. Each scenario is designed to either introduce a new challenge or combine aspects of previously defined scenarios. The average video length is 2 minutes and 22 seconds. An overview of all scenarios is provided in Table 3.3.

Although the camera is nominally stationary in the first set, it is mounted on a drone, resulting in minor motion and vibration. This introduces slight camera shake, making the footage more representative of real-world conditions.

In the second set, the camera undergoes motion. This motion can be categorized into two types: rotational movement, where the camera changes its orientation, and translational movement, where the camera changes its position in space. These motion types are considered both independently and in combination across different scenarios.

Despite the delimitation specifying that the system is not required to re-identify objects that have been out-of-view, such scenarios are occasionally present in some of the videos. In particular, they occur in every video of testset 2 with 2 drones present.

Table 3.3: Overview of simulated video scenarios. Maximum and minimum distance from the camera are rough estimations, while the remaining stats are acquired through QGroundControl.

ID	Set	Drones	Description	Cam Alt. (m)	Max Dist. (m)	Min Dist. (m)	Max Alt. Diff. (m)	Duration (min)
S1	Stationary	1	Simple motion	150	35	10	0	2.38
S2	Stationary	1	Altitude difference	150	35	10	10	3.25
S3	Stationary	1	Far away, Altitude difference	150	85	10	12	2.78
S4	Stationary	2	Simple motion	150	38	4	0	2.00
S5	Stationary	2	Altitude difference	150	38	8	11	3.00
S6	Stationary	2	Far away, Altitude difference	150	93	11	12	3.75
S7	Stationary	2	Minor occlusion	4.5	50	30	4.5	1.57
S8	Stationary	2	Crash	200	40	5	0	1.52
M1	Moving	1	Rotating camera	200	55	6	6	2.28
M2	Moving	2	Rotating camera	200	65	6	10	2.75
M3	Moving	2	Rotating camera	150	55	5	6	2.27
M4	Moving	1	Follow simple	200	14	14	0	0.8
M5	Moving	2	Follow and rotating camera, low altitude	15	20	20	0	0.98
M6	Moving	2	Follow and rotating camera, low altitude	25	15	25	0	1.63
M7	Moving	2	Follow, sharp turns	40	10	15	0	4.43

4
Result

This chapter presents the numerical results obtained from the various components of the system. First, the training and evaluation results of the classification and detection pipeline are presented, including performance on the simulated video dataset. Next, the tracking performance is assessed. Finally, the overall system performance is presented, including aspects such as computational costs.

4.1 CLIP Training

The results of the different IR CLIP-based approaches are presented in two tables. Table 4.1 shows per-class F1 score on the held-out split of the IR dataset using visual prototypes for evaluation, and Table 4.2 reports drone recall across all simulated evaluation scenarios.

The simple fine-tuned model is excluded from both tables. As expected, training exclusively on drone images collapses the embedding space: the model predicts the class `drone` for almost every input. This yields a drone recall of 1.00 across all scenarios, but a recall of 0.00 for all other classes, making it unusable as a multi-class classifier.

On the IR test split, LoRA + CMKD achieves the highest macro F1 of 0.695, outperforming all other approaches across every class. CMKD 1-class reaches a macro F1 of 0.500, while CMKD multi-class performs similarly at 0.427. IR-CLIP Base yields the lowest macro F1 of 0.380, with particularity poor drone F1 of 0.221.

The scenario evaluation reveals a larger separation between methods. IR-CLIP Base achieves an overall drone recall of only 0.049, indicating that the unadapted CLIP model is insufficent for IR deployment. CMKD 1-class reaches 0.230, and CMKD multi-class 0.151, which is noticeably lower than CMKD 1-class despite comparable performance on the IR training split. LoRA + CMKD achieves the highest overall recall of 0.617, outperforming all other approaches in every scenario. Scale augmentation was evaluated during the hyperparameter sweep but did not yield a consistent improvement in macro F1; the best-performing configuration setting $p = 0$, effectively disabling it. The cosine floor penalty did not lead to measurable gains in downstream student performance and was omitted from the final teacher training configuration.

The far-away scenarios (S3,S6) are the most challenging across all IR models, with even LoRA + CMKD achieving only 0.261 and 0.233 recall, respectively. In contrast, the follow scenarios with a moving camera (M4-M6) shows some of the highest recalls for LoRA + CMKD, reaching up to 0.944 in M4.

Table 4.1: Per-class F1 on training test split using visual prototypes for IR.

Model	Macro F1	Drone	Bird	Airplane	Helicopter
IR-CLIP Base	0.380	0.221	0.522	0.407	0.370
CMKD 1-class	0.500	0.639	0.394	0.618	0.351
CMKD multi class	0.4274	0.350	0.566	0.449	0.344
LoRA + CMKD	**0.695**	**0.970**	**0.608**	**0.670**	**0.532**

Table 4.2: Drone recall per scenario for IR-CLIP Base, CMKD variants, and LoRA + CMKD.

ID	Description	Drones	Camera	Drone Recall			
				IR-CLIP Base	CMKD 1-class	CMKD multi class	LoRA + CMKD
S1	Simple motion	1	Stationary	0.051	0.409	0.372	**0.759**
S2	Altitude difference	1	Stationary	0.003	0.274	0.303	**0.765**
S3	Far away	1	Stationary	0.000	0.072	0.058	**0.261**
S4	Simple motion	2	Stationary	0.065	0.315	0.219	**0.704**
S5	Altitude difference	2	Stationary	0.015	0.281	0.224	**0.719**
S6	Far away	2	Stationary	0.003	0.012	0.005	**0.233**
S7	Occlusion	2	Stationary	0.208	0.055	0.070	**0.529**
S8	Crash	2	Stationary	0.014	0.193	0.093	**0.627**
M1	Rotating camera	1	Moving	0.012	0.226	0.219	**0.753**
M2	Rotating camera	2	Moving	0.025	0.248	0.087	**0.674**
M3	Rotating camera	2	Moving	0.037	0.261	0.378	**0.773**
M4	Follow simple	1	Moving	0.000	0.054	0.036	**0.944**
M5	Follow, low altitude	2	Moving	0.121	0.424	0.092	**0.765**
M6	Follow, low altitude	2	Moving	0.025	0.415	0.043	**0.817**
M7	Follow, sharp turns	2	Moving	0.078	0.212	0.074	**0.621**
Overall				0.049	0.23	0.151	**0.617**

For the RGB pipeline, we compare the base RGB CLIP model against the LoRA-adapted RGB encoder used as a teacher in the CMKD setup. Table 4.3 reports per-class F1 on the held-out split of the RGB dataset using visual prototypes, and Table 4.4 shows drone recall across all simulated evaluation scenarios.

On the RGB training split, LoRA RGB substantially improves the macro F1 from 0.380 to 0.531. The most pronounced gains occur for the drone class (F1 increasing from 0.221 to 0.699) and the helicopter class (from 0.370 to 0.929), while bird and airplane remain more challenging. This indicates that LoRA successfully reshapes the RGB embedding space to better separate morphologically similar aerial objects that are most relevant for this work.

The simulated scenario evaluation shows an even cleared advantage for the adapted model. The base RGB CLIP model achieves an overall drone recall of 0.252 across all scenarios, whereas LoRA RGB achieves 0.6535 and dominates the base model in

every scenario (Table 4.4). As with the IR results, the far-away scenarios (S3, S6) remain the most challenging, but LoRA RGB still achieves higher recalls than the base model. The follow scenarios with a moving camera (M4-M7) are among the strongest, with recall up to 0.945 in M4 and above 0.84 in M7.

Taken together, these results show that adapting the RGB encoder with LoRA yields a more structured and separable embedding space for aerial objects, both on the test split and under the distribution shift induced by the simulated evaluation scenarios.

Table 4.3: Per-class F1 on training test split using visual prototypes for RGB

Model	Macro F1	Drone	Bird	Airplane	Helicopter
RGB-CLIP Base	0.456	0.516	0.445	0.017	0.845
LoRA RGB	**0.531**	**0.699**	**0.397**	**0.102**	**0.929**

Table 4.4: Drone recall per scenario for RGB-CLIP Base and LoRA RGB

ID	Description	Drones	Camera	Drone Recall	
				RGB-CLIP Base	LoRA RGB
S1	Simple motion	1	Stationary	0.168	**0.788**
S2	Altitude difference	1	Stationary	0.502	**0.778**
S3	Far away	1	Stationary	0.206	**0.432**
S4	Simple motion	2	Stationary	0.147	**0.706**
S5	Altitude difference	2	Stationary	0.403	**0.748**
S6	Far away	2	Stationary	0.128	**0.320**
S7	Occlusion	2	Stationary	0.288	**0.504**
S8	Crash	2	Stationary	0.307	**0.494**
M1	Rotating camera	1	Moving	0.290	**0.629**
M2	Rotating camera	2	Moving	0.247	**0.494**
M3	Rotating camera	2	Moving	0.362	**0.687**
M4	Follow simple	1	Moving	0.186	**0.945**
M5	Follow, low altitude	2	Moving	0.153	**0.898**
M6	Follow, low altitude	2	Moving	0.194	**0.899**
M7	Follow, sharp turns	2	Moving	0.179	**0.844**
Overall				0.252	**0.6535**

4.2 RF-DETR Training

To evaluate the detection model, the performance metrics recall, precision, F1 and AP50 is used. The performance metrics are defined as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

$$\text{AP50} = \frac{1}{101} \sum_{r \in \{0, 0.01, \ldots, 1.0\}} P_{\text{interp}}(r)$$

One important note is how TP is defined during the model evaluation. For recall, precision and F1 it is calculated by comparing center distances. When the predicted box center and ground-truth center is less than 3 pixels apart, it is considered a TP. AP50 follows the standard COCO convention, where a detection is considered a TP if its bounding box achieves an IoU of at least 0.5 with the ground-truth.

Table 4.5 shows the IR models performance on the different simulations. The most evident change between simulations is the models precision which drops from the high 0.8-0.99 to at its lowest 0.346, as the environment becomes more complex. The objects of interest are relatively small, and as the distance increases, their features are harder to distinguish. The models performance on greater distances can be seen in the S6, it manages to achieve 0.919 recall and 0.938 precision, S6 is the simulation with the greatest distance between camera and drone.

Table 4.5: Object Detection performance, on IR-frames

ID	Description	Drones	Camera	Recall	Precision	F1	AP50
S1	Simple motion	1	Stationary	0.999	0.997	0.998	0.0135
S2	Altitude difference	1	Stationary	0.979	0.979	0.979	0.0020
S3	Far away	1	Stationary	0.990	0.990	0.990	0.0017
S4	Simple motion	2	Stationary	0.779	0.903	0.837	0.0154
S5	Altitude difference	2	Stationary	0.954	0.984	0.969	0.0032
S6	Far away	2	Stationary	0.919	0.938	0.928	0.0023
S7	Occlusion	2	Stationary	0.835	0.472	0.603	0
S8	Crash	2	Stationary	0.896	0.948	0.922	0.0005
M1	Rotating camera	1	Moving	0.991	0.847	0.913	0.0102
M2	Rotating camera	2	Moving	0.969	0.737	0.838	0.0023
M3	Rotating camera	2	Moving	0.973	0.903	0.937	0.0007
M4	Follow simple	1	Moving	0.960	0.782	0.862	0.0123
M5	Follow, low altitude	2	Moving	0.838	0.430	0.568	0.0003
M6	Follow, low altitude	2	Moving	0.788	0.346	0.481	0.0008
M7	Follow, sharp turns	2	Moving	0.790	0.407	0.534	0.0001
Overall				0.911	0.778	0.824	0.0042

Table 4.6 showcases the RGB models performance. The models performance, it achieves high scores across all test simulations. A slight degradation on simulations M5-M7, where both recall and precision is affected. M5-M7 have more complex backgrounds and more drastic camera motion. The models detection capabilities of targets of greater distance from a stationary camera and simpler background was strong, which is shown on test S6.

Table 4.6: Object Detection performance, on RGB-frames

ID	Description	Drones	Camera	Recall	Precision	F1	AP50
S1	Simple motion	1	Stationary	0.998	0.997	0.998	0.1070
S2	Altitude difference	1	Stationary	0.994	0.994	0.994	0.0842
S3	Far away	1	Stationary	0.996	0.996	0.996	0.0481
S4	Simple motion	2	Stationary	0.741	0.894	0.810	0.1249
S5	Altitude difference	2	Stationary	0.962	0.993	0.877	0.0790
S6	Far away	2	Stationary	0.929	0.939	0.934	0.0328
S7	Occlusion	2	Stationary	0.918	0.993	0.954	0.0002
S8	Crash	2	Stationary	0.908	0.973	0.940	0.0315
M1	Rotating camera	1	Moving	0.991	0.991	0.991	0.1464
M2	Rotating camera	2	Moving	0.990	0.993	0.991	0.0732
M3	Rotating camera	2	Moving	0.972	0.980	0.976	0.0112
M4	Follow simple	1	Moving	0.926	0.926	0.926	0.1091
M5	Follow, low altitude	2	Moving	0.823	0.820	0.821	0.2764
M6	Follow, low altitude	2	Moving	0.685	0.755	0.719	0.4629
M7	Follow, sharp turns	2	Moving	0.820	0.859	0.839	0.3506
Overall				0.910	0.940	0.924	0.1291

4.3 Tracking

The tracking performance in RGB and IR mode is presented in Table 4.7 and Table 4.8, respectively. The evaluation metrics include HOTA, DetA, AssA, Frag, IDSW, and the number of identities (IDs) detected by the tracker. Since the proposed tracker is the only tracker evaluated on the dataset, the HOTA score is primarily useful for comparing performance across different scenarios. DetA and AssA represent the detection accuracy and association accuracy components of HOTA, respectively. Higher values of HOTA, DetA, and AssA indicate better tracking performance, while lower values of Frag and IDSW are desirable, as they reflect fewer track fragmentations and identity switches. Ideally, the number of detected IDs should correspond to the true number of drones present in each scenario.

Table 4.7: Tracking performance per scenario across evaluation metrics, on RGB-frames

ID	Description	Drones	Camera	HOTA	DetA	AssA	Frag	IDSW	IDs
S1	Simple motion	1	Stationary	30.95	30.95	30.95	73	0	1
S2	Altitude difference	1	Stationary	28.37	28.37	28.37	108	0	1
S3	Far away	1	Stationary	11.51	11.51	11.51	30	0	1
S4	Simple motion	2	Stationary	16.99	24.597	12.64	101	5	14
S5	Altitude difference	2	Stationary	22.85	26.80	19.82	184	1	7
S6	Far away	2	Stationary	7.77	9.63	6.65	60	1	17
S7	Occlusion	2	Stationary	11.03	16.34	7.04	23	4	10
S8	Crash	2	Stationary	16.15	16.33	16.12	35	0	7
M1	Rotating camera	1	Moving	23.74	23.74	23.74	66	0	1
M2	Rotating camera	2	Moving	17.47	18.51	16.50	37	2	9
M3	Rotating camera	2	Moving	17.62	23.84	13.05	39	3	12
M4	Follow simple	1	Moving	43.11	43.06	43.16	40	2	3
M5	Follow, low altitude	2	Moving	27.16	43.472	17.19	60	3	7
M6	Follow, low altitude	2	Moving	22.49	45.18	11.24	90	5	14
M7	Follow, sharp turns	2	Moving	21.13	39.52	11.37	137	8	25
Overall				19.98	22.12	18.34	1083	34	129

Table 4.8: Tracking performance per scenario across evaluation metrics, on IR-frames

ID	Description	Drones	Camera	HOTA	DetA	AssA	Frag	IDSW	IDs
S1	Simple motion	1	Stationary	23.94	23.94	23.94	32	0	1
S2	Altitude difference	1	Stationary	21.30	21.30	21.30	38	0	1
S3	Far away	1	Stationary	6.81	8.68	5.48	16	0	3
S4	Simple motion	2	Stationary	11.82	18.48	8.27	47	1	12
S5	Altitude difference	2	Stationary	16.17	19.13	13.92	62	0	6
S6	Far away	2	Stationary	5.01	6.80	4.05	21	0	15
S7	Occlusion	2	Stationary	3.99	5.54	2.91	0	0	22
S8	Crash	2	Stationary	8.49	9.83	7.90	12	0	9
M1	Rotating camera	1	Moving	18.18	18.18	18.19	39	0	2
M2	Rotating camera	2	Moving	9.15	12.84	6.83	10	0	18
M3	Rotating camera	2	Moving	14.00	16.96	11.56	1	0	10
M4	Follow simple	1	Moving	32.56	32.43	32.68	18	2	6
M5	Follow, low altitude	2	Moving	18.92	27.57	13.09	15	2	28
M6	Follow, low altitude	2	Moving	13.19	21.65	8.06	43	4	59
M7	Follow, sharp turns	2	Moving	11.41	23.50	5.61	89	10	108
Overall				13.74	14.97	12.81	443	19	300

For the system operating in RGB mode, Table 4.7 shows that the most challenging tracking scenario, based on the HOTA score, is the far away with two drones scenario. In 1 out of the 15 scenarios, the AssA score is higher than DetA, where the difference between them is 0.1. They are the same in 4 of the scenarios, with DetA being higher than AssA in 10 of the scenarios.

Based on the HOTA scores reported in Table 4.8, the occlusion scenario appears to be the most challenging for tracking in IR mode. The far away scenario also produces a comparatively low HOTA score relative to the other evaluated scenarios. Across the 15 scenarios, DetA is higher than AssA in 11 cases, while the two metrics are equal in two scenarios. In the remaining two scenarios, AssA exceeds DetA, with a maximum difference of 0.25. Overall, these results indicate that tracking performance in both RGB and IR mode is more constrained by association accuracy than by detection accuracy.

Compared with IR mode, the RGB mode results in a higher number of Frag and IDSW. However, the number of IDs is generally closer to the ground-truth number of drones. In addition, RGB mode achieves higher HOTA, DetA, and AssA scores, indicating superior overall tracking performance compared to IR mode.

4.4 System Performance

The systems performance was measured on a machine running EndeavourOS with GeForce RTX 3050 4GB (35W), Intel Core i7-12650H and 16GB of ram. Table 4.9 displays the performance of the IR pipeline. Table 4.10 shows the average computational time of each individual module in the M2 simulation, chosen because it achieved the median FPS and has at least 1 drone in the frame throughout the whole simulation. Table 4.11 shows the performance of the RGB pipeline, and Table 4.12 shows the RGB modules performance on the M2 simulation.

Table 4.9: System Performance Ir

ID	Description	Drones	Camera	FPS	Classification Recall
S1	Simple motion	1	Stationary	21.2	0.937
S2	Altitude difference	1	Stationary	20.9	0.910
S3	Far away	1	Stationary	20.9	0.908
S4	Simple motion	2	Stationary	20.3	0.938
S5	Altitude difference	2	Stationary	21.3	0.928
S6	Far away	2	Stationary	21.4	0.911
S7	Occlusion	2	Stationary	21.4	0.899
S8	Crash	2	Stationary	21.6	0.963
M1	Rotating camera	1	Moving	21.4	0.867
M2	Rotating camera	2	Moving	21.3	0.909
M3	Rotating camera	2	Moving	21.7	0.954
M4	Follow simple	1	Moving	21.5	0.793
M5	Follow, low altitude	2	Moving	21.2	0.912
M6	Follow, low altitude	2	Moving	20.4	0.718
M7	Follow, sharp turns	2	Moving	20.9	0.814
Overall				21.16	0.891

Table 4.10: IR Average Module Performance in Ms

ID	Description	Drones	Camera	Detection	Tracking	Classification	Flow Estimation
M2	Rotating camera	2	Moving	30.75	0.42	4.7	15.27

Table 4.11: System Performance RGB

ID	Description	Drones	Camera	FPS	Classification Recall
S1	Simple motion	1	Stationary	21.5	0.930
S2	Altitude difference	1	Stationary	21.5	0.862
S3	Far away	1	Stationary	21.6	0.887
S4	Simple motion	2	Stationary	21.3	0.934
S5	Altitude difference	2	Stationary	21.6	0.883
S6	Far away	2	Stationary	21.2	0.932
S7	Occlusion	2	Stationary	20.9	0.951
S8	Crash	2	Stationary	21.2	0.928
M1	Rotating camera	1	Moving	21.0	0.851
M2	Rotating camera	2	Moving	21.0	0.843
M3	Rotating camera	2	Moving	20.5	0.912
M4	Follow simple	1	Moving	21.0	0.599
M5	Follow, low altitude	2	Moving	20.8	0.665
M6	Follow, low altitude	2	Moving	21.2	0.770
M7	Follow, sharp turns	2	Moving	21.3	0.766
Overall				21.17	0.848

Table 4.12: RGB Average Module Performance in Ms

ID	Description	Drones	Camera	Detection	Tracking	Classification	Flow Estimation
M2	Rotating camera	2	Moving	30.72	0.39	6.1	15.93

Across all scenarios, both the IR and RGB pipelines maintain real-time performance, with an average throughput of approximately 21 fps. The end-to-end classification recall is high for both modalities (overall 0.891 for IR and 0.848 for RGB), showing that the full pipeline can reliably assign correct labels to active tracks under a variety of motion patterns and camera configurations. Recall is lowest in the most challenging moving-camera scenarios, where targets are small, move rapidly, and are occasionally partially occluded. Compared to the standalone CLIP evaluation in Section 4.1, these system-level recalls are higher, which is expected since classification is performed only on confirmed tracks and aggregated over time, making the metric less sensitive to occasional misclassifications in individual frames. Overall, the results show that the proposed system meets the real-time requirement while preserving strong classification performance in realistic operating conditions.

5

Discussion

This chapter discusses the performance of the proposed vision system based on the results presented in the previous chapter. Each system component is analyzed in a separate section, followed by an evaluation of the overall system performance and its applicability in real-world scenarios. The chapter concludes with a discussion of the system's societal and ethical implications, suggestions for future work, and methodological reflection.

5.1 Detection

Based on the results in section 4.2, this section discusses the detection performance and examines its strengths and limitations across different scenarios. The results show that both detection pipelines achieved strong overall performance across the simulated scenarios, as presented in Table 4.5 and Table 4.6. This is reflected in generally high precision, recall, and F1 scores. High recall is particularly important for the overall vision system, since missed detections interrupt target tracking and have larger negative effects than occasional false positives. These results indicate that the RF-DETR small model is capable of providing sufficiently reliable detections to maintain stable target tracking.

The RGB pipeline achieved the strongest overall performance, maintaining high recall and precision across most simulations (Table 4.6). Performance degradation was mainly observed in scenarios containing multiple overlapping drones, significant camera motion, or limited visibility of the target. In scenario S4, the recall dropped noticeably due to overlapping targets, which reduced the model's ability to separate the drones into individual detections. Analysis of the annotations showed that in approximately 10.2% of the simulation duration, one target was fully enclosed within the other target's bounding box. Similar reductions in recall were observed in M5–M7, where the drones were frequently partially visible or located near the image boundaries, reducing detection consistency. Figure 5.1 shows scenario M6 where one drone is fully visible while the other is partially outside the image frame.

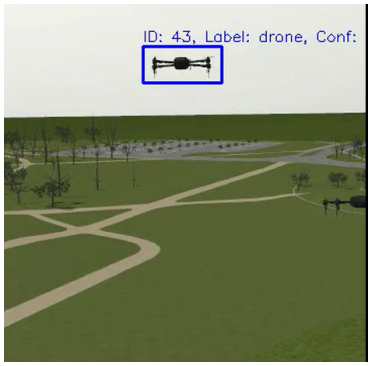

Figure 5.1: Partially visible target

The IR pipeline showed greater variation between scenarios, particularly in precision (Table 4.5). Scenarios containing trees and cluttered backgrounds, such as S7 and M5-M7, resulted in an increase in false positives. In the synthetic IR imagery, trees had visual brightness similar to the objects of interest and, at greater distances, could resemble drones, causing the detector to incorrectly classify them as targets. Figure 5.2 demonstrates this situation. This shows the limitations of post-processing as the method to obtain IR footage, as certain objects do not translate well. Despite the reduction in precision, the IR model maintained relatively high recall in most scenarios, including long-range cases such as S6. This suggests that the detector generalizes well to small targets even when visual information is limited.

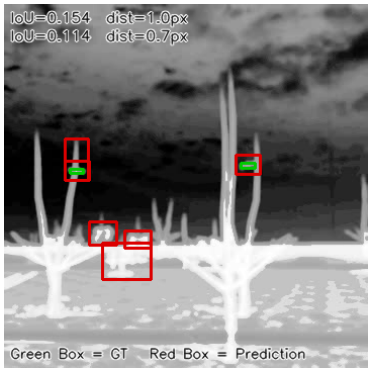

Figure 5.2: Tree in simulation

Both pipelines showed low AP50 scores despite strong recall and precision values. This is mainly caused by the ground-truth boxes generated from the simulations being very slim, while the predicted boxes were significantly larger. As a result, the IoU between predicted and ground-truth boxes were too small to count as a TP even if the predicted box fully covered the ground-truth box. In the IR pipeline, one contributing factor to the larger predicted boxes was that the ground-truth boxes were smaller than the actual drone. This is because of the way post-processing was applied on the RGB-frames to create the IR-frames. In particular, the gaussian blur causes the drones to appear bigger compared to the RGB-frame, and as the ground-truth is based on the RGB-frame, a mismatch occurs. Figure 5.3 illustrates the difference in box sizes and shows that the ground-truth annotations are smaller than the actual visible extent of the target.

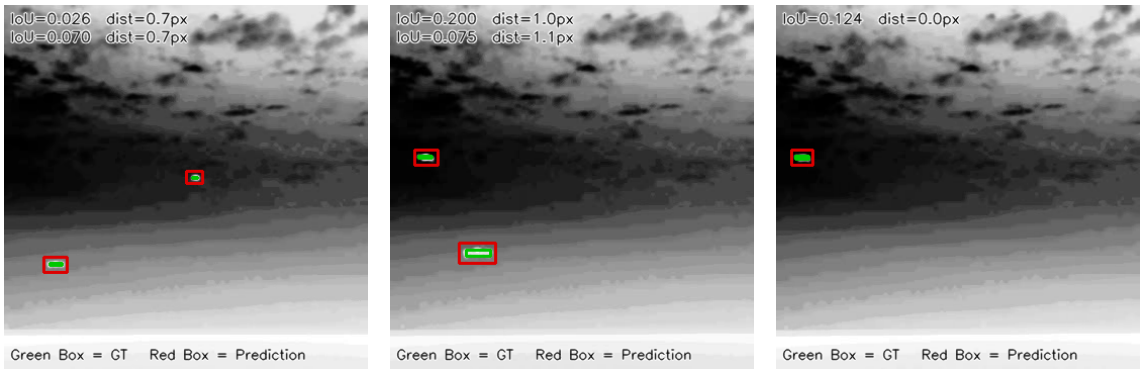

Figure 5.3: Difference in box sizes for IR

However, this does not explain the low AP50 score observed in the RGB mode. Inspection of the videos shows that the detector consistently predicts bounding boxes that are larger than the ground-truth. Moreover, the discrepancy appears to increase with distance to the drone. Figure 5.4 illustrates this effect even at relatively short distances. Compared with the images in Figure 5.5, it is evident that the difference becomes more pronounced as the drones move farther away. In addition, some ground-truth boxes are inaccurate, as they do not fully enclose the drone, as shown in the second and third images of Figure 5.5. Such cases have only been observed in the far away scenarios, suggesting that the quality of the ground-truth annotations degrades when the drones are sufficiently distant.

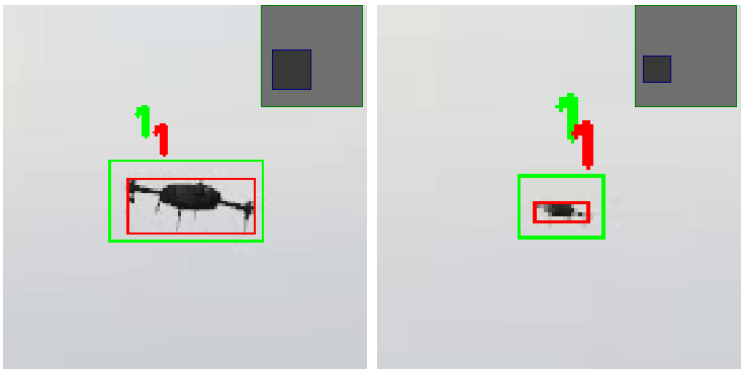

Figure 5.4: Frames from S1 (simple motion). Red box is ground-truth, green is predicted by the model. The item in the top-right corner of each picture is an artifact from OpenCV as the images were zoomed-in.

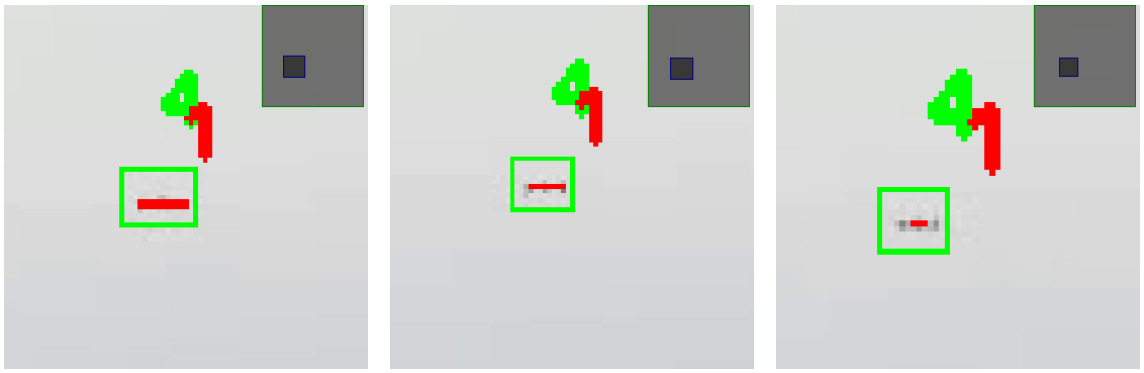

Figure 5.5: Frames from S3 (far away). Red box is ground-truth, green is predicted by the model. The third picture contains a particularly unreasonable ground-truth box. The item in the top-right corner of each picture is an artifact from OpenCV as the images were zoomed-in.

Overall, these results indicate that the detector is capable of localizing objects consistently and effectively in most cases. However, it fails to predict accurately sized bounding boxes when the objects are sufficiently far away. We hypothesize that the impact of this issue on the remaining pipeline is limited. For association in BYTE, consistently oversized boxes should still be matched reliably, since the algorithm is based on IoU. For classification, the additional context may even improve performance, as CLIP generally benefits from surrounding visual information. Therefore, the significance of the box size issue depends entirely on whether the intended application requires precise bounding box dimensions.

The RGB model performed slightly better than the IR model. One contributing factor to this is that the default weights are derived from training on RGB images, whereas the IR model had to adapt and learn to detect objects on IR images. Given that the IR model was fine-tuned on approximately 5300 images, the results are strong. Both models showed reduced performance on M5-M7, likely due to the second target being at greater distances with stronger camera motion.

In relation to the system requirements in section 1.3, the results indicate that the functional requirement for detecting airborne objects in both RGB and IR video is satisfied, as both models maintain reliable detections across most scenarios. However, the reduced performance in more challenging conditions, particularly those involving camera motion and long distance, shows limitations in robustness. This relates to the non-functional requirement for stable performance across varying environments.

5.2 Tracking

An unexpected result is the high number of Frag. For example, more fragmentations occur during simple motion compared to occlusion. Comparing Frag and IDs suggests that these fragmentations are caused by missed detections, since the simple motion scenario contains only one ID but 73 fragmentations. However, the tracker

produces 2872 bounding boxes, compared with 2874 in the ground-truth. Together with the high localization-based F1 score reported in Table 4.6, this indicates that the model generates a single trajectory corresponding to the ground-truth object. Inspection of the video footage with overlaid bounding boxes confirms that the drone is tracked consistently without visible fragmentation.

The discrepancy between the high number of fragmentations and the other performance metrics, together with the visual inspection of the annotated videos, can be explained by how Frag is computed. The metric uses IoU and the Hungarian algorithm to associate predicted and ground-truth bounding boxes. Consequently, a predicted box that fully contains the drone may still be considered unmatched if it is too large, causing the IoU to fall below the matching threshold. Because the model tends to produce systematically oversized bounding boxes, these detections are occasionally classified as false positives. This creates apparent gaps in the trajectory, which are counted as fragmentations. Therefore, the Frag metric should be interpreted with considerable caution when assessing the model's true fragmentation behavior.

The box size issue also affects the HOTA, DetA, and AssA scores. Because the association between predicted detections and ground-truth objects is based on the IoU of their bounding boxes, systematically oversized boxes can degrade the matching quality. Since all three metrics depend on these associations, imperfect matches lead directly to lower scores and may underestimate the model's actual tracking performance. Nevertheless, the metrics remain useful for relative comparisons between scenarios.

Beyond the dedicated occlusion scenario in S7, occlusions also occur in several other scenarios due to interactions between the drones. They are particularly common in the simple motion scenario with two drones (S4), as both drones remain at the same altitude throughout the sequence. In most cases, the model fails to recapture the occluded drone, resulting in a new ID for one of the drones. An example of this behavior is shown in Figure 5.6. These events can explain why the HOTA score drops in S4.

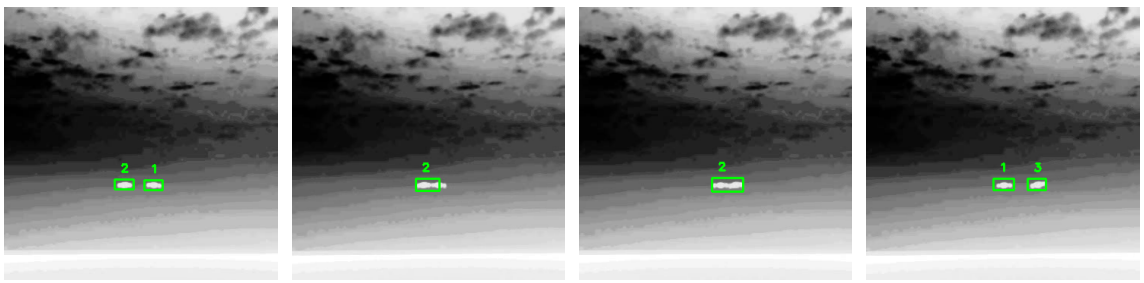

Figure 5.6: Occlusion example from scenario S4, resulting in the model assigning a new ID to one of the drones after occlusion is over.

Comparing S4 to the altitude difference scenario with two drones (S5), one might expect S5 to be more challenging and therefore to yield a lower HOTA score. How-

ever, the opposite is observed. As shown in Table 4.7 and Table 4.8, the DetA score remains largely unchanged between the two scenarios, whereas AssA is substantially lower in S4. Together with the higher number of identity switches and IDs in S4, this indicates that occlusion events are the primary cause of the performance degradation, primarily affecting association rather than detection. These results suggest that occlusion poses a greater challenge to the tracker than increased motion complexity.

For all scenarios, the HOTA score is higher in RGB mode compared to IR mode. The only difference in how tracking operates in the two modes is the change in the detector, as the association depends solely on the bounding boxes it receives. However, as previously discussed, it is hard to make a definitive conclusion on the IR detection performance given the problems with the IR-frames. Figure 5.7 shows an example where the unrealistic IR frames may have negatively affected the detection performance, and consequently the association performance. Part of the unrealistically bright tree is detected and tracked throughout the sequence. This explains why S7 for IR has the lowest HOTA score.

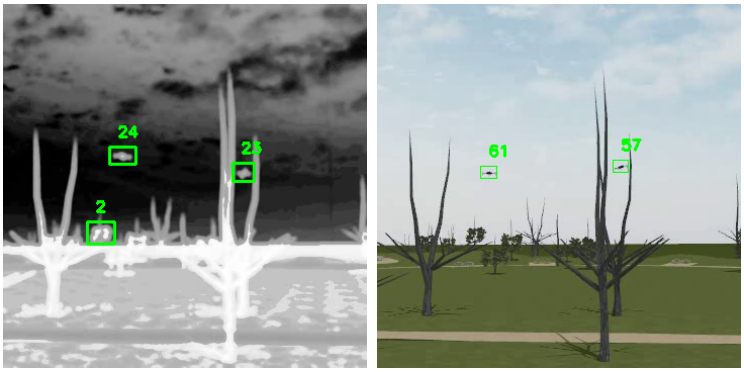

Figure 5.7: Example of false positive detection resulting in false positive tracking. Left picture (IR) shows part of a tree detected, marked with ID 2 across the entire video. Right picture successfully does not detect and track it.

In the scenarios where the camera is rotating (M1-M3), the HOTA score for RGB is similar to many of the scenarios where the camera is stationary (S1, S2, S4, S5, S8). This indicates that ego-motion compensation may have a positive effect on tracking performance. Nevertheless, without a direct comparison with and without ego-motion compensation, no definitive conclusion regarding its impact can be drawn. In addition, the tracker is sometimes able to maintain the correct identity through short-term occlusions, as illustrated in Figure 5.8.

In the follow scenarios (M4-M7), the HOTA score for RGB is also comparable to the scenarios where the camera is stationary. However, their DetA scores are substantially higher, all exceeding 40. This is likely because the drones remain closer to the camera throughout the follow sequences, which the detector is significantly better at predicting a correctly sized box for, ultimately enhancing the score. The AssA score being lower in M5, M6 and M7 can be explained by the higher number of

out-of-view events that occur, which the model cannot handle while it still punishes the AssA score.

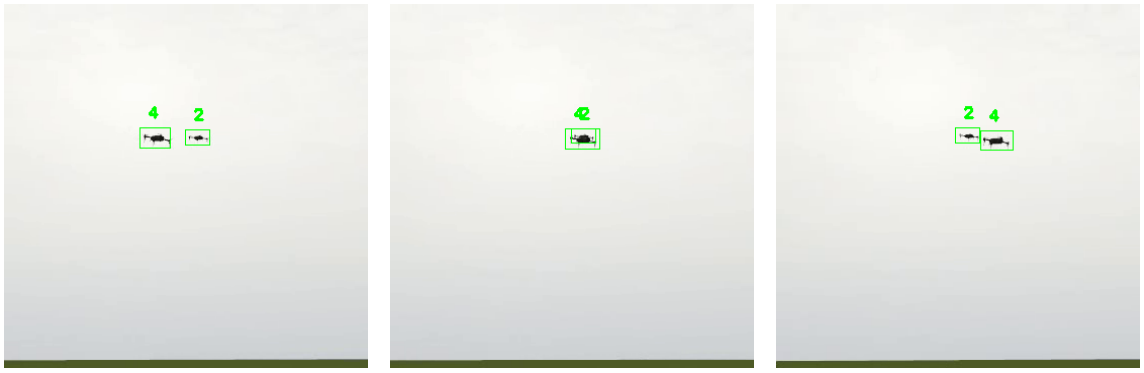

Figure 5.8: Model successfully maintaining ID during short-term occlusion, scenario M2

In conclusion, the system partially fulfills the tracking requirements. The system is capable of tracking multiple objects, provided that they remain within the field of view, as specified. It demonstrates partial robustness under varying conditions, although it struggles in certain scenarios. Additionally, while the performance of the IR mode is difficult to assess, the results indicate that the system is capable of tracking in IR mode, albeit with lower performance compared to RGB mode.

5.3 Classification

This subsection analyzes how different CLIP training strategies used in this work translate into the empirical results reported in Section 4.1. We relate the four IR models to their performance on the test split (Table 4.1) and on the simulated evaluation scenarios (Table 4.2).

On the test split, moving from IR-CLIP Base to CMKD 1-class and CMKD multi-class yields moderate gains in macro F1, but these improvements do not carry over to the more challenging simulated scenarios. Both CMKD variants still exhibit low drone recall across many videos, indicating that their embedding spaces remain poorly calibrated for small, low-contrast aerial targets under varying conditions. In contrast, the LoRA + CMKD configuration achieves the highest macro F1 on the test split, and more importantly, substantially higher drone recall across all scenarios. This suggests that first adapting the RGB teacher with LoRA and then distilling into the IR encoder is more effective than cross-modal distillation from an unadapted teacher.

Part of the discrepancy between the test split and the scenario evaluation can be attributed to a mismatch in data distributions. The training and test data consist of real IR imagery, whereas the evaluation scenarios use pseudo-IR images generated from simulated RGB frames (Section 3.5), with different intensity statistics and noise characteristics. Under these conditions, all models degrade in absolute

performance, but the relative ordering remains consistent: methods that induce a better-structured embedding space during training are also the ones that generalize best to the simulated scenarios.

Figure 5.9 shows the UMAP projection of the IR-CLIP embedding space for the base model and the LoRA + CMKD model, with the visual prototypes overlaid. In the base IR-CLIP model, which applies only IR preprocessing and no task-specific training, the four classes occupy a heavily overlapping central region. All four classes form a single entangled cluster with no clear boundaries between them. The prototypes reflect this poor separability, as they all lie in the same central region, with drone and helicopter prototypes nearly coinciding. This is expected behavior for a model that was never trained on IR data. CLIP learned a joint image-text embedding space from RGB images, and when fed IR-inputs it maps them into a compact region without the fine-grained structure to distinguish morphologically similar aerial objects.

After LoRA + CMKD training, the embedding space exhibits a markedly different geometry. The overall distribution is more spread out, and although the classes still overlap to some degree, the four prototypes are now substantially further apart. The drone prototype sits in a region with a higher density of drone samples, the helicopter prototype has moved toward the lower part of the space where helicopter samples are more concentrated, and the bird and airplane prototypes are similarly better aligned to their respective sample distributions. This wider prototype separation directly corresponds to larger classification margins at inference time and is consistent with the observed increase in macro F1 (Table 4.1). The effect of LoRA + CMKD training goes beyond a uniform shift in the embedding space; it appears to have reorganized the structure in a way the prototype-based classifier responds to more confidently.

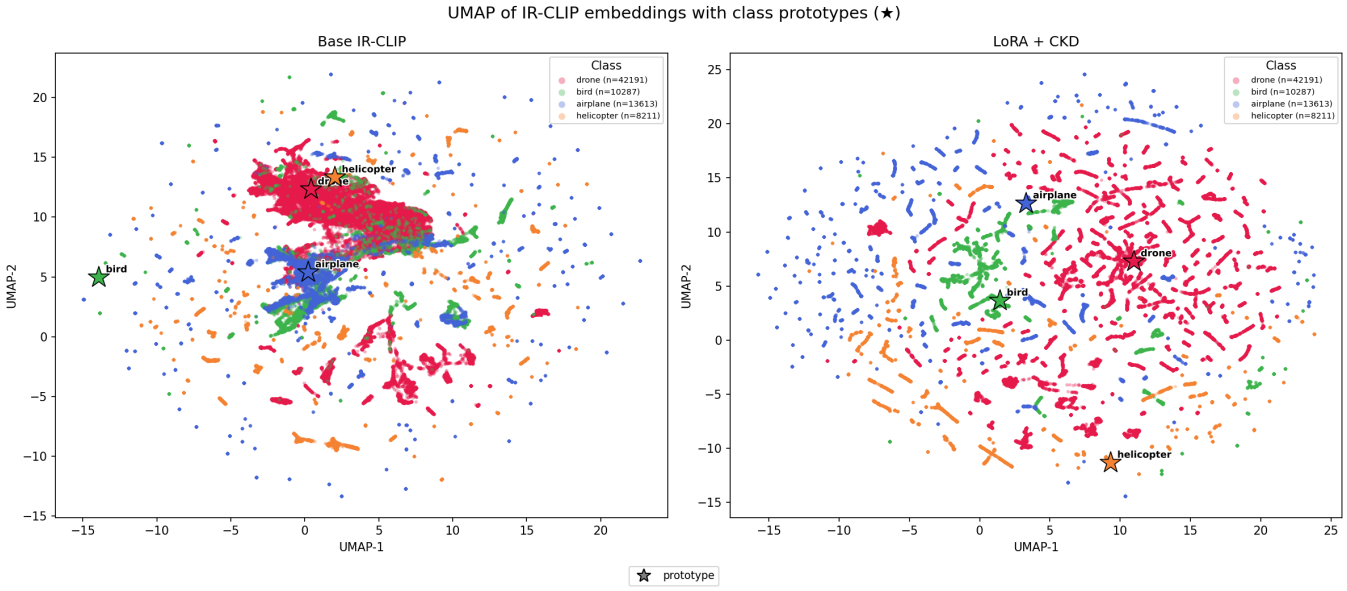

Figure 5.9: UMAP of embedding space for Base IR-CLIP and LoRA + CMKD with visual prototypes

To better understand how these changes manifest at the image level, GradCAM++ [73] was applied to visualize which spatial regions drive the classification decisions. For images that the base IR-CLIP model misclassifies but LoRA + CMKD classifies correctly, both models tend to focus on very similar regions around the target, which is expected since the query, key, and value projections in the image encoder are kept frozen during LoRA adaptation. The key distinction lies in how the attended features are represented in the embedding space: both models direct attention to the same locations, but encode those features differently.

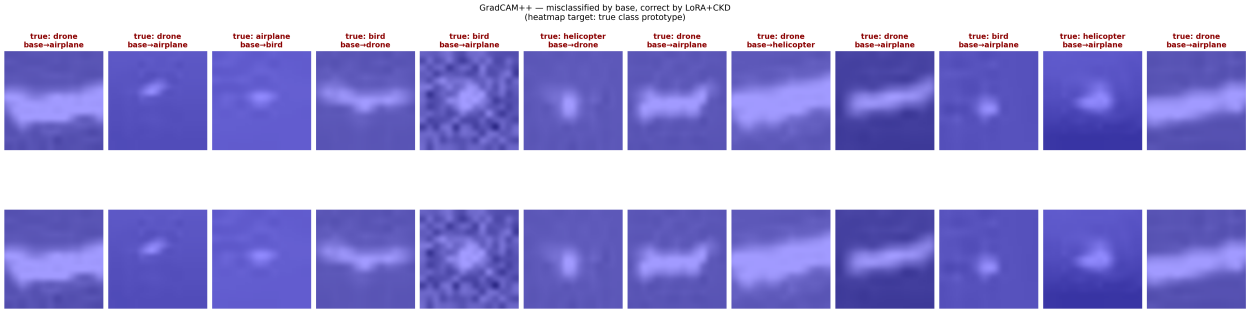

Figure 5.10: GradCAM++ heatmaps from images misclassified by base IR-CLIP (top) but correctly classified by LoRA + CMKD (bottom). The similar attention patterns between rows confirms that fine-tuning preserves the spatial attention structure by design.

These qualitative UMAP and GradCAM++ visualizations should be interpreted with some caution, since dimensionality reduction and attribution methods can distort distances and local neighborhoods. However, the improved separation between prototypes, the emergence of more distinct class regions, and the consistent attention patterns all align well with the quantitative metrics. Together, they indicate

that, for the data and models considered in this work, adapting the RGB teacher with LoRA substantially improves effectiveness of cross-modal distillation to IR compared to using an unadapted teacher. Practically, our results point to a useful design pattern for similar resource-constrained classification systems: first specialize a strong RGB teacher on the relevant aerial domain using a lightweight adapter such as LoRA, and then distill its representations into an IR student, rather than relying solely on cross-modal knowledge distillation from an unadapted teacher.

5.4 Real-World Applicability

The purpose of this work was to develop a modular, real-time vision system for UAV platforms capable of detection, classification, and multi-object tracking under resource constraints. The results demonstrate that the system can detect and track airborne objects in real-time, and that classification of drones versus other aerial objects is feasible, especially in scenarios where the drone is in relatively close distance and enough visual features are in the frame. However, several factors limit the system's applicability in real-world scenarios.

The decision to use visual prototypes as classification inference methods introduces both advantages and limitations. While the use of prototypes improves classification performance, it also hinders the zero-shot capabilities. The visual prototypes are computed using only the training data and thereby limiting classification to only classes seen by training. In our case, the system is limited to `"Drone"`, `"Helicopter"`, `"Plane"`, and `"Bird"` as predictable classes. In real-world use cases, such as airspace monitoring additional classes like `"zeppelin"` or `"human"` may also be needed. If such classes are not present in the training data, the system cannot recognize them without collecting new labeled examples and recomputing the visual prototypes.

Furthermore, the simulation environment limits how well the results transfer to real-world settings, as testing is carried out in a controlled and synthetic environment. The simulated footage contains some sensor noise and visual distortions, but at lower levels than typically observed in real UAV recordings, including camera vibrations, motion blur, compression artifacts, and rolling-shutter effects. The simulated scenarios also exclude varying weather or other complex environments identified as a challenge [15]. As a result, both detection and tracking may perform less reliably in real operational conditions, particularly during rapid ego-motion, occlusions, or cluttered scenes. The tracking results already demonstrate occasional ID switches within simulation, indicating that these issues would likely become more pronounced in real-world deployments.

Although the system is designed to operate online using only current and past frames, the evaluation was conducted on pre-recorded video rather than a live camera stream. In practice, transitioning to a live feed introduces additional constraints that were not captured during evaluation. Furthermore, processing time is not constant in real-world scenarios, as it depends on factors such as the number of active

tracks and complexity of the detection. Consequently, the worst-case latency may be higher than the average suggests. In a live feed scenario, the camera could therefore outpace the pipeline in complex scenarios, requiring frames to be either dropped, which might result in losing track of fast moving targets, or buffered, which introduces a latency between input and output that might be unacceptable in time-critical applications.

5.5 Methodological Reflection

The work followed the methodological plan from Section 1.4: an initial broad literature review, division into three subcomponents, iterative component-wise development, and final end-to-end evaluation in simulation. In retrospect, this staged approached proved effective for fast development and for isolating failure modes (e.g. classifier confusion between drones and birds). However, it also meant that interaction between the different components were only discovered late in the process. A tighter co-design of components might have mitigated this. Nonetheless, the adopted method provided a clear structure and separability given the tight time frame.

5.6 Societal and Ethical Aspects

The development of the vision system has been carried out with consideration for deployment in both civilian and military applications, including airspace monitoring, infrastructure protection, and defense-related scenarios. As the system is designed for automatic detection, classification, and tracking of airborne objects, its intended functionality is inherently connected to surveillance tasks. This dual-use aspect introduces ethical concerns regarding how the system may be used in practice.

A central concern is the risk of misclassification within the system. In practical deployment, confusion between visually similar classes, such as drones and birds, can directly affect situational awareness in airspace monitoring systems. False positives may lead to unnecessary responses, while false negatives may result in undetected aerial activity. This raises questions about where responsibility lies when incorrect outputs contribute to operational decisions, making human oversight essential, particularly in scenarios where decisions may escalate to real-world interventions.

Another ethical concern relates to surveillance and privacy. In civilian applications, continuous monitoring of airspace may raise privacy concerns if individuals or private activities are unintentionally captured in the camera field of view. The system does not perform identification of individuals or collect personal data, meaning that privacy implications primarily depend on the context in which the system is deployed.

Given the military application of the system and the associated risk of misuse, the system and its underlying implementation are not publicly available and access

is restricted. In Sweden, distribution and use are subject to national regulations, including the Military Equipment Act [74], and oversight by the Inspectorate of Strategic Products. These frameworks require licensing, end-user verification, and consideration of the recipient country's human rights situation, thereby providing mechanisms intended to reduce the risk of unauthorized use.

5.7 Future Work

The current preprocessing converts IR images to greyscale to match CLIP's expected RGB format. While functional, this might discard potential information. A direction for future work is to investigate alternative channel representations that could e.g. contain spatial information to aid detection and classification. Alternatively, incorporating a thermal camera in the simulation setup capable of generating IR frames from simulated temperature data could increase the realism of the environment compared to the approach used in this work.

Additionally, one of the systems central challenges is the handling of long range objects. Sensor fusion [15] has the potential to combine the detailed texture information form RGB while preserving the IR ability to highlight objects in low-light conditions. Such an approach could reduce some of the difficulties observed in the challenging scenarios, in particular the detection and classification of far-away targets.

Furthermore, new research centered around autonomous driving propose the method Telescope [75] for ultra-long-range detection. Telescope utilize a warping mechanism on far away objects to perform detection on higher resolution imagery, resulting in more accurate detection. Applying a similar approach for detection of smaller objects in aerial contexts could increase system performance among all system components in the more challenging scenarios with limited additional computation.

6
Conclusion

This project set out to address a gap in existing anti-UAV systems by designing a modular real-time vision pipeline for detection, classification, and multi-object tracking of airborne objects from a moving aerial platform. The proposed system integrates RF-DETR for object detection, ByteTrack for tracking, and a CLIP-based classifier. To bridge the gap between RGB-trained models and IR deployment, and to improve robustness in complex visual environments, the system utilizes RF-DETR fine-tuning, cross-modal knowledge distillation with Low-Rank Adaptation and optical-flow-based ego-motion compensation.

The system was evaluated across both RGB and IR modalities under varying conditions, including ego-motion, partial occlusions, and different target distance. The results demonstrated feasibility for air space monitoring deployment, achieving strong precision and recall for both detection and classification while maintaining real-time performance. A central finding was that adapting the RGB teacher with LoRA prior to cross-modal distillation improved classification performance.

The detection module systematically predicted bounding boxes that were larger than the ground-truth annotations, while still maintaining accurate object localization. A majority of tracking metrics rely on the IoU between predicted and ground-truth bounding boxes. Therefore there is a systematic overestimation which had an substantial negative impact on the reported scores. As a result, the reported tracking performance appears worse than the actual qualitative behavior suggests.

Although the system performance degraded in more challenging scenarios, the overall results are promising and indicate that the proposed system can handle multiple real-world constraints. However, the system was evaluated on simulations rather than real-world scenarios and not on embedded UAV hardware. As a result, the evaluation does not fully capture the complexities of real-world applications. Further validation in real operational environments is therefore required before the system can be considered deployment ready.

References

[1] R. Singh and S. Kumar, "A comprehensive insights into drones: History, classification, architecture, navigation, applications, challenges, and future trends," 2025. [Online]. Available: https://arxiv.org/abs/2501.10066

[2] O. Ozdemir, I. Guvenc, W. Khawaja, M. Ezuma, V. Semkin, and F. Erden, "A survey on detection, classification, and tracking of aerial threats using radar and communications systems," 2024. [Online]. Available: https://arxiv.org/abs/2402.05909

[3] S. Habibi, N. Ivaki, and J. Barata, "A systematic literature review of unmanned aerial vehicles for healthcare and emergency services," *arXiv preprint arXiv:2504.08834*, 2025.

[4] F.-Y. Wang, J. Huang, Y. Wang, Y. Lv, Y. Li, C. Tian, B. Li, Y. Tian, X. Dai, F. Lin, T. Zhang, Q. Zhang, X. Fu, and L. Kovács, "Uavs meet llms: Overviews and perspectives toward agentic low-altitude mobility," 2025. [Online]. Available: https://arxiv.org/abs/2501.02341

[5] J. Redmon, S. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Computer Vision and Pattern Recognition*, 2015, pp. 779–788.

[6] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C.-y. Li, J. Yang, H. Su, J.-J. Zhu, and L. Zhang, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2023, pp. 38–55.

[7] C. Chen and X. Pan, "Deep learning for inertial positioning: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 10, pp. 12 658–12 681, 2024.

[8] B. Jensen. (2025, Apr.) Fewer soldiers, more drones: What ukraine's military will look like after the war. Center for Strategic and International Studies. Commentary. [Online]. Available: https://www.csis.org/analysis/fewer-soldiers-more-drones-what-ukraines-military-will-look-after-war

[9] C. Schüpbach, S. Glüge, M. Nyfeler, A. Aghaebrahimian, and N. Ramagnano. (2024) Robust low-cost drone detection and classification in low snr environments. [Online]. Available: https://arxiv.org/abs/2406.18624

[10] Y. Dong, F. Wu, S. Zhang, G. Chen, Y. Hu, M. Yano, J. Sun, S. Huang, F. Liu, Q. Dai, and Z.-Q. Cheng, "Securing the skies: A comprehensive survey on anti-uav methods, benchmarking, and future directions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 6 2025, pp. 6725–6739.

[11] Z. Tang, T. Xu, Z. Feng, X. Zhu, C. Cheng, X.-J. Wu, and J. Kittler, "Revisiting rgbt tracking benchmarks from the perspective of modality validity: A new benchmark, problem, and solution," 2025. [Online]. Available: https://arxiv.org/abs/2405.00168

[12] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, pp. 12 878–12 895, 2022.

[13] X. Chen, B. Yan, J. Zhu, D. Wang, X. Yang, and H. Lu, "Transformer tracking," 2021. [Online]. Available: https://arxiv.org/abs/2103.15436

[14] M. A. Amer and M. Falaki, "Lightweight rgb-t tracking with mobile vision transformers," 2025. [Online]. Available: https://arxiv.org/abs/2506.19154

[15] D. Wang, Z. Jiang, L. Ma, Z. Liu, J. Liu, X. Fan, R. Liu, G. Wu, and W. Zhong, "Infrared and visible image fusion: From data compatibility to task adaption," 2025. [Online]. Available: https://arxiv.org/abs/2501.10761

[16] E. Granger, M. Pedersoli, H. R. Medeiros, A. Belal, and S. Muralidharan, "Visual modality prompt for adapting vision-language object detectors," 2024. [Online]. Available: https://arxiv.org/abs/2412.00622

[17] Chalmers University of Technology, "Regulations for the use of ai tools in thesis work," 2024. [Online]. Available: https://www.chalmers.se/en/education/your-studies/masters-and-bachelors-thesis/regulations-for-the-use-of-ai-tools/

[18] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *CoRR*, vol. abs/2005.12872, 2020. [Online]. Available: https://arxiv.org/abs/2005.12872

[19] U. Seidaliyeva, L. Ilipbayeva, K. Taissariyeva, N. Smailov, and E. T. Matson, "Advances and challenges in drone detection and classification techniques: A state-of-the-art review," *Sensors*, vol. 24, no. 1, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/1/125

[20] M. Nikouei, B. Baroutian, S. Nabavi, F. Taraghi, A. Aghaei, A. Sajedi, and M. E. Moghaddam, "Small object detection: A comprehensive survey on challenges, techniques and real-world applications," *Intelligent Systems with Applications*, vol. 27, p. 200561, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667305325000870

[21] P. Tsirtsakis, G. Zacharis, G. S. Maraslidis, and G. F. Fragulis, "Deep learning for object recognition: A comprehensive review of models and algorithms," *International Journal of Cognitive Computing in Engineering*, vol. 6, pp. 298–312, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266630742500004X

[22] T. Shehzadi, K. A. Hashmi, M. Liwicki, D. Stricker, and M. Z. Afzal, "Object de-

tection with transformers: A review," *Sensors*, vol. 25, no. 19, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/19/6025

[23] I. Robinson, P. Robicheaux, M. Popov, D. Ramanan, and N. Peri, "Rf-detr: Neural architecture search for real-time detection transformers," 2026. [Online]. Available: https://arxiv.org/abs/2511.09554

[24] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2024. [Online]. Available: https://arxiv.org/abs/2304.07193

[25] B. Sharma, "Rf-detr segmentation: Real-time detection & instance segmentation guide," https://learnopencv.com/rf-detr-segmentation-real-time-detection-instance-segmentation-guide/, 2026, accessed: 2026-04-21.

[26] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee, "Rf-detr object detection vs yolov12 : A study of transformer-based and cnn-based architectures for single-class and multi-class greenfruit detection in complex orchard environments under label ambiguity," 2025. [Online]. Available: https://arxiv.org/abs/2504.13099

[27] P. Robicheaux, J. Gallagher, J. Nelson, and I. Robinson, "Rf-detr: A sota real-time object detection model," https://blog.roboflow.com/rf-detr/#rf-detr-architecture-overview, 2025, accessed: 2026-04-21.

[28] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157 vol.2.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[30] A. Munir, F. Z. Qureshi, M. Ali, and M. H. Khan, "Compositional zero-shot learning: A survey," 2025. [Online]. Available: https://arxiv.org/abs/2510.11106

[31] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds., vol. 22. Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2009/file/1543843a4723ed2ab08e18053ae6dc5b-Paper.pdf

[32] C. H. Lampert, H. Nickisch, S. Harmeling *et al.*, "Learning to detect unseen object classes by between-class attribute transfer." in *Cvpr*, vol. 1, no. 2. Miami, FL, 2009, p. 3.

[33] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[34] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[36] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[37] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Vision transformers," https://d2l.ai/chapter_attention-mechanisms-and-transformers/vision-transformer.html, 2023, dive into Deep Learning, Chapter 11.8.

[38] C. Hu, X. Li, D. Liu, H. Wu, X. Chen, J. Wang, and X. Liu, "Teacher-student architecture for knowledge distillation: A survey," *arXiv preprint*, aug 2023, arXiv:2308.04268. [Online]. Available: https://arxiv.org/abs/2308.04268

[39] S. Stanton, P. Izmailov, P. Kirichenko, A. A. Alemi, and A. Wilson, "Does knowledge distillation really work?" *ArXiv*, vol. abs/2106.05945, 2021.

[40] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model compression," vol. 2006, 08 2006, pp. 535–541.

[41] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv*, vol. abs/1503.02531, 2015.

[42] J. Malik, S. Gupta, and J. Hoffman, "Cross modal distillation for supervision transfer," 2015. [Online]. Available: https://arxiv.org/abs/1507.00448

[43] C. Zhao, Y. Jin, Z. Song, H. Chen, D. Miao, and G. Hu, "Cross-modal distillation for widely differing modalities," 2025. [Online]. Available: https://arxiv.org/abs/2507.16296

[44] J. F. P. Kooij, E. Granger, A. Bhuiyan, and F. Hafner, "Cross-modal distillation for rgb-depth person re-identification," 2022. [Online]. Available: https://arxiv.org/abs/1810.11641

[45] D. Ienco and C. F. Dantas, "Discom-kd: Cross-modal knowledge distillation via disentanglement representation and adversarial learning," 2024. [Online]. Available: https://arxiv.org/abs/2408.07080

[46] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[47] M. Adžemović, "Deep learning-based multi-object tracking: A comprehensive survey from foundations to state-of-the-art," 2025. [Online]. Available: https://arxiv.org/abs/2506.13457

[48] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, Sep. 2016, p. 3464–3468. [Online]. Available: http://dx.doi.org/10.1109/ICIP.2016.7533003

[49] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960. [Online]. Available: https://doi.org/10.1115/1.3662552

[50] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Bytetrack: Multi-object tracking by associating every detection box," 2022. [Online]. Available: https://arxiv.org/abs/2110.06864

[51] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, "Bot-sort: Robust associations multi-pedestrian tracking," 2022. [Online]. Available: https://arxiv.org/abs/2206.14651

[52] P. R. Singh, R. Gottumukkala, and A. Maida, "An analysis of kalman filter based object tracking methods for fast-moving tiny objects," 2025. [Online]. Available: https://arxiv.org/abs/2509.18451

[53] N. Mahdian, M. Jani, A. M. S. Enayati, and H. Najjaran, "Ego-motion aware target prediction module for robust multi-object tracking," 2024. [Online]. Available: https://arxiv.org/abs/2404.03110

[54] J. Huang, W. Zou, J. Zhu, and Z. Zhu, "Optical flow based real-time moving object detection in unconstrained scenes," 2018. [Online]. Available: https://arxiv.org/abs/1807.04890

[55] A. Torralba, P. Isola, and W. Freeman, *Foundations of Computer Vision*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2024. [Online]. Available: https://mitpress.mit.edu/9780262048972/foundations-of-computer-vision/

[56] M. Kuklin, "Optical flow in opencv (c++/python)," 2021, accessed: 2026-05-13. [Online]. Available: https://learnopencv.com/optical-flow-in-opencv/

[57] L. Kong, C. Shen, and J. Yang, "Fastflownet: A lightweight network for fast optical flow estimation," 2021. [Online]. Available: https://arxiv.org/abs/2103.04524

[58] T. Xiao, Y. Lu, X. Di, Z. Ma, H. Qian, Y. Liu, and Y. Zhu, "A uav tracking framework via motion-decoupled trajectory prediction and occlusion handling," *Information Sciences*, vol. 726, p. 122784, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025500920X

[59] J. Luiten, A. Osep, P. Dendorfer, P. H. S. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe, "HOTA: A higher order metric for evaluating multi-object tracking," *CoRR*, vol. abs/2009.07736, 2020. [Online]. Available: https://arxiv.org/abs/2009.07736

[60] P. Dendorfer, A. Osep, A. Milan, K. Schindler, D. Cremers, I. D. Reid, S. Roth, and L. Leal-Taixé, "Motchallenge: A benchmark for single-camera multiple target tracking," *CoRR*, vol. abs/2010.07548, 2020. [Online]. Available: https://arxiv.org/abs/2010.07548

[61] Roboflow, "Train an rf-detr model," https://rfdetr.roboflow.com/learn/train/, 2026, accessed: 2026-05-17.

[62] M. Bilal, "Birds-ir," https://universe.roboflow.com/muhammad-bilal-c4xvp/birds-ir-wk0z8, 2022, licensed under CC BY 4.0.

[63] uav qnoms, "Uav computer vision model," https://universe.roboflow.com/uav-qnoms/uav-wqshy, 2025, licensed under CC BY 4.0.

[64] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 318–327, 2017.

[65] G. F. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, "Large margin deep networks for classification," 2018. [Online]. Available: https://arxiv.org/abs/1803.05598

[66] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *ArXiv*, vol. abs/2004.11362, 2020.

[67] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018. [Online]. Available: https://arxiv.org/abs/1802.03426

[68] F. Svanström, "Drone detection and classification using machine learning and sensor fusion," Master's thesis, Halmstad University, Halmstad, Sweden, 2020. [Online]. Available: https://hh.diva-portal.org/smash/get/diva2:1434532/FULLTEXT02.pdf

[69] N. Jiang, K. Wang, X. Peng, X. Yu, Q. Wang, J. Xing, G. Li, Q. Ye, J. Jiao, and Z. Han, "Anti-uav: A large-scale benchmark for vision-based uav tracking," *IEEE Transactions on Multimedia*, vol. 24, pp. 3552–3563, 2021.

[70] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

[71] KleinYuan, "Bytetrack standalone," https://github.com/KleinYuan/bytetrack-standalone, 2023, accessed: 2026-04-20.

[72] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," pp. 4077–4087, 2017.

[73] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 839–847.

[74] Swedish Government Offices, "Lag (1992:1300) om krigsmateriel," Swedish Code of Statutes (SFS 1992:1300), Ministry of Foreign Affairs, Stockholm, Sweden, 1992. [Online]. Available: https://www.riksdagen.se/sv/dokument-och-lagar/dokument/svensk-forfattningssamling/lag-19921300-om-krigsmateriel_sfs-1992-1300/

[75] P. Ewen, D. Rivkin, M. Bijelic, and F. Heide, "Telescope: Learnable hyperbolic foveation for ultra-long-range object detection," *arXiv preprint arXiv:0000.00000 (LINK TO COME)*, 2026.

[76] Open Robotics, "Building your own robot," https://gazebosim.org/docs/harmonic/building_robot/, 2023, accessed: 2026-04-10.

[77] Open Source Robotics Foundation, "Bounding box camera sensor," https://gazebosim.org/api/sensors/9/boundingbox_camera.html, 2023, gazebo Sensors API documentation, accessed: 2026-04-20.

[78] ArduPilot Development Team, "Ardupilot documentation," https://ardupilot.org/ardupilot/, 2024, accessed: 2026-04-10.

[79] Dronecode Project, "Mavlink developer guide," https://mavlink.io/en/, 2024, accessed: 2026-04-10.

[80] ——, "Qgroundcontrol user guide (v5.0)," https://docs.qgroundcontrol.com/Stable_V5.0/en/qgc-user-guide/, 2024, accessed: 2026-04-10.

[81] Open Robotics, "Use ros 2 to interact with gazebo," https://gazebosim.org/docs/latest/ros2_integration/, 2024, accessed: 2026-04-10.

A

Appendix

A.1 Simulation Setup

Gazebo Harmonic serves as the core component of the simulation environment, as it is responsible for handling the simulation itself. The environment is primarily configured using Simulation Description Format (SDF) files, which are based on an XML format [76]. These files define both the objects within the simulation and the surrounding environment, including aspects such as physics, visual elements, and sensor configurations.

To extend functionality, a number of plugins were used in the simulation setup. For example, sensor-related plugins are employed to configure camera behavior, including enabling the streaming of captured footage. Another plugin used is a bounding box camera, which records the ground-truth bounding box of labeled objects [77]. As described in section 3.5, it uses the configuration which restricts the box to only the visible portions. This is configured by setting the type to 2d.

Another plugin called gz-sim-label-system, is a way for the camera to identify which objects to calculate the bounding box from. The main configuration files in the simulation include the world file, which defines the environment, and the files responsible for specifying the drone model. The drone model used is referred to as iris-with-gimbal, which includes a camera sensor mounted on a gimbal. A variant of this model without the gimbal is also used, representing the objects to be recorded in the simulation.

Finally, Gazebo provides communication through a topic-based system, allowing components to exchange data via subscriptions. In this setup, two topics are of interest. First, the simulation publishes images captured by the camera sensor to a designated topic. Second, the bounding box camera publishes the generated bounding box data to a separate topic.

ArduPilot provides the software that controls how UAVs work [78]. In particular, ArduCopter is used to control the drones, handling aspects such as navigation and mission logic. Communication between Gazebo and ArduPilot is facilitated through a plugin, enabling interaction between the simulation environment and the flight controller. External communication is done through MAVLink, a lightweight messaging protocol for drone communication [79], with QGroundControl being one

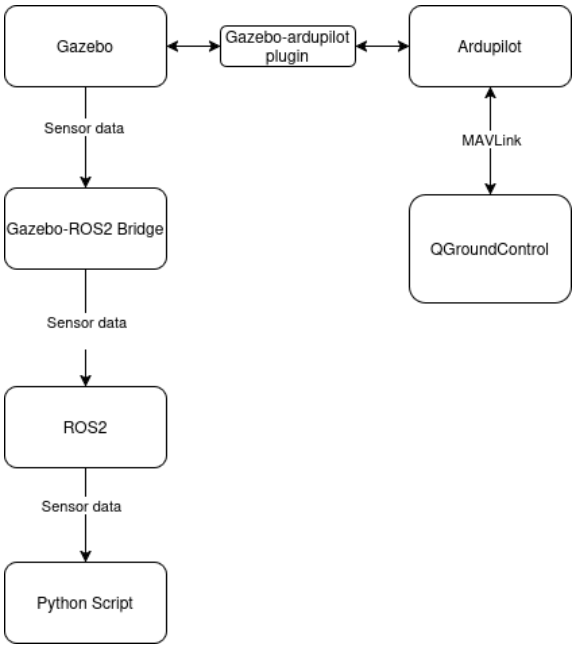

Figure A.1: Shows how information flows in the simulation environment

such external component.

QGroundControl provides functionality for controlling the flying of ArduPilot powered drones through a GUI [80]. To evaluate different conditions and behaviors, multiple missions were designed to represent a variety of scenarios. These included simple movement patterns as well as more complex situations, such as flights involving occlusion or varying altitudes. By creating missions that ranged from close-to-ground navigation to higher-altitude paths, as well as scenarios where the camera itself was in motion, a broader range of scenarios could be tested.

A.2 Acquisition of Sensor Data

To access and store images captured by the camera sensor, as well as the ground-truth bounding box data, a Python script is used. The script utilizes ROS2 libraries to interface with the image topics, acquiring data by subscribing to it. Additionally, a bridge between the Gazebo topic and the ROS2 topic is established, since the script communicates exclusively through ROS2 topics [81]. Once received, the images are processed and saved in a video format, enabling their use in subsequent applications such as testing and evaluation. The raw footage from the camera sensor is in 360x360 resolution with a horizontal FOV of 0.6. Choosing a lower resolution was done to optimize the performance of the vision system. The images are given in RGB format. Simultaneously, bounding box data are acquired at each iteration and stored in a text file.

Since the ground-truth bounding box data is used for evaluation, it is essential

that each bounding box corresponds to the correct video frame. Subscribing to a topic retrieves the available information at a given time, but this does not guarantee that different topics publish data synchronously. To address this, the script employs the ROS2 package message-filters, specifically the ApproximateTimeSynchronizer, which pairs and saves frames and bounding box data only if their timestamps fall within a defined threshold of 0.05 seconds. During testing of the script before the filter was applied, it was evident that the frames had the most trouble being aligned with the bounding boxes during erratic movement, both of the drones themselves and rotation of the camera. Consequently, when applying the filter, some frames are discarded in such scenarios, which may give the impression of an increased drone speed in the resulting video. However, in practice, the footage appears visually consistent and natural.