Design of an Autonomous UAV Tracking System

Scott Tollebäck

Vilhelm Persson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6314
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2026

Abstract

Drones are becoming more common everywhere, and with that comes the risk of unauthorized drones entering protected areas such as airports or military sites. One way to deal with this is to send out another drone to find and follow the unknown one for closer inspection.

This project built a system that does exactly that. A drone equipped with a camera is able to automatically spot another drone, estimate how far away it is, and follow it, all without a human controlling it. The system uses an object detection model to find the drone in the camera image and a tracking method to keep following it between detections.

Tests showed that the system could spot a drone from up to around 80 meters away when the sky was in the background, and around 20 meters when the background was rocky terrain. Once close enough, the system was able to follow the target drone at a distance of around 10 meters in real time.

The results show that this kind of autonomous drone detection and following is possible using standard hardware and commercially available components, though improvements to the camera pipeline and more diverse training data would make the system more robust in the future.

Sammanfattning

Drönare blir allt vanligare och med det ökar risken för obehöriga drönare i skyddsområden som flygplatser eller militära anläggningar. Ett sätt att hantera det här är att skicka ut en annan drönare för att hitta och följa den okända drönaren för närmare granskning.

Det här projektet byggde ett system som gör precis det. En drönare utrustad med kamera kan automatiskt hitta en annan drönare, uppskatta avståndet till den och följa efter den, helt utan att en människa styr. Systemet använder en objektdetekteringsmodell för att hitta drönaren i kamerabilden och en spårningsmetod för att hålla koll på den mellan detekteringar.

Tester visade att systemet kunde upptäcka en drönare på upp till cirka 80 meter när himlen var i bakgrunden och cirka 20 meter när bakgrunden bestod av stenig terräng. När systemet väl kommit nära nog kunde det följa måldrönaren på ett avstånd av cirka 10 meter i realtid.

Resultaten visar att den här typen av autonom drönardetektering och följning är möjlig med vanlig hårdvara och kommersiellt tillgängliga komponenter, även om förbättringar av kamerakoppling och mer varierad träningsdata skulle göra systemet mer robust i framtiden.

Acknowledgements

This project was conducted in collaboration with the Department of Automatic Control at Lund University (LTH) and Saab Surveillance. We are truly grateful for the guidance and support received throughout the project.

At LTH, we would like to thank our supervisor Kristian Soltesz for his invaluable input and experience, which played an integral role in shaping the design decisions made throughout the project. We would also like to thank Bo Bernhardsson for connecting us with a course on a related topic, through which we had the opportunity to present our work as guest lecturers.

At Saab, we are deeply grateful to Axel Müller, who supervised the project and without whom this work would not have been possible. His dedication in securing the necessary equipment, resolving practical challenges, and generously sacrificing his own time for the benefit of the project has been invaluable.

Finally, we would like to thank all those who assisted during field experiments and testing, and everyone who provided feedback, suggestions, or support along the way.

Contents

1. Introduction **13**
- 1.1 Background . . . 13
- 1.2 Problem Description . . . 14
- 1.3 Aim and Objectives . . . 14
- 1.4 Research Questions . . . 15
- 1.5 Scope and Limitations . . . 15

2. Theory **16**
- 2.1 Computer Vision . . . 16
  - 2.1.1 OpenCV . . . 16
- 2.2 Object Detection . . . 16
  - 2.2.1 YOLO: You only look once . . . 17
- 2.3 Tracking . . . 18
  - 2.3.1 Lucas–Kanade Optical Flow . . . 19
  - 2.3.2 OpenCV Implementation of Optical Flow . . . 20
- 2.4 Camera . . . 20
- 2.5 Distance Estimation . . . 21
  - 2.5.1 Width-based estimate . . . 22
  - 2.5.2 Distortion-Corrected Width-Based Estimate . . . 22
  - 2.5.3 Width-and-height-based estimate . . . 23
  - 2.5.4 Angular-width-based estimate . . . 23
- 2.6 Image Centering . . . 24
  - 2.6.1 Control on Pixel Error . . . 24
- 2.7 Feedback Controller . . . 25
  - 2.7.1 P Controller . . . 25
  - 2.7.2 PI Controller . . . 25
- 2.8 Video Streaming . . . 26
  - 2.8.1 Real-Time Streaming Protocol (RTSP) . . . 26
  - 2.8.2 Cosmostreamer . . . 27
  - 2.8.3 Video Link Overview . . . 27
- 2.9 Drone Communication . . . 28

2.9.1 MAVLink . . . 28
2.9.2 ArduPilot . . . 29
2.9.3 QGroundControl . . . 29
2.10 Drone Control via MAVLink . . . 30
2.10.1 GPS Coordinate Prediction . . . 30
2.10.2 Target Interception Logic . . . 30
2.11 Gimbal . . . 31

3. Methods **32**
3.1 System Overview . . . 33
3.2 System loop . . . 33
3.3 Dataset and Preprocessing . . . 33
3.3.1 Data Collection . . . 33
3.3.2 Annotation . . . 35
3.3.3 Data Processing . . . 35
3.4 Model Training . . . 37
3.4.1 YOLO26 . . . 37
3.5 Model Validation . . . 37
3.6 Video Streaming . . . 38
3.6.1 Transport Protocol Selection . . . 38
3.6.2 Buffer Management . . . 39
3.6.3 Fault Tolerance and Reconnection . . . 39
3.6.4 Hardware Latency . . . 39
3.7 Object Detection and Tracking . . . 40
3.7.1 Detection with YOLO . . . 40
3.7.2 Optical Flow Tracking . . . 41
3.7.3 Integration with the Detection Pipeline . . . 41
3.8 Distance Estimation . . . 42
3.8.1 Calculating Camera's Intrinsic . . . 43
3.8.2 Selecting a Distance Estimation Method . . . 44
3.9 Gimbal and Drone Control . . . 45
3.10 Display . . . 46
3.11 Control Channel . . . 46
3.11.1 Safety Through Control Isolation . . . 47
3.11.2 UDP Command Channel . . . 47
3.12 Drone Control . . . 47
3.12.1 Simulation . . . 48
3.12.1.1 Target Drone . . . 48
3.12.1.2 Interceptor Drone . . . 48
3.12.2 Real Situation . . . 48
3.12.2.1 Yaw Control . . . 48
3.12.2.2 Target Localization . . . 49

4. Results **50**

4.1 Image Validation . . . 51
4.1.1 Far Distance Image Validation . . . 51
4.1.2 Close Image Validation . . . 53
4.2 Model Detection Speed . . . 54
4.3 Tracking Validation . . . 56
4.4 Distance Estimation Validation . . . 59
5. Discussion **62**
5.1 Performance . . . 62
5.1.1 Network Performance . . . 62
5.1.2 Detection Performance . . . 63
5.1.3 Tracking Performance . . . 64
5.2 Detection Evaluation . . . 65
5.2.1 Detection Models . . . 65
5.2.1.1 Effect of Model Size . . . 65
5.2.1.2 Effect of Input Resolution . . . 66
5.2.2 Effect of Confidence Levels . . . 67
5.2.3 Detection Range . . . 67
5.2.3.1 Further Validation . . . 68
5.2.4 Limitations of the Model . . . 68
5.3 Estimate Distance Evaluation . . . 69
5.3.1 Difficulties with Distortion . . . 69
5.3.2 Future Distance Evaluation . . . 69
6. Conclusions **70**
6.1 At what distance can a drone be reliably detected and tracked using the developed system? . . . 70
6.2 How can detection models and tracking be combined to achieve reliable real-time drone detection and tracking? . . . 70
6.3 How does model size and input resolution affect detection? . . . 71
6.4 Reliability Through System Design . . . 72
6.5 Final Conclusion . . . 72
7. Future Work **73**
7.1 Dual Control through Gimbal and Drone . . . 73
7.2 Infrared Camera . . . 73
7.3 Distance Estimation . . . 73
7.4 Evaluating different detection algorithms . . . 74
7.5 Compute and control on device . . . 74
Bibliography **75**
A. Appendix **77**
A.1 Drone Components . . . 77
A.2 Video Link Components . . . 77
A.3 Radio Link Components . . . 78
A.4 Software . . . 78

1

Introduction

1.1 Background

There has been a significant increase in the use of unmanned aerial vehicles (UAVs), in both civil and military contexts [Rassler and Veilleux-Lepage, 2025]. This growth has created the need for improved methods for the surveillance and inspection of unknown UAVs. The ability to observe and analyze unknown UAVs at close range is essential to make informed and timely decisions regarding potential threats.

In addition, the decreasing cost of UAVs has led to their widespread commercial availability. As a result, individuals can access platforms with advanced sensing, mobility, and payload capabilities, increasing the potential for misuse in sensitive or protected areas. Such protected areas include critical infrastructure and sensitive sites such as airports, military installations, government facilities, and industrial environments, including nuclear power plants and energy production facilities. In these contexts, unauthorized UAV activity can pose risks related to safety, security, and information exposure, highlighting the need for reliable close-range surveillance, identification, and inspection capabilities.

The practical consequences of unauthorized UAV activity have already been observed in critical infrastructure environments. For example, Copenhagen and Oslo airports were temporarily closed due to reported drone sightings, resulting in flight delays and operational disruptions for passengers [Wilson, 2025].

One possible countermeasure to unauthorized drone activity is the deployment of highly skilled drone operators capable of manually tracking and intercepting intruding UAVs. Although effective, this approach presents significant challenges in terms of availability and cost. Maintaining highly skilled personnel on standby across multiple protected sites is economically impractical, particularly for large or geographically distributed critical infrastructure. In contrast, relying on human operators primarily for analysis and monitoring of sensor data significantly reduces operational costs. Consequently, there is strong motivation for deploying automated or semi-autonomous UAV systems as part of the surveillance and inspection infrastructure at secure installations.

1.2 Problem Description

The problem addressed in this thesis is the design of an autonomous control system capable of protecting secure and sensitive locations from unauthorized UAV activity. The system is intended to operate without human intervention, responding to external alerts by deploying a surveillance drone that navigates to the vicinity of an unknown aerial object and collects visual information for analysis.

Although existing technologies enable automated drone deployment in response to predefined alerts, current commercial systems rely on static missions and human supervision and do not support autonomous close-range detection and tracking of unknown UAVs using onboard perception. As a result, there is a gap between existing automated deployment capabilities and fully autonomous UAV-based observation of dynamic aerial targets, which this work aims to address.

The general problem can be decomposed into three main subproblems. First, in order to track an unknown UAV, the system must be able to detect aerial objects in complex and cluttered environments. This includes reliably distinguishing UAVs from other objects, such as birds or environmental artifacts, to avoid false detections.

Second, external sensing technologies, such as radar or radio-frequency signals, may be unreliable or noisy in stressed or complex environments. For this reason, this thesis focuses on estimating the state of the unknown UAV using visual information. Consequently, robust state estimation and prediction methods based on image data are required.

Finally, the surveillance UAV must plan and execute safe and efficient flight paths. The control system must ensure continuous tracking of the target while maintaining a safe standoff distance and respecting the physical and operational limitations of the UAV, as well as the safety constraints related to the surrounding environment.

The entire control loop of perception, state estimation, and path planning must be performed continuously in real time. This imposes strict requirements on computational efficiency and modular system design in order to achieve robust and reliable performance.

1.3 Aim and Objectives

The aim of this thesis is to design and evaluate an autonomous UAV system capable of detecting and tracking a target drone in real time. The objectives are to ensure correctness and reliability at each stage of the pipeline, from video acquisition and object detection to distance estimation and flight control, as well as to identify system bottlenecks and analyze how design choices affect the overall performance and safety of the system.

1.4 Research Questions

- At what distance can a drone be reliably detected and tracked using the developed system?
- How can detection models and tracking be combined to achieve reliable real-time drone detection and tracking, and how does model size and input resolution affect this performance?
- How do design choices in the detection and tracking pipeline affect the reliability of the overall autonomous system?

1.5 Scope and Limitations

- The drone program will start when the target drone is visible in the picture.
- The detection algorithm will not be used for targets further away than 20 meters.
- The size of the target drone is fixed.
- The project is implemented using commercially available components.

2

Theory

The theory chapter aims to capture the theoretical understanding behind the different techniques and methods used in the project. It covers subjects that may not be universally familiar, ensuring that all readers understand why the different approaches are suited to each task. Everything from equipment to networking to computer vision is addressed in the theory chapter.

2.1 Computer Vision

Computer vision is a field of artificial intelligence concerned with enabling machines to interpret and understand visual information from the world. By processing images and video, computer vision systems can identify objects, estimate distances, track motion, and make decisions based on what they observe, capabilities that are central to the system developed in this thesis.

2.1.1 OpenCV

Open Source Computer Vision Library (OpenCV) is a widely used open source library for real-time computer vision and image processing. It provides functions for processing images, working with video streams, and camera calibration, and is used extensively throughout this project for tasks such as reading camera feeds, running optical flow, and visualizing results. The library supports C, C++, Python, Java and assembly language [OpenCV Foundation, n.d.]

2.2 Object Detection

In order to enable autonomous tracking of an unknown aerial vehicle, a reliable detection mechanism is required. Object detection refers to the process of identifying and localizing objects within sensor measurements.

Several measurement technologies can be used for object detection. Active sensing approaches as radar (Radio Detection and Ranging) and LiDAR (Light

Detection and Ranging) directly measure physical properties of the environment. Radar operates by transmitting radio waves and estimating object distance and velocity from the reflected signal. LiDAR instead emits short laser pulses and computes distance using the time-of-flight principle, enabling reconstruction of a three-dimensional point cloud.

In contrast, passive sensing systems such as cameras rely on ambient light and capture two-dimensional image data. In this work, detection is performed using a vision-based approach. The following sections, therefore, introduce the theoretical foundations of image-based object detection.

2.2.1 YOLO: You only look once

YOLO is a detection and classification model. It takes in an image as input and outputs a list of objects defined in x and y coordinates, their respective height and width and their confidence score and predicted class. In more depth, the images are split into grids of boxes, there are SxS predefined boxes. It produces a set with bounding boxes of corresponding confidence with an object exists inside this bounding box. The model also generates for each grid cell a probability of which class might be in it. Then combining these two maps into one we get bounding boxes in which we believe an object exists, along with which probabilities those grid cells are certain classes, allowing it to detect and classify.

The architecture of the YOLO26 model can be viewed in Figure 2.1. It consists of the parts named backbone, neck and head.

The backbone serves as a feature extractor, generating low, medium and high level feature representations from the input image. These features capture increasingly complex visual information, ranging from basic edges and textures to more abstract object-level patterns.

The extracted features are then passed to the neck, which employs a feature pyramid network to enhance multi-scale feature representation. By combining and aggregating features from different levels, the neck enables the model to better detect objects of varying sizes[Lin et al., 2017]. Finally, the head performs the detection task using three separate detection layers. Each layer operates on a different scale of features, allowing the model to effectively identify small, medium and large objects within the image.

There are different weight sizes in YOLO. The ones that will be used in this study are the nano, small and medium models. As can be viewed in Figure 2.1 they have the same depth multiplier, but different width multipliers which varies the amount of weights that can be trained.

As seen in Figure 2.2, YOLO is capable of detecting multiple objects within the same image. The types of objects detected depend entirely on how the training data was configured, meaning any number of classes can be supported. As shown in Figure 2.3, multiple overlapping detections can be produced for the same object. YOLO resolves this by applying non-maximum suppression (NMS), which retains

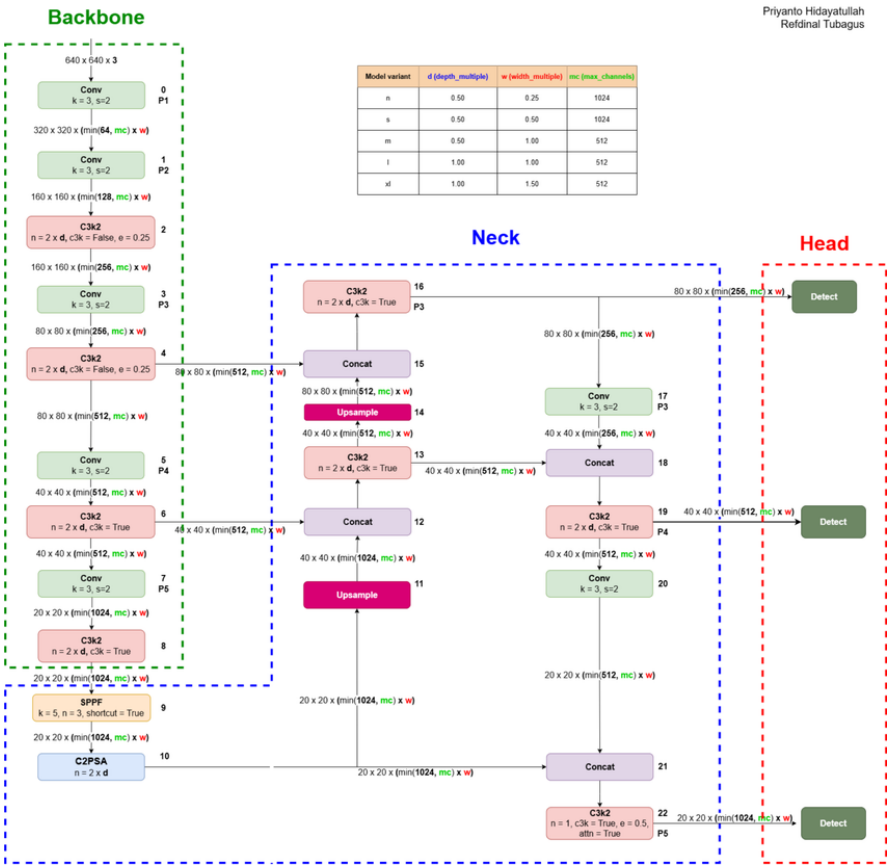

Figure 2.1 Model architecture of YOLO26 [Hidayatullah and Tubagus, 2026]

only the most confident detection for each object by comparing confidence scores and the ratio of the overlapping area between bounding boxes to their total combined area, known as Intersection over Union (IoU).

2.3 Tracking

Tracking an object means following its every move. However, in this report, the focus is less on traditional tracking and more on estimating the position of an object from image context, rather than formally defined tracking. The tracking section therefore covers methods such as Optical Flow and its sub-methods, exploring how image understanding can be used to predict where an object is based on how the background of the image changes.

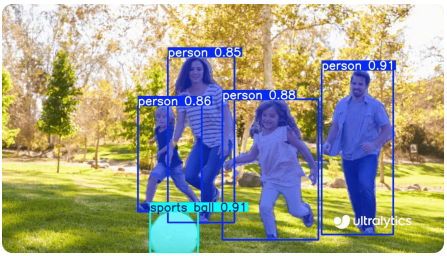

Figure 2.2 Example of YOLO [Vina, 2024]

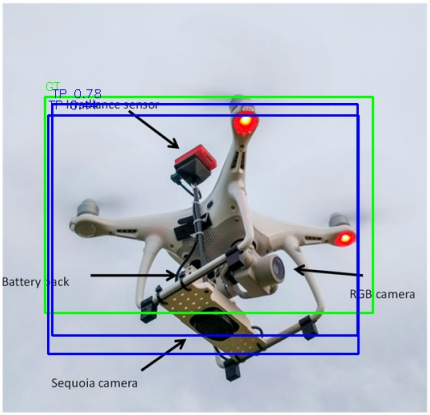

Figure 2.3 Example of YOLO

2.3.1 Lucas–Kanade Optical Flow

Optical flow is used for detecting motion between two consecutive images. Lucas–Kanade optical flow is a technique where one assumes that small patches of an image move consistently between frames. Instead of trying to compute motion for every pixel independently, it samples a close group of pixels and estimates a single motion vector for that region [Lucas and Kanade, 1981].

The method is built on three core assumptions: brightness constancy, meaning that a pixel's intensity does not change between frames, small motion, meaning displacement between frames remains limited, and spatial coherence, meaning neighboring pixels share similar motion. From brightness constancy, the optical flow constraint equation is derived:

$$I_x u + I_y v = -I_t \tag{2.1}$$

where I_x and I_y are the spatial intensity gradients, I_t is the temporal gradient, and

u, v are the unknown horizontal and vertical motion components. Since this yields one equation per pixel but two unknowns, the system is underdetermined. Lucas–Kanade resolves this by aggregating equations over a local pixel window, forming an overdetermined system solved via least squares:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^\top A)^{-1} A^\top b \tag{2.2}$$

where A contains the spatial gradients of all pixels in the window and b contains the corresponding temporal gradients.

2.3.2 OpenCV Implementation of Optical Flow

In practice, Lucas–Kanade optical flow is available directly through the OpenCV API via `calcOpticalFlowPyrLK`, which extends the base method with image pyramids to handle motion at multiple scales. The function tracks a set of 2D points from one frame to the next:

```
cv.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts,
                        status, err, winSize, maxLevel, criteria)
```

The function takes two consecutive frames and a vector of 2D points to track, and outputs their estimated positions in the next frame along with a status vector indicating whether each point was successfully tracked. The `winSize` parameter controls the local pixel window used for the least squares estimation, and `maxLevel` sets the number of pyramid levels, where higher values allow the method to handle larger inter-frame motion. Tracking is considered failed for a given point if its minimum eigenvalue falls below `minEigThreshold`, allowing unreliable points to be automatically discarded.

2.4 Camera

To work with computer vision, one must understand camera intrinsics. What this means is that every camera perceives the world differently depending on how it is built, in terms of depth, field of view, light sensitivity, and so on. What is most relevant to this project are two things: the camera matrix K and the fisheye distortion coefficients D.

The camera matrix describes a pinhole model of how the 3D world projects onto a 2D image plane. It captures the intrinsic geometric properties of the camera, namely the focal lengths f_x and f_y, which determine how strongly the scene is magnified along each axis, and the principal point (c_x, c_y), which defines the pixel coordinate of the optical center of the image. Together these parameters define the

mapping from real-world 3D coordinates to pixel coordinates in the image:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

The fisheye distortion coefficients define a model describing how the image deviates from an ideal fisheye projection, particularly toward the edges. Instead of assuming a perfect linear relationship between the viewing angle and image radius, the model applies a polynomial correction based on the angle θ.

The coefficients k_1, k_2, k_3, k_4 control how strongly points are radially displaced as a function of θ. Lower-order terms (especially k_1) capture the main distortion effect, while higher-order terms refine the behavior for large angles near the image boundaries.

Both parameters K and D are required in order to calculate distance while accounting for lens distortion. They are unique to each camera and cannot be obtained from the manufacturer, meaning they must be determined experimentally for the camera shown in Figure 2.4.

Figure 2.4 DJI O4 Air Unit Pro camera module [DJI, 2026b].

2.5 Distance Estimation

Distance estimation is the process of estimating the distance from a detected object in an image to our own position, without any prior knowledge of the surrounding space. Using only a single camera for this introduces inherent limitations. To estimate the size of a drone we need to know the distance to it, and to estimate the distance we need to know the size. This is a paradox. Therefore the method described here sidesteps this by assuming the size is already known, since resolving that is the ultimate goal. There are different methods taking different things into account as distortion, width, height and so on.

2.5.1 Width-based estimate

The first distance estimation method is the width-based estimate, which uses the horizontal extent of the detected bounding box. Given the pixel width $w = x_2 - x_1$, the known real-world width W, and the calibrated focal length f_x, the distance Z to the target object along the optical axis is derived through the pinhole camera model. This model describes how a 3D point projects onto the image plane, where all rays pass through the camera centre forming similar triangles between the object and its projection. This gives the relation:

$$\frac{w}{f_x} = \frac{W}{Z} \tag{2.4}$$

Rearranging gives the pixel width as a function of distance:

$$w = \frac{f_x \cdot W}{Z} \tag{2.5}$$

Solving for Z yields the estimated distance:

$$Z = \frac{f_x \cdot W}{w} \tag{2.6}$$

Given a known real-world width W and calibrated focal length F_x, the distance is thus computed directly from the bounding box pixel width.

2.5.2 Distortion-Corrected Width-Based Estimate

The second method extends the width-based estimate by accounting for lens distortion. Rather than operating directly in pixel space, the bounding box corners are first undistorted into normalised camera coordinates using OpenCV's `cv2.undistortPoints`, which takes the camera matrix K and distortion coefficients D as inputs. The camera matrix K encodes the intrinsic parameters of the camera, namely the focal lengths f_x and f_y and the principal point (c_x, c_y):

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Together with the distortion coefficients D, which model the radial and tangential distortion introduced by the lens, `undistortPoints` maps each pixel coordinate to a normalized camera coordinate that is free of lens distortion. This is done via the relation:

$$\tilde{x} = \frac{X}{Z} \tag{2.8}$$

where $\tilde{x}$ denotes the normalized x-coordinate, representing the angular position of a point independent of focal length. The horizontal extent of the object in normalized space is then:

$$\Delta\tilde{x} = |\tilde{x}_2 - \tilde{x}_1| \tag{2.9}$$

Since in normalized coordinates the projected width of an object with real-world width W at distance Z satisfies:

$$\Delta\tilde{x} = \frac{W}{Z} \tag{2.10}$$

solving for Z yields:

$$Z = \frac{W}{\Delta\tilde{x}} \tag{2.11}$$

By undistorting prior to estimation, the focal length cancels out and the method is no longer sensitive to pixel-space distortions introduced by the lens, making it more accurate than the raw width-based estimate particularly toward the edges of the image.

2.5.3 Width-and-height-based estimate

The third method extends the width-based estimate by incorporating both the horizontal and vertical extents of the bounding box. Two independent distance estimates are first computed, one from the width and one from the height, using the respective focal lengths f_x and f_y:

$$Z_w = \frac{f_x \cdot W}{x_2 - x_1}, \qquad Z_h = \frac{f_y \cdot H}{y_2 - y_1} \tag{2.12}$$

where W and H are the known real-world width and height of the object respectively. Since the two estimates are derived independently they may differ slightly due to bounding box noise or aspect ratio inconsistencies. To produce a single stable estimate, the geometric mean of the two is taken:

$$Z = \sqrt{Z_w \cdot Z_h} \tag{2.13}$$

The geometric mean is preferred over the arithmetic mean as it is less sensitive to outliers in either estimate, giving a more robust result when one dimension is noisier than the other.

2.5.4 Angular-width-based estimate

The fourth method approaches distance estimation through the angular subtense of the object rather than its projected pixel width. As in the distortion-corrected method, the bounding box corners are first undistorted into normalized camera coordinates using K and D. The angle each edge subtends relative to the optical axis is then computed as:

$$\theta_1 = \arctan(\tilde{x}_1), \qquad \theta_2 = \arctan(\tilde{x}_2) \tag{2.14}$$

giving the total angular width of the object as seen by the camera:

$$\Delta\theta = |\theta_2 - \theta_1| \tag{2.15}$$

For an object of known real-world width W subtending an angle $\Delta\theta$, the distance is recovered through the relation:

$$Z = \frac{W}{2\tan\left(\frac{\Delta\theta}{2}\right)} \tag{2.16}$$

This formulation is geometrically exact under the pinhole model, as it operates directly on angles rather than linearized projections. It is therefore expected to be more accurate than the plain width-based estimate, particularly for objects that are large relative to the field of view or located toward the periphery of the image where linear approximations break down.

2.6 Image Centering

A technique used in object tracking is image centering. It involves, through control and movement, always keeping the target as close to the center of the image as possible, where the center is defined as half the width along the x-axis and half the height along the y-axis. This is done to achieve smoother motion and to minimize the risk of the object leaving the frame and becoming untrackable.

2.6.1 Control on Pixel Error

One approach is to control based on the error between the center of the object's bounding box and the center of the image. This error can be defined along both the x-axis and y-axis separately as following:

$$c_{\text{box}} = \left(\frac{x_1 + x_2}{2}, \quad \frac{y_1 + y_2}{2}\right) \tag{2.17}$$

$$c_{\text{screen}} = \left(\frac{f_{\max,x}}{2}, \quad \frac{f_{\max,y}}{2}\right) \tag{2.18}$$

$$\boxed{\varepsilon_x = c_{\text{box},x} - c_{\text{screen},x}} \tag{2.19}$$

$$\boxed{\varepsilon_y = c_{\text{screen},y} - c_{\text{box},y}} \tag{2.20}$$

Note: ε_x and ε_y have opposing signs due to the misalignment between image coordinate space (origin top-left, y increases downward) and drone control space (positive yaw $\Rightarrow$ rotate right, positive pitch $\Rightarrow$ nose up).

2.7 Feedback Controller

A feedback controller is a feedback control mechanism that computes a control signal based on the error between a desired setpoint and a measured output. The general error is defined as:

$$\varepsilon(t) = r(t) - y(t) \tag{2.21}$$

where $r(t)$ is the reference setpoint and $y(t)$ is the measured process output.

2.7.1 P Controller

The proportional controller produces a control signal directly proportional to the current error:

$$u(t) = K_p \cdot \varepsilon(t) \tag{2.22}$$

where K_p is the proportional gain. A larger K_p produces a stronger response but risks oscillation, while a smaller K_p gives a more stable but slower correction. A pure P controller cannot eliminate steady-state error, since it produces no output when $\varepsilon \to 0$.

2.7.2 PI Controller

The PI controller extends the P controller with an integral term that accumulates error over time, allowing it to eliminate steady-state error that a pure P controller cannot correct:

$$u(t) = K_p \cdot \varepsilon(t) + K_i \int_0^t \varepsilon(\tau)\, d\tau \tag{2.23}$$

In discrete time, the integral is approximated using the Euler forward method:

$$u[k] = K_p \cdot \varepsilon[k] + K_i \cdot T_s \sum_{j=0}^{k} \varepsilon[j] \tag{2.24}$$

where T_s is the sampling period and the accumulated integral $I[k]$ is updated as:

$$I[k] = I[k-1] + \varepsilon[k] \tag{2.25}$$

A known issue with integral controllers is *integrator windup*, which occurs when the system is saturated and the integral term grows unbounded. This is mitigated by clamping both the integrator and the output:

$$I[k] = \text{clip}(I[k-1] + \varepsilon[k],\ I_{\min},\ I_{\max}) \tag{2.26}$$

$$u[k] = \text{clip}(K_p \cdot \varepsilon[k] + K_i \cdot T_s \cdot I[k],\ u_{\min},\ u_{\max}) \tag{2.27}$$

2.8 Video Streaming

To process the drone's camera feed in software, the video stream must be transmitted from the drone to the computer in real time. This section describes how that is achieved in this project.

2.8.1 Real-Time Streaming Protocol (RTSP)

RTSP is an application-layer control protocol, residing in the topmost layer of the OSI model as seen in Figure 2.5, designed for the delivery of real-time media such as audio and video, both live and on-demand. It uses a client-server model where the client sends commands to a media server, which responds and streams media accordingly, keeping track of sessions and states to ensure proper streaming [Schulzrinne et al., 1998]. RTSP shares similarities with HTTP in that it exchanges text-based messages, but is specifically designed for controlling media streams rather than transferring data. It provides a framework to manage multiple streaming sessions and supports selection of the underlying transport method, either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

In this project, RTSP operates over TCP to negotiate and manage the stream, while the actual video payload is carried separately via the Real-Time Transport Protocol (RTP) and decoded from H.264, a video compression standard, by FFmpeg before being passed to OpenCV as a raw image matrix. UDP can also be used in combination with RTSP and may appear faster at first glance; however, it can deliver corrupted or lower resolution images, which risks degrading performance in the detection pipeline.

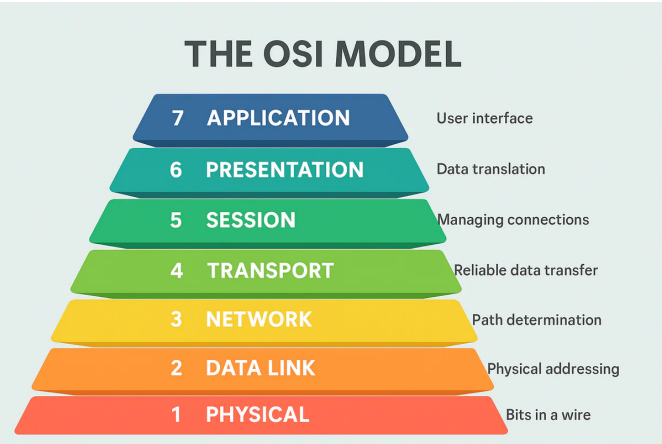

Figure 2.5 OSI Model [Network Supply, 2025]

2.8.2 Cosmostreamer

The Cosmostreamer, as seen in Figure 2.6, is essential for drone-to-computer communication. It connects to the DJI Goggles, as seen in Figure 2.7, via USB-C, receiving a low-latency video stream. The stream is displayed on the Cosmostreamer's local web interface. Through the Cosmostreamer's settings, it is also possible to configure different output connections, such as RTSP and others, allowing a computer to connect via a specified port and receive the video in the desired format.

Figure 2.6 Cosmostreamer GD1 [Cosmostreamer, 2026].

Figure 2.7 DJI Goggles 3 [DJI, 2026a].

2.8.3 Video Link Overview

The video feed follows a multi-step pathway from the drone to the computer, as illustrated in Figure 2.8. While the number of steps may appear unnecessary, it is a

consequence of the hardware available and the limitations of the DJI ecosystem.

The image is first captured by the camera mounted on the drone, which transmits the feed wirelessly to the DJI Goggles 3 using the DJI O4 Air Unit Pro. The goggles receive the signal and forwards it via a USB-C cable to the Cosmostreamer. The Cosmostreamer processes the incoming stream and exposes it over the local network as an RTSP stream, which the computer can connect to and receive as a continuous video feed. The computer then decodes the stream using FFmpeg and passes the resulting image frames to OpenCV for further processing in the detection pipeline.

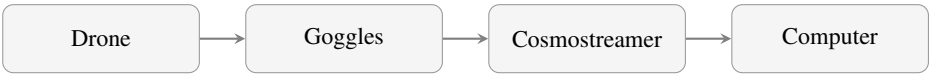

Figure 2.8 Drone video link

2.9 Drone Communication

Communicating with the drone is an entirely different process from receiving the camera feed. This is because the camera and the drone flight controller are independent devices, each requiring their own communication channel. As a result, two separate streams of communication are maintained with the drone simultaneously.

2.9.1 MAVLink

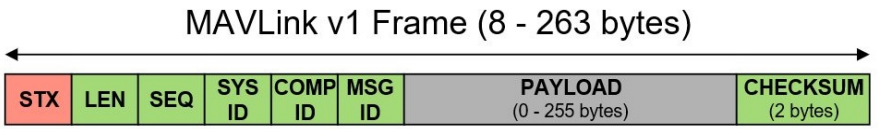

Figure 2.9 MAVLink v1 protocol frame[MAVLink, n.d.]

Micro Air Vehicle Link (MAVLink) is a lightweight communication protocol designed for small UAVs, providing reliable data exchange between the vehicle and ground systems through built-in mechanisms for packet loss detection, message authentication, and data integrity verification.

MAVLink 1, the type of protocol that is used for this project has only eight bytes of overhead per packet, as seen in Figure 2.9. It does not require any additional framing and is thus well suited for tasks with a limited communication bandwidth.

With MAVLink, it is possible to set a certain GPS coordinate and height as a target along with receiving current GPS positions from a drone. Also, it is possible to set a specific heading for a drone to point towards a certain direction.

2.9.2 ArduPilot

ArduPilot is an open source autopilot system for different vehicle types, including UAVs. It is built to be a complete replica of the real world mechanics in UAV flying.

It is possible to simulate UAVs directly on a computer in ArduPilot, using Software in the Loop (SITL) simulator. It simulates a flight dynamics model along with network packets that can be sent and received. This way, it is possible to write programs to control the drone and test it in a safe environment before deploying it in a real environment, reducing costly mistakes.

The drone software communicates with UDP packets. Hence, one can send commands to the drone to set different altitudes and headings as well as receive the GPS position of the drone. One benefit of this is that the program works independently of the drone. It communicates with a simulated drone the same way as with a real drone, making it easier to develop software for drone control and navigation.

Although the SITL Simulator is very capable, it is still a simulator and can not anticipate everything. Real testings also need to be conducted in order to evaluate the drone programs and their performances.

2.9.3 QGroundControl

Figure 2.10 QGroundControl when two drones are connected.

One way of getting a good overview of the drones is with QGroundControl. It is an open-source ground control station software used for configuring, controlling and monitoring UAVs. It communicates with the vehicle's flight controller through telemetry links, enabling real-time data transmission. QGroundControl supports vehicle setup and calibration, mission planning through setting waypoints and real-time flight monitoring. It can display vehicle data such as position, altitude and battery information, as seen in Figure 2.10. Additionally, it supports both manual

and autonomous control modes with safety features as return-to-launch (RTL). It is possible to connect to the drones through UDP, making it easy to get an overview of simulated drones before testing them in the real environment.

2.10 Drone Control via MAVLink

A drone with MAVLink support handles all low-level flight mechanics internally. Therefore, only high-level path planning is required. Since the onboard software manages stabilization, altitude holding, and positional control, the software developed here focuses on defining flight paths and ensuring the drone maintains correct heading throughout each maneuver.

2.10.1 GPS Coordinate Prediction

In order to calculate where to fly regarding the target drone's position, a heading and a distance are needed to estimate the position of the target drone. The heading can be calculated by measuring how far horizontally from the center the target is. By already knowing the camera's field of view (FOV) and the current heading of the drone, it is possible to calculate a new heading.

By combining the heading with a distance estimate (covered in section 2.5), a GPS coordinate can be calculated. This is done by using equations 2.28 – 2.31. The variables lat_{orig} and lon_{orig} are the starting latitude and longitude, lat_{new} and lon_{new} are the resulting latitude and longitude, d is the distance, θ is the bearing in radians and R is the radius of the earth.

$$\Delta x = d \cdot \sin(\theta) \tag{2.28}$$

$$\Delta y = d \cdot \cos(\theta) \tag{2.29}$$

$$\text{lat}_{\text{new}} = \text{lat}_{\text{orig}} + \frac{\Delta y}{R} \cdot \frac{180}{\pi} \tag{2.30}$$

$$\text{lon}_{\text{new}} = \text{lon}_{\text{orig}} + \frac{\Delta x}{R \cdot \cos\left(\text{lat}_{\text{orig}} \cdot \frac{\pi}{180}\right)} \cdot \frac{180}{\pi} \tag{2.31}$$

2.10.2 Target Interception Logic

One way to calculate how to steer when intercepting a target is by calculating how far the target drone can move before it can be reached by the interceptor drone. Equation 2.32 is used to calculate the target's heading and equation 2.33 calculates how far the target will move until the interceptor reaches it. θ is the heading angle, v_x and v_y are latitudinal and longitudinal velocity vectors of the target drone, d is the predicted distance, $d_{current}$ is the current shortest distance between the target and the interceptor drone and v_{max} is the maximum speed of the interceptor drone.

$$\theta = \text{atan2}(v_y, v_x) \tag{2.32}$$

$$d = \sqrt{v_x^2 + v_y^2} \cdot \frac{(d_{current})}{v_{max}} \tag{2.33}$$

2.11 Gimbal

A gimbal is a stabilization device that holds the camera and uses built-in sensors to keep it steady during flight. When a drone tilts, for instance when turning or adjusting its position, the camera would naturally tilt with it, resulting in a skewed or shaky view. The gimbal counteracts these movements in real time, ensuring the camera remains level and the footage stays smooth and correctly oriented for the viewer.

The HEQUAV G-Port 3-Axis Gimbal in Figure 2.11 is mounted on the drone to provide image stability. It can rotate in the pitch and yaw directions via two controllable servos. The yaw range of the gimbal is 270 ° and the pitch range is 180°. Both axes are controlled through Pulse Width Modulation (PWM) servo signals, with commands sent via MAVLink over the same connection as the drone, each control loop iteration, using the standard RC channel convention with values ranging from 1000 to 2000 corresponding to the full range of servo motion, where 1500 represents the neutral position. The input acts as a joystick, defining the rate of rotation rather than an absolute angle.

Figure 2.11 The HEQUAV G-Port 3-axis gimbal used in this system [HEQ UAV TECH, 2026].

3

Methods

This chapter describes the development process of the system, from initial design decisions to a fully integrated pipeline. It covers how each component was approached, what problems were encountered, and why certain solutions were chosen over others. The goal is to provide a clear understanding of not just what was built, but the reasoning behind each design choice, tracing the path from an empty project to a complete autonomous detection, tracking, and control system.

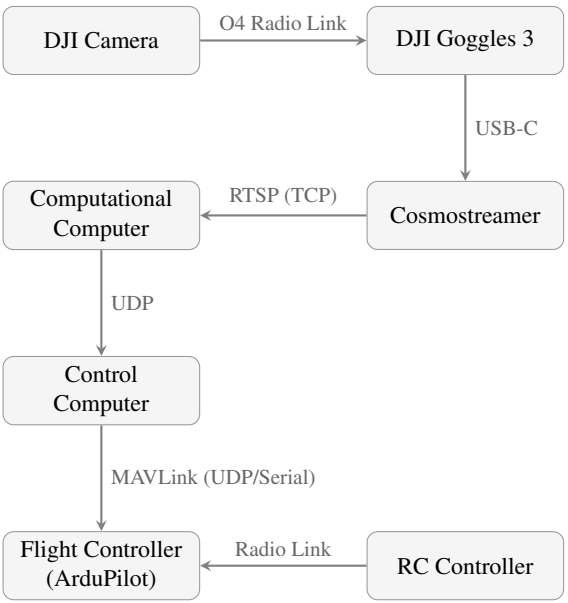

Figure 3.1 Full system including video pipeline and control path via MAVLink and RC link.

3.1 System Overview

Several components need to be connected in order to get the system working. As can be viewed in Figure 3.1, seven different components are connected to get the full system to work. First, the DJI Camera sends the video feed over O4 Radio Link to a pair of DJI Goggles 3. Then, the goggles are connected through USB-C to a Cosmostreamer that transmits the data further through either Wi-Fi or ethernet to the computational computer. Based on the camera feed, the computer calculates the next wanted position and heading for the drone and sends it to the control computer through UDP. Finally, the control computer sends the command further to the drone's flight controller using MAVLink. In case the operator wants to resume control of the drone, it is possible to switch to loiter mode on the controller and the controller input will be used to control the drone through radio link. Also, in case of switching to loiter mode, the control computer will not send any commands to the drone to avoid giving dual input to the drone.

3.2 System loop

Figure 3.2 illustrates the system loop executed on the computational computer. In each iteration, the latest frame is fetched from the RTSP stream and passed to the detection algorithm. The algorithm outputs a bounding box around the detected object. If an object was detected in the previous frame, optical flow is used in place of full detection to track the object more efficiently. From the width of the bounding box, a distance estimate is computed. Using both the estimated distance and the object's position within the frame, a command is issued to the drone to adjust its position relative to the target, maintaining the distance specified by the user.

3.3 Dataset and Preprocessing

Training an object detection model requires a dataset that closely reflects the conditions the model will encounter during deployment. For this system, that means aerial footage of drones captured from varying distances, angles, lighting conditions, and backgrounds. The dataset must also be sufficiently large and diverse to avoid both underfitting, where the model fails to learn meaningful features, and overfitting, where it memorizes the training data but fails to generalize to unseen footage. All images must also be annotated in the format expected by the YOLO training pipeline, with bounding boxes around the target drone.

3.3.1 Data Collection

Plenty of data was needed to train the computer vision model. There already exist complete datasets with pictures of drones along with labels containing information about the bounding box of the drone. These datasets were analyzed and put into this

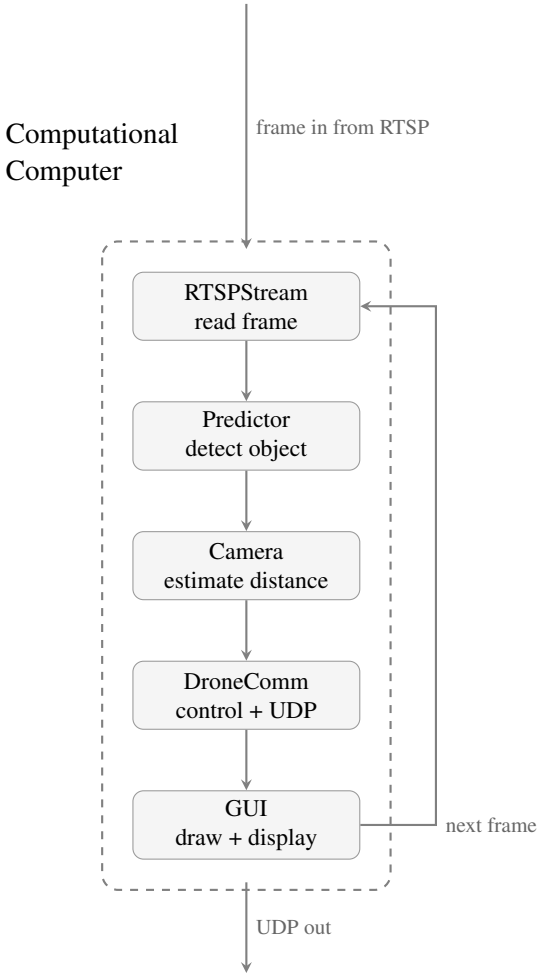

Figure 3.2 Main processing loop inside Computational Computer with input/output arrows (dotted outer box)

project's dataset if they were good enough. Datasets were sourced from open-source repositories, including images of various environments and objects that resemble drones in appearance, such as birds, as seen in Figure 3.3.

After training the first model, it was found that detection performance was poor for drones in certain positions. Closer inspection of the dataset revealed that most images had been captured from a human perspective looking upwards towards the drone. Additional images were therefore needed from ground level and from above the drone. Distance was also an issue, as the majority of the dataset consisted of

Figure 3.3 Bird in the dataset

images taken in close proximity, leaving long-range appearances underrepresented. This was addressed by recording video from one drone filming another across as many angles and distances as possible. The frames were subsequently annotated and added to the dataset. As a result, the retrained model showed improved detection across a wider range of orientations and distances.

3.3.2 Annotation

To complement the existing dataset, additional data collection was necessary as the model struggled to identify drones against the background of ground and trees at longer distances. The solution was to expand the training set with images representing these conditions.

Using the established system pipeline, frames could be extracted from recorded video and converted into MP4 file. However, due to long processing times, this approach was not ideal. A more efficient solution was to insert an SD card directly into the DJI goggles and record footage through them, significantly reducing overhead.

In total, 45 minutes of additional footage were collected. Changing weather conditions during filming proved beneficial, as the ground transitioned from snow-covered to bare over the course of the session. This natural variation added complexity to the dataset and improved the diversity of training samples.

Through the use of OpenCV, a custom annotation tool was developed. The tool plays back each frame from the recorded video, allowing you to step forward and backward through the sequence. By drawing a bounding box over the drone using the mouse, the frame is saved, along with a generated label file in YOLO format. All footage was then manually annotated frame by frame. A frame could look like Figure 3.4

3.3.3 Data Processing

When depending on external datasets, certain difficulties arise. With a set of over 12,000 images, it is not feasible to manually review each one. Two specific annotation errors can occur. The first is an image containing a drone with no annotation

Figure 3.4 Image taken by us, and annotated

attached, meaning the model trains on these as if no drone is present, which introduces confusion. The second is the reverse: an image with no drone but containing a bounding box annotation, meaning the model is taught to detect something that is not there. Both cases degrade model performance.

To avoid reviewing every image manually, a solution was developed using the model itself. To detect the first error type, images containing a drone but missing an annotation, the process starts by filtering to only images labeled as empty, meaning they should contain no drone. The model is then run on this subset with a very low confidence threshold of 0.01, capturing virtually all detections. The resulting images are then reviewed manually, where corrupted images are removed and false positives are skipped. This approach may not catch every corrupted image, but it captures the vast majority.

The reverse issue, labels present with no drone in the image, is solved similarly. The dataset is filtered to only images that have annotations attached. Detection is run on this subset once again, but this time the focus is on false negatives rather than detections. The resulting images are then reviewed manually and corrupted entries are removed.

Another issue was images that did not truly represent the intended definition of a drone within the scope of this project. In some cases, disassembled parts such as a body without wings were labeled as a drone. In others, overlapping text from editing software or similar tools was present on the drone itself. This type of error is harder to detect and clean, as there is no straightforward way to identify it automatically. Once again, the detection model proved useful. By filtering to only true positives and sorting by detection confidence score, starting from the lowest, images that

deviated most from a typical drone appearance could be identified and removed when deemed inappropriate for the project goals.

3.4 Model Training

The training process is time-intensive. The smaller 1280-resolution variant, for example, required approximately 3–4 hours to complete, while larger models, such as the medium variant trained at 1600 resolution, required approximately 24 hours. It is therefore important to plan ahead and schedule training sessions during nights and weekends to maximize utilization. All models were trained on a machine equipped with 32 GB RAM, a 13th Gen Intel Core i5-13500, and an NVIDIA GeForce RTX 4070 Ti SUPER.

3.4.1 YOLO26

The first model trained used an input resolution of 640×640, meaning all images were scaled to fit this resolution during training. Although the model successfully detected drones, the accuracy was insufficient, and it was decided to train a medium-sized model instead. Due to the lower resolution, both models had problems detecting drones further away than five meters. One downside of higher image size was that it takes longer time running the detection model. Despite this, it was decided to scale up the resolution to 1280.

With the new models having image sizes of 1280, it detected drones with the sky as background very well. When looking from above towards the ground, the models had trouble distinguishing the drone. First the image sizes was increased to 1600 but it did not fix this matter. As written in the data collection section, the dataset was instead updated which fixed this issue.

To further investigate the performance of the different model types and image sizes, nine models were trained. These were different combinations between the nano, small and medium types with image sizes of 1280, 1600 and 1920.

3.5 Model Validation

A significant challenge when developing a real-time detection algorithm is validation. A high score on the validation set after model training does not necessarily reflect real-world performance directly. This is especially relevant given that a large portion of the dataset was retrieved online, meaning the training distribution may not capture all real-world conditions. To address this, the model was tested live using a camera against varying backgrounds that may not have appeared in the dataset.

Due to weather conditions and limited access to equipment, outdoor testing could not always be conducted. The project ran from winter through to summer, and during the colder months weather conditions proved challenging. Strong winds

and frost affected the drone, reducing flight time and requiring frequent rest periods. Rain was also a recurring issue, as the project drone is a fragile prototype that could not be flown in even minimal moisture, requiring completely dry conditions. Additionally, a supervisor from Saab was required to be present for all outdoor flights with the project drone, further limiting testing opportunities.

One method used was indoor testing. Real size images of drones were printed on A3 and A4 paper and either hung from the ceiling or taped to a wall. This allowed testing with only the camera against different backgrounds, distances and camera movements without requiring outdoor access. For example, the camera could be held by hand and moved quickly at different angles to observe whether the bounding box in the software followed or lost track of the target. Distance could also be measured precisely indoors using a tape measure, making it possible to evaluate at what range detection became unreliable. This insight was used to supplement the dataset with more images at those distances, as the appearance of a drone differs notably between one meter, five meters and twenty meters away, where it becomes little more than a dot.

3.6 Video Streaming

As seen in Figure 3.5, a video frame must traverse several hardware and software boundaries before reaching the computational computer. The pipeline begins at the DJI camera, is relayed through the DJI Goggles 3 via the proprietary O4 radio link, and is subsequently ingested by the Cosmostreamer device, which exposes the stream over the local network. Since the camera-to-Cosmostreamer segment is managed entirely by proprietary firmware, no modifications to that portion of the pipeline were possible. Our engineering effort therefore focused on reliably consuming the stream from the Cosmostreamer onwards.

3.6.1 Transport Protocol Selection

The Cosmostreamer exposes the video stream via the RTSP at `rtsp://cosmostreamer.local:554/video`. An initial attempt was made to receive the stream over UDP, motivated by the fact that UDP is a connectionless protocol that omits acknowledgment handshakes, thereby minimizing per-packet latency. However, in practice this approach produced frequent frame corruption: because UDP provides no delivery or ordering guarantees, packets lost in transit caused the decoder to reconstruct incomplete frames, resulting in severe visual artifacts. For the purpose of object detection, a corrupted frame is significantly more detrimental than a dropped frame, since running inference on degraded imagery produces unreliable or entirely absent detections without reducing computational load.

TCP was therefore adopted as the transport layer. Unlike UDP, which offers no delivery guarantees, TCP enforces ordered delivery and retransmits any lost packets

before passing data to the application [Eddy, 2022]. This ensures that every frame delivered to the detection pipeline is complete and visually intact.

3.6.2 Buffer Management

A secondary problem emerged once TCP transport was in place: the detection pipeline was unable to process frames at the rate they were produced by the Cosmostreamer. At a nominal stream rate of 60 frames per second, the inter-frame interval is approximately 16.7 ms, whereas the combined latency of pre-processing, inference, and post-processing considerably exceeded this budget. The resulting backlog caused the internal frame queue to grow unboundedly, eventually exhausting available memory and stalling the RTSP connection.

This was addressed by discarding intermediate frames during each read cycle, as outlined in Algorithm 1. Rather than consuming a single frame per iteration, the client reads and discards $n-1$ consecutive frames before retaining the n-th, effectively sub-sampling the stream. For example, retaining every third frame reduces the effective frame rate from 60 to 20 frames per second, providing approximately 50 ms between successive detections. This approach is preferable to reducing stream resolution or introducing explicit sleep, as it keeps the network buffer drained without imposing artificial delays that could accumulate over time. The parameter `max_reads` in the implementation controls this sub-sampling factor and can be tuned to match the throughput of the detection model.

3.6.3 Fault Tolerance and Reconnection

A further reliability concern was identified whereby any transient disconnection upstream, such as a camera power cycle or a temporary loss of the radio link, caused the entire software pipeline to terminate. Since the system is intended for deployment in field conditions where such interruptions are plausible, this behavior was unacceptable. A watchdog mechanism was therefore introduced: the elapsed time since the last successfully decoded frame is tracked continuously, and if this interval exceeds a configurable threshold `max_stale_time`, the client releases the existing `VideoCapture` handle and attempts to re-establish the RTSP connection. This design ensures that the system recovers autonomously from transient faults without requiring operator intervention, and that the detection pipeline resumes as soon as the upstream source becomes available again.

3.6.4 Hardware Latency

A significant source of latency in the system arises from the video streaming pipeline. The captured camera feed is relayed through DJI Goggles and streamed over the local network via Cosmostreamer. Each step in this chain introduces additional delay: the wireless video link, encoding and decoding stages, and network transmission collectively result in an end-to-end latency of approximately 500–1000 ms. This delay means that the processed frame used for detection and tracking re-

flects the drone's state nearly a second in the past, necessitating more conservative and careful control.

Algorithm 1 RTSP Frame Acquisition

```
while running do
    for i = 1 to max_reads do
        read, decode, and discard frame
    end for
    retain final frame → detection pipeline
    if no frame received for max_stale_time seconds then
        reconnect to RTSP stream
    end if
end while
```

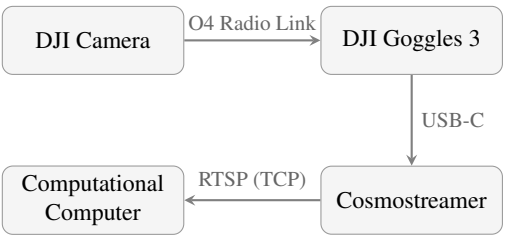

Figure 3.5 System video and communication pipeline.

3.7 Object Detection and Tracking

The detection of the target drone is the central task of the software pipeline, as all downstream behaviour, positional estimation, tracking, and control depend on a reliable position estimate. Running the detector on every frame, however, is not feasible within the system's computational budget, as inference time varies significantly across model variants, as shown in Figure 4.13. The system, therefore, combines periodic YOLO detection with lightweight optical flow tracking to maintain a continuous position estimate within the constraints of real-time operation.

3.7.1 Detection with YOLO

Detection is performed using a trained YOLO model selected based on the capabilities of the hardware on which it is deployed. The model is trained as a single-class detector, where class zero corresponds to the target drone, simplifying the detection output to a binary decision of present or absent. A confidence threshold is applied

to filter weak detections: only predictions exceeding this threshold are considered valid, below which the detector is treated as returning no detection. Among all returned predictions, only the highest confidence detection is retained, as the system tracks a single target at any given time.

3.7.2 Optical Flow Tracking

Optical flow was selected as the tracking method to complement the detection algorithm. The Lucas–Kanade method solves this estimation problem by assuming that neighboring pixels share the same motion, allowing it to construct an overdetermined system of equations across a small patch of pixels and solve for the displacement using least squares. Critically, the method tracks a small set of distinctive points in the image rather than processing every pixel, making it computationally inexpensive compared to running the full detector on each frame. This property made it an attractive candidate for maintaining target position between detections.

Figure 3.6 Optical flow tracking using Lucas–Kanade method [OpenCV, 2025].

3.7.3 Integration with the Detection Pipeline

Detection and consumption of the position estimate run at different rates. The main loop processes frames as fast as possible, meaning that on any given iteration, the position estimate fed to the rest of the system may come from either a fresh YOLO detection or an optical flow propagation, depending on where in the detection cycle that frame falls. The detection interval is adapted based on the outcome of each attempt: when no target is found, the interval is shortened so the detector runs more frequently in search of the target, whereas a successful detection resets it to the nominal rate, reducing unnecessary computation while the tracker maintains the position estimate. Optical flow is therefore a natural fit for filling the intermediate frames: it is computationally cheap, operates purely on pixel data already available, and over a short detection interval, the brightness constancy assumption holds well enough that the propagated bounding box remains a reliable position estimate. Any drift that accumulates is acceptable because the next detection resets the estimate before the error grows significant. Method is summarized Algorithm 2.

A maximum hold time is enforced on the tracked bounding box: if no successful detection has occurred within a fixed time window, the estimate is discarded entirely rather than propagated indefinitely, preventing a stale position from misleading the control pipeline.

Algorithm 2 Detection and Tracking Prediction Loop

```
if time since last detection ≥ detection interval then
    run YOLO detector on current frame
    if detection successful then
        update bounding box estimate
        initialise optical flow tracker
        reset detection interval to nominal rate
    else
        increase detection interval ×3
    end if
else if optical flow tracker is active then
    propagate bounding box via optical flow
end if
if time since last detection > hold time then
    discard bounding box estimate
end if
return current bounding box estimate
```

3.8 Distance Estimation

Since the system relies solely on the onboard camera, distance estimation must be vision-based. One component of the pipeline is therefore dedicated to continuously estimating the distance between the pursuing drone and the target.

The first challenge we encountered was that estimating absolute distance to an object using a single camera is inherently unreliable. Without knowledge of the object's real-world size, there is no unique mapping between its image size and distance, making exact depth impossible to determine from a single frame. This highlighted the need for alternative approaches, such as using known object dimensions, stereo cameras, or motion-based depth estimation, and we chose the most suitable option for our project: relying on known object dimensions.

The solution relied on the physical limits captured by the bounding box. By analyzing footage, we observed that the box consistently enclosed the full width of the drone, including wingspan, allowing us to define its physical boundaries in the image. With the real-world width known, the remaining task was to understand how the camera maps pixels to actual dimensions. For that we need to understand the camera's intrinsic.

3.8.1 Calculating Camera's Intrinsic

To determine the camera's intrinsic parameters and lens distortion, we conducted calibration experiments using known 2D patterns. We employed both a standard chessboard and a charuco board, seen in Figure 3.7, where the positions of the corners or markers are precisely defined in real-world coordinates. Multiple images of each pattern were captured from different angles and positions within the camera's field of view.

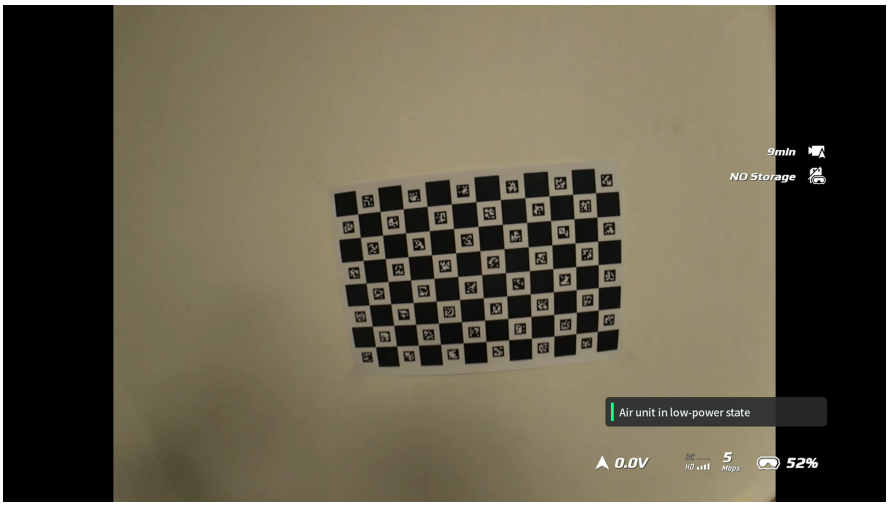

Figure 3.7 Example of a Charuco board used for camera calibration.

All of the calibration and distance estimation steps were performed using OpenCV. It provides built-in functions for detecting patterns like chessboards and Charuco boards, computing intrinsic camera parameters, and estimating lens distortion.

OpenCV functions such as `cv.findChessboardCorners()` and `cv.aruco.detectMarkers()` were used to locate the feature points in each image. These 2D image points were then matched to their corresponding 3D coordinates in real-world space and `cv.fisheye.calibrate()` solved for the camera matrix and distortion coefficients. Re-projection errors were computed using `cv.projectPoints()` to assess calibration accuracy.

Once the intrinsic parameters and distortion coefficients were obtained, images could be undistorted using functions such as `cv.fisheye.initUndistortRectifyMap()` and `cv.remap()`. This allowed us to correct the warped images and better understand the true pixel dimensions of objects in the frame. Figure 3.8 shows the result of applying undistortion to one of the calibration images.

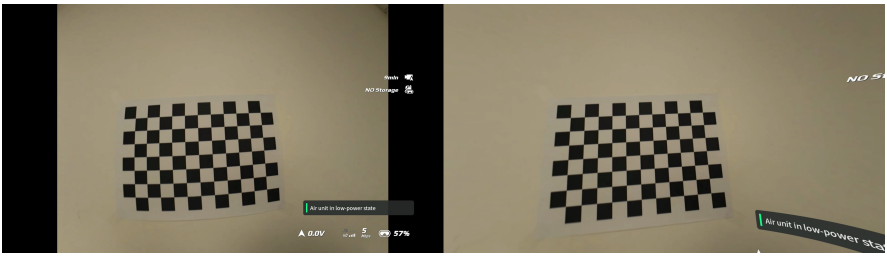

Figure 3.8 Comparison of camera images before and after undistortion. The left image shows the original frame with fisheye distortion, while the right image shows the same frame after applying lens undistortion.

The intrinsic camera matrix K and fisheye distortion coefficients D were derived from a calibration experiment conducted at a resolution of 1920×1080 pixels, where f_x and f_y are the focal lengths expressed in pixels and (c_x, c_y) is the principal point representing the optical center offset from the top-left corner of the image.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 660.01 & 0 & 958.19 \\ 0 & 660.47 & 433.29 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$D = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} 0.16445406 \\ 0.08007942 \\ -0.01450253 \\ -0.0254096 \end{bmatrix} \tag{3.2}$$

3.8.2 Selecting a Distance Estimation Method

The four methods introduced in the theory chapter were evaluated in a controlled setting: width-based estimation, distortion-corrected estimation, width-and-height estimation, and angular-width-based estimation. Each method was integrated into the system loop and tested against known ground truth distances, established by placing a measuring tape and a drone at fixed positions within the camera's field of view.

The evaluation was conducted in two stages. First, the drone was centered in the middle of the frame and each method was compared against the true distance to assess baseline accuracy. Second, the camera was angled so that the drone appeared toward the corners of the image, while the physical distance remained constant. This allowed us to observe how sensitive each method was to the drone's position within the frame, a critical factor in a real system where the drone will rarely be perfectly centered.

None of the methods proved reliable once the camera rotated beyond a moderate angle, which was precisely the range where accurate estimation would have been

most beneficial. Some methods performed better at certain angles but not consistently across all cases, resulting in unpredictable accuracy. This line of investigation was therefore abandoned in favour of focusing on estimation performance when the target is well centered in the frame.

Of the remaining approaches, the simplest method based solely on the width and height of the bounding box was preferred, as it requires less computation and benefits from the properties of the fisheye lens, where distortion is minimal near the center of the image. It was also observed that the height of the bounding box was less reliable than the width, frequently introducing noise that degraded the distance estimate. For these reasons, the final method uses only the bounding box width for distance estimation.

3.9 Gimbal and Drone Control

One approach considered was a combination of controlling the gimbal attached to the drone and the drone itself together. This was seen as a two-step process: first controlling the gimbal based on the pixel error from the camera, then controlling the drone based on the gimbal angle relative to the drone heading.

The gimbal is connected to the drone via two servos controlling horizontal and vertical rotation, with commands sent over MAVLink using PWM signals on channels R10, R11, and R12, controlling yaw, pitch, and position reset respectively. The accepted PWM range is 1000–2000, where 1000 commands maximum speed in one direction, 2000 in the other, and 1500 holds the gimbal stationary.

Initially, the gimbal was tested with a P controller alone, which proved insufficient as the response was either too sluggish or oscillatory depending on the chosen K_p. A PI controller was therefore introduced to eliminate steady-state error. To find suitable coefficients, an iterative tuning procedure was conducted by connecting via MAVLink to an RC controller, mapping joystick positions to K_p and K_i values during live testing. K_p was varied in the range 0.1–0.45 while K_i was adjusted simultaneously on a second joystick, with the integral term reset to zero whenever parameters were changed. Final values were selected based on observed tracking stability. This approach worked well, resulting in the ability to move the drone laterally while keeping the target in frame, with the gimbal recovering quickly as long as the target remained visible.

The primary limitation of this approach was the inability to determine the gimbal's current angle. Although the gimbal could be reset to its default position, its current orientation could not be read back, since the connection operates as a one-way serial bus over which data can only be sent to the gimbal and not received from it. A solution existed in the form of an additional cable with built-in MAVLink support that would have allowed the angle to be retrieved directly from the hardware; however, this was not available during the project and could therefore not be implemented. Had this capability been available, full dual control of both the gimbal and

the drone heading would have been possible, enabling smoother and more precise tracking.

3.10 Display

A visual interface was implemented to provide the user with real-time feedback from the detection system. The interface is built using OpenCV and displays incoming frames from the camera stream. When a drone is detected, bounding boxes, labels, confidence scores, and distance estimates are overlaid directly on the frame.

The interface also allows the user to interactively adjust parameters such as the detection confidence threshold through trackbars. This helps in fine-tuning the system's sensitivity during runtime.

Additionally, the display module supports video recording. When activated, frames are saved in memory and encoded into a video file upon completion of the session. This allows post-analysis or verification of system performance. The GUI provides color-coded annotations to differentiate between tracking and detection states, aiding in intuitive visualization of the system's behavior.

Rendering latency was measured to an average of 17.7 ms per frame using an exponential moving average. Importantly, the display module is not required for the core detection and tracking system to operate, it exists solely to support the human feedback loop during development and testing. Since rendering is decoupled from the control pipeline, it could be offloaded entirely if latency becomes a concern, for instance by streaming annotated frames over UDP to a separate display client, freeing the main processing thread from any rendering overhead.

In general, the display system serves as both a monitoring tool and a debugging aid, providing immediate visual feedback on detection accuracy, object location, and tracking performance.

3.11 Control Channel

Operating an unmanned aerial vehicle in real-world conditions introduces significant safety requirements that must be addressed at the architectural level. A single-computer design, in which the computational pipeline directly issues flight commands, presents an unacceptable risk: any software fault, hardware overload, or loss of the video stream would leave the drone without supervision, and no straightforward means of intervention. To mitigate this, the system adopts a two-computer architecture in which the computational computer and the control computer are assigned strictly separated responsibilities. The computational computer runs the vision pipeline, handling object detection and distance estimation, while the control computer takes that information and translates it into control outputs, communicating the resulting commands to the drone.

3.11.1 Safety Through Control Isolation

The computational computer is responsible solely for perception and decision-making: It ingests the video stream, runs the detection pipeline, and computes the desired control inputs. It does not, however, communicate directly with the drone. Instead, computed control signals are forwarded to a dedicated control computer, which acts as the authoritative interface to the flight controller. This separation provides two critical safety properties. First, the operator seated at the control computer retains the ability to intervene at any moment, overriding or discarding incoming commands and switching the drone to loiter mode or full manual control without any dependency on the state of the computational pipeline. Second, the control computer can implement a watchdog policy: if no command packet arrives within a defined time window, it can interpret the silence as a fault condition and assume safe control of the vehicle autonomously.

3.11.2 UDP Command Channel

Communication between the two computers is established over a dedicated Ethernet link, providing a low-latency, high-reliability physical channel that is isolated from the wireless video path. UDP was selected as the transport layer for command transmission. Although TCP was chosen for the video stream due to its delivery guarantees (see Section 3.6.1), the opposite trade-off applies here: for flight control, it is preferable to transmit the most recent command as quickly as possible rather than to retransmit a stale one. A missed command can simply be followed by the next, whereas a delayed retransmitted command risks issuing an outdated instruction to the flight controller.

Each packet encodes a set of control fields serialized into a compact binary representation using Python's `struct` module. This approach was chosen over text-based formats such as JSON or CSV because binary serialization produces a fixed-size, unambiguous payload that is faster to pack and unpack with no parsing overhead. The receiver unpacks the payload using the same format string, recovering the original values with no risk of encoding ambiguity. The set of fields is not fixed and can be redefined by updating the format string on both ends, making the protocol straightforward to extend as the control interface evolves.

Before deployment, the network reachability between the two computers is manually verified using `ping`, confirming that both machines are addressable on the shared Ethernet segment. This simple pre-flight check ensures that any physical layer fault, such as an unplugged cable or an incorrect IP assignment, is detected before the system is armed, rather than during operation.

3.12 Drone Control

A key part of this project was the ability to test and update how the drone moves, both when simulating and in a real situation. This section will cover how drone

control was implemented.

3.12.1 Simulation

Many simulations were performed to simulate how a target and interceptor drone could behave. ArduPilot was used to simulate the drones' behaviors and Python scripts with MAVLink commands were written to control the drones. In the simulation, the drones always flew on two different altitudes with a difference of two meters in order to not accidentally collide.

3.12.1.1 Target Drone. The script for the target drone was written such that it randomly selected predefined GPS coordinates and drove towards them. This was done to simulate unpredictable behavior.

3.12.1.2 Interceptor Drone. The interceptor drone had a script that fetched the current position and speed direction of the target drone. It then intercepted the target drone by predicting where it would be by the time it could reach its current position. The heading is predicted by calculating the target's heading from its speed vectors given in equation 2.32. The predicted distance can be viewed in equation 2.33. By knowing the target's current position, predicted interception distance and heading, a new coordinate can be created for the interceptor drone. This interceptor code is running in a loop every 0.1 seconds in order to follow the target properly.

3.12.2 Real Situation

In the real situation, the target's current position, speed and direction are unknown. Steering is relied upon object detection and tracking along with distance estimation, covered in sections 3.7 and 3.8.

3.12.2.1 Yaw Control. As described in Section 3.9, the gimbal cannot report its current angle due to incomplete cable connections. This means the system cannot compensate for gimbal offset when computing the drone's target heading. To avoid this ambiguity, the gimbal is kept fixed at its neutral position and the drone yaw is used exclusively for centering the target in frame.

The angular offset between the image center and the bounding box center is computed from the pixel error and the camera field of view:

$$\varepsilon_x = c_{\text{box},x} - \frac{f_{\max,x}}{2} \tag{3.3}$$

$$\theta_x = \frac{\varepsilon_x}{f_{\max,x}} \cdot \theta_{\text{FOV},x} \tag{3.4}$$

This angle is used to compute a new GPS target coordinate at distance d from the drone using equations 2.28 – 2.31. After that, the coordinate is sent to the drone via MAVLink, commanding it to rotate toward the target. Overshoot is less of a concern since the internal flight controller handles low-level stabilization.

The original intended design was a two-stage cascaded control strategy. The gimbal PI controller would handle fast, reactive centering of the target in frame, while a second PI controller would operate on the angle error between the gimbal heading and the drone heading. As the gimbal deviates from its neutral position, the drone would slowly rotate to realign itself, keeping the gimbal near center and available for further corrections. This approach would combine the quick response of the gimbal with the smoother, sustained rotation of the drone, reducing the risk of the gimbal reaching its mechanical limits during tracking. However, this strategy requires knowing the current gimbal angle at all times, which was not possible given the incomplete cable connection. It remains a direction for future work.

3.12.2.2 Target Localization. The target is localized by the detection model in the computational computer. The horizontal offset is measured from the middle of the screen to the center of the target's bounding box. With the knowledge of the camera having an FOV of 155°, a new course to the control computer can be sent. Also, the distance is calculated in the computational computer using distance estimation covered in Section 2.5. By having the interceptor drone's current heading, a new course and an estimated distance, the control computer can estimate the target's position using the Haversine Formula. This is covered in Section 2.10.1 and in equations 2.28 – 2.31.

4

Results

To validate the results of the methods and demonstrate the actual performance that the system can achieve, concrete evidence is required. Therefore, experiments were conducted on selected components of the system that were deemed to be the most relevant. The components were largely tested in isolation, where tracking was evaluated independently, object identification was evaluated independently, etc. All tests were conducted exclusively using real-world data and materials, ensuring that the results reflect real-world performance rather than controlled laboratory conditions. Figure 4.1 is an example of real-world data.

Figure 4.1 Drone detected by the software.

4.1 Image Validation

The detection model evaluation is split into two separate parts. The first focuses on long-range detection in difficult terrain, representing the initial phase of locating the target drone. The second covers close-up images, resembling the conditions encountered during active following once the desired distance has been reached. This separation is motivated by the significant difference in the apparent size of the drone between the two scenarios, as illustrated by comparing Figure 4.2 and Figure 4.12.

4.1.1 Far Distance Image Validation

To validate the model's ability to detect drones across varying distances and visual conditions, a dedicated validation dataset was captured. The collection process involved flying directly towards a drone from a distance at which it was initially not visible, continuing until it became clearly visible to the operator. Figure 4.2 shows the difficulty of identifying a drone at long range. This was repeated across multiple approach angles and background conditions, including approaches against open sky, tree lines, and ground-level backgrounds, as well as recordings taken from elevated positions above the target drone.

In addition, vertical flight sequences were recorded by launching one drone above the other and flying it to an altitude of up to 50 meters, or until the target drone was no longer detectable. Both ascending and descending passes were captured to account for varying perspectives and distances during vertical separation.

Figure 4.2 Example of drone in far-away dataset

The entire capture produced 15 minutes of video footage, which was subsequently annotated to form the validation set. This included both frames in which the drone was visible and labeled accordingly, as well as frames in which no drone was present, which were retained as negative samples. The resulting dataset provides a balanced basis for evaluating model performance across both detection and non-detection scenarios.

By calculating the IoU at the time of detection, it is possible to determine whether a detection is a true positive. A threshold of 0.3 was chosen, which is deliberately low given the nature of drone detection. False positives in this context tend to appear far from the actual drone location, meaning that any detection with sufficient spatial overlap with the ground truth is very likely a genuine detection rather than a spurious one.

The experiment was also evaluated across varying detection confidence thresholds. This allows for an understanding of how precision and recall are influenced by the minimum confidence required for a detection to be accepted, ultimately aiding in the selection of optimal model parameters. The chosen values were 0.2, 0.3 and 0.4, more is too limiting and less is too noisy.

Each trained model was evaluated against this validation set as seen in Table 4.1.

Table 4.1 Detection performance by model size and input resolution (IoU ≥ 0.3) on the long-range validation set

		Confidence 0.2				Confidence 0.3				Confidence 0.4			
Model	**Res.**	**Prec**	**Rec**	**F1**	**Acc**	**Prec**	**Rec**	**F1**	**Acc**	**Prec**	**Rec**	**F1**	**Acc**
Medium	1280	0.682	0.332	0.446	0.438	0.685	0.290	0.407	0.419	0.665	0.244	0.357	0.391
	1600	0.788	0.513	0.622	0.562	0.806	0.456	0.582	0.538	0.816	0.407	0.544	0.514
Small	1280	0.793	0.358	0.494	0.476	0.795	0.277	0.410	0.429	0.803	0.217	0.342	0.396
	1600	0.750	0.463	0.572	0.521	0.759	0.400	0.524	0.491	0.761	0.339	0.469	0.460
	1920	0.721	0.526	0.608	0.546	0.719	0.464	0.564	0.516	0.715	0.407	0.519	0.489
Nano	1280	0.657	0.264	0.377	0.389	0.679	0.226	0.339	0.376	0.696	0.190	0.299	0.363
	1600	0.763	0.408	0.532	0.498	0.771	0.317	0.449	0.449	0.770	0.234	0.359	0.401
	1920	0.774	0.475	0.589	0.542	0.793	0.414	0.544	0.513	0.801	0.344	0.481	0.474

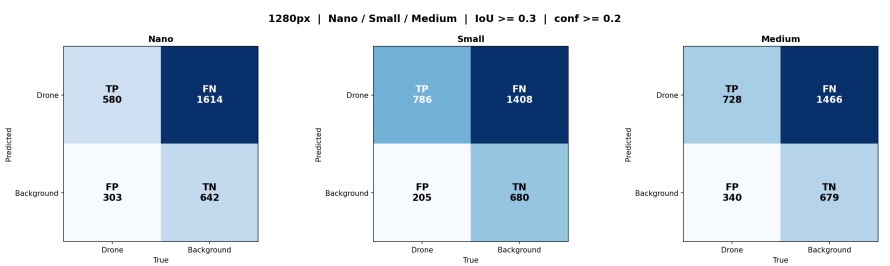

Figure 4.3 Confusion matrix with confidence threshold 0.2 and IoU > 0.3 for 1280 px input resolution

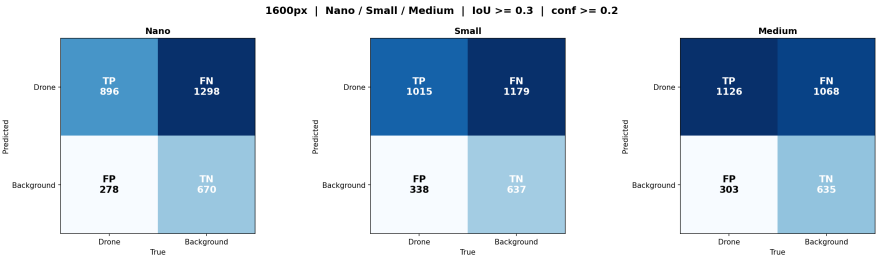

Figure 4.4 Confusion matrix with confidence threshold 0.2 and IoU > 0.3 for 1600 px input resolution

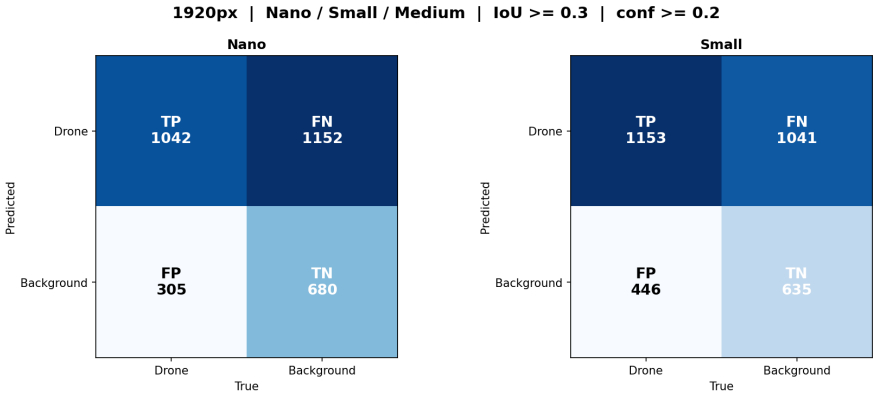

Figure 4.5 Confusion matrix with confidence threshold 0.2 and IoU > 0.3 for 1920 px input resolution

4.1.2 Close Image Validation

In addition to the difficult long-range tests at detection distance with challenging backgrounds, a separate evaluation was conducted on close-up images with clearer backgrounds, as seen in Figure 4.12. The reason for this separation is that the two datasets represent fundamentally different use cases. The previous section focuses on detecting a drone at range, in difficult conditions such as against trees and rocks, while this evaluation addresses how well the models perform when the drone is already close, typically at distances of 3–10 meters in most images. For comparison, the detection model was set to the same three confidence levels as in the previous experiment. Result can be viewed in Table 4.2.

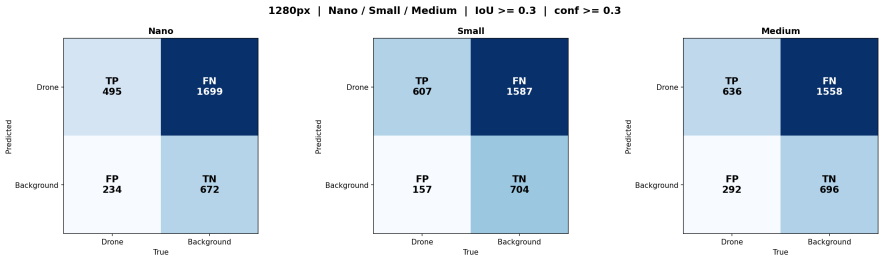

Figure 4.6 Confusion matrix with confidence threshold 0.3 and IoU > 0.3 for 1280 px input resolution

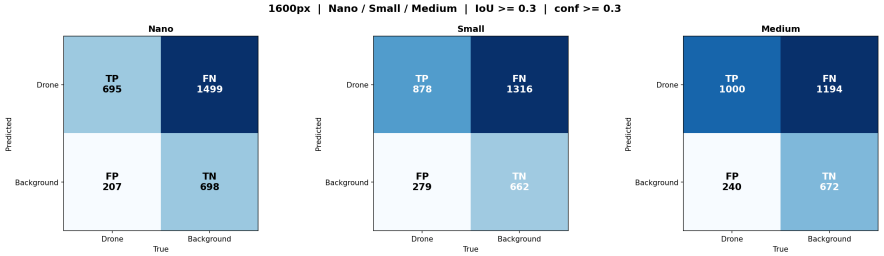

Figure 4.7 Confusion matrix with confidence threshold 0.3 and IoU > 0.3 for 1600 px input resolution

Table 4.2 Detection performance by model size and input resolution (IoU $\geq$ 0.3) on the close-range validation set

		Confidence 0.2				Confidence 0.3				Confidence 0.4			
Model	**Res.**	**Prec**	**Rec**	**F1**	**Acc**	**Prec**	**Rec**	**F1**	**Acc**	**Prec**	**Rec**	**F1**	**Acc**
Medium	1280	0.749	0.807	0.777	0.814	0.746	0.771	0.758	0.808	0.778	0.729	0.753	0.788
	1600	0.780	0.854	0.816	0.876	0.780	0.854	0.816	0.872	0.788	0.854	0.820	0.869
Small	1280	0.859	0.760	0.807	0.857	0.844	0.792	0.817	0.855	0.921	0.729	0.814	0.838
	1600	0.885	0.812	0.847	0.867	0.885	0.812	0.847	0.875	0.885	0.812	0.847	0.866
	1920	0.905	0.875	0.890	0.909	0.905	0.875	0.890	0.911	0.913	0.875	0.894	0.905
Nano	1280	0.800	0.667	0.727	0.746	0.828	0.500	0.623	0.681	0.944	0.354	0.515	0.644
	1600	0.848	0.814	0.831	0.863	0.848	0.812	0.830	0.865	0.900	0.750	0.818	0.843
	1920	0.855	0.833	0.844	0.878	0.851	0.833	0.842	0.882	0.884	0.792	0.835	0.866

4.2 Model Detection Speed

To understand how different hardware configurations interact with the model inference speed, an experiment was conducted in which each model was evaluated on the same validation set across multiple computers, with mean inference time recorded for each configuration.

The experiment was run on four different hardware setups to provide a broad comparison. First, a standard CPU on an Apple M3 was used as a baseline to il-

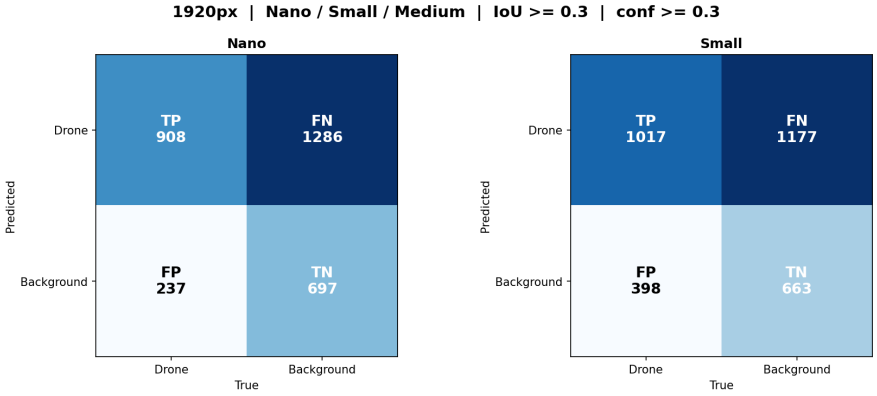

Figure 4.8 Confusion matrix with confidence threshold 0.3 and IoU > 0.3 for 1920 px input resolution

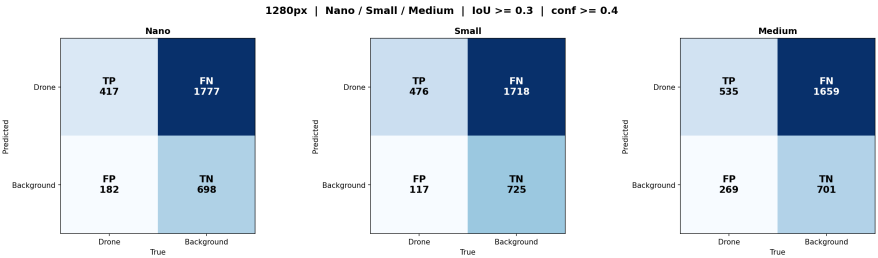

Figure 4.9 Confusion matrix with confidence threshold 0.4 and IoU > 0.3 for 1280 px input resolution

lustrate the performance gap between CPU and GPU inference. The same machine was then tested using Metal shaders, which leverage the M3's integrated GPU. To compare against dedicated NVIDIA graphics cards, two available options were selected, a GTX 1650 and an RTX 5070, representing a range of GPU performance levels from consumer-grade to higher-end hardware.

The results are presented in Figure 4.13 and Table 4.3, with exact inference times recorded for each device, model size, and resolution combination. Two additional tables supplement these results: Table 4.4 expresses each device's inference time as a speedup ratio relative to the GTX 1650, providing a direct comparison of hardware performance, while Table 4.5 shows how inference time scales with model size and resolution within each device, using the Nano model at 1280 px as the baseline.

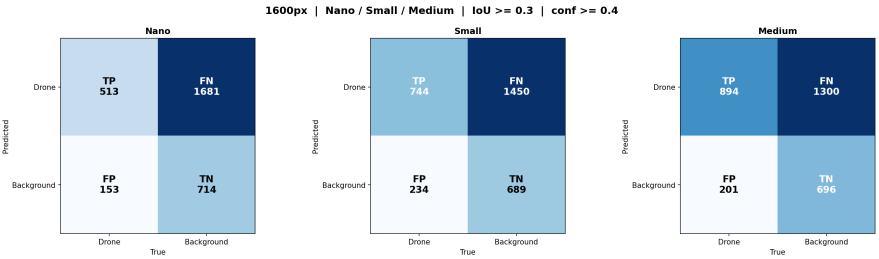

Figure 4.10 Confusion matrix with confidence threshold 0.4 and IoU > 0.3 for 1600 px input resolution

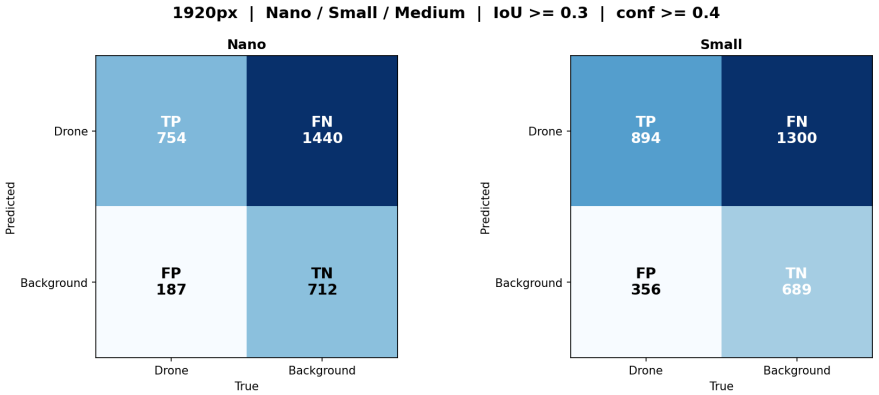

Figure 4.11 Confusion matrix with confidence threshold 0.4 and IoU > 0.3 for 1920 px input resolution

4.3 Tracking Validation

Due to computational limitations, a tracking functionality was introduced as a means to bridge the gap between detection frames, since detection is significantly more time-consuming than tracking. For the tracker to be considered valid, it must be evaluated against a reliable reference.

To evaluate the tracker, the key question is: how well can it follow what the detector would otherwise identify as the drone? Specifically, how long can the tracker coexist with the detector before drifting away from its output?

To answer this, an experiment was conducted using videos selected from the training set, ensuring that the detector would produce consistent and frequent detections. Upon first detection, the tracker was initialized to that state. From that point forward, the tracker was updated every frame while the detector ran simultaneously. By measuring the intersection of union between the tracker output and the

Figure 4.12 Image in close-up dataset

Table 4.3 Inference time (ms) by device, model size, and image resolution

Device	Model	1280 px	1600 px	1920 px
GTX 1650	Nano	14.6	23.1	30.1
	Small	35.6	56.5	75.3
	Medium	77.5	121.8	—
RTX 5070	Nano	7.0	8.8	9.2
	Small	8.8	9.0	12.6
	Medium	12.3	18.9	—
M3 Mac (MPS)	Nano	73.27	69.45	145.55
	Small	95.49	184.41	163.74
	Medium	215.41	266.59	—
M3 Mac (CPU)	Nano	119.90	190.70	370.39
	Small	238.45	534.04	650.03
	Medium	752.61	978.77	—

detector output over time, it was possible to determine how long the tracker remains in a valid state before drifting beyond an acceptable threshold.

Table 4.6 defines the three configuration profiles evaluated, where each successive profile increases the number of tracked feature points, the search window size, and the number of pyramid levels, while lowering the minimum point threshold required to maintain a track. Table 4.7 presents the evaluation results across all three

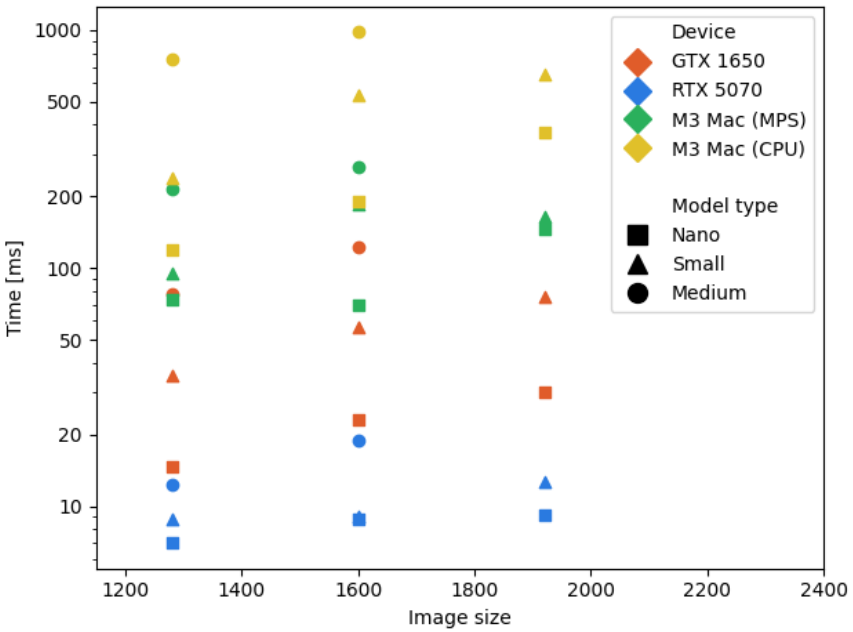

Figure 4.13 Performance of different model types on different computers

profiles, reporting the number of frames tracked, average IoU against the detector output, and streak statistics at two IoU thresholds ≥ 0.3 and ≥ 0.6 where a streak represents a consecutive sequence of frames in which the tracker remained within the acceptable overlap threshold.

Table 4.4 Inference time speedup ratio relative to GTX 1650

Device	Model	1280 px	1600 px	1920 px
RTX 5070	Nano	2.1×	2.6×	3.3×
	Small	4.0×	6.3×	6.0×
	Medium	6.3×	6.8×	—
M3 Mac (MPS)	Nano	0.20×	0.33×	0.21×
	Small	0.37×	0.31×	0.46×
	Medium	0.36×	0.58×	—
M3 Mac (CPU)	Nano	0.12×	0.12×	0.08×
	Small	0.15×	0.11×	0.12×
	Medium	0.10×	0.19×	—

Table 4.5 Inference time ratio relative to Nano at 1280 px (same device)

Device	Model	1280 px	1600 px	1920 px
GTX 1650	Nano	1.00×	1.58×	2.06×
	Small	2.44×	3.87×	5.16×
	Medium	5.31×	8.53×	—
RTX 5070	Nano	1.00×	1.26×	1.31×
	Small	1.26×	1.29×	1.80×
	Medium	1.76×	2.63×	—
M3 Mac (MPS)	Nano	1.00×	0.95×	1.99×
	Small	1.30×	2.52×	2.23×
	Medium	2.94×	2.91×	—
M3 Mac (CPU)	Nano	1.00×	1.59×	3.09×
	Small	1.99×	4.45×	5.42×
	Medium	6.28×	5.61×	—

A running average of processing times recorded during the experiment further showed that each tracking operation completed in approximately 3 ms on the CPU.

Table 4.6 Lucas–Kanade Tracker Configuration Profiles

Parameter	Original	Aggressive	Very Aggressive
Max feature points	50	150	250
Window size	31×31	41×41	51×51
Pyramid levels	3	4	5
Min points	10	6	3

4.4 Distance Estimation Validation

To validate the detection range of the system, a controlled field experiment was conducted using two drones, each equipped with a Qstarz BL-1000ST GPS logger. The device uses a GPS/GLONASS/QZSS multi-constellation engine and records at 1 Hz. The manufacturer specifies a positional accuracy of 3 m CEP under autonomous operation, or 2.5 m CEP with Satellite Based Augmentation System (SBAS) correction. CEP denotes Circular Error Probable, meaning that 50% of recorded positions fall within the stated radius of the true position. Under open sky conditions, as in this experiment, SBAS correction is typically available, placing the expected accuracy at approximately 2.5 m. With two devices in use simultaneously, the median combined positional uncertainty is on the order of 5 m, though individual measure-

Table 4.7 Lucas-Kanade Tracker Evaluation across Configuration Profiles

Metric	Original	Aggressive	Very Aggressive
Video statistics (shared across all configurations)			
Total frames		4869	
Frames with detection		4339	
Tracker performance			
Frames tracked	3497	3719	3800
Avg IoU	0.781	0.778	0.775
Streak analysis (IoU $\geq$ 0.3)			
Frames above threshold	3472	3625	3698
Max streak (frames)	298	293	282
Max streak (seconds)	4.97	4.88	4.70
Avg streak (frames)	33.7	29.7	24.8
Avg streak (seconds)	0.56	0.50	0.41
Number of streaks	103	122	149
Streak analysis (IoU $\geq$ 0.6)			
Frames above threshold	3330	3508	3543
Max streak (frames)	226	226	226
Max streak (seconds)	3.77	3.77	3.77
Avg streak (frames)	17.1	14.4	12.1
Avg streak (seconds)	0.29	0.24	0.20
Number of streaks	195	243	294

ments may exceed this as CEP is a 50th percentile figure with no guaranteed upper bound[Qstarz, 2026].

The stationary drone was placed at a known landmark, a water well, providing a fixed and easily verifiable reference position. The second drone flew away from the stationary one and then turned to approach it slowly, simulating a real detection scenario. One person monitored the detection software, calling out when the first detection occurred, while a second person simultaneously noted the time. The recorded time was then used to look up the GPS position of the approaching drone at that time, from which the distance between the two drones could be calculated.

One identified source of latency was a delay of approximately one second between the camera feed and the software display. This is accounted for within the timing uncertainty ± 2 s applied to all measurements.

The experiment was repeated three times per background condition to provide a more robust estimate of the detection range. Two background conditions were tested: open sky, representing a high-contrast, low-clutter scenario, and a gravel pit terrain, representing a more challenging and cluttered background. The uncertainty range was calculated using the GPS logs, which provide speed at each measurement point. Since the drone was not traveling at constant speed, the range is asymmetric for some measurements, extending further on one side than the other. The final

results of the experiment are presented in Table 4.8.

Table 4.8 Detection distances for the approaching drone under two background conditions. Uncertainty reflects a maximum timing error of ± 2 s at the recorded detection moment, translated to distance based on the drone's speed at each detection.

Run	Distance (m)	Range (m)
Against sky background		
1	$87.9^{+8.2}_{-4.0}$	83.9 – 96.1
2	$68.4^{+10.1}_{-6.1}$	62.3 – 78.5
3	$78.9^{+7.6}_{-6.8}$	72.1 – 86.5
Average	$\mathbf{78.4^{+8.6}_{-5.6}}$	**72.8 – 87.0**
Against ground background		
1	24.5 ± 20.0	4.5 – 44.5
2	21.9 ± 6.6	15.3 – 28.5
3	16.8 ± 7.8	9.0 – 24.6
Average	$\mathbf{21.1 \pm 11.5}$	**9.6 – 32.6**

5

Discussion

In this chapter, the results from the previous section are discussed and analyzed. The focus is on identifying the causes behind the results, evaluating what could be improved, and reflecting on what could have been done differently to achieve better outcomes.

5.1 Performance

One of the most important metrics is latency, the time it takes to process a single frame through the entire pipeline, from receiving it over the network, passing it through the system to the detection stage, and finally sending a control signal back to the drone. A significant part of the project has been dedicated to identifying and addressing the bottlenecks in this pipeline. By reducing latency, the system is able to send control signals to the drone at a higher rate, resulting in a considerably more responsive system.

5.1.1 Network Performance

Firstly, there is an inherent delay at the camera, spanning from the moment an image is captured until it arrives at the computer. This was an unavoidable limitation of the system. The image is captured by the drone, transmitted over a radio link to the paired goggles, which are then connected to a Cosmostreamer device that creates a local network to which the computer connects. This is a considerably long pathway for the image to travel. A likely solution in the future would be a system-on-chip approach, where a camera is mounted on the drone but connected directly to an onboard computer instead. This delay did become a notable problem during test flights. Since the delay was approximately 500 to 1000 milliseconds, as seen in Section 3.6.4, aggressive control inputs risked acting on outdated image data, causing the drone to overshoot in a given direction. If the drone had already begun correcting in the opposite direction, the delayed feedback would compound the error further. Resolving this delay would therefore significantly stabilize the overall system.

The next step is unpacking the incoming frames. Since the stream runs at 60 FPS, there is only 16.67 ms available to process each frame. As seen in Figure 4.13, the GTX 1650 which was primarily used in the field simply could not meet this requirement. The RTX 5070, while capable, was housed in a stationary workstation and could not be brought into the field. The consequence of failing to process frames in time is a buffer overflow, where an ever-growing queue of frames accumulates until the entire program collapses.

One potential solution would be to switch to a different network protocol such as UDP. The video stream was transmitted using RTSP, which by default operates over UDP. During testing this resulted in frequent packet loss, manifesting as corrupted, blocky, or blurry frames upon arrival at the computer. This is an inherent characteristic of UDP, as it offers no guarantee of packet delivery or retransmission of lost packets. As seen in Figure 4.11, even at the full input resolution of 1920 pixels it is already difficult to detect drones at greater distances under ideal conditions. Feeding corrupted or degraded frames into the detection pipeline would therefore make reliable detection even less feasible. Using RTSP over TCP ensures that every packet is delivered correctly and in order, preserving full image integrity, making it the only appropriate choice for this system.

The buffer overflow issue was mitigated through active frame management, which proved essential for maintaining pipeline stability during extended flight sessions. By ensuring the system consistently operates on the most recent frame rather than queued stale data. The effective latency between the physical scene and the resulting control signal was kept within a range that still permitted the system to follow and track the target, even if not in strict real-time. Without this measure, the compounding queue would have rendered real-time control infeasible on the available hardware.

5.1.2 Detection Performance

The next bottleneck is the detection stage itself, specifically how long it takes for a frame to pass through the YOLO network. Both model size and input resolution play a significant role here. Processing a single frame involves a substantial number of calculations, and since much of this computation can be parallelized, more capable hardware is able to resolve it considerably faster.

As shown in Figure 4.13 and Table 4.4, the choice of hardware has a profound effect on inference time. The M3 MacBook running on CPU achieves as little as 0.10 times the throughput of the GTX 1650, confirming that CPU-only inference is not viable for any real-time application. Even when leveraging the M3's integrated GPU through Apple's Metal Performance Shaders (MPS), throughput only reaches 0.20 to 0.58 times that of the GTX 1650, meaning it remains consistently two to five times slower despite being a modern chip.

The root cause of this gap lies in the underlying compute architecture. NVIDIA GPUs use CUDA, a framework that has been purpose-built and continuously re-

fined for deep learning workloads, supported by highly optimized libraries such as cuDNN and TensorRT. MPS, while functional, is comparatively immature in this regard and lacks equivalent kernel-level optimizations for neural network inference. Since Ultralytics YOLO is built on PyTorch, inference is dispatched through whichever backend the platform exposes, meaning the maturity of that software layer directly determines performance.

Beyond software, the hardware itself differs fundamentally. The GTX 1650 features 896 dedicated CUDA cores and 4 GB of dedicated graphics memory, while the RTX 5070 offers 6144 CUDA cores and 12 GB. The M3 chip, by contrast, has only 8 graphics cores and shares its 16 GB of RAM between the CPU and GPU. Regardless of the M3 being a newer generation chip, the lack of dedicated memory and the limited core count mean it cannot match the parallelism that a discrete GPU provides. This makes the available field hardware, the GTX 1650, the primary bottleneck in achieving consistent real-time inference performance.

The highest inference times across all results are produced by the YOLO medium model at 1920 pixel input resolution, as it represents the largest and most computationally demanding configuration. At CPU level, only one frame per second can be processed out of the 60 being received, which would require an exceptionally slow control system and is simply not feasible for real-time operation. The M3 MacBook GPU improves this to approximately 6 frames per second, however this remains far too slow for any meaningful control loop.

This underlines the necessity of adequate compute hardware. The GTX 1650 manages approximately 13 frames per second at this configuration, while the RTX 5070 reaches around 83 frames per second, technically exceeding the 60 FPS incoming stream rate. It is important to note however that these figures reflect inference time in isolation. In practice, the full processing loop encompasses additional steps including distance estimation, sending and receiving control signals, and rendering the user interface, all of which consume additional time each cycle. The true end-to-end throughput of the system will therefore be lower than the raw inference benchmarks suggest, meaning even the RTX 5070 may not sustain 60 FPS once all pipeline stages are accounted for.

5.1.3 Tracking Performance

Since detection is the largest addressable bottleneck in the pipeline, optical flow tracking was introduced to reduce the number of full detections required per second. This goal is clearly achieved according to the results in Table 4.7, which demonstrate that the system is capable of on average track a drone for over 500 milliseconds using only optical flow information under the original configuration. In practice, this means the system could run the detector as infrequently as four times per second, relying on the tracker for the intervening 250 milliseconds, while still maintaining sufficient positional accuracy for flight control.

Increasing the aggressiveness of the tracker, by raising the number of tracked

points and the search window size, did not yield better results. On the contrary, the original, more conservative configuration produced longer continuous tracking streaks. This suggests that tracking fewer points more carefully is more robust than attempting to track many points aggressively, likely because noisy or poorly matched points introduce drift that destabilizes the bounding box estimate over time.

Comparing the IoU ≥ 0.3 and IoU ≥ 0.6 thresholds reveals a meaningful difference in tracking quality over time. The average streak length at IoU ≥ 0.6 is approximately 0.29 seconds, compared to 0.56 seconds at IoU ≥ 0.3. This implies that during roughly the latter half of each tracking streak, the IoU falls between 0.3 and 0.6, indicating that the bounding box gradually drifts from the true drone position as the streak progresses. In practical terms, this drift means the system may slightly misestimate the drone's position or size toward the end of a tracking window. However, since true detections are interleaved frequently, the exact frequency depending on available compute, the tracker is regularly re-anchored to an accurate detection, making this accumulated drift negligible for real-world control purposes.

Several factors contribute to tracking streaks ending prematurely. The most common cause observed in the test footage is sudden acceleration or the drone moving out of frame, both of which violate the small motion assumption underlying Lucas–Kanade. In slower, more continuous motion sequences, drift accumulates gradually until the bounding box no longer sufficiently overlaps the true drone position, and the tracker is reset by the next detection.

5.2 Detection Evaluation

There are many aspects to consider when evaluating detection model performance. Latency is addressed separately in its own section. Here the discussion focuses on what makes a model well-suited for this application in terms of versatility, including how model size and input resolution present a combined trade-off against available computational resources. This includes how to balance detections against false detections, how the choice of dataset affects results, how different backgrounds influence performance, and how to select parameters that best suit the operational conditions.

5.2.1 Detection Models

There are several ways to evaluate model performance, whether by model size, input resolution, or the combination of the two. Each perspective offers different insights and all are worth discussing.

5.2.1.1 Effect of Model Size. Due to the number of variable combinations, the effect of model size is discussed using 1600 pixels as a fixed input resolution, as this is the largest resolution available across all three model classes.

The two tables show quite different results. Looking at accuracy alone, Table 4.1 shows that Medium performs best at long range, while Table 4.2 shows Small per-

forming best at close range, with Medium as a close second. A larger model may be better suited for long-range detection, where the drone occupies few pixels and must be distinguished from a more varied background, requiring greater representational capacity. At close range, the appearance of the drone is less varied, particularly given that a large portion of the dataset consists of close-up images, which may favor smaller, more specialized models.

This suggests that different model sizes could serve different roles within the same system. Based on these results, a switching strategy could be considered, where a Medium model is used when the target drone is not yet in close range to maximize detection capability, and a smaller model is engaged once the drone is within a certain distance, reducing computational load and energy consumption.

Another difference worth noting is precision, reflecting how many of the detected objects are actually drones. Medium performs worst at close range despite its larger capacity. One possible explanation is overfitting, where the model has learned to detect drones in a variety of contexts but struggles to reject false positives in the close-up dataset. This may also be related to the nature of the background in the close-up images, which consists largely of rocky terrain. Rocks viewed from above can resemble drones in shape, making false detections more likely. Paradoxically, this effect diminishes at longer range, where the drone appears as a small dot and is therefore easier to distinguish from background texture.

It should also be noted that the validation procedure counts every detection instance, meaning the model may produce multiple detections per image even when only one drone is present, which artificially lowers precision. In practice this is less of a concern, since the software selects the highest confidence detection above the confidence threshold, effectively filtering out redundant detections.

In terms of recall, Medium performs best at long range, correctly detecting the highest proportion of annotated drones. Taken together with its precision results, this means that the Medium model is the most capable of finding drones that are present in the image, making it the strongest overall performer for long-range detection.

5.2.1.2 Effect of Input Resolution. As can be viewed in the results, the larger the image size, the higher the accuracy on both long and short distances. On close image validation, viewed in Table 4.2, both small 1920 and nano 1920 perform very well compared to lower model resolutions.

Also, in the far distance validation seen in Table 4.1, small 1920 and nano 1920 have higher accuracies compared to all other models except the medium 1600 model. This may depend on that more features can be extracted when having a higher resolution. That can in turn result in the ability to detect more drones, resulting in higher accuracy.

In general, the larger the input resolution is, the better detection performance it has. One reason for this might be that the models can take better decisions with more trainable weights. The downside with larger input resolutions is longer latency.

In Figure 4.13 there is a steady trend of longer computing times with larger input resolutions.

5.2.2 Effect of Confidence Levels

The choice of confidence threshold has a significant impact on overall system performance, as seen across the detection results tables. There is a clear trade-off between precision and recall as the threshold changes: higher confidence leads to higher precision but lower recall. This is because a stricter threshold causes the model to produce fewer detections, but those that pass the threshold are more likely to be true drones, since the model only commits to a detection when it is highly certain based on what it has learned during training.

An interesting consideration is that the confidence threshold does not need to remain fixed throughout the operation of the system. When no drone has been detected for some time, lowering the threshold may help the system pick up a distant or partially obscured target and begin moving towards it for further investigation. Once the target is confirmed and the drone is in close range, raising the threshold would reduce the risk of confusing the target with background objects such as rocks, ensuring more reliable tracking. This kind of adaptive thresholding could therefore improve both the search and tracking phases of the system without changing the underlying model.

5.2.3 Detection Range

Background has a clear effect on the model's ability to differentiate a drone within an image. When comparing how far away a drone can be tracked in Table 4.8, the model can detect the drone with sky as background 40.2 – 77.4 meters further compared against ground background. This might reflect that there are fewer features to be extracted having a drone towards a sky compared to the ground. Objects on the ground as well as the background color could make it harder for the model to distinguish the drone, particularly when the drone is small at a long distance.

Another contributing factor may be the composition of the dataset. A large proportion of the training data contains drones against a sky background, which is relatively easy to collect since it does not require a two-drone setup, and the appearance of a drone against the sky changes little between seasons. Data featuring drones in rough terrain is comparatively harder to collect and was underrepresented in the dataset. As seen in Figure 4.2, such images are visually intensive even for the naked eye, requiring considerable effort to identify any pattern or pick out the drone from the surrounding terrain. With fewer training examples of this type, the model has less opportunity to learn the visual patterns needed to reliably detect drones in these conditions. It is therefore possible that with a more balanced dataset, performance against terrain backgrounds could have been comparable to that achieved against the sky.

These results also provide a basis for adding a safety layer to the system. Knowing the realistic detection range, the system can sanity check the distance estimates produced by the bounding box. If an estimated distance exceeds what the results show to be achievable, for example a reported distance of 100 meters which falls well outside the validated range, the system can flag this as unreliable. Rather than acting on an implausible distance estimate, it could instead indicate that a drone has been spotted in a given direction without providing a distance value, until a more credible estimate is available. The appropriate threshold for this would depend on the end system design and the background conditions in the operational environment.

5.2.3.1 Further Validation. Given the inherent positional uncertainty of the GPS devices, the experiment could be improved by combining the GPS measurements with a physical ground truth measurement. Since a hovering drone is subject to drift from wind and imperfect internal control, an alternative approach would be to approach the stationary drone very slowly and stop the moment a detection occurs. The approaching drone could then be landed at that position, and the distance between the two drones measured directly with a tape measure on the ground. While hover drift means that even the tape measure would carry some inaccuracy, comparing the two measurements against each other would still provide a valuable indication of their agreement, and would allow the reliability of the GPS-derived distances to be better understood.

5.2.4 Limitations of the Model

Dataset quality and size are always a limitation when training detection models. Ideally, a model would be exposed to an infinite variety of images to generalize across all possible conditions, but in practice data collection is constrained by time and effort. In this project, data was gathered through web scraping and images captured at the same test site, which may limit the diversity of the dataset. As a result, the model may struggle to generalize to environments or lighting conditions not represented in the training data, meaning the reported performance metrics may not fully reflect real-world detection capability across varied scenarios.

As an example, a large portion of the training data was captured during winter, resulting in bright white snowy backgrounds being overrepresented in the dataset. This introduced two limitations related to the season. First, the cold conditions made data collection difficult, as time outside was limited and the drone had to be protected from frost. Second, the validation data were captured primarily in spring, with greener and rockier backgrounds that differ significantly from the snowy conditions in the training data. This mismatch between the training and validation environments may have negatively affected the performance metrics reported, as the model was evaluated under conditions to which it had limited exposure during training.

5.3 Estimate Distance Evaluation

Since the scope limits the interaction distance to a range of 5–10 m, the distance estimation method was tested in an indoor environment rather than in the field using images of drones. This was primarily due to time constraints, as outdoor experiments require considerable planning in terms of equipment and available personnel. Indoor testing showed an accuracy of 10–30 cm deviation from the true distance, which was considered sufficient for the intended use case. Since the goal is to maintain a set following distance without losing or colliding with the target, an error of up to 0.5 m would still be acceptable in practice.

5.3.1 Difficulties with Distortion

Although the calibrated image in Figure 3.8 appears undistorted to the naked eye, visual inspection alone is insufficient to confirm the accuracy of the calibration. Subtle errors in the distortion coefficients are difficult to detect visually, yet can introduce significant errors in distance estimation due to the sensitivity of the method to small inaccuracies in the normalized coordinates.

A further source of error arises from the assumption that the drone's real-world width W is constant. The calibrated width was measured from the front or rear of the drone, however, when the drone is viewed from the side, its apparent width differs due to its rectangular shape. This means the assumed W may not reflect the true projected width at a given orientation, introducing a systematic error that varies with the drone's heading relative to the camera. Combined with the inherent noise in the bounding box coordinates produced by the detector, these factors contribute to the instability observed in the distance estimates.

5.3.2 Future Distance Evaluation

A potential improvement to distance estimation would be a more controlled field experiment. Both drones would hover in place, each equipped with a GPS logger, while the ground distance between them is measured independently to verify the GPS-derived separation. Detections would then be recorded systematically at a range of distances, allowing the displayed distance estimates to be compared against the known ground truth. With a sufficient number of measurements, this would enable a statistical confidence interval to be established for the system's distance estimation accuracy.

6

Conclusions

The question remains whether the research questions that initiated this project were successfully addressed. Can the tracking and detection system be considered valid in a real-time sense, and was it delivered as intended? Is the system well designed and reliable, and what fault control mechanisms are in place? Finally, what limitations exist that could serve as bottlenecks for a vision-based approach to drone detection?

6.1 At what distance can a drone be reliably detected and tracked using the developed system?

As discussed, detection capability varies depending on the background against which the target drone is observed. Against an open sky, the results in Table 4.8 show that reliable detection is achieved well within the 20 meter project constraint, with a substantial margin even accounting for the timing uncertainty. Against more challenging backgrounds, the picture is less clear. Detection within 10 meters can be stated with confidence given the error margins, while detection up to 20 meters remains possible depending on the specific image conditions.

Taking the most conservative result from Table 4.8, where the lower bound of the detection range reaches 9.6 meters, it can be concluded that the system is able to reliably detect and track a target drone within that distance regardless of background condition. Beyond that, performance against the sky is consistently strong, while more visually complex backgrounds remain a challenge at greater range.

6.2 How can detection models and tracking be combined to achieve reliable real-time drone detection and tracking?

As shown in Table 4.3 and Table 4.7, the combination of detection and tracking meets the latency requirements of a real-time system. For the system to function

reliably, a consistent and accurate estimate of the target position must be available at all times.

Considering latency alone, a GPU comparable to a GTX 1650 achieves between 8 and 13 frames per second depending on the model, as seen in Table 4.3, assuming there is no other processing load on the system. The tracker adds only 3 ms, as seen in Section 4.3, of overhead per frame and was shown to maintain a valid estimate for an average of approximately 500 ms, as seen in Table 4.7 between detections. This means that in theory, as few as two detection frames per second would be sufficient to keep the tracker anchored. However, since detections are inherently more accurate than tracker predictions due to the gradual drift introduced by optical flow, maximizing the detection frame rate remains desirable. The tracker should therefore be understood as a bridge between detections rather than a substitute for them, improving temporal continuity without replacing the accuracy that the detection model provides.

This understanding must be considered alongside Table 4.2 and Table 4.1, since meeting latency requirements is meaningless without reliable detection. Given that the project constraints place the system within 20 meters of the target drone at the start of operation, Table 4.8 indicates that detection at this range is achievable. Looking at the long-range dataset, while the drone is not detected in every frame, it is detected often enough to initiate movement towards the target. As the system closes in, conditions transition towards those of the close-up dataset, where accuracy reaches approximately 0.9. Combined with the tracker bridging gaps between detections, the system is therefore able to maintain a reliable detection rate and follow the target drone within the constraints of a real-time system.

6.3 How does model size and input resolution affect detection?

The results demonstrate that both model size and input resolution have a significant effect on detection performance, though the relationship is not straightforward and depends on the operational scenario.

Larger models generally perform better at long range, where the drone occupies few pixels and must be distinguished from a complex background. The Medium model achieves the highest recall in the far-distance dataset, meaning it finds the most drones that are actually present. At close range however, the advantage of a larger model diminishes, and the Small model performs best in terms of overall accuracy. This suggests that model size should ideally be matched to the operational phase, with a larger model used during the search phase and a smaller model once the target is nearby.

Higher input resolution consistently improves detection performance across both datasets and all model sizes. This is because more pixels provide the model with more features to work with, making it easier to distinguish a drone from

its background, particularly at distance. The trade-off is increased inference time, which grows steadily with resolution as shown in Figure 4.13.

The key insight is that model size and input resolution together form a combined trade-off against available compute. Choosing the right combination therefore requires balancing detection performance against the latency budget of the system.

6.4 Reliability Through System Design

A reliable system must account for what happens when individual components fail. Several safety measures were designed into the system with this in mind.

If the camera connection is lost, the compute unit enters a reconnection mode and attempts to re-establish the stream. During this time no commands are sent to the flight controller, which by design causes the drone to hover in place until a signal is received. If the connection is not restored within a defined time window, a return-to-home command can be issued, depending on the requirements of the end system.

The separation of the compute unit and the flight controller is another deliberate design choice. By keeping these as independent units, the operator retains full manual override at all times, regardless of the state of the autonomous pipeline. This effectively acts as a dead man switch, ensuring that a failure in the detection or tracking pipeline cannot result in uncontrolled behavior, and that a human operator can always intervene if needed.

Together these measures provide a layered approach to fault tolerance, where each failure mode has a defined and safe response. The system can be concluded to perform reliably within the scope of the project, maintaining safe behavior under both normal operation and foreseeable failure conditions.

6.5 Final Conclusion

Overall, this project demonstrates that autonomous vision-based drone detection and following is achievable using commercially available hardware and components. The developed system successfully detects, tracks, and follows a target drone in real time, with validated detection ranges and fault tolerant design. While dataset diversity and pipeline latency remain the primary limitations, the results provide a strong foundation for further development towards a fully operational counter-drone system.

7

Future Work

Due to limitations in scope, resources, and time, some ideas were not explored during the project. Additional suggestions emerged later in the project or came from people with different areas of expertise. This chapter summarizes potential directions for future work and ideas on which to build upon.

7.1 Dual Control through Gimbal and Drone

As described in the method, the gimbal could be controlled to minimize the positional error of the target within the image, keeping it centered in the frame. However, a limitation arose in determining the gimbal's orientation relative to the drone, as without this information it was not possible to know whether the camera was pointing straight ahead or to either side. A future improvement would be to implement a dual controller, where the gimbal is responsible for keeping the target centered in the image, while the drone continuously adjusts its heading to align with the gimbal orientation. This would result in smoother turns, reduce the risk of the target leaving the frame, and improve energy efficiency, since adjusting heading through gimbal rotation requires less energy than yawing the drone while hovering.

7.2 Infrared Camera

One problem with drone detection was that the target drone can blend into the background. This could be mitigated if an infrared camera were to be used instead. The drone will have a different temperature compared to the background, making it easier for the software to distinguish the target drone. It would be interesting to investigate this concept further.

7.3 Distance Estimation

To further improve distance estimation, the current assumption of a known target size could be revisited. In practice, the physical dimensions of an unknown drone

may not be available, making size-based estimation unreliable. Integrating a LiDAR sensor could address this limitation by providing direct range measurements independent of any assumptions about target size, potentially improving both accuracy and generalisability.

7.4 Evaluating different detection algorithms

This thesis has focused on YOLO models as detection models because they are the most advanced at the moment. However, there are competing algorithms such as Faster R-CNN and RTMDet that are more focused on performance on smaller devices. It would be interesting to further evaluate how small detection algorithm that can be made that retains a high accuracy.

7.5 Compute and control on device

Another interesting topic to research is how well the drone can compute the detection model on its own. There is a risk that the video and communication links could be interrupted and thus it would be advantageous for the drone to detect and navigate by itself. This could be done by mounting a small computer such as a Raspberry Pi. The major challenge for this concept is to run a detection algorithm on such a small computing device.

Another improvement would be optimizing the data pipeline. As confirmed by the report, the latency with which data arrives at the compute unit is quite large. Minimizing this pipeline delay by connecting the camera and compute unit directly on the same device would therefore be a significant improvement to the system as a whole.

Bibliography

Cosmostreamer (2026). *Cosmostreamer gd1 for dji goggles and drones*. https://cosmostreamer.com. Accessed: 2026-03-30.

DJI (2026a). *Dji goggles 3*. https://www.dji.com/se/goggles-3. Accessed: 2026-03-30.

DJI (2026b). *Dji o4 air unit series*. https://www.dji.com/se/o4-air-unit. Accessed: 2026-03-30.

Eddy, W. (2022). *Transmission Control Protocol (TCP)*. RFC 9293. IETF. URL: https://www.rfc-editor.org/info/rfc9293.

HEQ UAV TECH (2026). *Hequav g-port 3-axis gimbal*. https://www.hequav.com/products/hequav-g-port-3-axis-gimbal. Accessed: 2026-03-30.

Hidayatullah, P. and R. Tubagus (2026). *Yolo26: a comprehensive architecture overview and key improvements*. arXiv: 2602.14582 [cs.CV]. URL: https://arxiv.org/abs/2602.14582.

Lin, T.-Y., P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie (2017). *Feature pyramid networks for object detection*. arXiv: 1612.03144 [cs.CV]. URL: https://arxiv.org/abs/1612.03144.

Lucas, B. D. and T. Kanade (1981). "An iterative image registration technique with an application to stereo vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 674–679.

MAVLink (n.d.). *Packet Serialization*. https://mavlink.io/en/guide/serialization.html. Accessed: 2026-03-31.

Network Supply (2025). *The OSI model, explained*. https://www.network-supply.com/blogs/knowledge/the-osi-model-explained. Accessed: 2026-03-31.

OpenCV (2025). *Optical flow*. https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html. Accessed: 2026-03-30.

OpenCV Foundation (n.d.). *OpenCV: Open Source Computer Vision Library*. https://opencv.org/. Accessed: 2026-03-30.

Qstarz (2026). *Gps-based data recorder*. Accessed: 2026-05-12. URL: https://www.qstarz.com.

Rassler, D. and Y. Veilleux-Lepage (2025). "On the horizon: the Ukraine war and the evolving threat of drone terrorism". *CTC Sentinel* **18**:3.

Schulzrinne, H., A. Rao, and R. Lanphier (1998). *Real Time Streaming Protocol (RTSP)*. RFC 2326. URL: https://datatracker.ietf.org/doc/html/rfc2326.

Vina, A. (2024). *How to use ultralytics yolo11 for instance segmentation*. Accessed: 2026-03-31. URL: https://www.ultralytics.com/blog/how-to-use-ultralytics-yolo11-for-instance-segmentation.

Wilson, T. (2025). *Copenhagen and oslo airports forced to close temporarily due to drone sightings*. Accessed: 2026-01-28. URL: https://www.bbc.com/news/articles/cn4lj1yvgvgo.

A

Appendix

A.1 Drone Components

Flight Controller

MatekSys H743 Slim V4

Speed Controller

SEQURE E70 G1 2-8S 70A BLHeli_32 / AM32 128K 4in1 ESC

Radio Transmitter and Receiver

RadioMaster XR4 Gemini Xrossband Dual-Band ExpressLRS Receiver

GPS and Compass

MatekSys M9N-G4-3100

Battery Elimination Circuit

MatekSys BEC12S-PRO

Camera and digital video transmitter

DJI O4 Pro Air Unit

Gimbal

HEQUAV G-Port 3-Axis

A.2 Video Link Components

Video Receiver

DJI Goggles 3

Streaming Box

Cosmostreamer GD1

A.3 Radio Link Components

Transmitter and Receiver

RadioMaster Nomad Dual 1-watt Gemini Xrossband ExpressLRS Module

Radio Controller

RadioMaster TX16S

A.4 Software

The source code developed for this project is maintained in a private repository and is not publicly available.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
May 2026

Document Number
TFRT-6314

Author(s)
Scott Tollebäck
Vilhelm Persson

Supervisor
Axel Müller, Saab AB, Sweden
Kristian Soltesz, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Design of an Autonomous UAV Tracking System

Abstract

Drones are becoming more common everywhere, and with that comes the risk of unauthorized drones entering protected areas such as airports or military sites. One way to deal with this is to send out another drone to find and follow the unknown one for closer inspection.

This project built a system that does exactly that. A drone equipped with a camera is able to automatically spot another drone, estimate how far away it is, and follow it, all without a human controlling it. The system uses an object detection model to find the drone in the camera image and a tracking method to keep following it between detections.

Tests showed that the system could spot a drone from up to around 80 meters away when the sky was in the background, and around 20 meters when the background was rocky terrain. Once close enough, the system was able to follow the target drone at a distance of around 10 meters in real time.

The results show that this kind of autonomous drone detection and following is possible using standard hardware and commercially available components, though improvements to the camera pipeline and more diverse training data would make the system more robust in the future.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-78

Recipient's notes

Security classification

http://www.control.lth.se/publications/