CHALMERS
UNIVERSITY OF TECHNOLOGY

Automation of a Drone Swarm Surveillance System

Bachelor thesis in Electrical Engineering

Sebastian Backman
Filip Hansen
Eric Hägge Lundberg
Isac Lilja
Ludvig Magnusson
Lisa Nyström

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Bachelor thesis 2026

Automation of a Drone Swarm Surveillance System

Sebastian Backman
Filip Hansen
Eric Hägge Lundberg
Isac Lilja
Ludvig Magnusson
Lisa Nyström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2026

Automation of a Drone Swarm Surveillance System
Sebastian Backman
Filip Hansen
Eric Hägge Lundberg
Isac Lilja
Ludvig Magnusson
Lisa Nyström

Supervisors:
Emmanuel Dean, Department of Electrical Engineering
Mikael Enelund, Department of Mechanical Engineering
Robert Brenick, AstaZero
Examiner: Knut Åkesson, Department of Electrical Engineering

Bachelor Thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Picture of a flying drone.
Photo by Pok Rie [1]

Typeset in LaTeX
Gothenburg, Sweden 2026

Automation of a Drone Swarm Surveillance System

Sebastian Backman
Filip Hansen
Eric Hägge Lundberg
Isac Lilja
Ludvig Magnusson
Lisa Nyström

Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis addresses the need for a scalable and flexible drone-based surveillance platform for the autonomous vehicle testing industry. The platform should support cooperation between drones from different manufacturers through a unified communication interface and automatically generate missions in response to detected objects. The project was conducted as a collaboration between Chalmers University of Technology, Sweden, and The Pennsylvania State University, USA, for AstaZero, a research and testing facility for automated and connected vehicles. The developed platform integrates a backend system responsible for drone coordination, mission planning, and communication between system components with a web-based operator interface. The interface allows operators to define surveillance areas on an interactive map, monitor live video streams and telemetry data, and manage the generated missions when objects are detected. The system also evaluates available drones based on hardware capabilities and operational status to autonomously select the most suitable one for a selected mission, and determine appropriate flight parameters for effective surveillance coverage. Experimental validation, including on-site testing at AstaZero, demonstrated the platform's ability to communicate with multiple drones, dispatch missions, stream real-time video, and integrate object detection into the surveillance workflow. Limitations regarding object detection reliability and hardware dependencies were identified during the project. Nevertheless, the project shows the feasibility of a scalable multi-drone surveillance platform for autonomous vehicle testing environments.

Keywords: drone surveillance, multi-drone systems, drone swarm, MAVLink, DJI SDK, object detection, YOLO, AstaZero, mission planning

Automation of a Drone Swarm Surveillance System

Sebastian Backman
Filip Hansen
Eric Hägge Lundberg
Isac Lilja
Ludvig Magnusson
Lisa Nyström

Department of Electrical Engineering
Chalmers University of Technology

Sammandrag

Detta kandidatarbete behandlar behovet av en skalbar och flexibel drönarbaserad övervakningsplattform vid testanläggningar för autonoma fordon. Plattformen möjliggör samarbete mellan drönare från olika tillverkare genom ett enhetligt kommunikationsgränssnitt och kan automatiskt generera uppdrag när objekt detekteras. Projektet genomfördes som ett samarbete mellan Chalmers tekniska högskola i Sverige och The Pennsylvania State University i USA, på uppdrag av AstaZero, en forsknings- och testanläggning för autonoma och uppkopplade fordon. Den utvecklade plattformen kombinerar ett backendsystem med ett webbaserat operatörsgränssnitt. Backendsystemet ansvarar för drönarkoordinering, uppdragsplanering och kommunikation mellan distribuerade systemkomponenter. Gränssnittet möjliggör för operatören att definiera övervakningsområden på en interaktiv karta, följa videoströmmar och telemetridata från drönarna i realtid, samt hantera de uppdrag som genereras vid objektdetektering. Plattformen utvärderar dessutom de tillgängliga drönarna automatiskt utifrån deras hårdvarukapacitet och driftstatus för att välja den mest lämpliga för de givna uppdragen, samt bestämma dess flygparametrar för effektiv övervakningstäckning. Experimentell validering, däribland tester vid AstaZeros testanläggning, visade att plattformen kan kommunicera med flera drönare, skicka uppdrag, strömma video i realtid och integrera objektdetektering i övervakningsarbetsflödet. Under projektets gång identifierades begränsningar med avseende på objektdetekteringens tillförlitlighet, behovet av ytterligare drönare för fullskaliga tester samt storskalig validering i verkliga miljöer. Trots dessa begränsningar visar projektet att en utbyggbar plattform för övervakning med flera drönare är genomförbar vid testanläggningar för autonoma fordon.

Nyckelord: drönarövervakning, system med flera drönare, drönarsvärm, MAVLink, DJI SDK, objektdetektering, YOLO, AstaZero, uppdragsplanering

Acknowledgements

We would like to express our sincere gratitude to everyone who supported us throughout this project.

First, we would like to thank our supervisors at Chalmers University of Technology, Mikael Enelund and Emmanuel Dean, for their guidance, support, and insightful feedback. Their expertise and engagement have been invaluable throughout the entire project.

We would also like to thank Robert Brenick from RISE AstaZero for his availability and willingness to answer technical questions. His input helped clarify objectives and provided valuable structure for the project.

Furthermore, we would like to express our appreciation to our collaborators at The Pennsylvania State University - Rishab Srinivasan, Manit Garg, Kaisheng Cheng, Yonathan Solomon, and Aarya Soni - for their support, helpful discussions, and productive collaboration during the course of this work.

We would also like to extend our gratitude to Darryl Farber for the invitation to participate in the workshop "Strategic Foresight of Engineering Systems and Risk Management for National Security: Anticipating Technical Surprise of Autonomous and Intelligent Systems", which covered interesting topics such as strategic risk management, advanced manufacturing, and ethical dimensions of global trends such as AI.

Finally, we would also like to express our deepest gratitude to the Herbert & Karin Jacobsson Foundation for their generous financial support, which gave us the incredible opportunity to both participate in the workshop above which took place in Washington, D.C., and to visit State College, the location of The Pennsylvania State University, to finalize the project together. It truly was an unforgettable experience.

Preface

The work presented in this thesis originates from a joint capstone project between Chalmers University of Technology in Gothenburg, Sweden, and The Pennsylvania State University in State College, United States of America. From January to May 2026, six students from Chalmers and five students from The Pennsylvania State University collaborated on the development of a dynamic multi-drone surveillance system for autonomous vehicle testing at AstaZero, a world-leading proving ground for autonomous vehicles located outside of Gothenburg, Sweden.

The partnership relied on continuous communication, as well as weekly virtual meetings, laying a solid foundation for exchanging ideas, code and overcoming challenges encountered throughout the project. The collaboration was further strengthened through a visit by the Chalmers team to Washington, D.C., and State College, where both teams participated in the Strategic Foresight Workshop organized by Business Executives for National Security (BENS) in collaboration with The Pennsylvania State University.

Sebastian Backman
Filip Hansen
Eric Hägge Lundberg
Isac Lilja
Ludvig Magnusson
Lisa Nyström
Gothenburg, May 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
API	Application Programming Interface
ATOS	Automated vehicle Test Operating System
CUDA	Compute Unified Device Architecture
DJI	Da Jiang Innovations, a drone manufacturing company
FIFO	First-In, First-Out
FOV	Field of View
FPV	First-Person View
GDPR	General Data Protection Regulation
GitHub	Developer platform for managing and sharing code
GUI	Graphical User Interface
HOG	Histogram of Oriented Gradients
HTTP	Hypertext Transfer Protocol
IMY	Integritetsskyddsmyndigheten (Swedish Authority for Privacy Protection)
IP	Internet Protocol
IPC	Inter-Process Communication
MAVLink	Micro Air Vehicle Link, a protocol for communication with drones
OBB	Oriented Bounding Box
R-CNN	Region-based Convolutional Neural Network
RISE	Research Institutes of Sweden
ROS	Robot Operating System
SDK	Software Development Kit
SIFT	Scale-Invariant Feature Transform
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
WebRTC	Web Real-Time Communication
YOLO	You Only Look Once

Contents

1
Introduction

While the automotive industry continues to advance both technically and competitively, there is an increasing demand for efficient proving grounds capable of supporting safety-critical testing of autonomous vehicles. One way to improve such facilities is to develop robust surveillance systems that enhance operational uptime. This project investigates the possibility of a scalable drone-based surveillance system, utilizing drones equipped with cameras and speakers for monitoring, interactions, and intruder responses.

A prominent example of such a facility is AstaZero, located outside Borås, Sweden. As a non-profit organization owned by RISE (Research Institutes of Sweden), AstaZero provides a full-scale testing environment for the testing and validation of diverse autonomous systems. To maintain high safety standards, continuous surveillance is essential, a task where drones can play a critical role. By utilizing real-time object detection, these drones can effectively detect intruders and be dispatched by an operator to quickly intercept them, thereby contributing to a more modular and automated surveillance system.

The project is a collaboration between Chalmers University of Technology (Chalmers) and Pennsylvania State University (Penn State) and builds upon prior project iterations conducted at AstaZero. The objective is to develop a flexible digital environment that enables multi-drone operations across different drone platforms and communication standards. The final system was evaluated through on-site testing at AstaZero's proving ground, executing the defined surveillance and safety missions.

1.1 Objectives

The primary objective of this project was to enhance the safety and security of autonomous vehicle testing at the AstaZero facility by implementing an automated drone-based surveillance system. The system was designed to autonomously identify intruders within the test area and provide a rapid response by coordinating several drones. By integrating automated mission planning when an intruder is detected, the system can dynamically assign tasks, such as audible warnings or visual illumination, to the most suitable drones, ensuring that potential risks are mitigated without disrupting ongoing tests.

Central to this solution is the development of a user-friendly frontend interface that serves as a command-and-control dashboard. This interface enables operators to define the testing area, monitor live video streams, and manage autonomous mission proposals. By providing the ability to accept or reject proposed interception missions in real time, the dashboard ensures that the operator remains an important part of the decision-making process.

1.1.1 Key Project Milestones

To provide a structured framework for the project, the main objective was divided into five concrete goals:

- Develop a system capable of automatically connecting to multiple types and numbers of drones and retrieving their model specifications, available add-ons (e.g., cameras, speakers), and telemetry (e.g., battery percentage, altitude, coordinates) through a generalized communication protocol.
- Integrate an interactive map within the frontend, allowing operators to define a surveillance area. Based on the selected area, the system autonomously calculates optimal flight altitudes and positioning to ensure maximum coverage.
- Implement a mission planning function that creates missions when an intruder is detected for available drones based on their current capabilities and telemetry data.
- Develop a user-friendly frontend interface that provides an overview of the surveillance area, live video streams with detected objects overlay, a list of connected drones and their capabilities, and where the active and proposed missions are shown.
- Integrate existing object detection, enabling data transmission to mission planning functionality.

To further simplify the evaluation of the project, the key milestones were translated into concrete and measurable requirements and desires. This technical specification is presented in the requirement table found in Appendix B.1.

1.2 Limitations and Delimitations

To ensure a manageable scope, several delimitations have been applied in this project. In addition, a number of limitations affecting the project have been identified. These are outlined below.

1.2.1 Software

In this project, the team only engaged in software development, and no hardware development occurred. The object detection software was only able to detect and recognize objects such as bikes, pedestrians, and motor vehicles that are relevant for the test site at AstaZero, as implemented in the prior work [2], and was not further

developed.

AstaZero currently uses an in-house developed software called ATOS (Automated vehicle Test Operating System), further described in Section 2.7. Full integration of ATOS would be ideal for differentiating between test objects and intruders. However, due to its complexity, ATOS has not been fully integrated in the prior project and was not further integrated as it would be too time-consuming and not a priority.

1.2.2 Drones

Two DJI (Da Jiang Innovations) Mavic 2 Enterprise drones were provided by AstaZero to the Chalmers team, while Penn State was provided with one drone utilizing a Holybro Pixhawk 4X flight controller. This was sufficient to develop and test communication between the two types of drones, although it limited testing other drone types. The two drones provided by AstaZero were equipped with cameras and various additional equipment, such as speakers, spotlights and strobes. The drone provided by Penn State initially lacked both a camera and additional equipment; however, a SIYI A8 mini camera was acquired during the course of the project.

1.2.3 Access to AstaZero

Access to AstaZero's test track was highly constrained, as it is fully booked daytime by companies. In combination with limited daylight hours during early spring in Sweden and the dependence on suitable weather conditions, the possibilities for system testing at AstaZero were expected to be very limited [3].

1.3 Collaboration with The Pennsylvania State University

This project was carried out in collaboration with Penn State. The purpose of the collaboration was to leverage the complementary strengths and perspectives of each institution, while reinforcing the academic ties between Chalmers and Penn State. This international partnership introduced additional dimensions to the work, requiring effective coordination across time zones and cultural differences, as well as effective and continuous communication between the two teams.

The workload was divided based on the hardware available to each team, as defined in Section 1.2.2, with each team responsible for integrating their respective hardware into the system. Beyond this division, both teams collaborated on the same set of shared components and worked towards the same goals.

1.4 Background

The following section provides the necessary context for this project. It begins with a description of the testing environment and the specific facilities involved, followed by an overview of previous work that serves as the basis for the current system's development.

1.4.1 AstaZero

AstaZero [4] is the first full-scale autonomous vehicle testing site in the world, located outside of Borås, Sweden. It is a non-profit organization fully owned by RISE. The facility provides testing in all types of traffic environments and enables various autonomous systems to be tested before they are certified for use on public roads. A map of the site is shown in Figure 1.1. The wide range of testing opportunities makes this testing site valuable for vehicle manufacturers, legislators, universities, colleges, and suppliers [5].

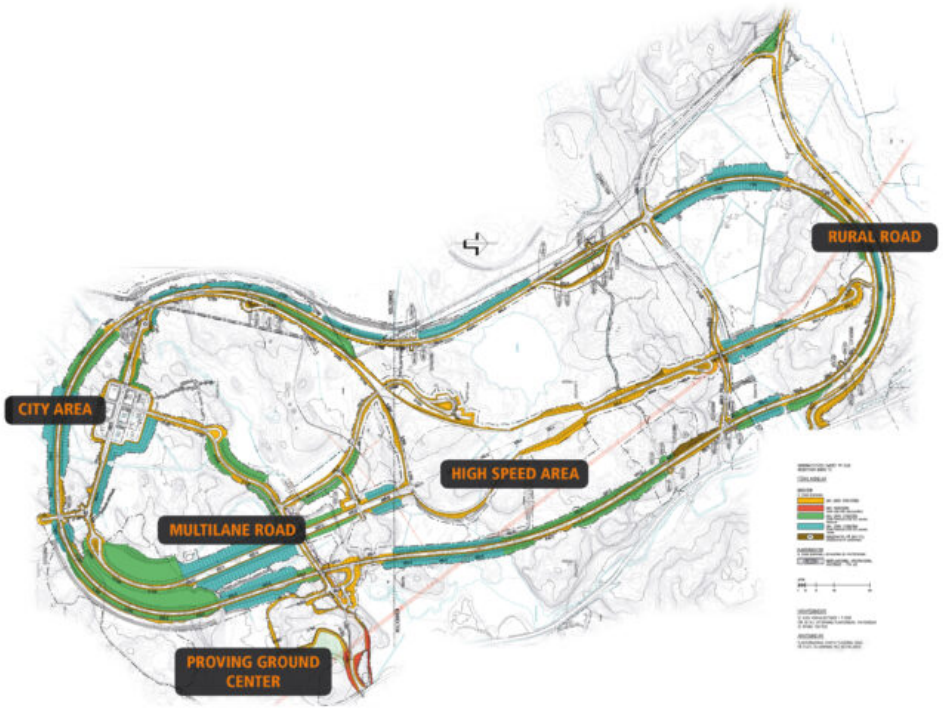

Figure 1.1: Map of the test center at AstaZero. Image taken from [4].

1.4.2 Previous Work

The project is mainly built upon two previous Chalmers and Penn State joint bachelor thesis capstone projects [2], [6]. Below is a summary of these projects, highlighting the specific components developed and how they played an important role for this project.

Work of 2024

From the 2024 project *Automate the Video Documentation of a Euro NCAP Test by Use of Drones* [6], image detection techniques were developed to support automated analysis of recorded footage. These techniques enable the detection of foreign objects as well as the estimation of their positions using drone cameras. This functionality provides a foundation for further development of automated monitoring and mission suggestion. A more detailed theoretical background on object detection is presented in Section 2.10. In addition to object detection, an Android application was developed. The application functions as a communication relay between the backend and the drone's controller, which is essential for sending commands within the DJI ecosystem.

Work of 2025

The 2025 project *Drone Platform for Safety in Testing* [2] established the foundation for the 2026 project. The system architecture included the use of a Redis database for data management, see Section 2.2.1, as well as Docker to ensure a consistent runtime environment, see Section 2.4. Furthermore, a frontend interface was developed to simplify drone operations, enabling control without the need for a traditional controller. The system also included an image stitching module, which enabled two drone video streams to merge into a single, larger FOV (Field of View). While this functionality remains part of the inherited architecture, it was not the primary focus of this year's development due to the limited number of drones and focus on interception missions. The full system architecture is presented in Figure 1.2.

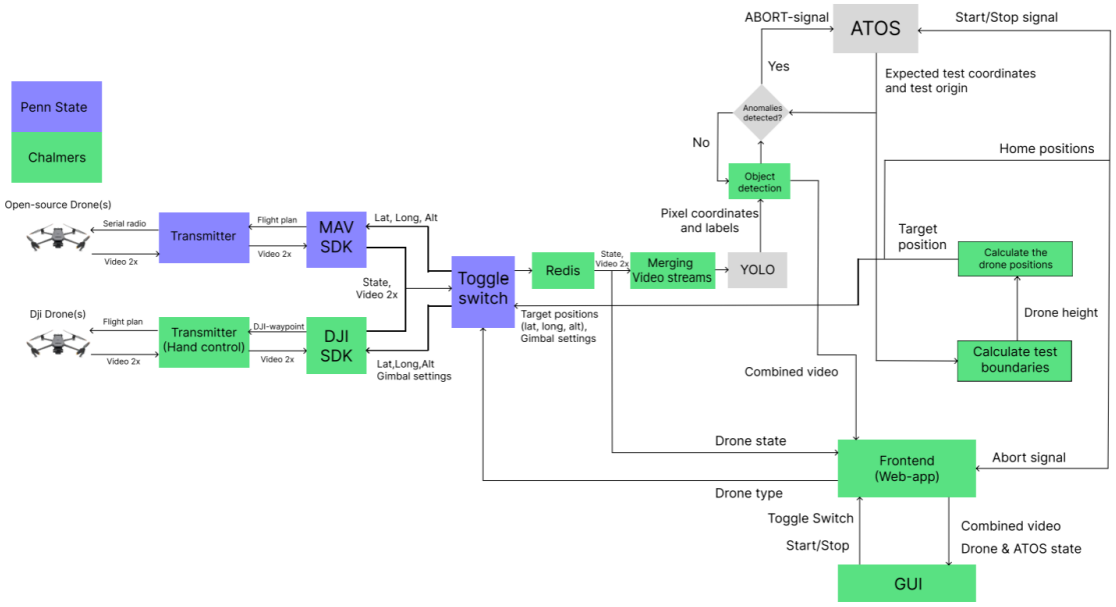

Figure 1.2: 2025 project system architecture. Taken from [2].

2
Technical Background

This chapter presents the theoretical foundation necessary to carry out the project. It describes key concepts, methods, and background knowledge about the project's design and implementation.

2.1 Hardware and Communication Systems

This section presents the hardware and communication systems central to the project. The primary hardware components are the drones used by each team, which serve as the physical foundation of the surveillance system. For these drones to function as part of a larger automated system, it is essential to have reliable communication interfaces. These interfaces enable the exchange of commands, telemetry data, and video streams between the drones and the rest of the system. Without such interfaces, autonomous operation and coordination between multiple drones would not be possible.

2.1.1 DJI Mavic 2 Enterprise

The Chalmers team used two DJI Mavic 2 Enterprise drones, developed by DJI. The package includes the drone itself, a remote controller, and a range of attachable equipment such as a speaker, spotlight, and strobe. An Android phone is also part of the setup, used as an interface between the remote controller and the system.

DJI SDK

Communication with the DJI Mavic 2 Enterprise is handled through the DJI SDK (Software Development Kit), a collection of pre-built tools developed by DJI for building applications that communicate with and control DJI drones [7]. The SDK supports automation of flight missions and enables retrieval of telemetry data such as GPS position, altitude, speed, and battery level. As the SDK is not supported on computers for most drone models, an Android or iOS device is required to use the SDK [7]. The phone is connected to the remote controller via a USB cable, through which an application on the phone can interact with the drone.

2.1.2 MAVLink Drone Platform

The Penn State team used a custom-assembled drone platform based on the Holybro Pixhawk 4X flight controller, paired with a SIYI A8 mini camera and a Raspberry

Pi, which is a small single-board computer used for onboard processing. As the platform is assembled from individual components rather than being a commercially available product, it is not tied to a proprietary software environment, allowing greater configurability and flexibility in development.

MAVLink Communication Protocol

Communication is handled through MAVLink (Micro Air Vehicle Link), which is an open protocol designed for the exchange of data between drones and other integrated components. The protocol is highly regarded for its reliability and its compatibility with a wide range of microcontrollers, drone types, and programming languages, with messages defined using XML files [8]. As MAVLink is not limited to mobile devices, communication can be established through any suitable interface, such as a radio antenna, making the development process more flexible compared to the DJI SDK.

2.2 Backend

The backend in software engineering constitutes the server-side component of a web application and is responsible for the underlying functionality and logic of the system. It manages data processing, application logic, and communication with databases and external services. Unlike the frontend, Section 2.3, the backend does not typically interact directly with the user but instead processes requests generated by user interactions. For example, when a user clicks a button on the frontend, the backend determines how the system should respond by executing the appropriate operations and returning relevant data to the frontend [9].

2.2.1 Redis

Redis is an open-source, in-memory key-value database system that enables fast data access. Rather than storing data on a disk or a solid-state drive, Redis instead stores data in memory, which enables higher performance and reliability [10]. The data is stored as values, each associated with a unique identifier called a key, which is used by the software to retrieve the desired data.

2.2.2 IPC

IPC (Inter-process communication) refers to mechanisms that allow multiple running processes to communicate with one another [11]. Since processes execute within isolated memory spaces that are normally inaccessible to other processes for security reasons [12], operating systems provide safe communication mechanisms such as *pipes* and *sockets*.

To share data between processes, a common approach is to use a dedicated process responsible for storing and managing the data, while other processes communicate with it through pipes or sockets to store and retrieve information. Redis is well

suited for this purpose due to its fast data access and support for request-response communication patterns.

2.2.3 Communication Protocols

WebSockets allows a constant two-way connection between a browser client and a server over TCP (Transmission Control Protocol) [13]. This enables data to be sent between them without needing to establish a new connection each time. WebSockets are therefore used to enable fast and reliable real-time communication.

HTTP (Hypertext Transfer Protocol) is a stateless protocol and the foundation of any data exchange on the web [14]. It uses a Request-Response model between a client and a server, where a request is initiated by the client using a method, a URL, and optionally a body. The body is protocol-specific data and is arbitrary. Most common methods include:

- **GET**: Used to retrieve data from a server. It should not modify the server's state.
- **POST**: Used to send data to the server to create or update a resource. Unlike GET, it often results in a change in state or side effects on the server.
- **DELETE**: Used to remove the specified resource from the server.

How a request with a specific method and URL is handled is determined by the server. A response is returned by the server, containing a status code and the requested resource. Because HTTP is stateless, each transaction is independent; the server retains no session memory between requests unless external mechanisms like cookies or tokens are employed.

2.3 Frontend

In software engineering, the frontend refers to the part of a web application with which the user directly interacts. It is responsible for the presentation layer, including the layout, visual elements, and user interface components. Frontend development is commonly implemented using technologies such as HTML, CSS, and JavaScript, often supported by frameworks and libraries such as React and Bootstrap. The primary objective of the frontend is to provide a user-friendly, responsive, and efficient interface that facilitates interaction between the user and the underlying system [9].

2.4 Docker

Docker is a container-based system that allows users to test, build, and deploy software applications smoothly and quickly. The platform stores the desired software into standardized units called containers, which also store everything needed to run the software, including libraries, system tools, code, and runtime [15].

One of the main advantages of using Docker is its portability. Since the containers store all dependencies needed to run the desired software, it's easy to migrate applications between environments. Another advantage is its capability of isolation between containers, which means that each container runs independently and therefore with high security, which is necessary for applications in shared environments to prevent one compromised or malfunctioning container from affecting others [16].

2.4.1 Docker Compose

Docker Compose is a tool for defining and running multi-container applications. Compose simplifies the control of an application stack and facilitates managing Docker services, networks, and volumes in a single configuration file. All the services can be created and started with a single command [17].

2.5 Android App

The Android application functions as a communication relay between the backend and the drone. Within the DJI ecosystem, drones typically lack the functionality to communicate directly with external servers via Wi-Fi or 4G [7]. Instead, all telemetry data and mission commands must pass through the remote controller, which is physically connected to a mobile device running the application.

By leveraging the DJI SDK, the application translates high-level instructions from the backend into low-level commands that the drone can execute. In the opposite direction, it captures real-time telemetry and video streams from the drone and forwards them back to the backend, ensuring a continuous data loop despite the hardware's connectivity constraints.

2.6 Communication Technologies

To ensure seamless interaction between the decentralized components of the system, several communication technologies and protocols are employed. These technologies enable the backend to coordinate with multiple drones across a shared network. This section outlines the fundamental principles, focusing on IP (Internet Protocol) Multicast for automated connectivity and the UDP (User Datagram Protocol) for low-latency transmission.

2.6.1 IP Multicast

IP multicast is a method of network communication in which one sender transmits data to a selected group of receivers simultaneously, rather than sending separate copies to each device. In IP networks, communication can occur as unicast (one-to-one), broadcast (one-to-all), or multicast (one-to-many), with multicast designed

for efficient group delivery [18]. Devices that wish to receive a multicast stream explicitly join a multicast group, and only those members receive the transmitted data [18]. This approach conserves bandwidth because the source sends a single stream into the network, and the network replicates packets only where needed instead of carrying multiple identical streams across the core network [18]. IP multicast is commonly used for applications such as IPTV, video distribution, and internet radio, where the same content must be delivered to many users at the same time [18]. A multicast group is identified by a dedicated IP address; in IPv4, these are Class D addresses from 224.0.0.0 to 239.255.255.255. [19]

A subset of the IP multicast protocol is *Administratively scoped IP multicast*. Packets sent to an administratively scoped IP address are required to not be forwarded by routers at the edge of a network, preventing the packets from leaving the network. The administratively scoped address space is defined to be the range 239.0.0.0 to 239.255.255.255. [20].

2.6.2 UDP

UDP is one of the core communication protocols for sending data on the internet. It allows data to be sent with a minimum of protocol mechanisms as well as delivery and duplicate protection not being guaranteed [21]. It also has no connection requirement, meaning data can be sent to the host without a confirmation that it is ready to receive. To ensure the data arrives, the host may have to send the same data repeatedly depending on the network congestion.

The transport layer used for sending the packets is not defined by the Internet Engineering Taskforce, but the lack of a connection requirement for UDP packets makes it a suitable protocol for sending to many receivers. Therefore, multicast transmission usually employs the UDP transport layer. [22].

2.7 ATOS

ATOS is an open-source ROS2-based scenario testing system developed by and used at the AstaZero testing center. ROS (Robot Operating System) is a set of open-source software libraries and tools that aid in building robots [23]. All participants used in testing scenarios, both vehicles in testing and surrounding vehicles as well as pedestrians, are controlled and monitored by ATOS [24].

2.8 WebRTC

WebRTC (Web Real-Time Communication) is an open-source project that enables real-time video, audio, and data communication between web browsers and mobile applications. It is supported by all modern browsers and ensures low-latency communication through the JavaScript APIs (Application Programming Interfaces) [25].

2.9 Data Validation

Pydantic is a Python library for data and type validation in Python. A model is a class that defines the fields of the data and their types. Untrusted data can be passed to a model, and after parsing and validation, Pydantic guarantees that the fields of the resulting model instance will conform to the field types defined on the model [26]. It is well suited for validating data formats, such as JSON.

2.10 Object Detection

Object detection is a fundamental and challenging task within the field of computer vision, and a central part of the project's system. The field has evolved considerably over the past decades. Early methods were largely based on handcrafted feature descriptors, such as the SIFT (Scale-Invariant Feature Transform) proposed by Lowe [27] and the HOG (Histogram of Oriented Gradients), introduced by Dalal and Triggs [28]. These methods offered a certain degree of robustness to variations in scale and orientation, forming the basis for classical detection pipelines.

A major advancement in this area is the introduction of the YOLO (You Only Look Once) framework, which established a new approach to object detection. Unlike earlier models, such as R-CNN (Region-based Convolutional Neural Network) [29], which rely on a sequence of steps including region proposal, classification, and post-processing, YOLO formulates object detection as a single regression problem. In this approach, both class probabilities and bounding box coordinates are predicted directly from the input image in a single forward pass [30].

A key advantage of YOLO is its ability to process the entire image during both training and inference. This global perspective enables the model to incorporate contextual information, reducing the risk of incorrectly classifying background regions. As a result, YOLO is particularly effective in complex and dynamic environments where multiple objects may be present simultaneously [30].

2.11 Threading

Threading refers to the use of multiple threads within a program to enable concurrent execution of tasks. In Python, which is the main programming language used for this project, threading is supported through the standard threading module [31], which is used in this project to handle communication, data processing, and user interaction concurrently, improving performance and responsiveness. Similarly, the Android ecosystem relies heavily on multi-threading. Android operates on a strict main thread (User Interface thread) model. To prevent the app from not responding, all intensive operations that can block the main thread, such as network calls or heavy computations, should be offloaded to background worker threads [32].

2.12 Git Hooks

Git Hooks are a way to fire off custom scripts when a certain important action occurs. There are client-side and server-side hooks. Client-side hooks are, for example, triggered when making commits and server-side hooks could be triggered by receiving pushed commits [33]. Both client and server-side hooks can be used to enforce certain policies for a repository, such as preventing non-ASCII filenames or checking commit messages [34].

3

System Design and Implementation

3.1 Project Management

The division of work and project management used the Scrum framework based on agile methodology. Both the Chalmers team and the Penn State team mostly worked on the same set of work items. However, the Penn State team was dedicated to working on the MAVLink protocol implementation for the drone adapter layer, as they have access to a MAVLink-compatible drone. Similarly, the Chalmers team implemented the DJI drone adapter.

The system consists of several different modules, which made it suitable to create Epics for each module. An Epic (or called a Milestone) is a large piece of high-level work, such as a module or working functionality of connected modules. It is a considerable amount of work that will be broken down into smaller stories. Stories are work that is manageable and is often a feature. Stories are sometimes referred to as issues. The stories were broken down so that they were atomic, at the smallest practical level. This improved the ability to work in parallel without dependencies on each other, which was crucial for large teams as well as when members were in different time zones. Stories can have different labels and priorities.

The stories were organized using a kanban board, which is "a visualization tool that shows work in progress to help identify bottlenecks and overcommitments, thereby allowing the team to optimize the workflow" [35]. When a team member started working on a story, they placed it in the *In Progress* column. After it was finished, it was moved into the *Review* column, waiting for at least one reviewer to accept it. Finally, it will be moved into the *Done* column once it is merged into the main branch. The board is shown in Figure 3.1.

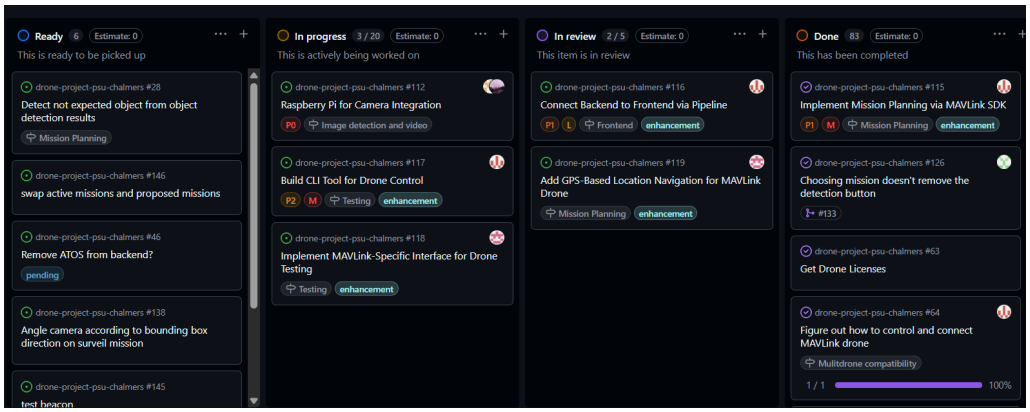

Figure 3.1: Kanban board used for this project.

3.2 Usage of AI

Throughout the project, AI (Artificial Intelligence) was used as a supportive tool. The primary usage of AI was assisting the team in understanding unfamiliar code, and exploring possible approaches to extending existing functionality. In addition, AI tools were used to review and refine the written report, for example by suggesting clearer phrasings, identifying inconsistencies, and improving overall readability. All AI-generated suggestions, whether related to code or written content, were reviewed and validated before being implemented, ensuring that responsibility for code implementations, and the final text remain with the authors. This usage pattern reflects a broader trend in the software industry, where 68% of professional software developers report using AI tools daily or weekly for development [36].

3.3 Testing and Verification

To enforce code standards and quality, the Python linter Ruff and the spell checker Codespell were run as a Git Hook (Section 2.12) each time the developer wanted to make a commit. If the toolchain failed, the commit failed, and the developer had to fix the presented issues. The code editor used (Visual Studio Code) had *Format on save* enabled for all team members, using a configuration file in the repository, leading to consistent formatting.

When a developer created a pull request, a second developer examined the proposed changes for code quality, potential bugs, and overall maintainability. The pull request required approval from the reviewer before it could be merged. If the code reviewer identified issues or suggested improvements, the original developer addressed the feedback and updated the pull request.

To avoid unnecessary review, a checklist had to be manually filled by the developer before being reviewed, and functioned as a self-reminder. These were for adding or updating tests, tests passing, linking an issue, among others.

3.3.1 Mock Drone

To test how the various components of the system interact, it was required to have one or more drones connected to the system. As access to drones was limited and inconvenient to connect, a simple mock drone was created with a Python script. The script connects to the backend and registers itself as a drone, continuously sending telemetry, streaming a predefined video file as the camera feed, and responding to all tasks according to the defined protocol. Multiple terminal windows could run the script to simulate multiple drones. This drastically facilitated the testing of system components and the system at large during the development.

3.3.2 DJI Assistant

To test the full system in a safe environment without risk of accidents, the software DJI Assistant was used to evaluate and validate the drone behavior in a virtual environment before deploying it in real-world conditions. Mission logic, navigation, and various tasks such as the spotlight or speaker can all be simulated. The camera feed is not simulated and uses the drone's real camera.

The drone is connected via a USB-C cable to a computer running the simulator. The controller and Android phone with the developed application had to be connected. However, when starting the simulation, the drone only flies in the simulator.

3.3.3 Unit Tests

For testing code during development, multiple techniques were used. Unit tests for parts of the code, such as the different Pydantic models, were created early on to continuously validate that no regressions took place and that the tests correctly showed how the data is structured.

Tests for components and subsystems, such as the drone selection, automatic mission suggestion, and object detection, were written alongside development of those features. These tests were run during the development to verify the components behaved as expected for various test cases. This made it easy to test the components worked in isolation without having to integrate them into the complete system first.

3.4 Version Control

The project used Git for version control. The project's Git repository was hosted on GitHub, the most widely used platform for code management and collaboration in both personal and professional contexts [37]. The repository is publicly available at GitHub [38]. For new features or bug fixes, developers created dedicated feature branches from the main branch. When development was completed, a pull request was initiated to merge the feature branch into main. This process follows the steps outlined in Section 3.3.

3.5 Software

The complete system, illustrated in Figure 3.2, comprises various subsystems and incorporates existing components from prior work as well as building upon them. Red represents existing components from prior work [2] and green and gray represent systems developed during this work. The Backend and Image stitching containers are connected and communicate via the Redis database. The mission planner subsystem is outlined with the dashed purple line and handles all mission logic.

The backend was aimed to be developed jointly between the two teams. The communication flow and interaction with drones is illustrated in Figure 3.3. The Chalmers team (green boxes) focused on the DJI Adapter as well as establishing a common communication protocol, while the Penn State team (blue boxes) worked on creating the MAVLink adapter in Python, along with integrating video streaming with their acquired camera. For the sake of clarity, detailed discussion of the MAVLink adapter along with their camera and video streaming are beyond the scope of this thesis. Interested readers are referred to the work by Penn State for further information [39].

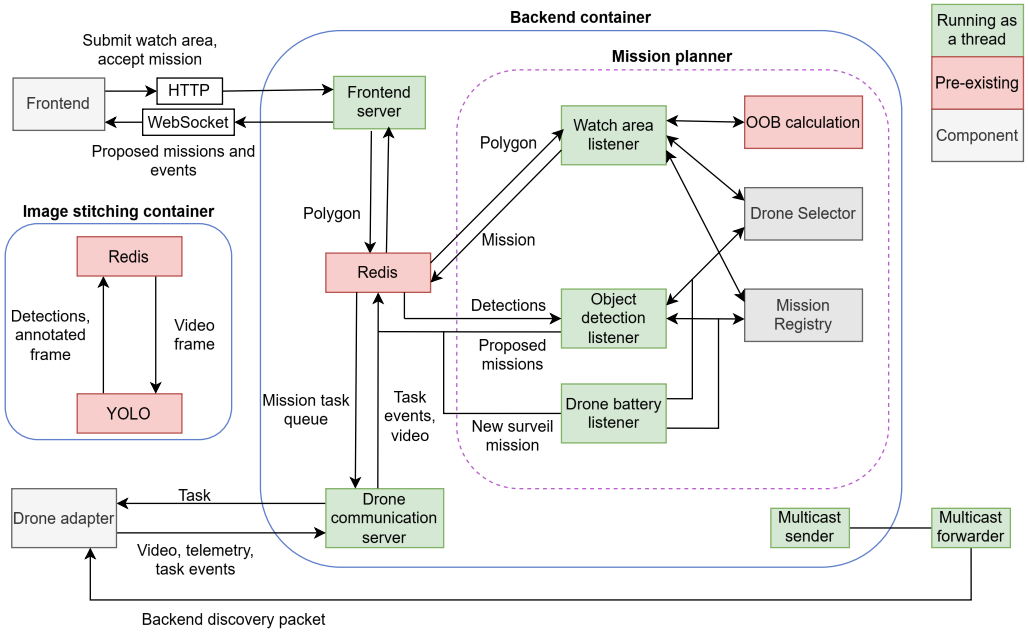

Figure 3.2: Overall system architecture.

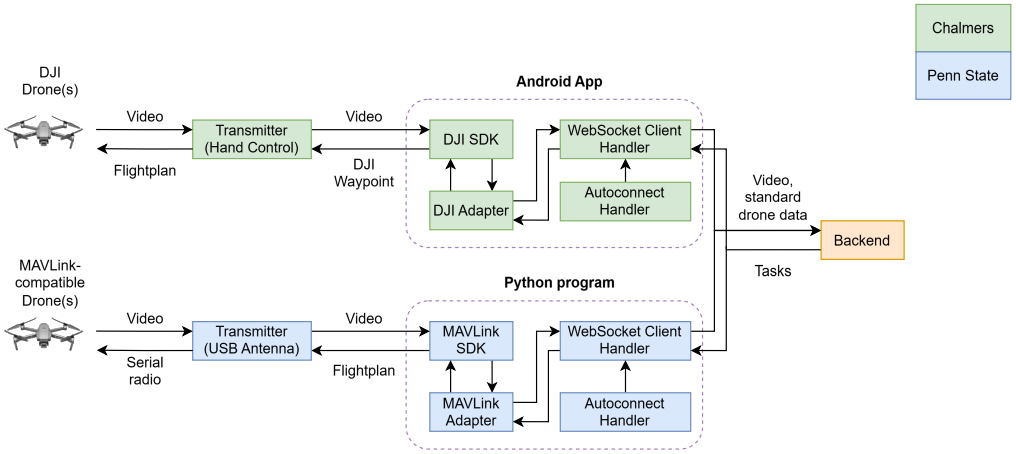

Figure 3.3: Drone communication flow and adapter implementations for DJI and MAVLink drones.

3.5.1 Docker

The architecture builds directly on the previous year's work [2], retaining Docker as the core mechanism for achieving modularity and flexible deployment (Section 2.4). The system is still organized into three distinct containers, representing the backend, the image stitching module, and the frontend. The image stitching module was kept as it contains critical logic essential to the system. While the specific image stitching functionality was not utilized this year, the module's underlying framework remains a vital component of the overall software architecture.

By encapsulating these components in separate Docker images [40], the solution enables reliable and repeatable execution across Linux AMD64 systems, independent of host-specific configurations. Communication between components is facilitated through Docker's internal networking, as in the prior work.

A Docker Compose configuration was used to coordinate the multi-container setup. This file acts as a blueprint, describing how the individual services, networks, and volumes are defined and managed within the system.

An improvement was the use of Docker Compose volumes. The prior work required rebuilding the containers when any code was changed, but with the use of volumes it is sufficient to restart the containers, facilitating rapid iteration.

3.5.2 Frontend

The frontend of the project is a web-based interface designed to monitor and control drone missions in real time. It is built using HTML, CSS, JavaScript and Bootstrap providing a responsive and user-friendly layout. The Leaflet.js library enables interactive map functionality. Communication with the backend occurs through a WebSocket connection and HTTP requests, enabling real-time interaction with the drones. The static content is served by the frontend Docker container using nginx.

The interface is divided into several key sections:

1. Live Video Feed: Displays real-time video streams from the drones, allowing operators to switch between different viewing layouts such as single drone or side-by-side display. An annotated view can also be selected, where detected objects are displayed with visual overlays. This is shown in Figure 3.4.
2. Drone status panel: Displays essential telemetry data for each connected drone, including altitude, speed, battery level, along with hardware capabilities such as camera, spotlight or speaker information. Shown in Figure 3.5.
3. Control panel: Provides operational controls such as return home and abort mission. Shown in Figure 3.5.
4. Mission Planning: Allows users to select and dispatch predefined missions generated from detections. The list of all active missions and status of each task in a mission are displayed in Figures 3.6 and 3.5.
5. Interactive Map: Implemented using Leaflet, this feature enables users to define surveillance areas by drawing polygons, which are then sent to the backend for processing and dispatching a drone. Shown in Figure 3.4.
6. User Feedback System: The interface includes visual notifications that inform the operator about the success or failure of actions such as mission dispatch or abort operations which is shown in Figure 3.5.

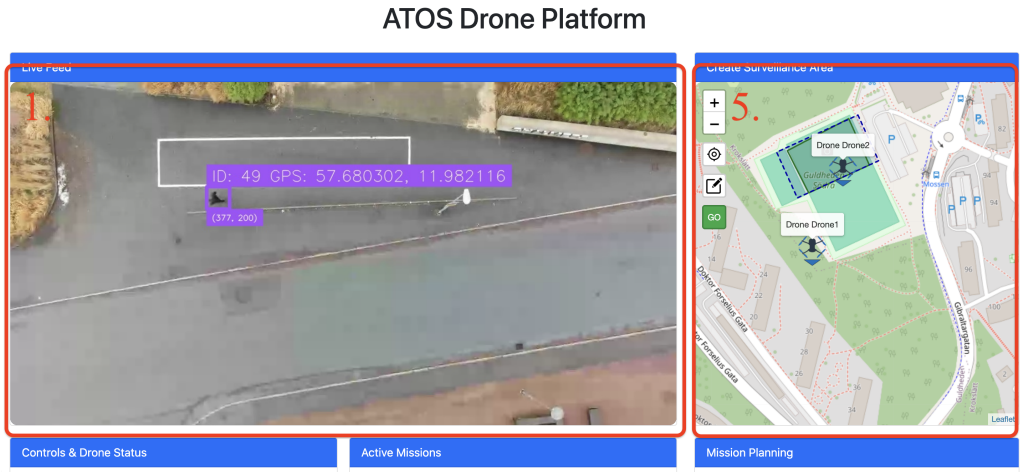

Figure 3.4: 1. Real-time video streams from drones shown with detections. 5. Interactive map for defining surveillance areas and displaying drone positions.

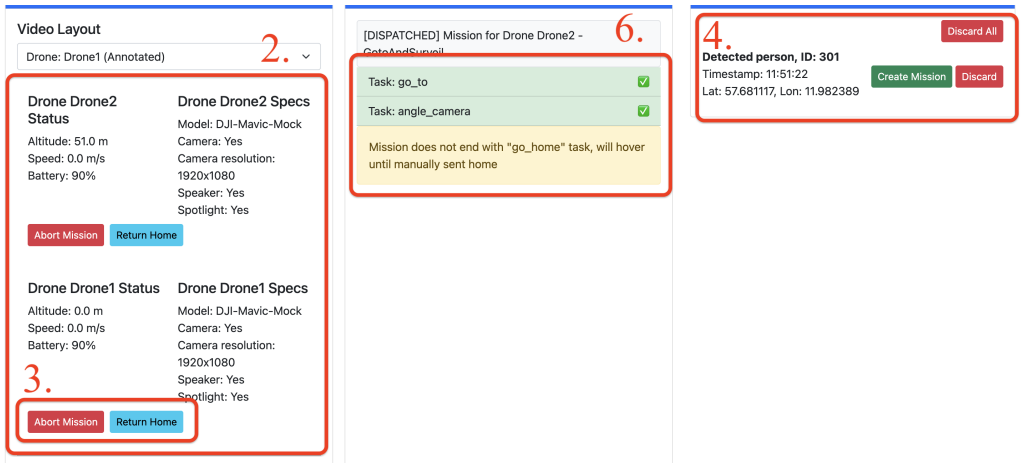

Figure 3.5: 2. Drone telemetry and capabilities.
6. Active missions, showing the status for each task.
4. List of detected objects from the drone video stream.

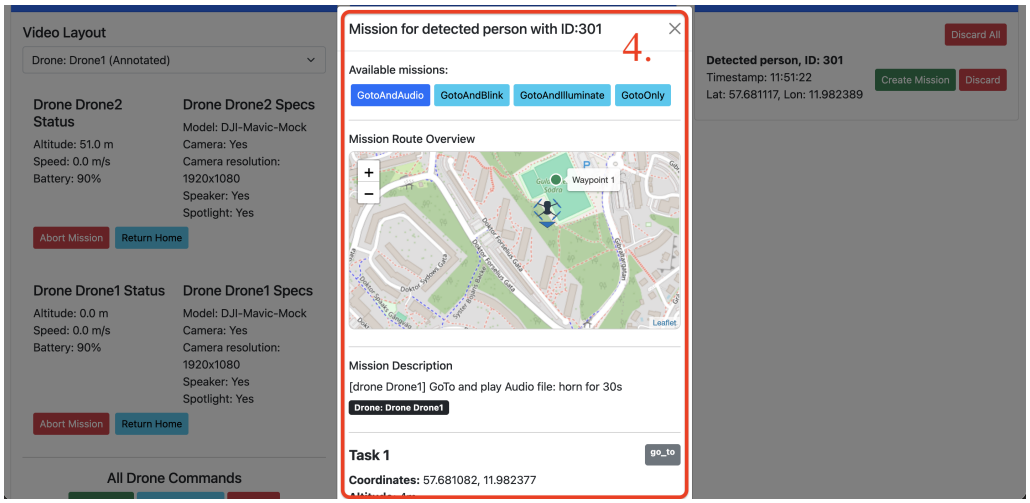

Figure 3.6: 4. Proposed missions for a detection, displaying all tasks of the chosen mission type.

Overall, the frontend serves as an intuitive command and monitoring dashboard, facilitating efficient mission management and real-time drone control through seamless communication with the backend services.

3.5.3 Backend

The backend of the ATOS Drone Platform is responsible for mission management, communication with drones and coordination of system components. It is implemented using Python.

Key functionalities of the backend include:

1. Mission Management: The backend handles the creation, storage, dispatch and cancellation of drone missions. Missions are managed through a class called *MissionRegistry*, which stores mission data and status (e.g., *Dispatched*, *Pending*, *Completed*) in Redis.
2. Drone Selection: Drones are automatically selected for missions based on various criteria, which differ for each mission. The drones are ranked by a score based on distance to destination, battery level, camera resolution among others. There are also strict filters for missions requiring certain hardware capabilities. The drone with the highest score gets selected for the mission.
3. Drone Communication: Handles drones connecting, video streams and forwards commands to the drones from other parts of the backend. Also sends the next task in a mission once the previous task has been completed
4. API Endpoints: HTTP endpoints allow the frontend to control mission execution, setting surveillance areas and retrieving mission data.
5. Message Validation: The backend uses Pydantic models to ensure that all incoming and outgoing messages follow a consistent and validated structure, improving reliability and maintainability.

Overall, the backend acts as the central control hub of the ATOS Drone Platform, ensuring reliable mission orchestration and scalable integration with drone hardware and external services.

3.5.4 Message Formats

Central to the backend is the various defined message formats for communication with drones, the frontend and between threads using Redis. These all have in common that they communicate or store text-based data, which makes JSON suitable. A shortcoming of JSON is the inherent lack of enforced structure and validated data types in the format. This introduces the constant need to manually validating individual fields, checking for their existence and types, everywhere in the code where they are read. This was one of the suggested improvements from the prior project [2].

To avoid this, the Python library Pydantic was used to create schemas, often referred to as models, for all message formats, using Python classes (Section 2.9). The construction of a new message instance enforces the correct and valid fields. The message is converted into a JSON string with the method `model_dump_json()`. Parsing a message is handled by a single function call on the model type, `model_validate_json()`. Another benefit of using Pydantic was its expressive type system, with inheritance and discriminated unions. With inheritance, all messages to drones could inherit from the same common class to share fields, for example `drone_id` is common for all messages sent to drones. Discriminated unions allow distinguishing messages by a field, in this case the `msg_type` attribute all messages have. This allows parsing a message and the corresponding model for that message type will be returned [41].

Redis Communication

To communicate between containers and threads, Redis's ability to aid with IPC (Section 2.2.2) was leveraged. Four Redis channels are used in the backend to enable asynchronous communication between different system components. The `DRONE_COMMANDS_CHANNEL` is used to publish commands from the backend to the drones, such as mission tasks or abort instructions. The `DRONE_EVENT_CHANNEL` is used to receive events and status updates from the drones, including telemetry data and task-related events. This is received by the frontend server and sent to the frontend via the WebSocket connection. The `SURVEIL_AREA_CHANNEL` is used to transmit information when a surveillance area is defined, allowing different parts of the system to access and react to it. `DETECTIONS_CHANNEL` is used to notify the backend when a new object has been detected.

All other state stored to Redis is listed in Appendix A.1.

3.5.5 Missions

A mission is a sequence of tasks to be executed by a drone. These tasks contain different parameters defining how this task is supposed to be executed.

Table 3.1: The different tasks that are chained to create a mission.

Task	Parameters	Drone action
go_to	lat, long, altitude, heading	Flies to the specified location
play_audio	audio_file, volume, duration	Plays the specified file through the speakers
led	type, duration	Activates that type of LED
spotlight	pattern, duration	Turns on and off the spotlight with a specified pattern
angle_camera	pitch, yaw, duration	Angle the camera. Duration allows for panning
hover	duration	Acts as a delay before the next task is sent to the drone
go_home		Flies back to its starting point and lands

A mission has a field named `tasks` that contains a list with combinations of the tasks listed in Table 3.1. In addition to this a mission has the additional fields listed in Table 3.2.

Table 3.2: Key-value pairs for a mission.

Key	Value
drone_id	The drone's serial number
mission_id	UUID string
mission_type	ex. GotoAndAudio
description	Verbose description to show user
tasks	List of tasks that compose the mission

All tasks sent to the Android application are low-level tasks, and the backend keeps track of the broader mission. This was done to ensure more control from the backend. If the connection is lost to the drone, no new tasks can be executed until the connection is established again.

When a mission is executed, the first task is sent from the backend to the Android application via the WebSocket connection. If a task has a specified duration, the Android application initiates the task, and after that duration the task execution is stopped, and a `task_complete` message is sent to the backend. If there are remaining tasks in the mission the next task in the list gets sent to the Android app. If a task doesn't have a specified duration, the `task_complete` message will be sent as soon as the Android application starts executing the task, and the tasks will be terminated only when the user chooses to end the mission via the frontend. The mission is considered complete either once the drone has returned home and landed or the mission has manually been aborted. At that point the drone once again becomes available to be sent on new missions.

3.5.6 Drone Selection

To find the most suitable drone for a mission, a drone selection process is used to prioritize available drones in multiple steps. The steps are: hard filters, mission-specific hardware evaluation, and multi-criteria scoring.

Hard filters

All connected drones are filtered based on the following criteria:

- Telemetry and Capabilities: The drone must have active telemetry and capabilities registered in Redis.
- Availability: The drone cannot be currently deployed on another mission.
- Battery level: The drone must have a battery level above 30%.

Mission Specific Hardware Evaluation

Based on what mission is to be executed, different capabilities are used. The drones are further filtered based on what capabilities are mandatory for that mission type, see Table 3.3.

Table 3.3: Mandatory hardware for each mission type.

Mission	Required Hardware
GoToAndSurveil	Camera
GoToAndAudio	Speaker
GoToAndBlink	LED (beacon)
GoToAndIlluminate	Spotlight
GoToOnly	None

Then a mission-specific score is calculated. The mission-specific scoring consists of three different scores: Resolution score (S_{res}), FOV score (S_{FOV}), and No hardware score (S_{NH}).

The resolution score is used to reward better image quality from the drone, and is calculated with a logarithmic scaling:

$$S_{\text{res}}(n) = a \cdot \ln(n) + b$$

where n denotes the total number of pixels, and a and b are fitted parameters with values $a = 20.094$ and $b = -246.0$. The values for a and b were determined using the scipy.optimize.curve_fit method, which employs a non-linear least squares approach to estimate the coefficients [42]. The curve was fitted onto common resolutions ranging from 480p (854 x 480) to 5.4K (6016 x 3384) with appropriately assigned values between 20-100. This is illustrated in Figure 3.7.

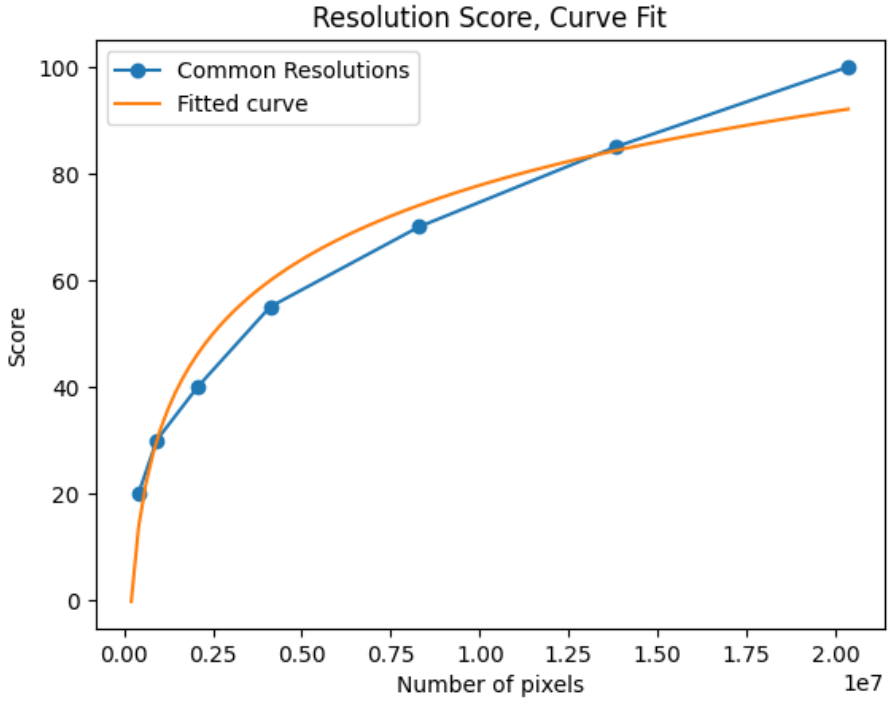

Figure 3.7: Logarithmic fit used to map camera resolution, measured in pixels, to a resolution score.

A logarithmic scaling was used since the difference between 720p and 1080p is a lot more noticeable than the difference between 4K and 5.4K. A logarithmic scaling captures this behavior.

The FOV score is used to reward a wider FOV, since this means the drone can be at a lower altitude and still cover an area. This indirectly gives better camera quality. The FOV score is calculated via a linear normalization to values between 0 and 100 using the drone's diagonal FOV and the function:

$$S_{\text{FOV}} = \min\left(\frac{FOV_{\text{actual}}}{FOV_{\text{max}}}, 1\right) \cdot 100$$

where FOV_{actual} is the drone's diagonal FOV and $FOV_{\text{max}} = 120°$, since most non-FPV (First-Person View) drones have a FOV below this value. No hardware score, S_{NH} serves the purpose of rewarding drones that are not over-equipped for the current mission and thus saving these drones for other missions. First, an OES (Over-Equipment Score) is calculated by taking the average of the scores for its equipment that is not used in this mission (Ex. S_{res} and S_{FOV}). If a drone is equipped with a speaker, beacon, or spotlight that is not used for the mission, that equipment item contributes a score of 100. If the item is used for the mission, or if the drone is not equipped with that item, its contribution is 0.

$$OES = \frac{1}{n}\sum_{i=1}^{n} S_i$$

where $n \in \mathbb{Z}^+$ is the number of scores being added.

The OES represents how over-equipped the drone is. This then gets inverted to give a final score between 0-100:

$$S_{\text{NH}} = 100 - OES$$

To calculate hardware quality (HQ), a weighted average is calculated, using the mission-specific weights, see Table 3.4:

$$HQ = S_{\text{res}}W_{\text{res}} + S_{\text{FOV}}W_{\text{FOV}} + S_{\text{NH}}W_{\text{NH}}$$

Table 3.4: Hardware quality weights for each mission type.

Mission	**Resolution (%)**	**Field of View (%)**	**No Hardware (%)**
GoToAndSurveil	30%	50%	20%
GoToAndAudio	50%	50%	0%
GoToAndBlink	50%	50%	0%
GoToAndIlluminate	50%	50%	0%
GoToOnly	0%	0%	100%

The mission types GoToAndAudio, GoToAndBlink, and GoToAndIlluminate all use 50% for both Res and FOV. This decision was made to prioritize drones with good

cameras for these mission types. When the drone goes to the detected object it will angle its camera towards the object. Having a good camera gives the user the ability to further identify the object and track where the object goes. It is also because there is no dynamic scoring for other specific equipments; The score for a piece of equipment is 100 or 0, which makes weights for these capabilities not very useful.

Multi-Criteria Scoring

The final score is given by calculating a weighted average for Battery score, Distance score and Hardware quality score.

The battery score (B) rewards drones with a high battery level and equals the drone's battery percentage. The Distance score (D), that rewards drones near the objective to minimize the travel distance, is calculated using a linear normalization and inverting the score to give higher score to the nearest drone:

$$D = \min\left(1 - \frac{D_{\text{actual}}}{D_{\max}}, 1\right) \cdot 100$$

where $D_{\max}$ is set to 1000 m since this is around the maximum distance the hardware supports.

The three scores: B, D and HQ are now gathered into one single score using a weighted average with the weights $W_{\text{B}}, W_{\text{D}}$ and W_{HQ}, see Table 3.5:

$$\text{Final Score} = B \cdot W_{\text{B}} + D \cdot W_{\text{D}} + HQ \cdot W_{\text{HQ}}$$

Table 3.5: Weights used in the final multi-criteria drone-selection score.

Attribute	**Weight (%)**
Battery level	35%
Distance	20%
Hardware quality	45%

The drone that passes all the filtering and receives the highest final Score is selected for the mission.

3.5.7 Dispatching Surveillance Drone

When the backend receives an HTTP request to surveil an area, it publishes the payload to the `SURVEIL_AREA_CHANNEL` Redis channel. A Python thread called *Surveillance area listener* listens on the same channel. When a message is received, it tries to create a GoToAndSurveil mission using the system explained in Section 3.5.6 and dispatches it, that is, if no drone is already on a surveillance mission.

Surveillance Area Calculation

An algorithm developed and tested by the prior work was used. It computes the optimal OBB (Oriented Bounding Box) for the user-specified surveillance area polygon, along with the optimal altitude to cover it [2]. Some minor modifications where made to not use hard-coded drone specifications, such as camera FOV and aspect ratio. The minimum altitude was also changed from 30 to 10 meters to allow more flexibility.

3.5.8 Object Detection

To detect objects in real-time from the drone video feeds, models such as YOLO are essential. YOLO performs direct object detection, minimizing the computational load compared to traditional network-based approaches [30]. By using YOLO, both stationary and moving objects can be detected and tracked across each drone camera stream. To enable tracking of objects detected by YOLO, the coordinates of the objects need to be calculated. This method was originally developed in the work from 2024 [6] and focuses on saving the center pixel from each bounding box. Using the drone's altitude, FOV, and GPS position, an estimated position for all objects is calculated. The model itself was also trained as part of that work.

The object detection is run in its own container, called `image_stitching`, inherited from previous years' work. It remains mostly unchanged, apart from being refactored to be dynamic and support a varying number of drones. The image stitching code remains intact, but extended video surveillance was out of scope for this project and was not integrated into the system, so it has not been tested. When a drone's telemetry and capabilities are stored to Redis for the first time, it begins streaming the video frames from Redis and runs them through the YOLO model described previously. The output of the model, a list of detections, is then stored in Redis for processing by the Object detection Listener , see Section 3.5.9.

An optimization to reduce detection latency was also added; A FIFO (First-In, First-Out) queue for frames with a max capacity. When the queue was full, the oldest frame was dropped. If not in place, every frame would be processed, resulting in unnecessary latency when the inference time is slower than the video stream. Latency testing was performed to find a balance between latency and not dropping too many frames, as each dropped frame is a missed opportunity to detect objects. A max capacity of 60 frames was deemed satisfactory for our available hardware.

3.5.9 Object Detection Listener

To provide real-time mission proposals to deter intruders, a subsystem was developed that acts on the detections generated by the YOLO model. A Python thread called *Object detection listener* listens on the `DETECTIONS_CHANNEL` Redis channel, where objects are sent once they are detected. To prevent the system from flooding the operator with redundant mission suggestions for the same object, a multi-stage filtering process is implemented. When a detection is received, it is processed through

two separate logic steps:

- Temporal Cooldown: Unique object IDs are cached and ignored for a set period of 20 seconds, ensuring that a single tracked person or vehicle does not trigger multiple alerts in rapid succession.
- Spatial Deduplication: If an object does not have a persistent ID, as the detection algorithm can be inconsistent, the system compares its GPS coordinates against recent events. If a detection of the same type occurs within five meters of a previous event, it is treated as the same object and discarded.

Spatial Constraints

The system utilizes the user-specified surveillance area polygon to restrict responses to that zone. To determine if a detection warrants a mission, the system checks if the detection's GPS position falls within that polygon. This ensures that missions are not created for events occurring outside the drone's operational boundaries, in case the calculated position from a detection is erroneous.

Mission Proposal

When a valid detection, such as a person or vehicle, is confirmed within the surveillance area, the system tries to create a mission of each type with the best available drone, using the system explained in Section 3.5.6. These missions are sent to the frontend via the `DRONE_EVENT_CHANNEL` Redis channel to be further evaluated by the operator for which mission should be dispatched, or discarded.
To ensure optimal camera angles and safety, target coordinates are not set directly on the object's position. Instead, the system calculates an offset, typically three meters away from the object and at an altitude of three meters to provide the best vantage point while maintaining a safe distance from the subject.

3.5.10 Auto Surveillance Drone Swap

Drone batteries are often limited in capacity and constrained to 15 to 35 minutes of flight time [43], therefore a single drone cannot surveil an area for an extended period of time. To allow longer surveillance periods, a system to swap the surveilling drone was implemented. A Python thread called *Drone battery listener* monitors the connected drone's battery levels. If a drone's battery level is less than or equal to a predefined value, set to 20%, the system tries to select a new drone for the same surveillance area. If one is found, the new drone gets assigned the mission, but is not yet dispatched, and the mission status is set to *Waiting*. To avoid a potential drone collision, the original drone's mission gets aborted and assigned a GoHome mission, which is dispatched. The original drone goes home even if no replacement drone was found, due to the low battery level. Once the drone_communication subsystem receives a `task_completed` message for the `go_home` task, the new surveillance drone's mission is dispatched, if it was created. The operator could then replace or charge the original drone's battery, and the swapping would eventually repeat itself.

3.5.11 Android Application

An Android device with an application using the DJI SDK is connected to the remote controller for the drone, as described in Section 2.1.1. The application is built upon the work from 2024 and 2025 [6] [2]. The application has support for limited manual control of the drone, as well as a live video feed. However, the main purpose is not to be used by the operator, but rather to implement the defined drone message protocol and execute tasks using the DJI SDK. The application structure and communication flow is illustrated in Figure 3.8. The application receives tasks from the backend via a WebSocket connection and handles all the lower-level and finer parts of drone control. Examples include automatically ascending to a safe altitude after takeoff, navigating towards specified coordinates using suitable flight parameters and returning home by retracing the recorded flight path using SDK functionality. The application itself has no concept of a complete mission and is only aware of the currently active task.

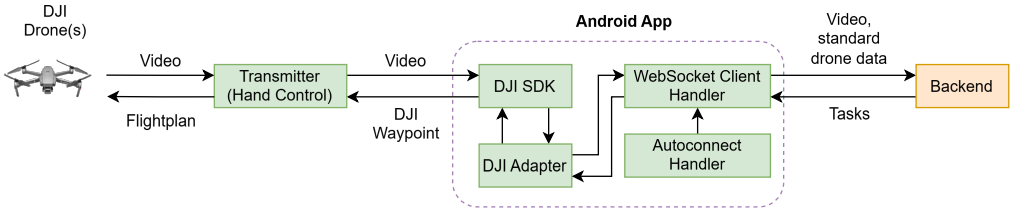

Figure 3.8: Communication flow between the DJI drone, transmitter, Android app, and backend.

Automatic Connection

The prior work had the user manually enter the IP address and port for the backend every time the application was started. It was also bothersome having to find the IP and port, especially as the local IP can change periodically due to Dynamic Host Configuration Protocol [44]. To improve the user experience and make drone connection faster, auto connect was added leveraging administrative scoped IP multicast, see Section 2.6.1. The backend sends a multicast packet every second containing its name (to distinguish from other potential backends on the network), IP address, and port stored as a JSON object. The Android phone might receive many kinds of multicast packets, so to only handle and try to parse the JSON data for the discovery packets, the arbitrary string `CTH` is prepended to the JSON string. The receiver then checks that the first three characters match this string before parsing the rest of the packet data, otherwise the packet is ignored. This approach is common for file formats, and this string is called a magic number or magic string [45].

During development it was noticed that multicast packets were not being received by other devices. This was due to Docker not being able to forward them onto the host computer's network, they were effectively being contained inside the container. To work around this, following two steps were implemented:

1. The backend sends the JSON data in a UDP packet to `host.docker.internal`

and port `50000`. Docker's network stack resolves this to the IP address of the host computer. The IP address to multicast had to be specified beforehand, as the host's interface for receiving traffic could not be decided from within the container. The environmental variable `HOST_IP` is set by running the Python script `export_host_ip` on the host computer before starting the containers. The Docker compose file then forwards this value to the container's environmental variables.

2. A Python script called `forward_multicast` is continuously running on the host computer listening on the same port. It constructs a multicast packet with the received data and sends it to the predefined administratively scoped multicast group with IP address `239.255.42.99` that the Android application listens on.

The Android application utilizes APIs from the Android SDK to join the multicast group upon application startup, to receive the discovery packets. When a packet arrives the magic string gets checked as described previously. In case the multicast packets don't arrive for any reason, the auto connect system is designed to have a prioritized multi-layer connection strategy. The system periodically evaluates available connection options and selects one based on priority. The highest priority is manual override, where a user-specified IP address and port are always preferred. If no manual configuration is set, the system uses multicast-based discovery, selecting the first backend from the list of discovered services. If multicast discovery fails, a fallback mechanism uses the last known connection, which is stored locally on the device and reused when needed.

4

Experimental Validation and Results

This chapter presents the experimental validation and results obtained throughout the project. The system is evaluated through simulation-based testing, off-site field testing and final on-site testing at AstaZero's facility.

A number of tests were conducted to evaluate the project's requirements. Each test is presented using the following sections: Setup, Metrics, Outcome, and Results. Setup describes the testing scenario, while Metrics specifies the requirements and desires against which the test is evaluated. All requirements and desires are taken from the total requirement table which can be found in Appendix B.1. Outcome details the observations from the test, and finally, the Results section provides a comparison between the Metrics and the Outcome to determine if the criteria were met.

4.1 Simulation Testing

To test the system in a safe environment without risking accidents, simulation-based testing was done to evaluate and validate the drone behavior in a virtual environment prior to real-world deployment. This was done through DJI Assistant, calculation tests, and the use of mock drones.

4.1.1 Mock Drone Test

This test case focuses on verifying system connectivity, communication and frontend functions. Mock drones are used to ensure that messages, telemetry data, and video streams are correctly transmitted between system modules without flying the drone.

Setup

The Docker environment was running with all Docker components active. Multiple mock drones were instantiated to simulate drone system interaction and telemetry data. The mock drones were configured with different specifications, telemetry data, and battery levels to simulate different test conditions. Each mock drone was also equipped with a pre-recorded video stream to simulate object detection scenarios.

Metrics

The requirements evaluated using the mock drone setup are presented in Table 4.1. Since the test was performed as an integrated system test, all listed requirements were evaluated during the same test run and under the same conditions.

Table 4.1: Mock drone requirements and desires.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
1. Map				
1.1 Area				
1.1.1	The user shall be able to create a rectangular surveillance area.	Mock drone	A rectangular surveillance area can be created successfully on the map.	R
1.1.2	The user shall be able to create a polygonal surveillance area.	Mock drone	A polygon with arbitrary shape and number of edges can be created in the interactive map.	D
1.2 Map Update				
1.2.1	The user shall be able to create a new area	Mock drone	When creating a new area the points defining the selected area updates and overwrites the previous values.	R
1.2.2	The system shall save the last area on page refresh.	Mock drone	The previously created area remains visible after refreshing the page.	D
1.3 Live Position				
1.3.1	The system shall display the drone position on the map.	Mock drone	The drone position is continuously updated and shown correctly on the map with a precision of 3 m.	D
1.3.2	The system shall display users position on the map.	Mock drone	The users position is displayed with a precision of 5 m.	D
2. Missions				
2.2 Mission Control				
2.2.3	The system shall generate mission proposals upon object detection.	Mock drone	A mission proposal is generated automatically after object detection.	R
3. Information				

ID	Requirement	Validation Method	Acceptance Criteria	R/D
3.1	The system shall display battery level.	Mock drone	The battery level is displayed correctly for connected drones.	R
3.2	The system shall display telemetry information.	Mock drone	Lat, long, altitude and speed is displayed for all connected drones and continuously updated.	D
3.3	The system shall display drone specifications.	Mock drone	All equipment and capabilities for all connected drones are displayed on the frontend.	D
3.5	The system shall display all connected drones.	Mock drone	All connected drones are listed successfully on the frontend.	R
4. Video Stream				
4.1.1	The user shall be able to switch between connected drones.	Mock drone	The video stream changes successfully when selecting another drone.	R
4.1.2	The user shall be able to switch between normal and annotated video views.	Mock drone	The frontend switches successfully between normal and annotated video views.	D

Outcome

The system allowed users to create polygonal surveillance areas directly on the map interface, as shown in Figures 4.1, 4.2, and 4.3. The drones were also displayed on the map, allowing users to monitor their positions in relation to the defined surveillance area. Existing polygons could be redrawn, or new polygons could be created, until the intended surveillance area was achieved. The most recently defined area also persisted after a page refresh, confirming that the area was stored correctly.

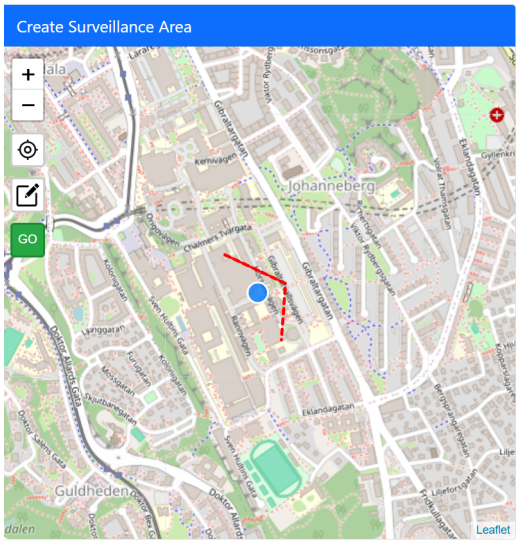

Figure 4.1: Drawing polygon on the map.

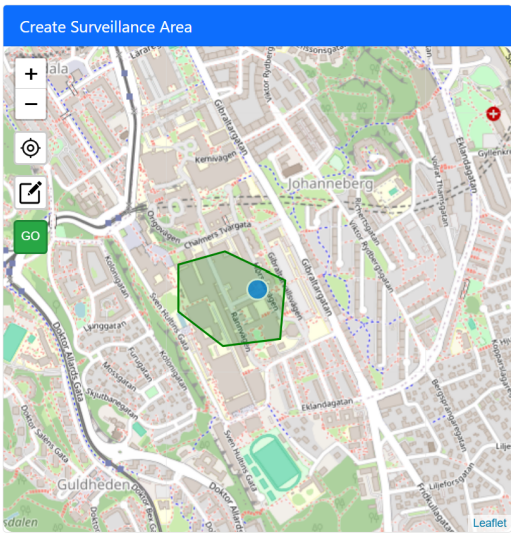

Figure 4.2: Completed polygon surveillance area.

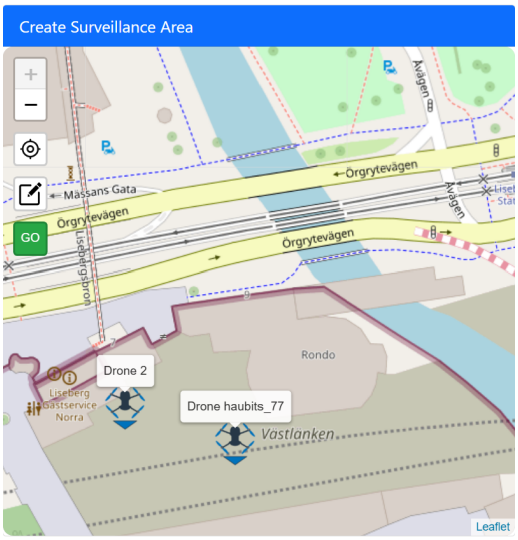

Figure 4.3: Drone positions displayed on the map.

The object detection triggers a mission that appears on the frontend, shown in Figure 4.4. Different missions are proposed to the operator depending on the available drone specifications, illustrated in Figure 4.5.

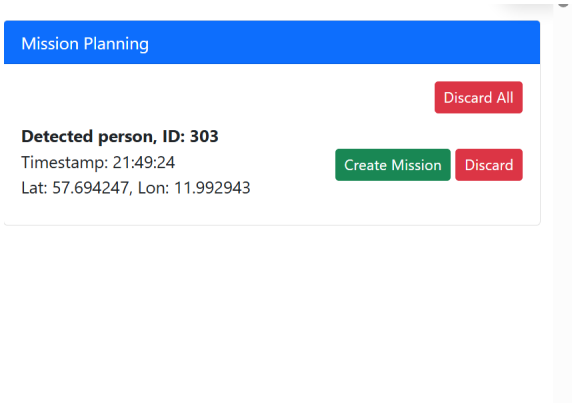

Figure 4.4: Object detections trigger mission creation.

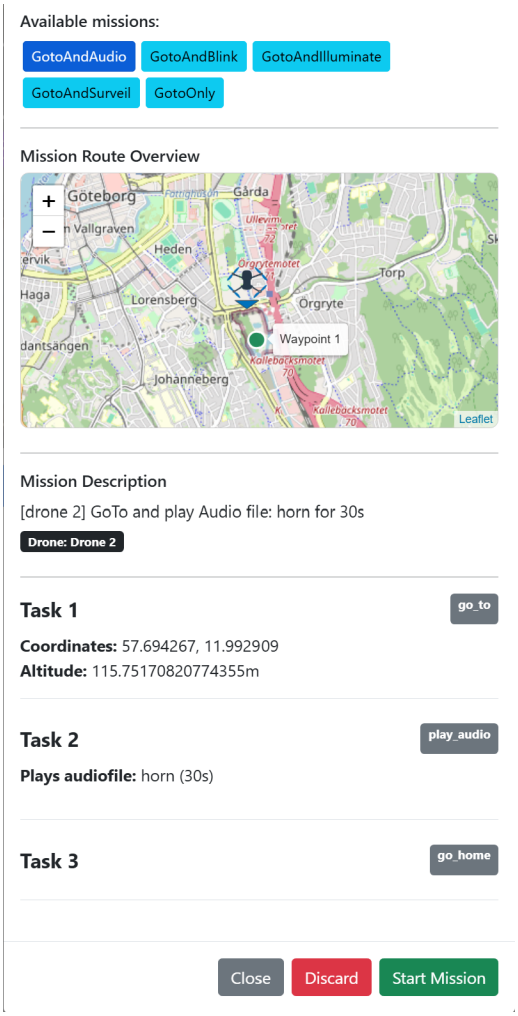

Figure 4.5: The proposed missions, depending on the drone capabilities.

The frontend allows users to view all connected drones, including their telemetry, battery levels, and specifications which can be seen in Figure 4.6.

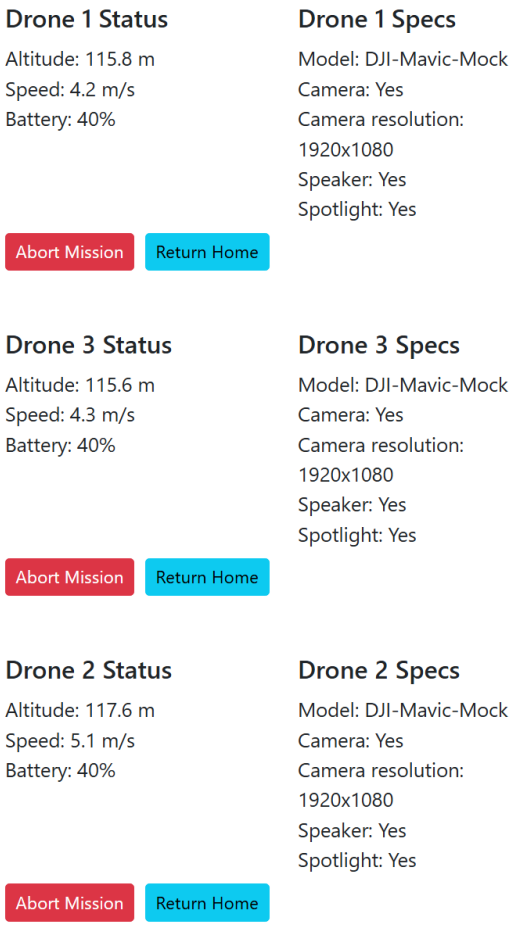

Figure 4.6: Connected drones displayed on the frontend with their respective information.

The frontend allows users to switch between different drone video streams. Operators can also choose whether to use the annotated or normal view which is illustrated in Figure 4.7.

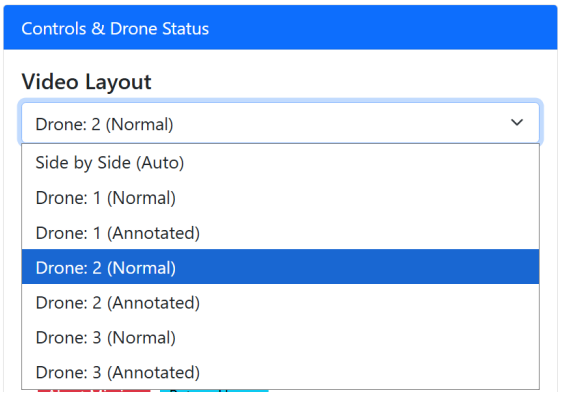

Figure 4.7: Frontend interface for switching between different drones and views.

Results

All requirements and desires evaluated in Table 4.1 were fulfilled, except Desire 1.3.2. Although the user position was displayed on the map, the positional precision exceeded the acceptance criterion of 5 m. Desire 1.3.2 was therefore considered not fulfilled.

4.1.2 Auto-connect Test with DJI Simulation

This case tests the connection time when using the auto-connect functionality with the help of DJI simulation. This feature is essential for a plug-and-play compatible system.

Setup

The Docker environment was active, with all system components running and connected to the same local network. A DJI Mavic 2 drone was connected to a computer via USB-C. The DJI Assistant simulator was used to simulate GPS connectivity during indoor testing. An Android phone running the application was connected to the remote controller via USB.

The Android app was started on the phone. When the application interface became visible, a stopwatch was started. The auto-connect functionality initialized automatically, after which the drone telemetry and capabilities were retrieved and sent to the backend. The stopwatch was stopped once the drone specifications became visible on the frontend. The test was repeated six times.

Metrics

The test is evaluated against Requirement 7.1, in Table 4.4

Table 4.2: Connection requirement used for the autoconnect test.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
7. Connection				
7.1	The drone shall automatically register to the backend when started.	DJI simulation	Drone telemetry and specifications are displayed on the frontend within 10 seconds after the drone has started.	R

Outcome

The auto-connect system successfully identified the backend via IP multicast in all six trials, demonstrating the reliability of the discovery protocol. Once the connection was established, telemetry data began populating the frontend dashboard with negligible latency. As shown in Table 4.3, the connection times averaged approximately 6 seconds. The slightly higher duration for the first trial (7.19 s) is possibly attributed to initial network service discovery, whereas subsequent trials benefited from existing network routing information.

Table 4.3: Measured times from starting the application until the drone is shown on the frontend.

Test	$\mathbf{\Delta t}(s)$
1	7.19
2	5.53
3	5.78
4	5.25
5	6.36
6	5.81
Avg	5.99

Results

The experimental results yielded an average connection time of 5.99 seconds, representing a 40% margin relative to the 10-second threshold defined in Requirement 7.1. Since all six trials were completed within the specified limit, Requirement 7.1 is considered fulfilled. The consistency of these results confirms that the automated registration logic is robust enough for operational use at AstaZero.

4.1.3 Manual Connection Test with DJI Simulation

This case tests that the operator is able to enter an IP address and manually connect to the backend if the IP multicast does not work.

Setup

The Docker environment was active, with all system components running, except the multicast sender, and connected to the same local network. A DJI Mavic 2 drone was connected to a computer via USB-C. The DJI Assistant simulator was used to simulate GPS connectivity during indoor testing. An Android phone running the application was connected to the remote controller via USB.

The Android app was started on the phone and the IP address for the backend was manually entered, and the connect button was pressed. The test was repeated six times.

Metrics

The test is evaluated against Requirement 7.2, in Table 4.4

Table 4.4: Connection requirement used for the manual connection test.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
7. Connection				
7.2	The user shall be able to manually register a drone to the backend through the Android application.	DJI simulation	A drone can be successfully registered to the backend through manual user input of port and IP-address in the Android app.	R

Outcome

The Android app successfully connected to the backend in all six tests, verified by the drone appearing on the frontend as well as the Android app logging the successfull connection.

Results

The system achieved a 100% success rate for manual connections across all trials. These results confirm that Requirement 7.2 is fully satisfied, providing a robust and reliable alternative for drone registration in environments where automated discovery via IP multicast may be restricted or non-functional.

4.1.4 Latency Test with DJI Simulation

This test case evaluates the latency of the video stream between the drone and the frontend interface, including both the regular and annotated video views.

Setup

One DJI Mavic 2 drone with a camera was connected to a computer via a USB-C cable. The DJI Assistant simulator was running in order to simulate GPS-connection while being indoors. The controller and Android application were powered on, with the Android device connected to the backend. The frontend was displayed on the left half of the computer screen and a stopwatch was displayed on the right half of the screen. The drone was placed so that the camera records the stopwatch and streamed this video to the frontend, see Figure 4.8. The screen was recorded for a duration of 30 seconds, capturing both the stopwatch and the streamed video of the stopwatch. A frame was sampled every five seconds from the screen recording. The latency is the difference between the actual stopwatch and the streamed video of the stopwatch. This gives six data points of latency. The latency is calculated as the average of these data points. The same test was run for both the regular and the annotated video. The backend was running on a computer with an AMD Ryzen 5700U CPU.

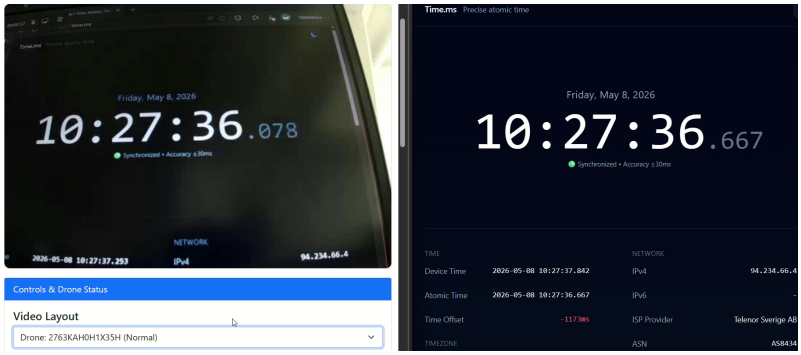

Figure 4.8: Screenshot of the latency test for the regular video feed.

Metrics

The test is evaluated against Desire 4.1.3 and Desire 4.1.4 presented in Table 4.5.

Table 4.5: Requirements for video latency in the DJI simulation environment.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
4.1.3	The system shall provide real-time video streaming.	DJI simulation	Video latency shall not exceed 2 seconds for the regular view.	D
4.1.4	The system shall provide real-time video streaming with the annotated view.	DJI simulation	Video latency shall not exceed 3 seconds for the annotated view.	D

Outcome

The measured latency data for both regular and annotated video streams is presented in Table 4.6. The regular video stream exhibited an average latency of approximately 0.97 seconds, though individual data points showed significant variance, likely due to fluctuations in the local network throughput. In contrast, the annotated video stream showed a more consistent but higher average latency of 3.51 seconds. This suggests that the processing bottleneck is constant and primarily governed by the inference time of the object detection model rather than network instability.

Table 4.6: Measurements of video latency over a 30 seconds period for regular and annotated video stream.

Test	Regular video (s)	Annotated video (s)
1	0.736	3.606
2	1.929	3.531
3	1.445	3.550
4	0.549	3.390
5	0.643	3.437
6	0.515	3.550
Avg	0.9695	3.510

Results

Based on the experimental data, Desire 4.1.3 was successfully met, as the regular video latency remained well below the 2 seconds threshold. However, Desire 4.1.4 was not fulfilled, with the annotated stream exceeding the 3 seconds limit by 0.51 seconds. As the backend was running on a CPU-bound environment (AMD Ryzen 5700U) without hardware acceleration for the YOLO model, the additional latency is attributed to the computational cost of real-time image processing and bounding box overlay.

4.1.5 Surveillance Area Calculation Test

The calculations for the surveillance area is a fundamental part of the surveillance mission, as it directly determines where the drone is positioned and at what altitude it operates. An incorrect calculation would result in the drone not covering the entire defined area, which could potentially leave intruders undetected. The prior work tested the implementation, but their algorithm was only tested for the FOV and aspect ratio of a DJI Mavic 2 [2]. It is therefore critical to verify that the algorithm produces correct and consistent results across a variety of polygon shapes, FOV and aspect ratios before relying on it in a real-world environment.

Setup

A script was developed to randomly generate between 2 and 10 coordinates within a circle boundary, 5 to 10 meters from the origin. A set seed was used to allow

reproducibility. The FOV and aspect ratio were randomly generated within the interval $[65, 155]$ and $[1.1, 2.3]$ respectively. The drone coverage area was subsequently calculated using the convex hull algorithm. The script was executed 100 times, producing 100 graphs each visualizing the input coordinates, the computed OBB, and the resulting drone coverage area. The test script can be found in file `convex_hull_visiual_test.py` in the repository on GitHub [38].

Metrics

The test is evaluated against Requirement 6.1 presented in Table 4.7. A coverage area is considered optimal if the longest side of the drone coverage area is aligned with the longest side of the OBB, thereby minimizing the total coverage area. A graph exhibiting an optimal coverage area is counted as a successful calculation.

Table 4.7: Requirement for OBB calculation and camera coverage.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
6.1	The system shall calculate the correct OBB and camera orientation.	Surveillance area calculation script.	The calculated OBB and camera coverage shall be correct at least 90% of the time.	R

Outcome

The 100 generated graphs were visually inspected and classified according to the criteria defined in section 4.1.5. Of the 100 graphs, 90 were classified as successful, as illustrated in Figures 4.9, while 10 were classified as unsuccessful, as illustrated in Figures 4.10. This corresponds to a success rate of 90%.

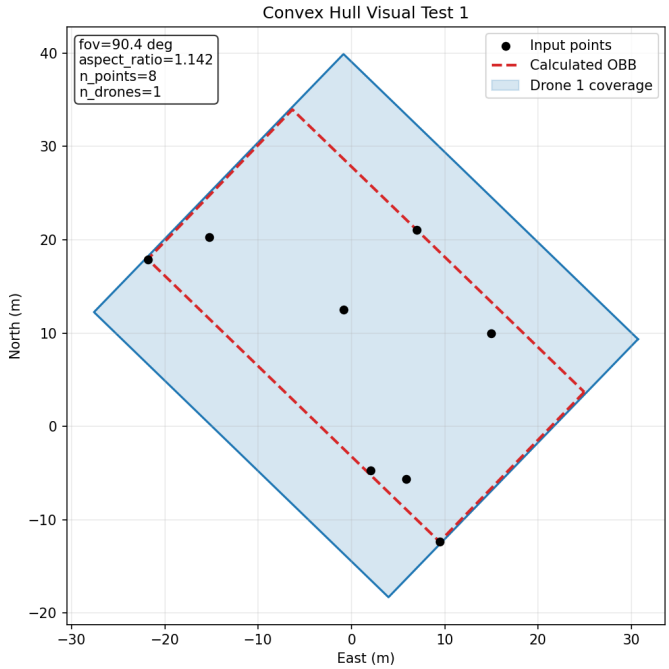

Figure 4.9: Example of a successful coverage calculation

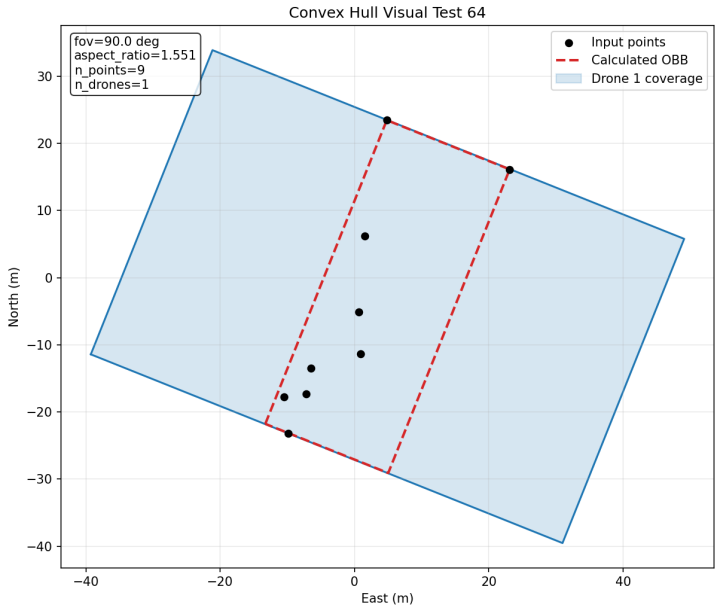

Figure 4.10: Example of a failed coverage calcuation

Results

The algorithm achieved a success rate of 90%, meeting the threshold defined in Requirement 6.1. The calculations for the surveillance area can therefore be considered sufficiently reliable for use in a real-world surveillance scenario. In the unsuccessful cases, the OBB was correctly calculated, however, the drone coverage area was rotated 90 degrees, resulting in a suboptimal coverage orientation.

4.1.6 Drone Selector Test

The drone selector module is tested with a script registering mock drones with the same telemetry and capabilities, apart from isolated attributes that vary between test cases. These attributes are the GPS position, battery level, camera quality and additional connected equipment. These attributes should make the drone selector choose one predefined drone.

Setup

The drone selector was tested in five different cases:

1. **Proximity:** A drone was selected for a GotoOnly mission. For the test to pass the drone closest to the objective has to be chosen.
2. **Battery:** One of the drone's battery level is set to 10%. Then a drone is selected for a GotoOnly mission. For the test to pass the drone with low battery level has to be filtered out.
3. **Camera Quality:** One of the drones' camera resolution is set to 4K while the other drones have 1080p. A drone is selected for a GotoAndSurveil mission. For the test to pass, the drone with the highest resolution has to be chosen.
4. **Hardware Missions:** For each of the mission types (GotoAndAudio, GotoAndIlluminate, GotoAndBlink, GotoAndSurveil) only one of the drones gets equipped with the required hardware. For the tests to pass, the drone equipped with the required hardware has to be chosen.
5. **No hardware:** One of the drones has no additional equipment. A drone is selected for a GotoOnly mission. For the test to pass the drone without any equipment has to be selected.

The test script can be found in file `drone_select_tests.py` in the Repo [38].

Metrics

This test is subject to Requirement 2.2.5, presented in Table 4.8. For this requirement to be satisfied all the tests must be passed, which means that the script should choose the most optimal drone for each case.

Table 4.8: Requirement for autonomous drone selection logic.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
2.2.5	The system shall select the best drone for the mission.	Drone select test script.	The intended drone is selected for a set of cases with different mission types and drone specifications.	R

Outcome

The drone selector module correctly chose the intended drone for each of the test scenarios. In each case, the system prioritized the intended attribute, such as proximity or hardware capability, while maintaining the hard filters.

Results

Requirement 2.2.5 is considered fulfilled, as the system consistently identified the optimal drone for each mission profile. Although the test is somewhat limited in its scope and primarily tests one attribute at a time, it proves the system capable of handling the most important selection scenarios.

4.2 Real-world Testing

Following the results in the simulated environment, real-world field tests were initiated. These tests were used to evaluate the system's performance and confirm that it was ready for the final on-site testing at AstaZero.

4.2.1 Off-site Surveillance Test

This test case was designed to validate video coverage of the selected area and the object detection of a person in a real-world environment. The tests were conducted at the Guldheden Södra football field, a large open area near Chalmers. Although the field lacked the asphalt surfaces typical of the AstaZero proving grounds, it provided a safe and accessible location to verify the system before the final test.

Setup

One DJI Mavic 2 drone, equipped with a camera was sent to surveil a specific area of a smaller football field. A person then entered the frame. The test was repeated five times.

Metrics

The tests success is compared against the requirements in Table 4.9.

Table 4.9: Person detection and area coverage requirements.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
5.2	The system shall detect people in the video stream.	Off-site testing	People shall be detected within 3 seconds from an altitude between 25–35 m at least 80% of the time.	D
6.2	The system shall translate polygons into drone positions.	Off-site testing	The drone shall cover the entire polygon correctly at least 90% of the time.	R

Outcome

The experimental data for person detection, conducted at an altitude of 30 m, is summarized in Table 4.10. The system successfully detected a person in only two out

of five trials, with detection times of 6 s and 2 s respectively. In the remaining three trials, the person remained within the frame for the duration of the test without triggering a detection event.

Table 4.10: Measured times until the person was detected. "-" indicates the person was not detected at all.

Test	t(s)
1	-
2	-
3	6
4	-
5	2

In contrast, the spatial translation logic demonstrated high reliability. In all five trials, the drone correctly positioned itself such that the video feed aligned with the boundaries of the polygon defined in the interactive map. Figure 4.11 illustrates the successful coverage of the Guldheden Södra football field, confirming that the camera's coverage matched the user-defined surveillance area.

Figure 4.11: Screenshot of the video feed covering the area selected in the interactive map and detecting two persons.

Results

The field tests yielded mixed results. Requirement 6.2 was fully met, with a 100% success rate in translating user-defined polygons into accurate drone positioning and area coverage. However, Desire 5.2 was not satisfied, as the person detection success rate (20%) fell significantly below the 80% acceptance threshold.

The poor detection performance might be due to the fact that the test was conducted on a grassy area while the YOLO model was primarily trained on asphalt. Additionally, the small pixel footprint of a human at 30 m altitude, combined with the lack of high computational power for faster frame-processing, might have contributed to the missed detections. These environmental and computational constraints are further analyzed in Chapter 5.

4.2.2 Mission Safety Test

To verify that the system was safe to operate, this test focused on whether the user could abort tasks and send the drone back to its starting position. To reduce the risk of collisions with other drones or objects during landing, it was important that the drone landed near its starting position.

Setup

The test was conducted using one DJI Mavic 2 drone. It was placed on a ground marker and assigned a `GotoAndSurveil` mission. During flight toward the objective, the operator selected `Abort Mission` on the frontend, followed by `Return Home`. After landing, the distance from the takeoff position was measured. The test was repeated five times. Figures 4.12 and 4.13 show the drone before takeoff and after landing, respectively.

Figure 4.12: Drone position before takeoff.

Figure 4.13: Drone position after landing.

Metrics

The landing precision was evaluated against Requirement 2.1.2, shown in Table 4.11, which specified that the drone should land no more than 1 m from its takeoff position. This criterion had to be met in all five trials to satisfy the requirement.

The mission aborting functionality was evaluated against Requirement 2.1.1 in Table 4.11. The mission had to be successfully aborted and the drone had to land successfully in all five trials to satisfy the requirement.

Table 4.11: Selected safety requirements.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
2.1.1	The user shall be able to abort a mission.	Off-site safety testing	The active mission is cancelled successfully on users command.	R
2.1.2	The user shall be able to return the drone to its original position.	Off-site safety testing	The drone returns to its takeoff position with a precision of 1 m when the user activates the Return Home command.	R

Outcome

The experimental data for the mission safety tests is presented in Table 4.12. In all five trials, the system demonstrated immediate response to the `Abort Mission` command, aborting the active task and returning to its take-off location as soon `Return Home` command was executed. The measured landing offsets were remarkably low, ranging from 0.02 m to 0.25 m. This high level of precision indicates that the DJI flight controller's internal GPS and visual positioning systems were effectively utilized by the Android application's safety logic.

Table 4.12: Measured distances from the drone's original position after the drone has landed.

Test	Distance (m)
1	0.07
2	0.08
3	0.15
4	0.02
5	0.25

In all five trials, the mission was aborted successfully and the drone landed safely.

Results

Requirements 2.1.1 and 2.1.2 were both fully fulfilled. The system successfully demonstrated the ability to override autonomous missions via manual operator intervention in 100% of the trials. Furthermore, the landing precision significantly exceeded the acceptance criteria; even the least accurate landing (0.25 m) provided a 75% safety margin relative to the 1 m threshold. These results confirm that the system can be safely operated in environments where landing space is constrained or where multiple drones are deployed in close proximity.

4.2.3 Off-site Intercept Drone Mission Test

This test case was designed to validate the execution of the various types of intercept missions. The tests were conducted at the Guldheden Södra football field.

Setup

One DJI Mavic 2 drone equipped with a camera was sent to surveil a specific area of a smaller football field. A second DJI Mavic 2 drone was connected three times, each time with different attached equipment: a beacon, a spotlight, and a speaker. In each trial, a person entered the video frame, and the operator selected the mission corresponding to the attached equipment: `GotoAndBlink`, `GotoAndIlluminate`, or `GotoAndAudio`.

Metrics

The test results are compared against the requirements in Table 4.13.

Table 4.13: Requirements for interception missions.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
2.2.7	The drone can execute a GotoAndBlink mission.	Off-site testing	The beacon is observed blinking during mission execution.	D
2.2.8	The drone can execute a GotoAndIlluminate mission.	Off-site testing	The spotlight is observed illuminating the area during mission execution.	D
2.2.9	The drone can execute a GotoAndAudio mission.	Off-site testing	Audio is clearly audible through the drone speaker when the corresponding action is executed.	D

Outcome

Each of the three interception mission types were executed successfully under field conditions. In all trials, the secondary drone autonomously navigated to the coordinates of the detected person. Upon arrival, the system correctly triggered the corresponding hardware payload: the beacon maintained a high-visibility blink pattern for the `GotoAndBlink` mission, the spotlight provided clear illumination of the subject for `GotoAndIlluminate`, and the speaker delivered a pre-recorded audio warning for `GotoAndAudio`.

Results

Desires 2.2.7, 2.2.8, and 2.2.9 were all successfully fulfilled. The test validated functionality of the automated mission suggestion module, the hardware specific commands in the Android app and the GUI (Graphical User Interface) interactions to choose and start an interception mission.

4.3 On-site Test at AstaZero

The final test at AstaZero was conducted to evaluate the complete system under conditions representative of the intended operating area. Although many subsystems had already been tested and verified at Guldheden, certain results had to be validated on asphalt to reflect the conditions of the final test site. This ensured that the integrated system functioned correctly in its intended operating environment.

4.3.1 Vehicle Detection Test

This test evaluates the system's ability to identify vehicles in a real-world environment using the on-site facilities at AstaZero. The objective was to investigate whether there was a noticeable difference in the object detection model performance compared to when smaller targets entered the surveillance area.

Setup

One DJI Mavic 2 drone equipped with a camera was used to monitor a large asphalt area at the AstaZero test site. The drone was assigned the area using the interactive map and its polygon functionality. During each trial, a car entered the camera frame. The test was repeated five times.

Metrics

This test was measured by Desire 5.1, shown in Table 4.14. The time was started as soon as the car had entered the frame entirely.

Table 4.14: Requirement for vehicle detection performance during on-site testing.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
5.1	The system shall detect cars in the video stream.	On-site testing	Cars shall be detected within 2 seconds from an altitude between 25–35 m at least 80% of the time.	D

Outcome

The experimental results for on-site vehicle detection are detailed in Table 4.15. In three out of five trials (60%), the system demonstrated high performance, detecting the car before it had even fully entered the camera's FOV as seen in Figure 4.14. However, the system also exhibited inconsistency: in one trial, detection was delayed to 2.5 seconds, and in another, the vehicle was not detected at all. These results suggest that while the YOLO model is capable of fast vehicle identification on asphalt, its reliability is sensitive to varying factors such as the car's angle relative to the camera and lighting conditions or fluctuations in video stream resolution and framerate during on-site testing.

Figure 4.14: Detection of a car, partially in frame.

Table 4.15: Measured times until the car was detected. ”-” indicates the car was not detected at all and 0 indicates the car was detected before it had entered the image entirely.

Test	**t**(s)
1	0
2	0
3	-
4	2.5
5	0

Results

Desire 5.1 was not fulfilled, as the successful detection rate within the 2 seconds threshold was 60%, failing to reach the 80% acceptance criterion. Despite this, the on-site vehicle detection showed a marked improvement over the off-site person detection (20%), likely due to the larger visual footprint of vehicles and the test being executed on an asphalt area. The failure of the test further confirms the need for a more robust object detection for a fully automated surveillance system.

4.3.2 Auto Surveillance Drone Swap Test

This test was conducted to verify that a surveilling drone with a low battery level could be replaced by another available drone. To ensure a safe swap and avoid potential collisions, the replacement drone should only be dispatched after the previous drone has landed.

Setup

The first DJI Mavic 2 drone was connected with a camera and an initial battery level above 30%. It was assigned to surveil a large asphalt area at the AstaZero test

site using the interactive map and its polygon functionality. A second DJI Mavic 2 drone was then connected, also equipped with a camera, but with an initial battery level above 60%. The first drone hovered over the assigned area until its battery level reached 20%.

Metrics

The test results were evaluated against Desire 2.2.6, shown in Table 4.16.

Table 4.16: Requirement for automated drone replacement based on battery level.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
2.2.6	The system shall send a replacement drone when the active drone reaches low battery level.	On-site testing	A secondary drone takes over the mission successfully when the active drone reaches 20% battery.	D

Outcome

When the active drone's battery reached the 20% threshold, the backend's battery listener successfully triggered the swap logic. The system automatically transitioned the first drone into a `GoHome` mission while simultaneously generating a `Pending` surveillance mission for the secondary drone, using the same coordinates. The secondary drone remained grounded until the backend received the `task_complete` event from the first drone's landing. Upon confirmation, the replacement drone was autonomously dispatched, reaching the surveillance area and resuming monitoring without operator intervention.

Results

Desire 2.2.6 was successfully fulfilled. The on-site test validated that the system can maintain long-term surveillance through autonomous drone rotation. Crucially, the "wait-for-landing" logic functioned as intended, eliminating the risk of mid-air collisions during the swap. While this sequencing introduces a brief gap in live surveillance, the successful preservation and transfer of mission parameters between different drones confirms the robustness of the backend's mission management architecture.

5
Discussion

This chapter discusses the performance of the individual subsystems and the system as a whole, highlighting both strengths and limitations identified during development and testing. Reflections on the overall design process, lessons learned, and potential areas for improvement are also presented. In addition, the chapter addresses ethical aspects related to the project and proposes directions for future work.

5.1 Data Serialization

In previous work, the code utilized JSON as the message format. One of the disadvantages of JSON is that it does not enforce a schema or guarantee the correct types. This was significantly improved upon by using Pydantic for the schemas and data stored in Redis. However, not all messages and Redis data developed towards the end of the project use schemas. An example of this is the Mission data, which is stored to Redis as a standard Python dictionary. As long as all code writing to Redis uses the methods in Mission Registry, no problem should occur. The issue is that nothing is preventing a new piece of code from doing something wrong to the data, which would cause errors in unrelated places from where the erroneous data was written to, leading to difficulties debugging. Accessing invalid fields on the dictionary is not checked by the Python linter, leading to runtime errors that could have been prevented during coding. The simple solution would be to switch to Pydantic models, however, the Mission classes have state, methods, and inherit from each other, which is incompatible with the data-only models in Pydantic. This can most likely be solved, but was never investigated.

An issue with using Mission classes and serializing data to Redis, is that when read from Redis we lose the ability to call the methods on the class, leading to inconsistent code regarding accessing mission data. For example a mission is created as a Mission object. It is then dispatched using the Mission Registry, and the return value is the dictionary stored to Redis.

Within the DJI Adapter, there is no validation logic for JSON objects prior to their transmission to the backend. This lack of client-side validation is mitigated by the fact that the backend validates all incoming messages, ensuring that any structural errors or bugs are identified. Nonetheless, this architectural detail is noteworthy to mention.

5.2 Drone Connection Persistence

The system in the prior year's work used an incrementing identifier for each drone adapter that was connected. That is, each new drone that was connected got an ID one larger than the last connected drone. This is a reasonable system, but raises some potential problems if the drone has to reconnect, if the application crashes while the drone is on a mission for example. With this system, when the application is restarted the backend will register it as a new drone and fail to send it more tasks or to even to go home, resulting in the drone hovering until it is sent home automatically due to a low battery level. To prevent these risks, our implementation uses persistent IDs for each drone, utilizing the unique model ID each drone has. This results in substantial improvements in safety and reliability as the system can recover from potential drone connection losses and crashes.

This was illustrated when testing at AstaZero, where the phone hosting a WiFi hotspot accidentally went too far away from the controllers and the application lost connection and had to be restarted.

However, the current system could use some improvements. It is up to the drones to ensure their model ID is unique among the other drones. If it isn't, the backend refuses the connection. A system where drones can negotiate a unique and persistent ID using the backend would be preferable.

5.3 Object Detection

It was discovered during testing that the object detection is sometimes unreliable or unable to find objects. For example, in a scenario where a drone is surveilling a football field, it fails to identify a moving person. Some issues could be because of the limitations of the model's training data. It might have been trained on videos of vehicles on tarmac and have difficulties with different surfaces. It was also trained on images with an altitude of 30 meters, which would make it perform worse for other altitudes [2].

Another issue is the computing power required to run the model. The students' computers did not have Nvidia GPUs to be able to utilize the speedup provided by using CUDA (Compute Unified Device Architecture) for the YOLO model inference, restricting the model to the CPU. The latency of detecting an object was approximately 3.5 seconds on average, from the results in Section 4.1.4. Were an intruder to enter the test track, the response might be outdated when the operator makes a decision. The FIFO queue for frame processing with a fixed maximum size (discussed in Section 3.5.8) improves latency by limiting the number of frames processed. Without this mechanism, every incoming frame would be processed, resulting in significantly higher latency. However, every frame not processed is a missed chance of detecting objects. This could be a partial reason for the unreliable object detection. As a temporary workaround, a manual detection trigger was

implemented to ensure that interceptive missions could still be initiated even when automatic object detection failed.

5.4 Extended Video Surveillance

The surveillance with multiple drones developed in the prior work was not integrated into the system since it was not a priority. This feature could be added as an option when drawing the watch area polygon. The developed mission system would be able to handle it without much issue.

5.5 Suggestions on Future Work

This section outlines key areas where additional development could further optimize the operational efficiency and safety of drone-based surveillance at AstaZero.

5.5.1 Integration with ATOS

A full integration with the ATOS ecosystem is a critical next step for making the system fully applicable in a real-world testing environment. By establishing a direct data link with ATOS, the surveillance system could cross-reference all detected objects against the official list of scheduled test participants in real time.

Currently, the lack of this integration presents a significant operational challenge: every authorized test object, including vehicles and pedestrians involved in active scenarios, would be flagged as a potential intruder. This would result in a high rate of false positives, forcing the operator to manually evaluate each detection to distinguish between planned participants and actual threats.

5.5.2 Object Detection

To improve upon the shortcomings of the YOLO model discussed previously, a better and more reliable model could be trained. One that used more varied training data along with capturing real world conditions from more altitudes. Increasing the resolution and training it using color video instead of grayscale could be investigated as it may yield better results.

5.5.3 Surveillance Blind Spot

When a drone gets sent home due to a low battery level, the next drone gets dispatched once the previous drone lands to avoid potential collisions. This introduces a temporary time-based blind spot in the surveillance coverage where an intruder could go undetected. A safer solution would be to replace them more seamlessly, in such a way that the previous drone keeps surveilling the area until the new drone arrives.

5.5.4 Distributed Processing

With a centralized backend architecture, scalability is ultimately limited by the computational resources of a single machine. Although the overall system architecture is designed to support a larger number of drones, the backend processing workload increases as additional drones are connected.

In particular, the latency of object detection scales approximately linearly with the number of active video streams. In the context of this project, a partial workaround identified after the on-site testing at AstaZero would be to allow the operator to manually disable selected drone cameras through the frontend interface. By reducing the number of active video streams processed simultaneously, more computational resources could be allocated to the remaining streams, improving the efficiency and reliability of the object detection for the active drones. A more permanent and robust solution would be to distribute the YOLO inference using a job system to multiple computers, which would allow the system to scale to significantly larger drone fleets and surveillance areas. Furthermore, the distributed nodes could utilize CUDA-capable GPUs to accelerate inference, allowing more video frames to be processed in real time and thereby potentially improving both detection performance and reliability.

5.5.5 Dynamic Coordinate Updates

One limitation of the current system is the latency between the initial detection of an intruder and the arrival of the responding drone. During this flight interval, a mobile intruder can move significantly further into the testing track, potentially making the original coordinates inaccurate.

To address this, an advantageous improvement would be the implementation of dynamic coordinate updates. By configuring the backend to recalculate an intruder's latest position at regular intervals the responding drone could continuously adjust its trajectory.

Furthermore, integrating a target-following algorithm would allow the drone to lock onto the intruder once in range. This would ensure persistent surveillance and proactive deterrence, as the drone would autonomously shadow the intruder's movements rather than merely hovering at a static location

5.6 Social and Ethical Aspects

While the developed software provides clear benefits for traffic safety, it also raises some social and ethical challenges. These are presented in this section.

5.6.1 Safety and Reliability

The purpose of the system is ultimately to ensure safety during the testing of autonomous vehicles. A product that fails to provide robust monitoring of the tests, or fails to convey its limitations in doing so, may lead to a false sense of security.
On the other hand, the system offers significant potential in other sectors beyond its application in autonomous vehicle testing. For instance, the system could be adapted for security surveillance to detect unauthorized activities contributing to public safety.

5.6.2 Personal Integrity

An important part of the testing is to record drone footage of the tests. Parts of these tests were done in an urban environment and might have captured bystanders. Although there isn't a legal limitation on recording in a public setting, according to the IMY (Swedish Authority for Privacy Protection) [46], there are many guidelines on how it should be done. The legal limitations primarily concern the storage of recorded material in accordance with the GDPR (General Data Protection Regulation) [47].

To mitigate these risks, recordings were restricted to appropriate settings. Efforts were made to minimize the capture of identifiable individuals by choosing test environments with minimal bystanders and recording only when necessary. Any collected footage was shared or published strictly on a "need-to-know" basis within the two student groups, their supervisors, and sponsor at AstaZero. Data storage was limited to material that serves a clear, predefined purpose for the project's objectives.

5.6.3 Dual-use and Open-source

In recent years, small FPV drones have been weaponized. Their capabilities as reconnaissance assets or offensive weapons have been widely used in the Russo-Ukrainian war. It is estimated that between 70% and 80% of casualties in the war are attributed to drones [48]. This raises ethical concerns with the potential use of the software in offensive applications.

Since AstaZero is a public research institution and works with an open-source approach, the final product will be made public which makes the product available to any actor. The software is specifically optimized for safety during autonomous vehicle testing in a controlled environment. This limits its direct utility in offensive applications compared to existing, general-purpose FPV platforms.

6
Conclusion

This thesis set out to extend the existing drone surveillance platform at the AstaZero autonomous vehicle testing facility into a more scalable and flexible multi-drone system capable of coordinating heterogeneous drone platforms through a generalized communication framework. Building upon previous project iterations, the work focused on improving interoperability between drone platforms, expanding operator functionality through a web-based interface, and establishing a modular backend architecture suitable for future large-scale surveillance operations.

The project's key milestones defined at the beginning of the thesis were largely achieved through the developed system architecture and implemented functionalities. The platform demonstrated support for communication with multiple drone platforms through generalized adapter layers for both DJI SDK and MAVLink-compatible drones. However, due to the geographic separation between the Chalmers team and Penn State team, full end-to-end interoperability between DJI and MAVLink drones could not be completely validated during the project.

The developed frontend enables operators to define surveillance areas, monitor telemetry and live video streams, and manage active and proposed missions in real time. Furthermore, the implemented mission planning framework demonstrated autonomous drone selection based on telemetry data and mission-specific hardware requirements.

Integration of the existing object detection enabled automatic generation of interceptive missions based on detected objects within the monitored area. However, object detection remained one of the primary technical challenges throughout the project, as detection reliability and performance were sensitive to factors such as computational load and video stream quality. During testing, the detection models also showed signs of limited generalization outside the data on which they had been trained. Since object detection was inherited from previous work and considered outside the primary scope of this project, the focus was placed on integration and system-level functionality rather than optimization of the detection model itself.

Overall, the project represents a meaningful extension of the AstaZero drone surveillance platform and leaves the system in a significantly stronger position for future development. The most significant contributions of this work are the generalized multi-drone communication architecture, the mission planning and drone selection framework, and the integration of heterogeneous drone platforms within a unified

system architecture. Together, these contributions provide both a practical implementation and a scalable foundation for future autonomous surveillance systems operating at larger multi-drone or swarm scales.

Bibliography

[1] P. Rie. "Flying drone." Accessed: 2026-05-13. [Online]. Available: https://www.pexels.com/sv-se/foto/flygande-kamera-teknologi-dronare-13310694/.

[2] A. Arshad, A. Boisen, J. Eriksson, V. Forsell, A. Hallander, and E. Rödin, "Drone platform for safety in testing," Bachelor's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2025.

[3] R. Brenick, *Personal communication*, Jan. 2026.

[4] RISE Proving Ground AstaZero. "Welcome to the automated transport system of the future – AstaZero," Accessed: Feb. 11, 2026. [Online]. Available: https://astazero.ri.se/.

[5] AstaZero. "About us," RISE Research Institutes of Sweden, Accessed: Feb. 9, 2026. [Online]. Available: https://astazero.ri.se/en/about-us/.

[6] M. E. Bilal, D. Ferreira, D. Mörck, F. Sandström, A. Svenske, and A. Särnholm, "Drone platform for safety in autonomous vehicle testing," Bachelor's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2024.

[7] DJI. "Mobile SDK introduction," DJI Developer, Accessed: May 11, 2026. [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/introduction/mobile_sdk_introduction.html.

[8] Dronecode Project. "MAVLink developer guide," Linux Foundation, Accessed: May 12, 2026. [Online]. Available: https://mavlink.io/en/.

[9] IT-Högskolan. "Frontend och backend – vad är skillnaden?" Accessed: Apr. 22, 2026. [Online]. Available: https://www.iths.se/frontend-vad-ar-det/.

[10] IBM. "What is Redis?" Accessed: Apr. 23, 2026. [Online]. Available: https://www.ibm.com/think/topics/redis.

[11] Microsoft. "Interprocess communications," Microsoft Learn, Accessed: May 11, 2026. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/ipc/interprocess-communications.

[12] Harvard University. "Kernel 1: Processes and isolation," Harvard John A. Paulson School of Engineering and Applied Sciences, Accessed: May 11, 2026. [Online]. Available: https://cs61.seas.harvard.edu/site/2022/Kernel1/.

[13] I. Fette and A. Melnikov, "The WebSocket protocol," Google, Inc. and Isode Ltd., Tech. Rep., Dec. 2011. DOI: 10.17487/RFC6455. [Online]. Available: https://www.rfc-editor.org/info/rfc6455.

[14] MDN Web Docs. "An overview of HTTP," Mozilla, Accessed: Apr. 26, 2026. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview.

[15] Amazon Web Services. "What is Docker?" Accessed: Apr. 11, 2026. [Online]. Available: https://aws.amazon.com/docker/.

[16] Docker Inc. "What is Docker?" Accessed: Apr. 24, 2026. [Online]. Available: https://docs.docker.com/get-started/docker-overview/.

[17] Docker Inc. "Docker compose documentation," Accessed: May 12, 2026. [Online]. Available: https://docs.docker.com/compose/.

[18] D. Minoli, *IP Multicast with Applications to IPTV and Mobile DVB-H*. John Wiley & Sons, Inc., 2008, ISBN: 978-0-470-25815-6. DOI: 10.1002/9780470383612.

[19] S. E. Deering, "Host extensions for IP multicasting," IETF, RFC 1112, Aug. 1989. DOI: 10.17487/RFC1112. [Online]. Available: https://www.rfc-editor.org/info/rfc1112.

[20] D. Meyer, "Administratively scoped IP multicast," IETF, RFC 2365, Jul. 1998. DOI: 10.17487/RFC2365. [Online]. Available: https://www.rfc-editor.org/info/rfc2365.

[21] J. Postel, "User datagram protocol," IETF, RFC 768, Aug. 1980. DOI: 10.17487/RFC0768. [Online]. Available: https://www.rfc-editor.org/info/rfc768.

[22] L. Eggert, G. Fairhurst, and G. Shepherd, "UDP usage guidelines," IETF, RFC 8085, Mar. 2017. DOI: 10.17487/RFC8085. [Online]. Available: https://www.rfc-editor.org/info/rfc8085.

[23] ROS2 GitHub Organization. "ROS 2 — GitHub organization," Accessed: Feb. 11, 2026. [Online]. Available: https://github.com/ros2.

[24] AstaZero. "AstaZero announces the release of ATOS." Automated vehicle Test Operating System, Accessed: Feb. 10, 2026. [Online]. Available: https://astazero.ri.se/astazero-announces-the-release-of-atos/.

[25] WebRTC Project Authors. "WebRTC: Real-time communication for the web," Google, Mozilla, and Opera, Accessed: May 12, 2026. [Online]. Available: https://webrtc.org/.

[26] Pydantic Team. "Models - Pydantic documentation," Accessed: Apr. 22, 2026. [Online]. Available: https://pydantic.dev/docs/validation/latest/concepts/models/.

[27] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004. DOI: 10.1023/B:VISI.0000029664.99615.94.

[28] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2005, pp. 886–893. DOI: 10.1109/CVPR.2005.177.

[29] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[31] Python Software Foundation. "Threading — thread-based parallelism," Python Documentation, Accessed: May 11, 2026. [Online]. Available: https://docs.python.org/3/library/threading.html.

[32] Google Developers. "Processes and threads overview," Android Developers Documentation, Accessed: May 10, 2026. [Online]. Available: https://developer.android.com/guide/components/processes-and-threads.

[33] S. Chacon and B. Straub, "Customizing Git - Git hooks," in *Pro Git*, 2nd, Apress, 2014, ch. 8. Accessed: Apr. 22, 2026. [Online]. Available: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks.

[34] Git Contributors, *Githooks: Hooks used by Git*, Version 2.54.0, Git Project, 2026. Accessed: Apr. 22, 2026. [Online]. Available: https://git-scm.com/docs/githooks.

[35] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 7th. Newtown Square, PA: Project Management Institute, 2021, ISBN: 978-1-62825-664-2.

[36] Stack Overflow. "Stack overflow developer survey 2025: AI," Accessed: May 11, 2026. [Online]. Available: https://survey.stackoverflow.co/2025/ai#sentiment-and-usage-ai-sel-prof.

[37] Stack Overflow. "Stack overflow developer survey 2022: Version control systems," Accessed: May 11, 2026. [Online]. Available: https://survey.stackoverflow.co/2022/#version-control-version-control-system-prof.

[38] L. Magnusson. "Drone-project-psu-chalmers," Accessed: Apr. 19, 2026. [Online]. Available: https://github.com/lud99/drone-project-psu-chalmers.

[39] K. Cheng, M. Garg, Y. M. Solomon, A. N. Soni, and R. Srinivasan, "Real-time distributed processing for drone swarms," Unpublished report, Pennsylvania State University, Department of Computer Engineering, University Park, PA, May 2026.

[40] Docker Inc. "What is an image?" Accessed: Apr. 24, 2026. [Online]. Available: https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-an-image/.

[41] Pydantic Team. "Types/unions - Pydantic documentation," Accessed: Apr. 22, 2026. [Online]. Available: https://pydantic.dev/docs/validation/latest/concepts/unions#discriminated-unions.

[42] SciPy Community. "scipy.optimize.curve_fit — SciPy v1.17.0 Manual," Accessed: May 12, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html.

[43] DroneBundle. “Drone flight time: Complete guide to battery life, performance optimization, and mission planning,” Accessed: Apr. 22, 2026. [Online]. Available: https://dronebundle.com/blog/flight-time-of-a-drone.

[44] EfficientIP. “What is DHCP and why is it important?” Accessed: Apr. 28, 2026. [Online]. Available: https://efficientip.com/glossary/what-is-dhcp-and-why-is-it-important/.

[45] S. Purohit. “Beneath the bytes: A deep dive into magic numbers for file identification,” Accessed: Apr. 19, 2026. [Online]. Available: https://medium.com/@shailendrapurohit2010/beneath-the-bytes-a-deep-dive-into-magic-numbers-for-file-identification-4bff213121c4.

[46] Integritetsskyddsmyndigheten. “Drönare,” Accessed: Feb. 11, 2026. [Online]. Available: https://www.imy.se/privatperson/kamerabevakning/regler-for-dig-som-kamerabevakar/dronare/.

[47] Integritetsskyddsmyndigheten. “Hantering av inspelat material,” Accessed: Feb. 11, 2026. [Online]. Available: https://www.imy.se/privatperson/kamerabevakning/regler-for-dig-som-kamerabevakar/nar-gdpr-galler-for-din-kamerabevakning/hantering-av-inspelat-material/.

[48] E. Hartog. “Drones caused 3 out of every 4 Ukraine war casualties, Latvian spies say,” Accessed: May 11, 2026. [Online]. Available: https://www.politico.eu/article/latvian-report-drones-are-mass-killers-on-the-ukraine-front/.

A

Redis Data

Table A.1: Redis key-value pairs used for the storing system state

Key	**Value**
frame_drone{id}	Encoded JPEG bytes
frame_drone_merged	Encoded JPEG bytes
frame_drone{id}_annotated	Encoded JPEG bytes
frame_drone{id}_detections	DetectionsSchema
telemetry_drone{id}	TelemetrySchema
capabilities_drone{id}	CapabilitiesSchema
model_drone{id}	String
watch_area	Points
watch_area_min_rect	Points
watch_area_coverage_area	Points
mission_{mission_id}_task_queue	AnyTaskAction
mission_{mission_id}_state	Mission

B

Requirements and Desires

Table B.1: System requirements and desires.

ID	Requirement	Validation Method	Acceptance Criteria	R/D
1. Map				
1.1 Area				
1.1.1	The user shall be able to create a rectangular surveillance area.	Mock drone	A rectangular surveillance area can be created successfully on the map.	R
1.1.2	The user shall be able to create a polygonal surveillance area.	Mock drone	A polygon with arbitrary shape and number of edges can be created in the interactive map.	D
1.2 Map Update				
1.2.1	The user shall be able to create a new area.	Mock drone	When creating a new area the points defining the selected area updates and overwrites the previous values.	R
1.2.2	The system shall save the last area on page refresh.	Mock drone	The previously created area remains visible after refreshing the page.	D
1.3 Live Position				
1.3.1	The system shall display the drone position on the map.	Mock drone	The drone position is continuously updated and shown correctly on the map with a precision of 3m.	D
1.3.2	The system shall display users position on the map.	Mock drone	The users position is displayed with a precision of 5m.	D
2. Missions				
2.1 Safety				

ID	Requirement	Validation Method	Acceptance Criteria	R/D
2.1.1	The user shall be able to abort a mission.	Off-site safety testing.	The active mission is cancelled successfully on users command.	R
2.1.2	The user shall be able to return the drone to its original position.	Off-site safety testing.	The drone returns to its takeoff position with a precision of 1 m when the user activates the Return Home command.	R
2.2 Mission Control				
2.2.1	The user shall be able to trigger a manual "Go To" mission.	DJI simulation	The drone travels to the selected destination successfully.	D
2.2.2	The user shall be able to trigger manual object detection.	Off-site testing	Object detection starts successfully when triggered by the user.	D
2.2.3	The system shall generate mission proposals upon object detection.	Mock drone	A mission proposal is generated automatically after object detection.	R
2.2.4	The selected mission is initiated.	DJI simulation	When a mission is selected in the frontend and the user chooses to start the mission, the correct mission is sent to the backend.	R
2.2.5	The system shall select the best drone for the mission.	Drone select test script.	The intended drone is selected for a set of cases with different mission types and drone specifications.	R
2.2.6	The system shall send a replacement drone when the active drone reaches low battery level.	On-site testing	A secondary drone takes over the mission successfully when the active drone reaches 20% battery.	D
2.2.7	The drone can execute a GotoAndBlink mission.	Off-site testing	Beacon can be seen blinking when a GotoAndBlink mission is executed.	D
2.2.8	The drone can execute a GotoAndIlluminate mission.	Off-site testing	The spotlight illuminates the object when GotoAndIlluminate mission gets executed.	D

ID	Requirement	Validation Method	Acceptance Criteria	R/D
2.2.9	The drone can execute a GotoAndAudio mission.	Off-site testing	Audio is played through the drone speaker when a GotoAndAudio mission is executed.	D
3. Information				
3.1	The system shall display battery level.	Mock drone	The battery level is displayed correctly for connected drones.	R
3.2	The system shall display telemetry information.	Mock drone	Lat, long, altitude and speed is displayed for all connected drones and continuously updated.	D
3.3	The system shall display drone specifications.	Mock drone	All equipment and capabilities for all connected drones are displayed in the frontend.	D
3.5	The system shall display all connected drones.	Mock drone	All connected drones are listed successfully in the frontend.	R
4. Video Stream				
4.1.1	The user shall be able to switch between connected drones.	Mock drone	The video stream changes successfully when selecting another drone.	R
4.1.2	The user shall be able to switch between normal and annotated video views.	Mock drone	The frontend switches successfully between normal and annotated video views.	D
4.1.3	The system shall provide real-time video streaming.	DJI simulation	Video latency shall not exceed 2 s for the regular view.	D
4.1.4	The system shall provide real-time video streaming with the annotated view.	DJI simulation	Video latency shall not exceed 3 s for the annotated view.	D
5. Object Detection				
5.1	The system shall detect cars in the video stream.	On-site testing	Cars shall be detected within 2 s from an altitude between 25–35 m at least 80% of the time.	D

ID	Requirement	Validation Method	Acceptance Criteria	R/D
5.2	The system shall detect people in the video stream.	Off-site testing	People shall be detected within 3 s from an altitude between 25–35 m at least 80% of the time.	D
6. Surveillance Calculations				
6.1	The system shall calculate the correct OBB and camera orientation.	Surveillance area calculation script	The generated OBB and camera coverage shall be correct at least 90% of the time.	R
6.2	The system shall translate polygons into drone positions.	Off-site testing	The drone shall cover the entire polygon correctly at least 90% of the time.	R
7. Connection				
7.1	The drone shall automatically register to the backend when started.	DJI simulation	Drone telemetry and specifications are displayed in the frontend within 10 s after the drone has started.	R
7.2	The user shall be able to manually register a drone to the backend through the Android app.	DJI simulation.	A drone can be successfully registered to the backend through manual user input of port and IP-adress in the Android app.	R

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY