Diploma Project

Drone Delivery System With ArUco marker Autonomous Landing

Nestoras Gregoriou

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), and with full accountability on my part for the accuracy, integrity, and validity of all content.

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

Drone Delivery System With ArUco marker Autonomous Landing

Nestoras Gregoriou

Advisor: Dr. Panayiotis Kollios

The Thesis was submitted in partial fulfillment of the requirements for obtaining

the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2026

Acknowledgements

I would like to express my sincere gratitude to the individuals whose support and contributions have been instrumental in the successful completion of this research.

First and foremost, I extend my heartfelt thanks to my advisor, Dr. Panayiotis Kollios, for their invaluable guidance, continuous support, and insightful feedback throughout this project. Their mentorship played a vital role in shaping the direction and outcome of this work.

I am also thankful to the KIOS Drones Team, whose collective efforts and collaborative spirit created a productive and supportive research environment. Being part of this team has significantly enriched my experience and the quality of this thesis.

In particular, I would like to express my deepest appreciation to Giorgos Pettemeridis, Yiannis Grigoriou, Panagiotis Chrysanthou and Charalambos Soteriou of the Drones Team. Their dedicated assistance, expertise, and encouragement were essential in carrying this project to completion. This work would not have been possible without them.

Thank you all for your contributions and unwavering support.

ABSTRACT

This work presents a system of an autonomous drone delivery system capable of executing remotely defined missions and performing visual marker-based precision landings. The system integrates the DJI Matrice 300 RTK (Real-Time Kinematic) drone with an NVIDIA Jetson Xavier onboard computer, which serves as the main processing unit for mission execution and precision tasks. Mission planning is performed through the AIDERS platform, a mission management system developed by KIOS Research and Innovation Center of Excellence. Waypoints selected through the platform's map interface are transmitted to the drone using JSON-based telemetry communication channel via WebSockets, enabling dynamic generation of delivery missions. The onboard control logic processes these waypoints and executes them sequentially while verifying arrival at each waypoint using a Haversine distance calculation to ensure positional accuracy within an acceptable error radius before landing execution. To enable autonomous landing, the system employs ArUco marker detection using the OpenCV library, utilizing imagery from the DJI H20 camera mounted on the drone. When the marker is detected, the system identifies its position through bounding box detection and triggers a custom landing procedure based on visual feedback. Real-World flight tests successfully demonstrated the delivery mission's functionality and waypoint navigation. Due to temporary airspace restrictions in Cyprus during the evaluation period, the autonomous landing component was demonstrated through simulation-based experiments and real flight experiments. The results indicate that the proposed system provides a feasible framework for integrating centralized mission planning with onboard visual perception for autonomous drone delivery operations.

Keywords: ArUco marker detection; Mission Planning systems; UAV navigation; Precision Landing; Autonomous drone delivery

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Important abbreviations that have been used in the text and need explanation are briefly presented.

UAV	Unmanned Aerial Vehicle
RTK	Real-Time Kinematic
SDK	Software Development Kit
PSDK	Payload Software Development Kit
JSON	JavaScript Object Notation
UDP	User Datagram Protocol
WS	WebSocket
GCS	Ground Control Station
IMU	Inertial Measurement Unit
GNSS	Global Navigation Satellite System
FPV	First Person View
RRT	Rapidly-exploring Random Tree
RRT*	Rapidly-exploring Random Tree Star
PRM	Probabilistic Roadmap
IoD	Internet of Drones
GNC	Guidance, Navigation, and Control

Chapter 1 Introduction

1.1 Background Information

Advancements in Unmanned Aerial Vehicles (UAVs) have led to the transition from specialized military and research platforms to practical commercial applications for civilian use [1]. In recent years, one of the most active application areas has been autonomous package delivery, where UAVs are used to transport items rapidly from point to point. This development has been driven by the increasing demand for faster delivery services, lower operational costs, and reduced reliance on manual transportation workflows. [2]

An autonomous delivery mission can be seen as an end-to-end process in which an UAV receives a delivery task, navigates to one or more target locations and completes the mission with minimal human intervention. A core component for this is waypoint navigation that allows the UAV to follow a predefined sequence of geographic coordinates and altitude constraints, creating a predictable mission path from takeoff to delivery completion.

A major challenge in delivery missions is landing at each waypoint in a precise and reliable way. GPS-based navigation can guide the UAV close to the target but environmental factors, like sensor noise and local disturbances can reduce the final positional accuracy. For this reason, precision landing techniques are often required. One widely used approach is computer-vision-assisted landing with ArUco markers [3], [4], where the UAV detects a known visual marker and corrects its position during descent. This method improves landing a lot and helps reduce risk during delivery completion.

Based on these considerations, this thesis focuses on integrating autonomous mission execution, waypoint-based navigation, mission planning, and ArUco-based precision landing within a single drone-delivery workflow.

1.2 Problem Statement

Autonomous drone delivery systems are increasingly studied, yet reliable real-world deployments remain challenging. In practice, many systems can demonstrate some form of autonomy but still depend on a lot of manual supervision or intervention during critical phases. The challenge is not only individual component performance, but a consistent integration of all mission stages into a dependable end-to-end pipeline.

Three technical difficulties are particularly important. First, waypoint navigation must remain stable and accurate throughout the route, even when environmental conditions are imperfect. Second, mission planning must manage mission stages and constraints in a structured way so that execution is safe, predictable, and recoverable when deviations occur. Third, landing must be precise enough for delivery completion, since small final-position errors can prevent safe or successful package drop-off.

Although works in these areas are frequently examined independently in literature rather than as parts of a unified operational system. This separation creates a practical gap for deployment-oriented applications, where the entire system must function as a unified system. For campus-scale delivery scenarios in particular, this integration affects mission success, safety and reproducibility of results. [5] [6] [3] [4]

Therefore, the problem addressed in this thesis is the design and evaluation of an integrated autonomous drone-delivery mission framework that combines waypoint navigation, structured mission planning, and computer-vision-based

precision landing (using ArUco markers). The work is validated in campus-like short range conditions. Long range conditions are considered outside the primary scope and is left for future investigation.

1.3 Objectives

1. **AIDERS platform communication:** Bidirectional communication with the AIDERS platform is established for the real-time reception of mission requests and waypoint coordinates, as well as for the transmission of mission status updates from the drone (e.g., “landed” or “in-mission”) and requests to resume or continue a mission after landing while awaiting takeoff.
2. **Generate waypoint mission:** Navigation mission generation, from the coordinates provided by the platform, including start/end conditions like number of repetitions and other custom settings like different original altitude.
3. **Execute mission:** Upon transmission and acceptance of the selected waypoints in the AIDERS platform, the mission is autonomously initiated. The drone is oriented toward the first waypoint, and the mission sequence is carried out automatically. Transitions between mission phases, such as orientation toward the next waypoint, ascent, descent, and landing, are handled by the system. In addition, a safety confirmation request is issued before each take-off to ensure safety of individuals in the surrounding area.
4. **Implement precision landing:** At each delivery point, specifically at the second of the three waypoints, precision landing is achieved through the use computer vision and ArUco marker detection for final-position correction.
5. **Evaluate key metrics:** Mission telemetry data are stored in a MySQL database and transmitted from the drone to the AIDERS platform through WebSocket connection. Based on these data, battery levels are monitored throughout the mission, and the contribution of different mission phases to battery consumption is assessed. In addition, the duration of each phase and

key flight parameters, such as longitude, latitude, altitude, and velocity are analyzed.

1.4 Structure of the Thesis

This thesis is organized into 6 chapters:

Chapter 1 introduces the research domain, defines the core problem, and presents the objectives and scope of the work.

Chapter 2 provides a comprehensive literature review, on UAV delivery systems, autonomous navigation, mission planning approaches, and precision landing techniques with emphasis on computer-vision-based methods.

Chapter 3 explains the technical background necessary to understand the system's components, including an exploration of the UAV platform (AIDERS), software stack, navigation concepts, and vision components used in the implementation.

Chapter 4 presents the design and implementation of the system architecture, detailing each step of the waypoint navigation, mission planning, and ArUco-assisted landing and how they are integrated into a complete mission workflow, including diagrams and tables.

Chapter 5 reports the experimental setup, evaluation methodology, and results from campus-scale testing, followed by analysis of performance and observed limitations.

Chapter 6 concludes the thesis by summarizing the main findings and future work.

Chapter 2 Literature Review

2.1 Overview of UAV and Autonomous Delivery Missions

UAVs (Unmanned Aerial Vehicles) have evolved into practical platforms for civil applications, including inspection, mapping, emergency support, and logistics. Among these [7] domains, autonomous delivery has received significant attention because it addresses time-critical and cost-sensitive transport tasks while reducing fully manual workflows [7] [8].

Autonomous delivery missions are typically defined as end-to-end operational processes in which a UAV receives a task, executes a flight plan, reaches one or more mission points, performs pickup and drop-off actions, and terminates the mission safely. In contrast to manual remote piloting, autonomous missions require coordinated interaction between onboard control logic, mission planning constraints, sensing modules and communication with external systems [9] [10].

A delivery mission is not only a navigation problem. It combines multiple functions, such as route or waypoint generation, mission-state transitions, battery life and safety checks, communication with a mission control and creation platform, and final delivery completion at the target location with the possibility of repeating the mission. This means reliability depends heavily on how well components are integrated, rather than on any single algorithm [9] [11].

A common and used deployment is waypoint-based navigation, where the vehicle follows a sequence of latitude/longitude/altitude targets. While waypoint navigation provides structure and repeatability, it can still face issues such as wind, GPS uncertainty, communication interruptions, and battery limitations. Therefore,

operationally usable systems must include additional control logic for interruption handling, mission continuation, and safe fallback behavior [10] [12] [13].

In this thesis, autonomous delivery missions are structured, multi-phase processes that include mission assignments and control from an external platform (AIDERS), waypoint parsing and validation, autonomous waypoint execution, status/telemetry feedback, precision landing at the delivery destination, and return or recovery behavior when required. The workflow in this project follows this structure, where platform commands are bridged to the drone's flight execution, and telemetry is returned continuously for mission supervision.

2.2 UAV Navigation and Waypoint Following

UAV navigation and waypoint following are very important for achieving autonomous delivery missions. In practical systems, autonomy depends on the coordinated operation of Guidance, Navigation, and Control (GNC) modules rather than on a single algorithm. Guidance determines where the vehicle should go, navigation estimates the vehicle's state (position, velocity, and altitude), and control commands the actuators so the UAV follows the intended path under real-world disturbances [14].

Although this thesis does not propose a global path planning algorithm, path planning literature is relevant because it provides theoretical context for how waypoint routes are generated and adapted in autonomous UAV systems.

2.2.1 Navigation Fundamentals for Autonomous UAVs Architecture Overview

Navigation Provides the state information needed for mission execution, including position, heading, velocity, and altitude. In many outdoor deployments, UAVs rely on fused measurements from Global Navigation Satellite System (GNSS), Internal Measurement Unit (IMU), barometer and other onboard sensors to estimate these states

in real time. However, localization quality can degrade due to signal blockage, multipath effects, or poor satellite geometry, and this directly impacts path-following quality [6].

Recent survey studies highlight that robust autonomous navigation requires explicit handling of uncertainty, including noisy perception and incomplete environmental information. This is important for delivery mission scenarios where the UAV must preserve safety margins while still tracking mission waypoints with acceptable accuracy [14] [6].

2.2.2 Waypoint Following Strategies

Waypoint following is commonly implemented by defining an ordered list of altitude, longitude, latitude targets and generating motion commands that guide the UAV between consecutive waypoints. Classical approaches include geometric and guidance-law-based methods such as line of sight and vector field variants, as well as controller centric strategies that track path errors over time. These methods are widely used because they provide a structured and repeatable way to execute missions [13] [14].

From a system's perspective, waypoint following and path planning are linked because, path planning determines the feasible route and waypoint sequence, while the follower must execute that route despite vehicle constraints and external disturbances. Literature reviews show that this interaction is a usual source of mission degradation when planning assumptions do not match real flight conditions [5] [14].

Operational studies also show that waypoint execution quality depends on platform dynamics, controller design, and wind robustness. Waypoint missions usually require anti disturbance control behavior and continuous correction of cross-track and altitude errors, especially during transitions and constrained maneuvers [12] [15].

2.2.3 Real World Limitations and Robustness Requirements

Waypoint navigation is straightforward to use, but real deployment faces usual problems: wind disturbance, localization drift, communication interruptions, battery constraints, and mission state synchronization problems. These effects can accumulate and reduce mission reliability if the system does not include robust supervision and fallback behavior [6] [2] [15].

That is why recent mission planning literature shows dynamic decision support, runtime adaptation, and operator-aware control flows rather than static one-shot mission execution. In delivery operations, pause/resume logic, mission cancelation handling, and safe recovery policies are not optional features but core reliability mechanisms [11] [9] [16].

2.3 UAV Path Planning

Path planning is a core component of autonomous UAV delivery. While waypoint following is determining how a vehicle tracks predefined points, path planning defines where the UAV should fly in the first place under mission constraints such as obstacle avoidance, flight safety, energy usage, and route quality. In delivery operations, the planner must balance competing objectives: short distance, low energy consumption, low risk, and robust feasibility under environmental uncertainty [17] [18].

2.3.1 Graph-Based Path Planning

Graph-based methods model the environment as nodes and edges and then compute minimum cost routes using deterministic search. In UAV literature, variants of A* style search remain popular because they are interpretable, easy to constrain, and practical for structured environments [18] [19]. In urban delivery scenarios, these methods are often adapted with custom cost functions that include risk, altitude restrictions, and energy-aware edge penalties [19].

2.3.2 Sampling-Based Planning

Sampling-based generates feasible trajectories by exploring the state space without full enumerating all possible paths. Important examples include the **Rapidly-exploring Random Tree (RRT)** family and **Probabilistic Roadmap (PRM)** family, which are widely used in high-dimensional planning problems because they can handle geometric and dynamic constraints affectively [20]. Asymptotically optimal variants such as **Rapidly-exploring Random Tree Star (RRT*)** improved path quality guarantees and became a baseline in many trajectory generation pipelines [20].

For UAVs, sampling-based methods are used when obstacle geometry is complex or partially known. Their key strength is feasibility in difficult spaces. However, path smoothness and convergence speed can require post processing or integration with optimization layers [17] [18].

2.3.3 Optimization and Evolutionary Planning

Optimization planning formulates path generation as a cost-minimization problem under constraints, enabling direct tradeoffs between safety, distance, and energy. Recent surveys report a strong increase in evolutionary and computational-intelligence methods for UAV route optimization, especially in non-convex and multi objective mission spaces [17] [21].

Evolutionary approaches are useful when cost landscapes are complex and differentiability assumptions do not hold. They are used for multi-objective problems such as risk vs distance but can require careful parameter tuning and convergence management [21] [22].Recent literature also emphasizes integration with **Internet of Drones (IoD)** environments, where drones operate as part of connected network that includes communication, telemetry, and mission-management infrastructure. In such settings, planner outputs must remain compatible not only with flight constraints but also with broader system-level coordination requirements [18].

2.3.4 Dynamic Replanning and Obstacle Avoidance

In real deployments, planned routes may become invalid due to moving obstacles, wind shifts, localization uncertainty, or temporary airspace constraints. Therefore, robust delivery systems need replanning or local avoidance mechanisms that can adapt online. Obstacle avoidance path planning studies show that safety guarantees and uncertainty-aware modeling are critical for practical autonomy [23].

The broader robotics literature also established key real time avoidance concepts, including potential field-based formulations for online correction behavior, which influenced many modern UAV local planners [24]. In addition, dynamically feasible trajectory generation methods are required to convert geometric paths into executable motion with bounded derivatives and controllable tracking behavior [25].

In summary, literature shows that no single family solves all mission delivery requirements. Delivery level systems typically combine global planning, local adaptation, and executable trajectory generation. This motivates the following section on mission planning and platform integration, where route generation is connected to command interfaces, mission state control, and operational supervision.

Table 1: Comparison of Representative UAV Path-Planning families used in UAV missions

Method family	Example algorithms	Typical strengths	Typical limitations	Common use in delivery missions
Graph-based	Dijkstra, A*	Deterministic, interpretable, easy to encode constraints.	Sensitive to map resolution and cost discretization	Structured maps, corridor/route planning
Sampling-based	RRT, RRT*	Feasibly in high dimensional or clustered spaces	Raw paths may be jagged and require smoothing	Complex geometry, obstacle-rich environments
Optimization/evolutionary	GA/PSO and other CI optimizers	Handles multi objective costs (risk, time, energy)	Parameter tuning and convergence time sensitivity	Energy-aware and mission-utility optimization
Reactive local avoidance	Potential-field style local correction	Fast online reaction to nearby hazards	Can suffer local minima without hybrid planning	Short horizon correction and emergency avoidance

2.4 Mission Planning Systems and Platform Integration

Mission Planning systems are the layers that connect high-level objectives to executable UAV behavior. In autonomous delivery context, this layers get mission requests, make them into structured plans, send commands, monitor execution, and coordinate planning when conditions change. Literature shows that robust performance depends not only on route quality but also on planning-system architecture, communication reliability, and supervision design [26] [27].

2.4.1 Mission Modeling and Planning Workflows

A mission plan is shown as a structured set of tasks, constraints, and execution states such as, dispatch, transit, action, return/recovery. Dynamic mission-planning research

emphasizes that that plans should support both pre-mission optimization and in-mission adaptation, since route assumptions can become invalid during execution [26].

UAV task allocation surveys show that mission-level performance depends on how effectively assignment, ordering, and timing decisions are made under limited resources. This is relevant when multiple objectives compete, such as completion time, energy usage and risk [28], [29].

In practice mission workflows benefit from explicit state transitions and event -driven triggers, because these improve debuggability and controllability under real operations. Ground control embedded planning studies also show the value of integrating planning and verification functions inside the mission interface rather than treating planning as an isolated offline stage [27].

2.4.2 Ground Control and External Platform Integration

Ground control software acts as an intermediary between mission-control applications and flight execution modules. Modern architecture increasingly supports external control center integration, making high-level systems to define tasks while the ground control stack handles mission translation, dispatch, and monitoring [30].

A Ground Control Station (GCS) is the operator-facing software environment used to plan, monitor, and control UAV missions. Work on QGroundControl demonstrates that integrating mission-design, decision-support, and automated planning functions into operator-facing tools can improve mission preparation and operational usability [31] .

Architecture studies show that communication layers are needed to connect software services, autopilot interfaces, and payload subsystems consistently. This reduces coupling and improves reuse across mission types and vehicle configurations [32].

Table 2: Comparison of Mission Planning and Platform Integration Literature

Focus area	Representative works	Main contribution	Relevance to this thesis
Dynamic mission planning	[26], [27]	Planning/replanning workflows under changing mission conditions	Support phase-based delivery execution and recovery logic
Task allocation and scheduling	[28], [29]	Allocation methods under competing mission objectives	Supports structured mission configuration and repeatable execution
Ground control extensibility	[31], [30]	GCS as integration hub for planning, control, and external systems	Aligns with AIDERS-to-drone orchestration model
Communication architecture	[32], [33]	Robust command/telemetry channel design and fleet-level coordination	Aligns with bidirectional command + telemetry pipeline
Human supervision	[34], [35]	Operator support and workload aware supervision	Aligns with pause/resume/cancel and safe intervention needs

2.4.3 Communication and Telemetry Integration

Autonomous missions require reliable bidirectional exchange: command flow platform to UAV and telemetry flow from UAV to platform. Recent architecture work indicates that communication design directly affects mission continuity, fault recovery, and operator situational awareness [32] [33].

Also transport reliability, integration quality depends on message semantics and system state synchronization. Mission fleet management studies show that coordination quality degrades when communication and mission-state models are weakly coupled, especially in dynamic deployments [33].

2.4.4 Runtime Supervision and Operator-in-the-Loop Control

Even in autonomous mission pipeline, operator supervision remains critical for safety and resilience. When humans are involved in the loop of the execution, studies in multi-UAV planning and control show that operators need clear rationale, intervention levers, and controllable automation behavior to handle any deviations of the planned operational parameters effectively [34] [35].

For deployment-oriented systems, this implies explicit supervision mechanisms such as pause/resume/cancel, approval gates before critical transitions, and visible mission-state reporting. These mechanisms do not reduce autonomy, they make autonomy operationally trustworthy [26] [34].

All these literatures on mission planning systems and platform integration indicate that dependable autonomy emerges from architectural coherence: mission models, communication channels, runtime supervision, and execution control must be designed as one system. This directly motivates the implementation choices discussed in later chapters, where external mission assignment, command bridging, telemetry feedback, and operator-aware mission control are integrated into a single delivery workflow.

2.5 Precision Landing with Computer Vision and Visual Markers

Precision landing is a critical final stage in autonomous UAV delivery missions. Although global navigation can guide a UAV near a destination, terminal landing accuracy often requires local visual feedback. Literature reviews consistently identify vision-based landing as reliable touchdown under real operational uncertainty [36].

2.5.1 Why Precision Landing Is Required

In delivery missions, the final meters are safety-critical because small lateral or altitude errors can lead to unsafe touchdown or failed drop-off. GPS and inertial fusion are effective for route-scale guidance but can be insufficient for centimeter-level endpoint accuracy in cluttered or disturbed environments [36].

Autonomous-landing studies show that robust terminal guidance must combine detection, relative localization, and closed loop correction during descent, instead of relying on a single absolute-position source [37] [38]. This motivates marker-assisted visual landing strategies in which the UAV estimates relative offsets to a known target and applies incremental corrections.

2.5.2 Marker-Based Detection and Pose Estimation

Visual markers provide a practical visual reference for precision landing because they are low-cost, easy to deploy, and directly usable for relative pose/offset estimation. ArUco marker families are widely used due to configurable dictionaries and robust identification behavior under partial occlusion and perspective changes [39].

Algorithmic improvements in marker detection pipelines have also reduced latency and improved frame-rate viability, which is important for real time onboard control loops [36]. Alongside ArUco, other marker systems such as April Tag variants and multimodal visuals have been used to improve robustness in different visual conditions [36] [40].

UAV-focused studies further show that marker design and placement strategy affect detection continuity across altitude changes. Approaches such as embedded/multi-scale marker patterns and pose-estimation-focused landing aids have been used to increase tracking reliability over a wider range of approach distances [40].

Table 3: Comparison of Marker-Based Precision Landing Approaches

Approach family	Representative works	Typical strengths	Typical limitations
Classical ArUco dictionaries	[39]	Simple deployment, low cost, efficient detection	Performance drops with blur, glare, or poor scale tuning
AprilTag/ robust fiducials	[36]	Strong identification robustness and pose utility	Can require heavier computation or marker tuning
Multi-scale/embedded marker design	[40]	Better detection continuity across wide altitude ranges	Additional marker design complexity and calibration effort

2.5.3 Vision-to-Control Landing Pipeline

In practice, many autonomous landing systems use a similar sequence of steps. The onboard camera captures frames, the visual marker is detected, and the system calculates the drone's relative position to the marker. From there, control commands are produced in a closed loop so the drone can keep adjusting its motion during landing. As long as the marker is visible, the drone keeps correcting its horizontal position, while the descent speed depends on how accurately the marker is aligned and on altitude-related limits [37] [38].

This landing pipeline is usually divided into phases such as approach, alignment, descent, and touchdown. Research shows that this phase-based logic makes the system easier to manage and improves recovery when the marker is briefly lost during landing [37].

2.5.4 Robustness Challenges and Fallback Behavior

Even though marker-based landing can work very well in controlled tests, real world conditions make it harder. Issues like blurry images, lighting changes, short periods where the marker is blocked, and delays in detection can reduce the system's reliability. The

literature shows that these should be treated as risks for the full landing system, not just as isolated vision problems [36].

Because of this, dependable landing systems add protection mechanisms. These include checking whether a detection is still recent enough to trust, verifying detection confidence, changing behavior depending on altitude, and using fallback actions if the system loses visual contact with the marker. Similar recommendations appear in both review papers and practical studies [37] [38].

In general, the literature supports marker-based precision landing as a useful and effective approach for improving the final landing stage of UAV delivery, especially when it is supported by strong control logic and clear fallback handling.

2.5.5 Relevance to This Thesis

In this thesis, precision landing is handled as the last stage of the autonomous delivery mission. The earlier parts of the system, such as mission planning and waypoint navigation, are used to guide the drone along the route, while marker-based vision is used near the end to help the drone align properly and land more accurately. This matches the general approach in literature, where high-level mission execution is combined with local visual correction during the final landing stage [36] [37].

2.6 Related Work and Research Gap

Related Work relevant to this thesis spans four connected areas: navigation and waypoints, path planning, mission planning/platform integration, and precision landing. The goal of this section is to synthesize what has already been solved and identify the practical gap that remains for deployment-oriented autonomous delivery.

There is considerable literature addressing UAV navigation robustness and waypoint-following control. Survey and experimental studies report approaches for state estimation, guidance design, and disturbance-aware tracking, including wind-aware path-following and mission-state supervision support [6], [13], [14], [15], [16]. These works show clear improvements in navigation reliability through better estimation and control design. However, most of them evaluate navigation as a mostly standalone subsystem

rather than as one phase in a full delivery pipeline connected to an external mission platform and terminal landing logic.

Path-planning literature is extensive and technically rich. Existing studies cover graph-based planning, sampling-based planning, optimization/evolutionary planning, and dynamic obstacle-avoidance formulas, [5] , [17] , [18], [20], [21] , [22], [23] , [24], [25] . These contributions provide strong algorithmic foundations for route generation and replanning under constraints. Still, many works prioritize planner-level optimization and simulation evaluation, with less emphasis on practical integration with real mission-command workflows, operator control semantics, and final visual landing behavior. In this thesis, such methods are reviewed as alternatives, while the implemented delivery workflow uses platform-defined operator waypoints, mission-state supervision, and arrival verification, in AIDERS, separate path-planning logic is mainly in specific mission modes (e.g., search and rescue) rather than in the normal point-to-point delivery flow.

Mission-planning and platform-integration studies emphasize architecture, task allocation, ground control station design, telemetry exchange, and human supervision [9], [11], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35]. These works confirm that mission reliability depends heavily on coherent architecture, not only algorithm quality. They also highlight runtime supervision requirements such as pause/resume/cancel handling and communication robustness. Even so, relatively few studies combine these elements directly with real delivery execution and marker-assisted precision landing in one validated operational chain.

Marker-based and vision-based landing methods are also well established. Foundational visual-marker detection work and recent autonomous landing studies show that robust visual touchdown is feasible when detection quality, closed-loop control, and fallback behavior are designed together [3], [4], [36], [37], [38], [39], [40]. Most of this literature focuses on the terminal stage itself, which is necessary, but often treated separately from platform mission assignment, mission-state transitions, and supervision logic.

Table 4: Capability Coverage in Related Work vs. This Thesis Scope

Work family	Navigation & waypoint execution	Path planning	Mission planning + platform integration	Vision precision landing	End-to-end delivery integration
Navigation/ control-focused works ([6], [13]- [16])	Strong	Limited/contextual	Limited	Rare	Limited
Path-planning-focused works ([5], [17]- [25])	Contextual	Strong	Limited	Rare	Limited
Mission/platform-focused works ([9], [11], [26]- [35])	Contextual	Contextual	Strong	Rare	Partial
Precision-landing-focused works ([3], [4], [36]- [40])	Contextual	Rare	Rare	Strong	Limited
This thesis target scope	Integrated	Integrated at mission level	Integrated	Integrated	Strong Focus

Based on the reviewed literature, the main gap is not the absence of methods for individual components. The gap lies in practical integration. Few works present a unified, deployment-oriented pipeline that starts from external mission assignment, continues through autonomous waypoint execution and runtime supervision, and concludes with marker-assisted precision landing under one consistent mission-control architecture.

More specifically, the following integration needs remain under addressed in many prior studies:

1. Direct coupling between external mission-planning platforms and onboard mission execution.

2. Consistent bidirectional command/telemetry flow aligned with mission-state logic.
3. Runtime operator intervention semantics (pause/resume/cancel/approval) integrated with autonomous execution.
4. Terminal precision landing embedded as mission phase rather than treated as isolated demo component.

This thesis addresses this gap by integrating the above capabilities into a single autonomous delivery workflow. The system combines platform-driven assignment (AIDERS context), waypoint parsing and execution, telemetry/state reporting, supervised runtime control, and ArUco-based precision landing as part of one mission pipeline. Therefore, the contribution is primarily system-level integration and operational coherence, supported by real mission workflow implementation and evaluation, rather than proposing a new standalone path-planning or vision algorithm.

Chapter 3 Technical Background

3.1 UAV Control Foundations

Autonomous mission operation is built on core UAV foundations that include airframe dynamics, propulsion-driven stabilization, navigation/state estimation sensors, onboard power systems, and payload imaging channels. These foundations determine how high-level mission commands are physically realized in flight and how operational state is observed during mission execution.

3.1.1 Airframe, Propulsion, and Flight Stabilization

The motion of a multirotor UAV is produced by the coordinated operation of multiple electric motors and propellers. By changing the speed of individual rotors, the aircraft can move and rotate around the roll, pitch, and yaw axes. In practical UAV systems, this motion is continuously stabilized by embedded flight-control loops.

At mission level, direct control of motors is usually not required. Instead, higher-level commands (e.g. takeoff, movement toward a waypoint, or landing, etc.), are interpreted by the flight controller and converted into low-level control actions. This layered structure is a basic characteristic of autonomous UAV operation, since it allows mission software to focus on task execution while the flight controller maintains stable flight.

3.1.2 Navigation and State Estimation Foundations

Reliable autonomous flight depends on continuous estimation of the UAV state. In modern platforms, this is achieved through the combined use of internal sensors, global navigation signals, and altitude measurements. From these sources, the system derives mission-relevant information such as position, velocity, heading, and overall flight status.

Altitude may be expressed in absolute or relative form, depending on the operational need. In mission supervision, relative altitude with respect to the home point is often more practical, since it provides a consistent reference for takeoff, waypoint travel, and landing.

In addition, indicators such as GPS quality and satellite availability are useful for assessing the reliability of the navigation solution during flight.

3.1.3 Energy and Endurance Foundations

Battery-powered UAV missions are limited by the available battery, and the total duration of the flight. For this reason, battery condition is an important operational factor in autonomous missions. It affects not only how long the UAV can remain airborne, but also when recovery or return actions should be considered.

In practical systems, battery thresholds are often used to support safe mission supervision. When the available energy drops below predefined levels, the system may interrupt the mission, initiate return to home behavior, or prevent further task execution. Such measures are important for reducing risk, especially in repeated or longer missions.

3.1.4 Payload Camera and Gimbal Foundations

Visual functions in UAV systems depend on the availability of a stable imaging source. For this purpose, camera and gimbal subsystems are used to capture scene information while reducing the effect of aircraft motion on image quality. This is especially important in tasks that require visual monitoring or image-based guidance.

Depending on the platform configuration, different image streams may be available, such as a First Person View (FPV) stream or a gimbal-mounted camera stream. These provide the visual input needed for observation, monitoring, and close-range landing support. In this thesis, such visual data are particularly important for marker-based landing operations.

3.2 DJI Payload SDK (PSDK)

The DJI Payload SDK (PSDK) provides the software interface layer through which onboard mission applications interact with aircraft capabilities. In this model, mission software is executed on the onboard computing unit, while flight stabilization and low-level control remain handled by the aircraft's internal flight controller. In this way, the

PSDK acts as a bridge between high-level mission logic and the main services provided by the UAV platform.

3.2.1 Flight-Control Command Interfaces

The PSDK provides high-level flight control interfaces for actions such as takeoff, return-to-home, and guided control modes. These interfaces are used by onboard software to supervise the execution of different mission phases during autonomous operation.

At the same time, the SDK does not replace the low-level control loops of the aircraft. Instead, it allows mission software to issue control commands at a higher level, while the internal flight controller remains responsible for stable and safe flight. This separation is important because it supports the development of mission logic without requiring direct interaction with motor level control.

3.2.2 Waypoint Mission Interfaces

The PSDK also provides waypoint mission interfaces that support mission initialization, waypoint upload, mission start, pause and resume actions, mission stop, and mission-state callbacks. Through these functions, the UAV can execute missions that are defined as an ordered sequence of waypoint targets.

These interfaces are important in autonomous mission systems because they combine navigation execution with mission-state monitoring. As a result, onboard software can not only start and control a waypoint mission but also observe its progress and respond when necessary. This is particularly useful in structured delivery workflows, where mission phases must be clearly defined and supervised.

3.2.3 Telemetry Subscription Interfaces

The PSDK telemetry interface provide topic-based access to aircraft state information. Typical subscribed data include position, altitude, velocity, heading, flight status, and battery readings. These data are essential for monitoring the condition of the UAV during mission execution.

Through topic-based telemetry access, onboard software can continuously receive updated state information without sending repeated individual requests for each variable. This supports real time mission supervision and allows the software to make decisions based on the current condition of the aircraft. For this reason, telemetry access is a key part of reliable autonomous operation.

3.2.4 Live view and Camera-Manager Interfaces

The PSDK also includes live view and camera-management interfaces that provide access to video streams and camera-related control functions. These interfaces allow onboard software to receive image data from the UAV camera system and manage camera operation when required.

This capability is important for visual processing tasks, since it provides the software path through which image data become available for computer-vision methods. In this thesis, these interfaces are relevant because they support the visual input needed for marker detection and precision landing.

3.3 Platform Communication Technologies (WebSocket, UDP, JSON)

WebSocket, UDP, and JSON are widely used communication technologies in distributed software and autonomous system platforms. WebSocket provides persistent bidirectional communication, UDP offers lightweight and low-latency data transport, and JSON provides a structured and human readable format for message exchange. Together, these technologies support the transmission of commands, telemetry data, and event information between software modules and external control platforms.

3.3.1 Publisher-Subscriber

In publisher-subscriber model, one part of the system publishes data, while the other parts subscribe to the data they need. This approach allows information to be shared continuously without requiring direct repeated requests between all components. In UAV mission software, the publisher-subscriber model is especially useful for state monitoring

and event-driven operation, since multiple software modules may need access to updated aircraft information at the same time.

:

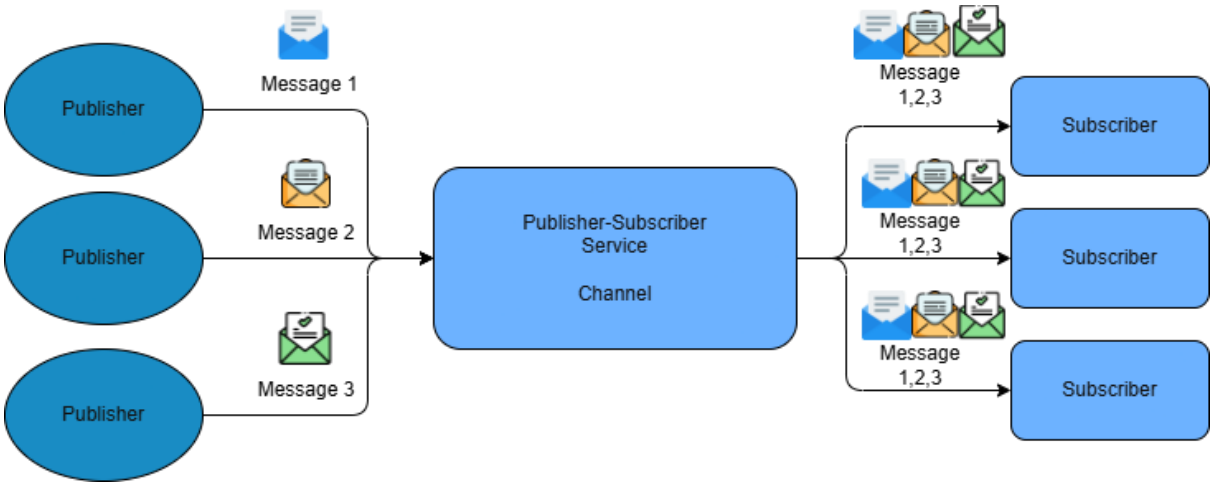

Figure 1: Publisher-Subscriber Model

3.3.2 WebSocket Protocol for Platform Sessions

WebSocket is a bidirectional communication protocol that supports persistent connections between two endpoints. Unlike traditional request-response communication models, WebSocket allows both sides to exchange messages continuously during the same session. This makes it suitable for mission systems that require real time supervision, command transmission, and monitoring updates. In UAV platforms, WebSocket is commonly used for command-and-control communication because it supports event driven interaction and continuous session management. It also allows the implementation of mechanisms such as keepalive monitoring, timeout detection, and reconnection when communication is interrupted.

3.3.3 UDP as Local Transport Layer

UDP is a connectionless transport protocol designed to transmit data with low overhead and low latency. Since it doesn't establish a persistent connection before sending data, it is often preferred in local communication scenarios where speed and simplicity are important. In robotics and UAV systems, UDP is commonly used for internal

communication between modules or processes, especially when lightweight data exchange is required. Although UDP does not provide built-in reliability mechanisms such as guaranteed delivery or retransmission, these can be handled at the application layer when necessary. For this reason, UDP is often used as a local transport solution within layered mission architectures.

Since this communication takes place locally on the same host, packet loss is extremely unlikely, so the reliability overhead of TCP is unnecessary. UDP therefore provides a lightweight and low-latency transport mechanism, which makes it suitable for fast message exchange between internal system components.

3.3.4 JSON Message Model for Commands, Telemetry, and Events

JSON is a structured, human-readable data format that is widely utilized for the exchange of information between software components. In UAV mission systems, JSON messages are commonly used to represent commands, telemetry data, and system events in a consistent format. Command messages typically contain mission control actions and related parameters, telemetry messages provide periodic updates about system state, and event messages describe discrete changes such as status update, alerts, or acknowledgments. The use of a common message model simplifies integration between different system components and improves readability during debugging and monitoring. In supervised autonomous systems, the interpretation of these messages depends on the current mission state, since the meaning and correctness of an action may vary according to the execution context.

3.4 OpenCV Foundations for UAV Vision Processing

OpenCV (Open Source Computer Vision Library) is an open-source software library widely used for image processing, computer vision, and real-time video analysis. It provides a large set of tools for handling visual data, including image transformations, filtering, feature extraction, object detection, and camera calibration. In UAV systems, OpenCV is often used as the main processing layer between raw camera input and higher-level functions such as target detection, tracking, and visual guidance.

3.4.1 OpenCV as a Computer-Vision Framework

OpenCV provides a modular framework for building computer-vision pipelines. Its functions support tasks such as frame acquisition, color-space conversion, filtering, thresholding, geometric transformations, feature extraction, and object or marker detection. Because of its flexibility and relatively efficient execution, OpenCV is widely used in embedded systems, robotics, and UAV applications.

In vision-based landing tasks, OpenCV is commonly used to process raw camera frames and convert them into useful visual information. Such information may include the location of a target in the image, its relative position, and other measurements that can later be used by the guidance control logic of the system.

3.4.2 Classical and Model-Based Detection in OpenCV

OpenCV supports different approaches for visual detection. In general, these can be into two categories: classical computer-vision methods and model-based detection methods.

Classical methods rely on image features, shapes, edges, patterns, or coded targets, and do not require training on a dataset. In contrast, model-based methods use trained machine-learning or deep-learning models to recognize specific objects or classes in incoming frames. In such workflows, OpenCV is often used for image preprocessing, result handling, and integration of the detection output into the wider perception pipeline.

Both approaches can be used in UAV vision tasks. The choice depends on the requirements of the application, such as computational cost, robustness, and the type of target that must be detected.

3.4.3 OpenCV Processing Architecture

OpenCV-based pipeline follows most of the time a sequence of processing stages. First, image frames are acquired from the camera stream. Next, preprocessing operations are applied, such as resizing, filtering, normalization, or contrast improvement. After that, the main detection or recognition stage is performed. The detected result is then used for geometric extraction and validation, so that structured measurements can be produced for the next part of the system.

This staged architecture is important because it shows that vision processing is not based on a single function call, but on a sequence of connected steps. In UAV applications, this helps improve clarity, debugging, and control over the perception pipeline.

3.4.4 Imaging Quality and Runtime Constraints

The performance of an OpenCV-based perception pipeline depends strongly on image quality and runtime conditions. Factors such as lighting variation, motion blur, camera noise, compression artifacts, target size, and viewpoint angle can affect detection performance. For this reason, preprocessing and validation stages are usually added in order to improve the reliability of the output under non-ideal conditions.

3.5 ArUco Fiducial Marker Technology

ArUco is a marker-based visual detection method used to identify coded square targets and estimate their position in the image. It is implemented within OpenCV and is widely used in robotics and UAV applications because it provides both marker identification and geometric information. In autonomous landing tasks, this makes ArUco suitable for detecting a known landing target and supporting close range alignment.

3.5.1 ArUco Marker Concept

An ArUco marker is a square binary pattern that contains a unique encoded identifier. During detection, the system searches the image for candidate quadrilateral regions and attempts to decode them according to a selected marker dictionary. If decoding is successful, the marker ID is recovered, and the corner coordinates of the marker are also obtained.

This is particularly useful because the output includes both the identity of the target and its image geometry. As a result, ArUco can be used not only to determine which marker is visible, but also to estimate the relative position of the UAV with respect to that marker in the camera frame.

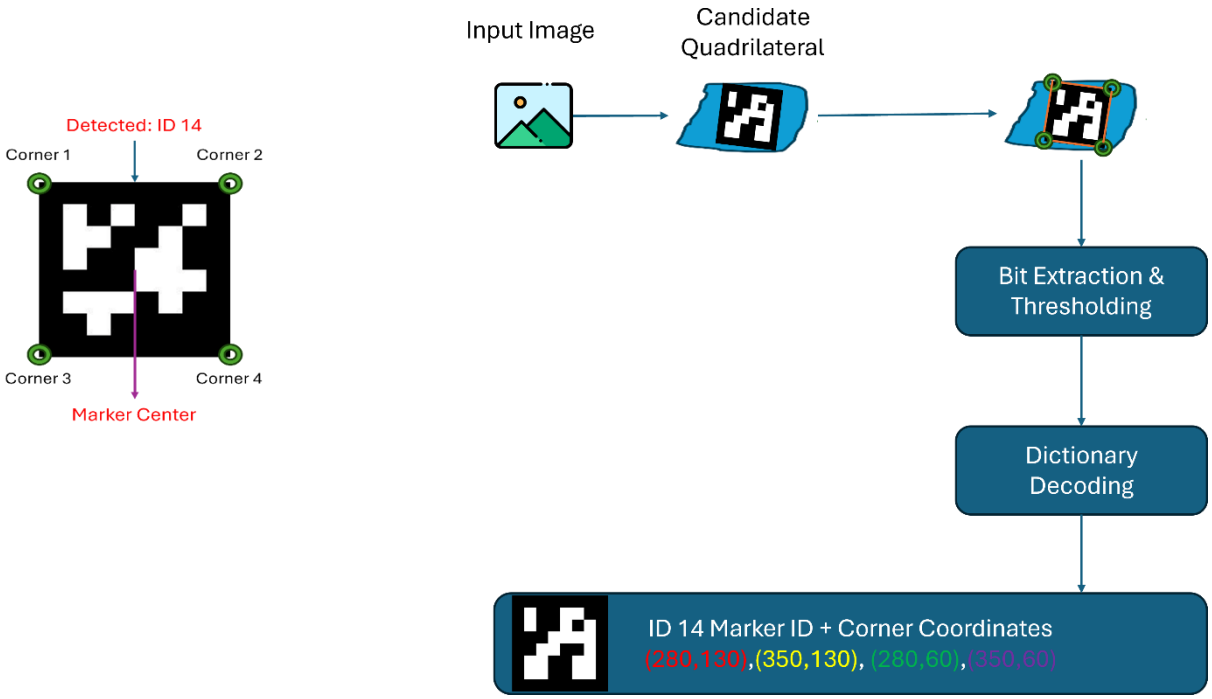

Figure 2: Conceptual illustration of an ArUco marker, showing its binary matrix, corner ordering, and the main detection stages leading to marker identification and corner extraction.

3.5.2 Dictionaries, Marker IDs, and Target Selection

ArUco markers are grouped into predefined dictionaries, such as 4x4, 5x5, or 6x6 marker families. Each dictionary contains a specific number of unique marker codes. The choice of dictionary affects the number of available marker IDs, the complexity of the encode pattern, and the robustness of detection in practical conditions.

The marker ID plays an important role in target selection. In scenes where multiple markers may appear, the system can use the decoded ID to determine whether the detected marker is the intended landing target. This helps reduce ambiguity and supports controlled operation in more complex visual environments.

3.5.3 ArUco Detection Pipeline OpenCV

In OpenCV, ArUco detection is performed through a sequence of processing steps rather than a single operation. First, the input frame is examined to locate square candidate regions that may correspond to markers. These candidate quadrilaterals are then perspective-corrected so that each one can be viewed in a normalized front-facing form. After that, the internal binary pattern of the candidate is sampled and converted into a grid of black and white cells.

The marker ID is not produced by simply counting white cells. Instead, it is determined by the specific arrangement of black and white cells inside the marker. This binary arrangement is compared against the selected ArUco dictionary, and if a valid match is found, the corresponding marker ID is returned. Once a marker has been successfully decoded, OpenCV also provides its corner coordinates, which can then be refined and used for marker-center calculation, image annotation, and relative position estimation.

In the context of UAV landing, this output is particularly useful because it provides both target identity and image geometry. The center of the detected marker and its offset relative to the image center can be used as simple visual guidance signals, helping the system estimate alignment with the landing target and generate corrective actions during the final landing phase.

Figure 3: Example of ArUco marker detection in an image frame using OpenCV, showing the detected marker boundary, decoded marker ID, and geometric reference used for localization.

3.5.4 Detection Reliability and Validation Criteria

In control related applications, marker detections are usually not accepted blindly from isolated image frames. Instead, validation criteria are often applied to improve reliability and reduce unstable behavior. Common checks include verifying the expected marker ID, confirming temporal consistency across consecutive frames, checking the geometric shape and size of the marker, and ensuring that the detection is recent enough to be trusted.

These validation steps help reduce false detection, output jitter, and unstable corrections during landing. In UAV applications, this is important because even a visually correct method can lead to poor control behavior if detections are noisy or inconsistent over time.

3.6 AIDERS Platform Technical Background

AIDERS is a platform-centered environment for mission management, live monitoring, and service orchestration in multi-system UAV operations. AIDERS is presented under the KIOS Research and Innovation Center of Excellence identity. From a technical background perspective, AIDERS can be understood as a containerized, service-oriented platform that integrates web services, communication services, robotic interfaces, data processing services, and persistent storage.

3.6.1 Institutional and System Context

In research and operational settings, mission platforms are used as the supervisory layer above field devices and onboard UAV software. In this layer, mission lifecycle management, operator interaction, monitoring dashboards, and service integration are centralized.

This architectural role is reflected in AIDERS by the presence of dedicated platform views (e.g., platform and monitoring pages), platform API routing, and operation-centered data models. As a result, AIDERS functions as an orchestration and supervision environment rather than a low-level flight-control firmware layer.

3.6.2 Service-Oriented and Containerized Architecture

AIDERS is deployed as a multi-container stack, where services are separated by responsibility. Based on the platform architecture documentation and container configuration, representative service groups include:

- web and API services
- WebSocket services
- live-stream and computer vision services
- algorithm processing services
- map-processing and geo services
- database

In such architecture, modularity and scalability are supported by service separation. Inter-service communication is performed over container networking and defined interfaces.

3.6.3 Communication Interface Layer

The platform interface layer combines REST-style service endpoints and websocket channels. REST endpoints are used for service actions and control requests across subsystems (for example mission, streaming, and vision service calls), while websocket channels are used for real-time bidirectional updates.

This dual-interface model is common in mission platforms:

- REST is used for explicit request/response actions.
- WebSocket is used for continuous asynchronous status and event exchange.

The interface layer therefore provides both control operations and real time monitoring updates

3.6.4 Data Management and Persistence Layer

A Persistent data layer is provided through PostgreSQL storage, with proxy/load-balancing components used in the database-access path. Platform entities are modeled around operations, connected assets, and mission-relevant records, enabling mission traceability and state consistency across services.

From a technical standpoint, persistent storage is necessary so that mission history, platform state, and cross-service outputs remain available beyond temporary runtime sessions.

3.6.5 Monitoring and Operational Supervision Concepts

Platform supervision in AIDERS is based on continuous status visibility of connected resources and operational-level context. Monitoring views and state fields are sued to track connectivity, mission-relevant transitions, and service-health indicators.

In mission platform architecture, this supervision layer is essential because distributed services and field assets must be observed under a unified operational context.

Chapter 4 System Architecture and Implementation

The implemented architecture and execution flow of the autonomous delivery system is organized as a distributed pipeline that connects an external mission-planning platform (AIDERS) with onboard mission execution software running on the UAV. The architecture combines mission-command reception, autonomous mission execution, telemetry and event transmission, and vision-based landing support.

At a high level, the implemented system consists of the following main components:

- Mission interface AIDERS
- Onboard delivery mission controller (Payload SDK side)
- Telemetry and event generation modules
- Camera stream and ArUco detection modules
- Precision landing controller

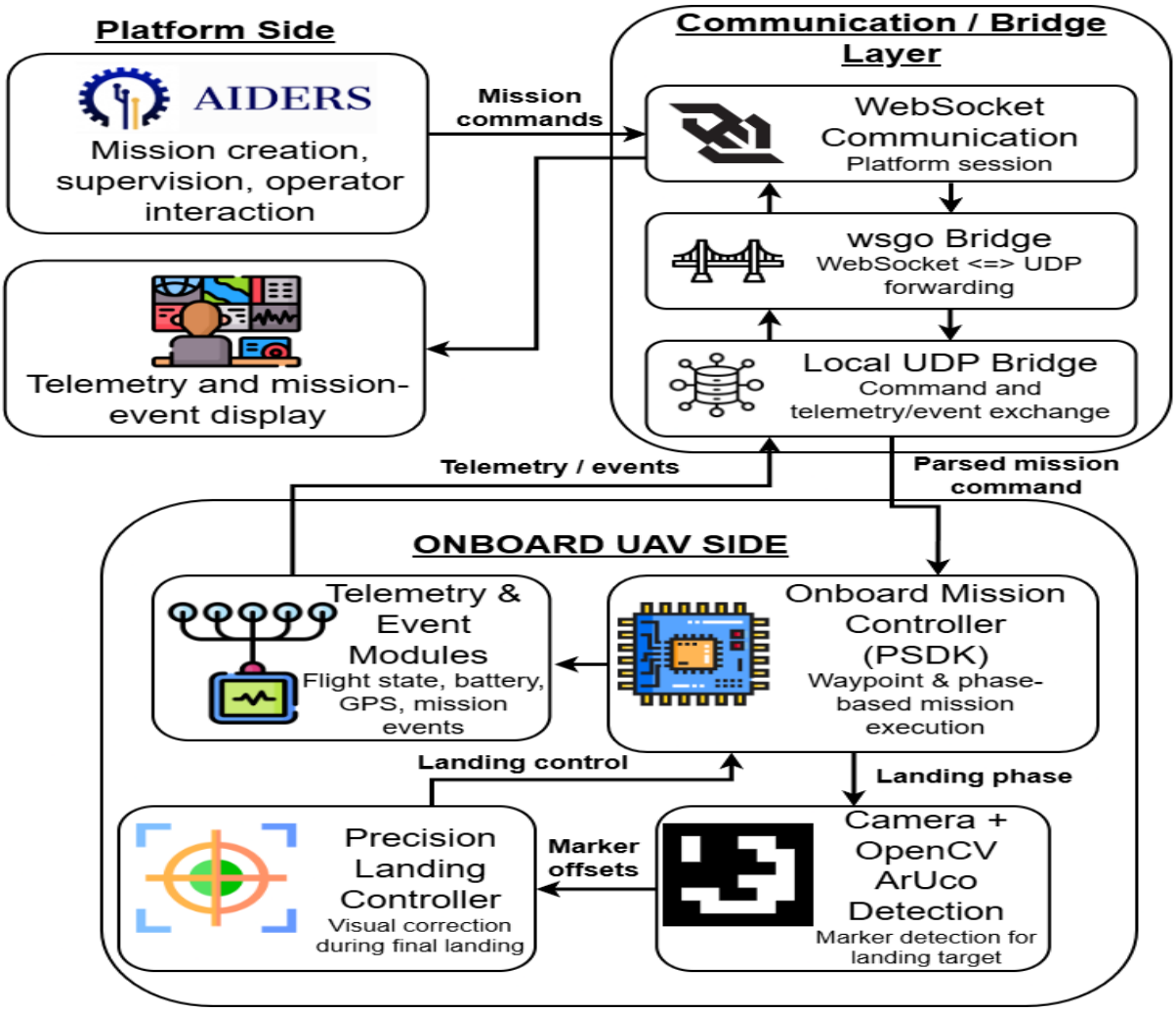

Figure 4: End-to-end architecture of the autonomous drone delivery system.

4.1.1 End to End Workflow

The system workflow starts from mission assignment at platform level and ends with onboard mission execution, landing behavior, and mission-state feedback.

1. A mission command is created in AIDERS and transmitted through a WebSocket connection.
2. The wsgo bridge receives the JSON message and forwards it to the local UDP command endpoint.
3. The onboard UDP bridge receives the packet, parses the mission information, and activates the mission-execution logic.
4. The mission controller carries out the delivery workflow using waypoint-based and phase-based control.

5. At the delivery waypoint, the mission controller transitions from normal waypoint navigation to the precision landing routine, where ArUco marker detections from the camera stream are used to guide final alignment and descent. If marker guidance is unavailable or a control failure occurs, the system falls back to a regular landing.
6. During mission execution, telemetry data and mission-related events are continuously generated onboard.
7. These outgoing telemetry and event messages are sent through local UDP to wsgo, which then forwards them back to AIDERS through WebSocket.

In practice, command handling and telemetry transmission run continuously and at the same time. This allows the system to remain responsive to commands while also providing live monitoring data to the platform.

Algorithm 4.1 End-to-End Mission Command and Delivery Execution

Input: Mission and approval JSON messages from AIDERS

1: **procedure** COMMAND INGESTION AND FORWARDING
2: Receive WebSocket message through the bridge
3: Validate the incoming JSON format
4: Forward valid command to onboard UDP endpoint (127.0.0.1:5005)
5: **end procedure**

6: **procedure** ONBOARD COMMAND HANDLING
7: Parse incoming UDP JSON packet
8: If operatorApproval exists, forward approval response to mission controller
9: Read mission command type (START / CANCEL / PAUSE / RESUME)
10: Apply control action for CANCEL, PAUSE, or RESUME
11: **end procedure**

12: **procedure** MISSION CONFIGURATION

13: Parse gpsInput waypoints and mission options

14: Map first waypoint, gpsInput[0], to pickup point

15: Map second waypoint, gpsInput[1], to delivery point

16: Map third waypoint, gpsInput[2], to return point, or use home fallback

17: Launch mission execution asynchronously

18: **end procedure**

19: **procedure** DELIVERY LANDING EXECUTION

20: Navigate to the delivery waypoint

21: Start marker-assisted precision landing if enabled

22: Read latest ArUco detection from the vision module

23: While landing is not completed do

24: Read the latest accepted ArUco detection from the shared vision state

25: Compute the horizontal and vertical marker offset from the image center

26: If the marker is outside the accepted center tolerance, send joystick velocity commands to reduce

the offset

27: If the marker is centered within tolerance, continue controlled vertical descent

28: If no valid marker is available for a timeout period, enter search or fallback landing behavior

29: end While

30: **end procedure**

31: **procedure** TELEMETRY AND RUNTIME FEEDBACK

32: Publish telemetry and mission-state updates to platform

33: Continue command listening while mission execution runs

34: **end procedure**

Output: Mission state transitions, telemetry stream, and delivery completion/failure status

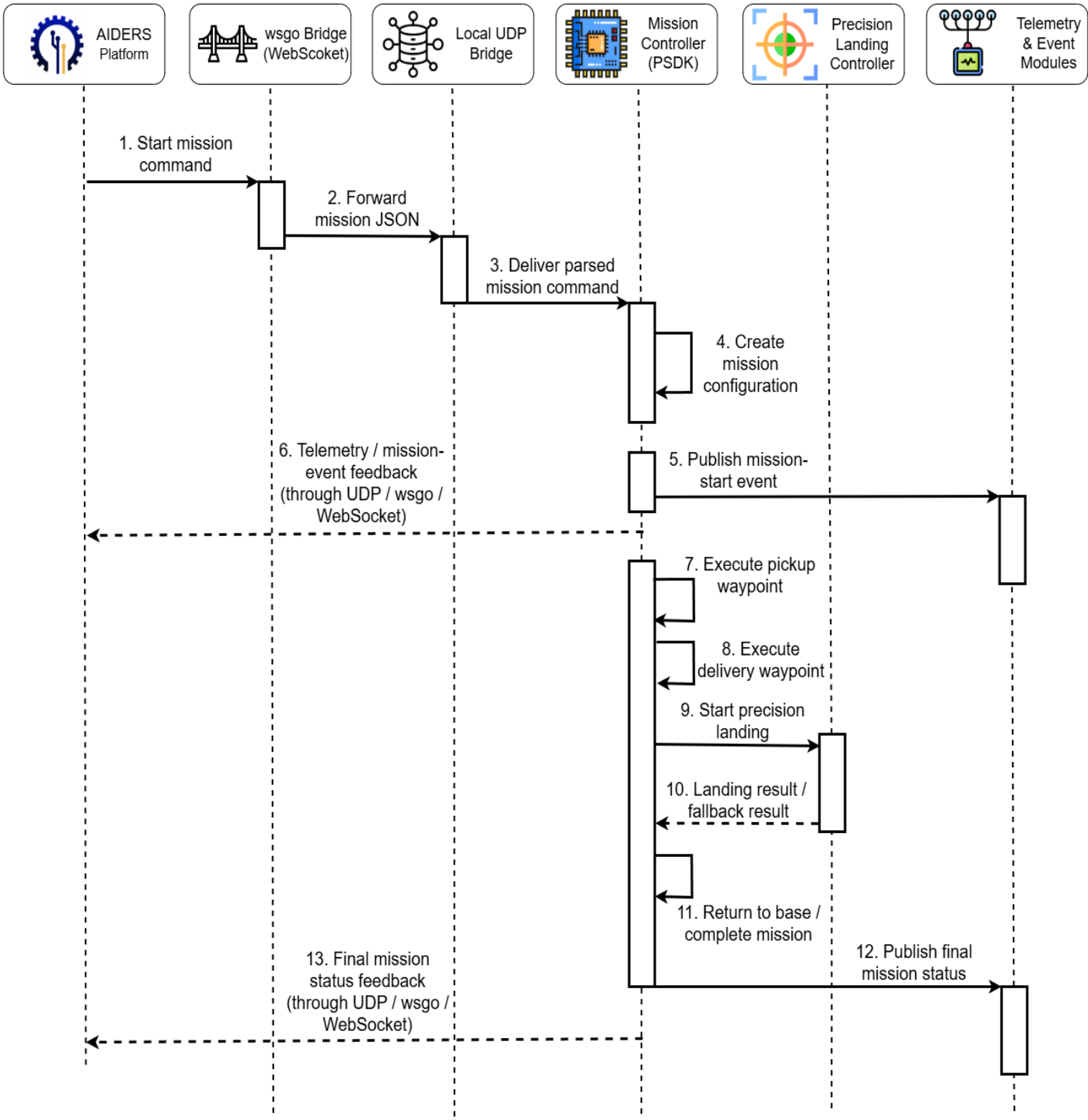

Figure 5: Sequence diagram of mission command handling, execution, and telemetry feedback.

4.1.2 Architecture Boundary

A clear separation is maintained between the external platform and the onboard UAV components.

- External side: AIDERS is responsible for mission creation, mission supervision, and operator interaction. Through the platform, the operator defines the mission, sends commands, and monitors the progress of the UAV during execution.
- Onboard side: The onboard system is responsible for receiving the mission data, executing the mission, generating telemetry, event messages, and running the landing-related control logic during flight.

This separation is important because it keeps the system structure clear. The platform is responsible for mission management and supervision, while the onboard side is responsible for execution and aircraft-side behavior during runtime. It also makes the architecture easier to maintain and extend, since changes on one side do not directly affect the other.

4.2 AIDERS Integration in the Implemented Workflow

AIDERS is integrated as the mission management and supervision endpoint of the implemented system. In this integration, mission requests and control actions are received from platform channels, while UAV telemetry and mission/operator events are returned through the outbound communication path.

4.2.1 Mission Assignment and Waypoint Source

Mission geometry is provided by the platform as waypoint data in JSON payload form. The onboard parser reads waypoint entries from the "gpsInput" field and extracts navigation parameters used by the mission controller (latitude, longitude, altitude, and speed).

In the implemented delivery mapping, waypoint order is interpreted as mission intent:

- First waypoint: pickup location
- Second waypoint: delivery location,
- Third waypoint: return/charge location.

Mission repetition flags are also received from command fields and mapped to mission configuration before execution. Precision landing is initialized as enabled by default on the onboard side. It is overridden only if a valid "enablePrecisionLanding" Boolean is explicitly provided in the incoming command payload.

Table 5: Mission payload mapping from AIDERS to the onboard mission configuration.

Payload field	Description	Onboard interpretation
gpsInput[0]	First GPS waypoint received from AIDERS	Pickup location
gpsInput[1]	Second GPS waypoint received from AIDERS	Delivery location
gpsInput[2]	Third GPS waypoint received from AIDERS	Return / base / charging location
latitude	Latitude coordinate of each waypoint	Stored as the waypoint latitude
longitude	Longitude coordinate of each waypoint	Stored as the waypoint longitude
altitude	Target flight altitude for the waypoint	Stored as the waypoint altitude

speed	Desired travel speed for waypoint navigation	Stored as the waypoint speed
repeat	Number of times the mission should be repeated	Stored in the mission configuration
enablePrecisionLanding	Optional flag for precision landing behavior	Determines whether precision landing is attempted during the landing phase

4.2.2 Platform Command Types Used

The integrated command interface supports mission lifecycle control and operator-interaction signaling

Mission lifecycle control is received through the "missionCommand" object and includes:

- Start (GO) / PAUSE / RESUME / CANCEL

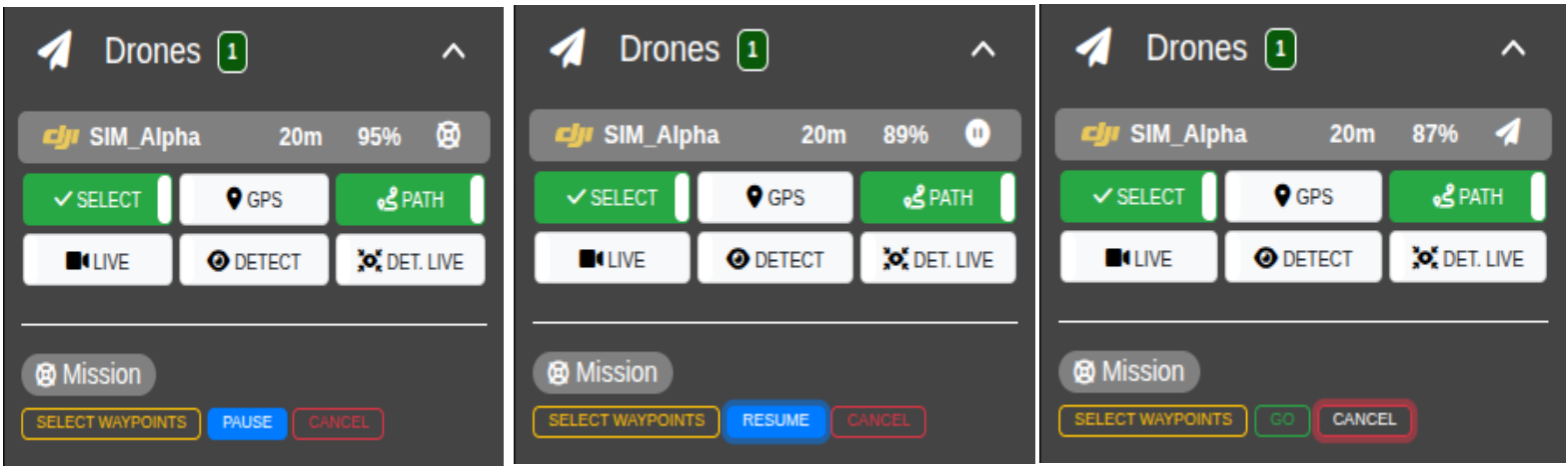

Figure 6: AIDERS Operator buttons interaction

Operator interaction signals are received through "operatorApproval" payloads. These approvals are used at gated transition points in the workflow, such as controlled continuation after landing.

Mission options, including “repeat”, are interpreted during command parsing and mapped to mission configuration parameters before execution begins. Precision-landing behavior follows a default enabled policy with optional command level override.

Table 6: Mission command mapping used by the onboard control layer.

missionCommandValue	Command meaning	Onboard handling behavior
0	Start mission	Parses the received mission payload, creates the mission configuration, and begins mission execution
1	Cancel mission	Stops the active mission workflow and prevents further phase execution
2	Pause mission	Temporarily blocks or pauses the active mission execution
3	Resume mission	Continues the mission from the paused or waiting state

4.2.3 Telemetry and Event Return Path

Telemetry is generated onboard and returned to the platform continuously during runtime. The telemetry payload includes navigation, flight, and health indicators such as latitude, longitude, altitude, velocity, heading, gimbal angle, GPS quality indicators, drone state, and battery percentage.

The system also sends event-based JSON messages. These messages are used to report mission related updates and operator-relevant events, such as mission status changes.

The outgoing data, follow the reverse communication path:

- Onboard telemetry and event producer. Then local UDP telemetry channel. Then wsgo bridge. Finally, platform WebSocket session.

This allows AIDERS not only to send mission commands, but also to act as a alive monitoring endpoint throughout the mission.

4.2.4 Integration Scope Statement

The integration scope of this work is limited to using AIDERS as an external platform and connecting its services to the onboard execution system.

No modifications were made to the AIDERS source code for the implementation of the delivery system pipeline. The implementation work was focused on the bridge modules and the Payload SDK side, where command reception, mission execution, telemetry transmission, and landing logic were developed.

4.3 Communication Bridge Implementation

4.3.1 WebSocket-UDP Bridge Design (wsgo)

The wsgo bridge is responsible for connecting the platform's WebSocket communication with local UDP interfaces used by the onboard system. Its main purpose is to act as an intermediate communication layer between AIDERS and the Payload SDK side modules. In this way, platform-side communication remains separated from the onboard mission-execution logic, while the overall communication path stays simple and lightweight.

At startup, wsgo loads runtime parameters (platform WebSocket URL, UDP command target, and UDP telemetry listener settings) through environment-variable overrides with default values. The bridge then opens an outbound WebSocket client connection toward the platform endpoint, and a local UDP listener for telemetry/events.

In the command direction, text WebSocket frames are validated as JSON objects and forwarded to the local UDP command endpoint. In the monitoring direction, UDP telemetry/event JSON payloads are read from the local listener and published back to the platform through the active WebSocket session.

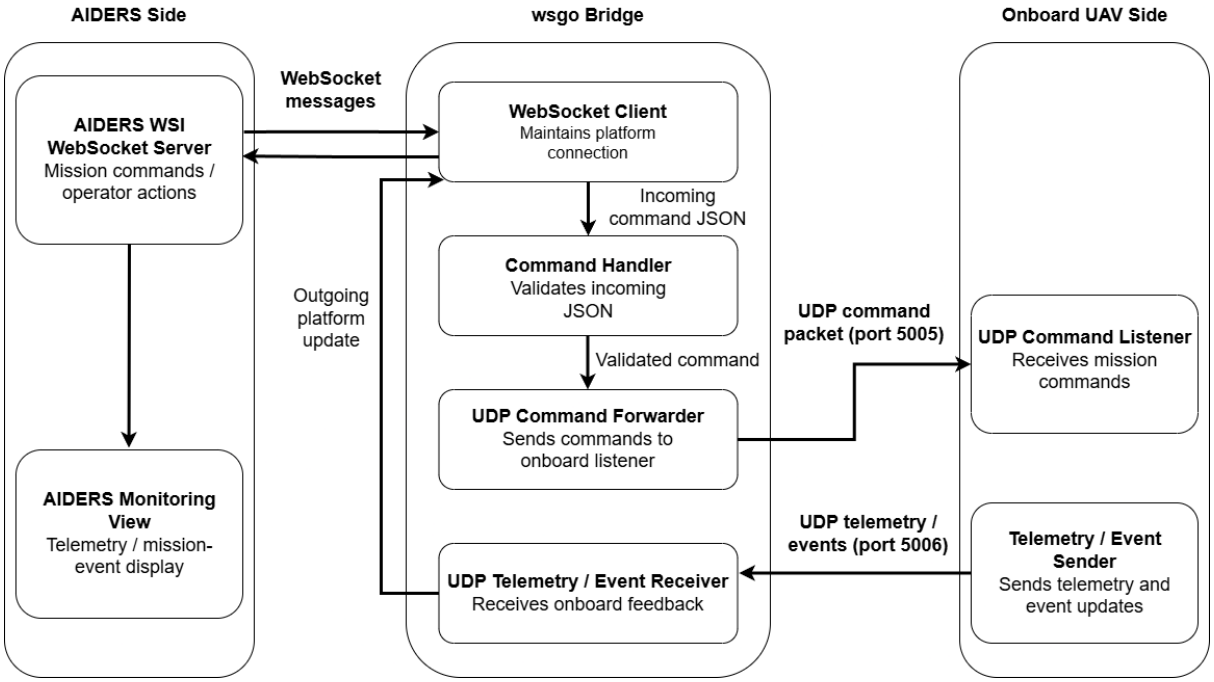

Figure 7: Communication bridge between AIDERS WSI, wsgo, and onboard UDP interfaces for mission command forwarding and telemetry/event feedback.

4.3.2 Onboard UDP Bridge and Command Handling (udp_bridge)

On the onboard side, the UDP bridge initializes local communication and handles command reception together with telemetry forwarding. Mission-control packets are received on the local command endpoint, decoded, and mapped to mission actions (start, cancel, pause, resume). Mission execution is dispatched asynchronously so the command channel remains responsive during long mission phases.

The command types are interpreted as:

- 0: start mission execution

- 1: cancel active mission
- 2: pause waypoint mission
- 3: resume waypoint mission and release any pending approval wait

4.3.3 JSON Payload Parsing and Mission Field Mapping

Mission payload parsing is centered on “missionComand” and “gpsInput”:

- gpsInput entries are parsed into (latitude. longitude, altitude, speed, repeat) waypoint tuples.
- missionCommand.repeat field is checked before use. If its value is less than or equal to zero, it is replaced with 1. It is also limited to the valid uint16_t range to avoid invalid values during execution.

Waypoint mapping for delivery mode is positional:

- waypoint 1 => pickup point
- waypoint 2 => delivery point
- waypoint 3 => return/charge point.

If only one waypoint is received, the implementation executes a regular waypoint mission path instead of the delivery-phase workflow.

Algorithm: UDP command handling

1. Open UDP listener on 127.0.0.1:5005.
2. Repeat while running:

   Receive packet;

   Parse packet as JSON;

   If invalid then log error and continue.
3. If operatorApproval exists:

   Forward approval to mission controller and continue.
4. If missionCommand is missing:

   Ignore packet and continue.

5. Read missionCommand.missionCommandValue (default: START=0):
   CANCEL =>set cancel flag, stop waypoint mission, release approval waits as denied.
   PAUSE => pause active waypoint mission.
   RESUME => resume waypoint mission, release approval waits as approved.
   START => parse gpsInput and mission options;
   sanitize repeat (min 1, clamp to uint16);
   set precision landing (default enabled, optional override);
   map waypoint[1]=pickup, waypoint[2]=delivery, waypoint[3]=return;
   if fewer than 2 waypoints, reject;
   if no mission is running, launch mission asynchronously.
6. Continue listening for new commands while mission execution runs in background.

4.4 Onboard Mission Controller Implementation (Payload-SDK Side)

4.4.1 Mission Configuration Model and Waypoint Structure

The delivery controller uses a structed mission model (DeliveryMissionConfig) composed of JSON that looks like:

```
{
  "pickupPoint": {
    "latitude": 35.145200,
    "longitude": 33.412800,
    "altitude": 25.0,
    "speed": 3.0,
    "repeat": 1
  },
  "deliveryPoint": {
    "latitude": 35.145800,
    "longitude": 33.413500,
```

```
    "altitude": 25.0,
    "speed": 3.0,
    "repeat": 1
  },
  "returnPoint": {
    "latitude": 35.145000,
    "longitude": 33.412300,
    "altitude": 25.0,
    "speed": 3.0,
    "repeat": 1
  },
  "autoReturn": true,
  "enablePrecisionLanding": true,
  "cruiseAltitude": 25.0,
  "missionRepeats": 1,
  "lowBatteryThreshold": 22
}
```


Each waypoint entry uses the same waypoint schema (latitude, longitude, speed, repeat), while mission-level fields control repetition, return behavior, energy safety, and precision-landing policy.

4.4.3 Delivery Phase State Machine

The delivery workflow is implemented as an explicit phase sequence:

1. fly to pickup waypoint
2. pickup operations (stop waypoint mission, land, wait approval, take off)
3. fly to delivery waypoint
4. delivery operations (approach, landing, wait approval, take off)
5. return behavior (specific return waypoint that works as home-point/charge-point)

Phase control includes repeated cancelation checks, arrival checks with horizontal-error tolerance, and flight-status verification before phase transitions. Delivery missions with multiple repeats are executed as a series of single-run delivery sequences.

4.4.4 Operator-Gated Transitions and Mission Commands

Operator-gated transitions are implemented through explicit approval waits:

- pickup departure approval
- delivery departure approval
- charge-recovery resume approval

When approval is requested, an operator event is published to the platform and the mission thread blocks until approval/denial is received. Approval messages are delivered through the bridge command path and synchronized using shared approval-state variables.

During the approval waiting periods, telemetry state is forced to a paused mission label, so the platform reflects hold state semantics consistently.

4.4.5 Battery-Aware Return-to-Charge and Recovery Behavior

Battery-Aware logic is applied at multiple checkpoints (mission start and in-flight/arrival transitions). If battery percentage is at or below the configured threshold, then the current mission phase is interrupted safely and the controller transitions to return-to-charge mode. The UAV navigates to the return/charge waypoint and executes landing. After landing, a resume approval gate is activated. When operator approval is received and takeoff is confirmed, the mission continues from the stored pre-return-to-charge phase instead of restarting from the beginning.

4.5 Precision Landing Implementation

4.5.1 Camera Stream Pipeline and Frame Flow

Camera frames are received through the livestream callback path and processed in a staged pipeline:

- frame geometry/payload validation
- Optional resizing of the image
- Optional ArUco detection if its enabled
- Store ArUco detection in a shared variable so the landing controller can read it in real time.
- Stream forwarding and optional preview output

In the integrated workflow of this thesis, ArUco detection is enabled by default through the startup script (start_all_in_one.sh) when the livestream process is launched. The stream module reads this setting at startup and then keeps the same detection mode during execution. Therefore, ArUco operation is startup-configured rather than continuously reconfigured while the mission is running.

4.5.2 OpenCV and ArUco Detection Module Integration

The detector module is initialized with configurable ArUco parameters (dictionary, target marker ID, marker-size limits, and temporal confirmation constraints). The stream controller publishes the latest detection output to a shared thread-safe state (PublishArucoDetection), which is consumed by the precision landing controller.

Detection output includes:

- Marker presence flag
- Marker ID
- Image-plane center offsets (offsetX, offsetY)
- Marker size and frame metadata

4.5.3 Marker-Offset to Control-Action Logic

The execution of PrecisionLandingOnHelipad function implements vision-based control using joystick velocity commands:

- Obtain joystick authority. Joystick authority means the flight controller grants the onboard application permission to send direct motion commands (e.g., horizontal velocity and vertical descent rate). Without this control permission, the precision-landing module cannot apply marker-based corrections and must fall back to a safer non-guided landing behavior.
- Set velocity-control joystick mode
- Read latest shared ArUco detection
- Convert image offsets to body-frame horizontal velocity commands

- Apply clamp, so horizontal velocity is limited and cannot exceed 0.8 m/s and deadband logic, to treat small marker offsets near the image center as zero, so the UAV does not jitter when it is almost aligned.
- Trigger controlled descent after stable centered detection

If marker data are not available or stale, the controller enters a search behavior and applies slow descent after search timeout. Control updates run at a fixed period until landing completion conditions are met.

4.5.4 Landing Completion and Fallback Behavior

Precision landing completion is triggered when descent is active, and altitude reaches the configured final threshold. At that point, the controller executes force-landing and waits for landed-state confirmation.

Fallback logic is explicitly implemented for critical failure paths (for example, joystick authority failure or joystick command execution failure). In such cases, the controller switches to regular force landing and still attempts safe completion.

Result metadata (success, usedFallbackLanding, marker-detection history, elapsed time, and reason string) are logged for runtime traceability.

4.6 Telemetry and Runtime Monitoring Implementation

4.6.1 Telemetry Generation and Published Fields

Telemetry is generated periodically on the onboard side and forwarded through the UDP-to-WebSocket bridge path. Published fields include:


```
{
  "telemetry": {
    "latitude": 35.14531,
    "longitude": 33.41302,
    "altitude": 21.4,
    "velocity": 2.3,
    "heading": 87.5,
```

```
    "gimbalAngle": -45.0,
    "gpsSignal": 5,
    "satelliteNumber": 18,
    "homeLatitude": 0.0,
    "homeLongitude": 0.0,
    "droneState": "In_Mission",
    "batteryPercentage": 62
  }
}
```


State-label generation combines raw flight status with the mission callback state and reports an explicit paused state while waiting for operator approval.

4.6.2 Mission Events and Operator Notifications

Besides periodic telemetry, the onboard software can transmit auxiliary mission-notification messages for operator awareness (e.g., approval wait and low-battery return-to-charge conditions). These messages are sent over the same outbound communication path as telemetry.

In the current integrated platform flow, this notification channel is treated as secondary support information, while the primary monitored and persisted runtime channel is the telemetry stream described in Section 4.6.1. For this reason, system analysis and reporting are centered on telemetry behavior.

4.6.3 Runtime Diagnostics

Runtime diagnostics are embedded at multiple layers:

- Waypoint event/state logging
- Mission-phase logs and battery checkpoints
- Flight-status snapshots with mission context
- Communication-path error logging (JSON, socket, forwarding)
- Stream callback stage/frame diagnostics

- Crash-signal handlers with backtrace output in the livestream application entry point.

Chapter 5 Experimental Results

The system analysis was divided into five parts:

- Experimental Setup
- Mission execution Analysis
- Communication and Telemetry Analysis
- ArUco marker detection Analysis
- Precision Landing behavior analysis.

5.1 Experimental Setup

The system consists of the AIDERS platform as the mission-planning and supervision interface, the wsgo bridge for WebSocket and UDP communication, and the DJI Payload SDK onboard modules for mission execution, telemetry publishing, camera streaming, ArUco detection, and precision landing control. The delivery mission profile uses three mission waypoints. The first waypoint represents the pickup location, the second waypoint represents the delivery location, and the third waypoint represents the return or charge-station location. The mission command is created through the AIDERS interface and transmitted to the onboard system as a JSON payload. After parsing, the onboard mission controller executes the delivery sequence using waypoint navigation and phase-based mission logic.

Table 7: Mission Waypoints and their roles in the mission

Waypoint from AIDERS	System interpretation	Role in the mission
Waypoint 1	Pickup point	First mission destination that works as the pickup location
Waypoint 2	Delivery point	Delivery destination and precision landing target area
Waypoint 3	Return / charge point	Final return point and recovery Destination when battery recovery is required

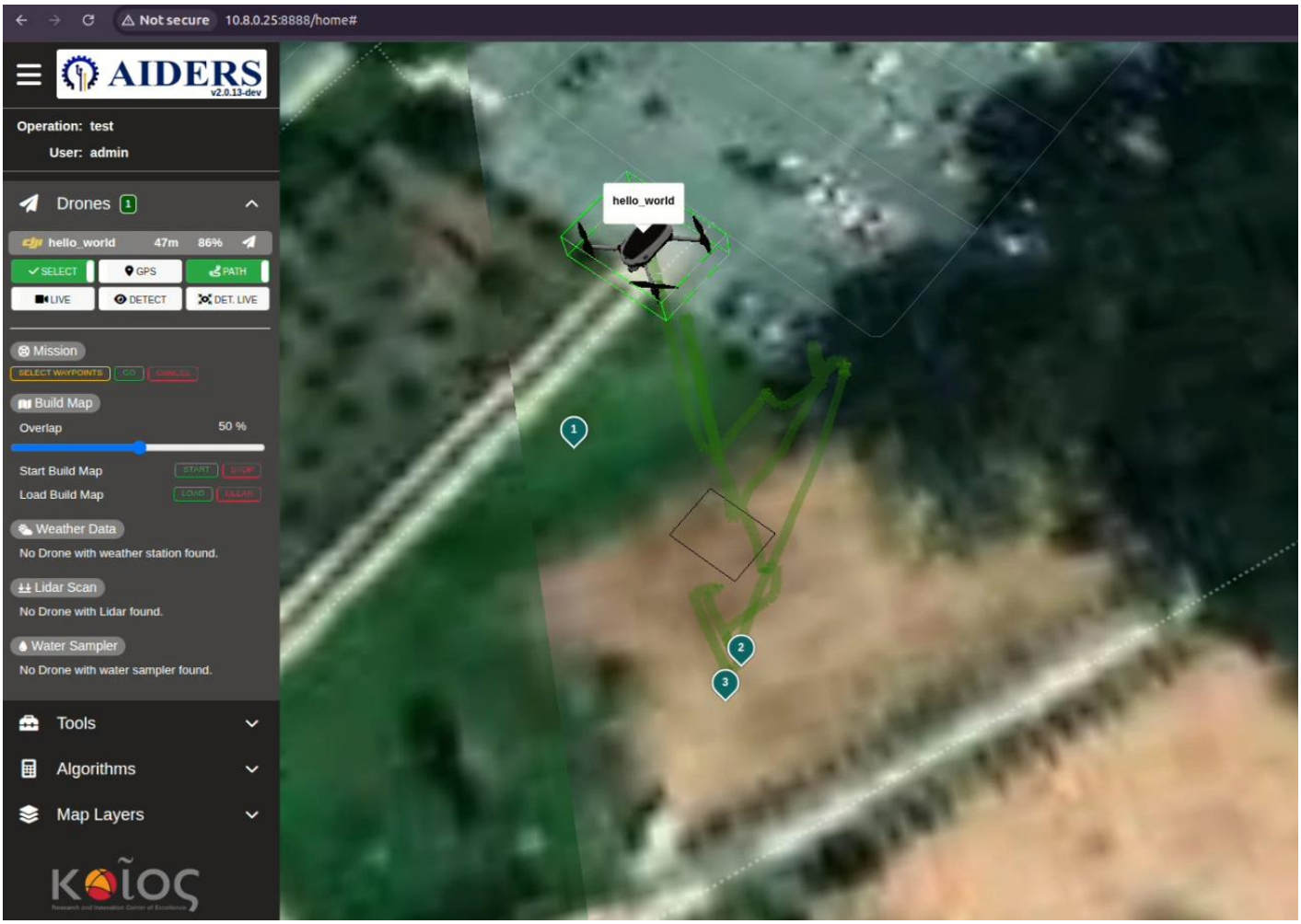

Figure 8: Outdoor mission test area/map-view showing pickup, delivery, and return points. Also, AIDERS mission interface showing the selected waypoints before mission dispatch.

For the vision part of the evaluation, an ArUco marker was used as the landing reference. Recorded flight footage was also processed offline using the same detection logic as the onboard system, so that marker-detection behavior could be evaluated under controlled video analysis conditions.

Hardware:

DJI Matrice 350 RTK, with the NVIDIA Jetson Xavier for the onboard control, alongside the DJI Zenmuse H20T camera.

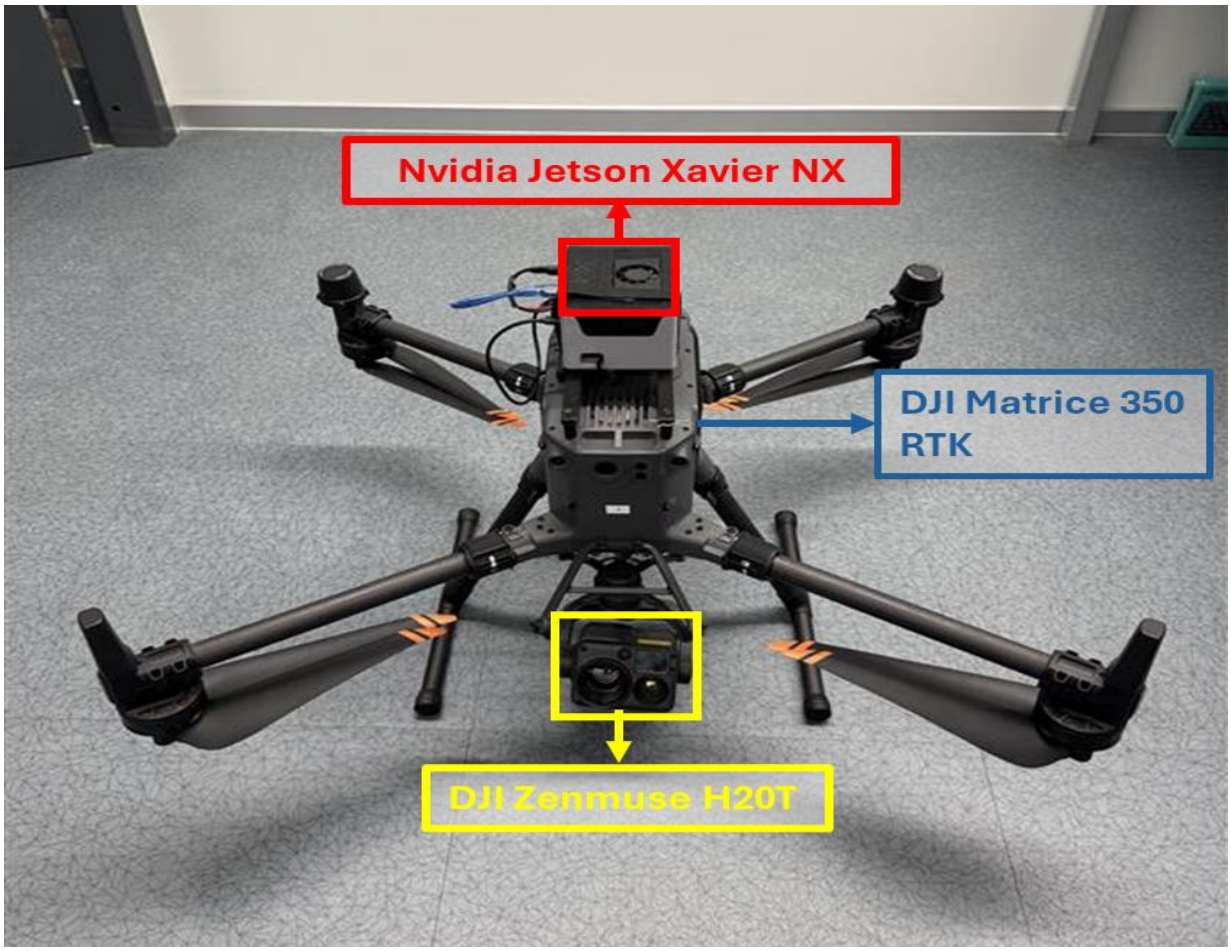

Figure 9:Hardware used for experiment

5.2 Mission Execution Analysis

This experiment evaluates whether the system can execute a delivery mission after the operator defines the mission waypoint in AIDERS.

The tested mission followed the delivery workflow implemented in chapter 4. First, the operator selected the pickup, delivery, and return points through the AIDERS interface. The platform then transmitted the mission command to the onboard system through the WebSocket to UDP bridge. On the UAV side, the waypoints received were parsed and mapped into the internal delivery mission model. The mission controller then executed the flight workflow using waypoint navigation and phase-based control.

The executed delivery sequence follows these main phases:

1. Receive mission command from AIDERS
2. Parse gpsInput waypoint and mission options
3. Fly to the pickup waypoint
4. Stop Waypoint mission and land for pickup
5. Wait for operator approval before departing the pickup point
6. Move to the delivery waypoint location
7. Execute marker assisted precision landing or fallback landing behavior
8. Wait for operator approval before departing the delivery point
9. Return to the configured return/charge waypoint.

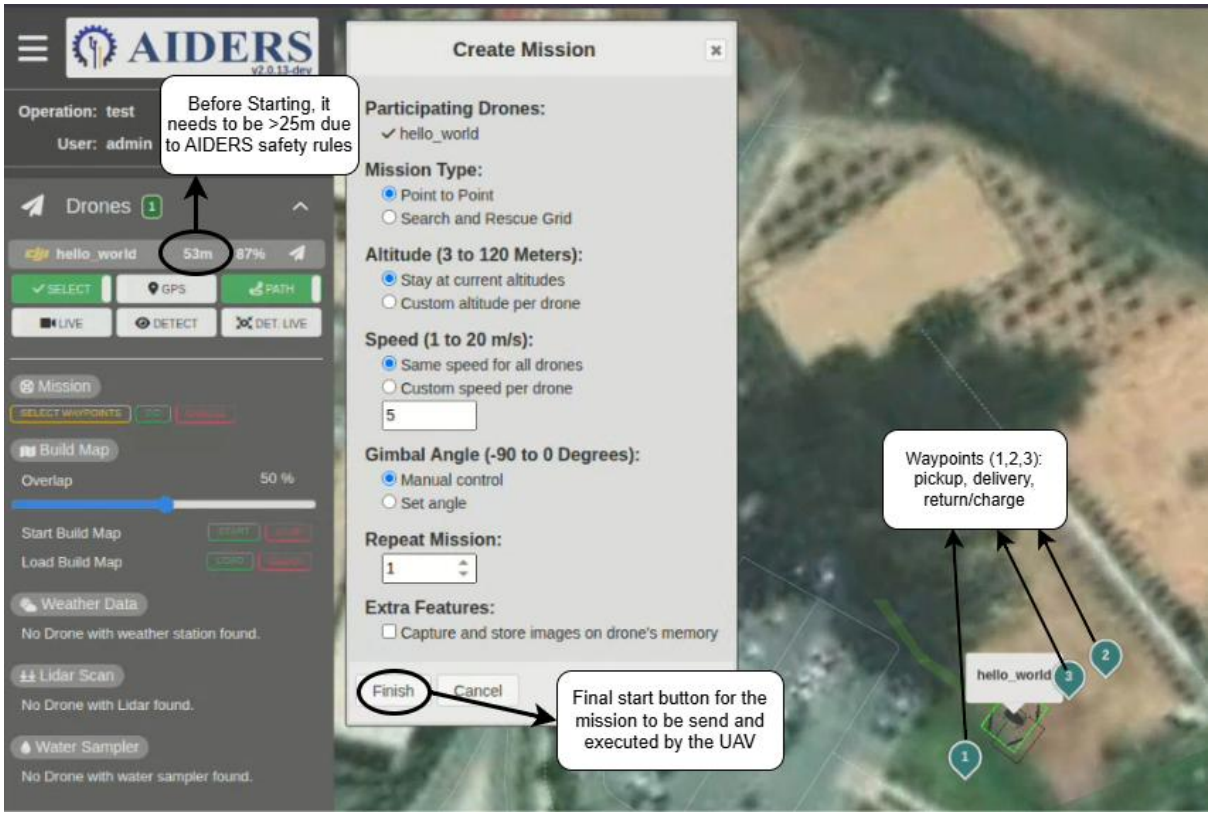

Figure 10: Placement of mission waypoints and interface right before final command sent to start the mission.

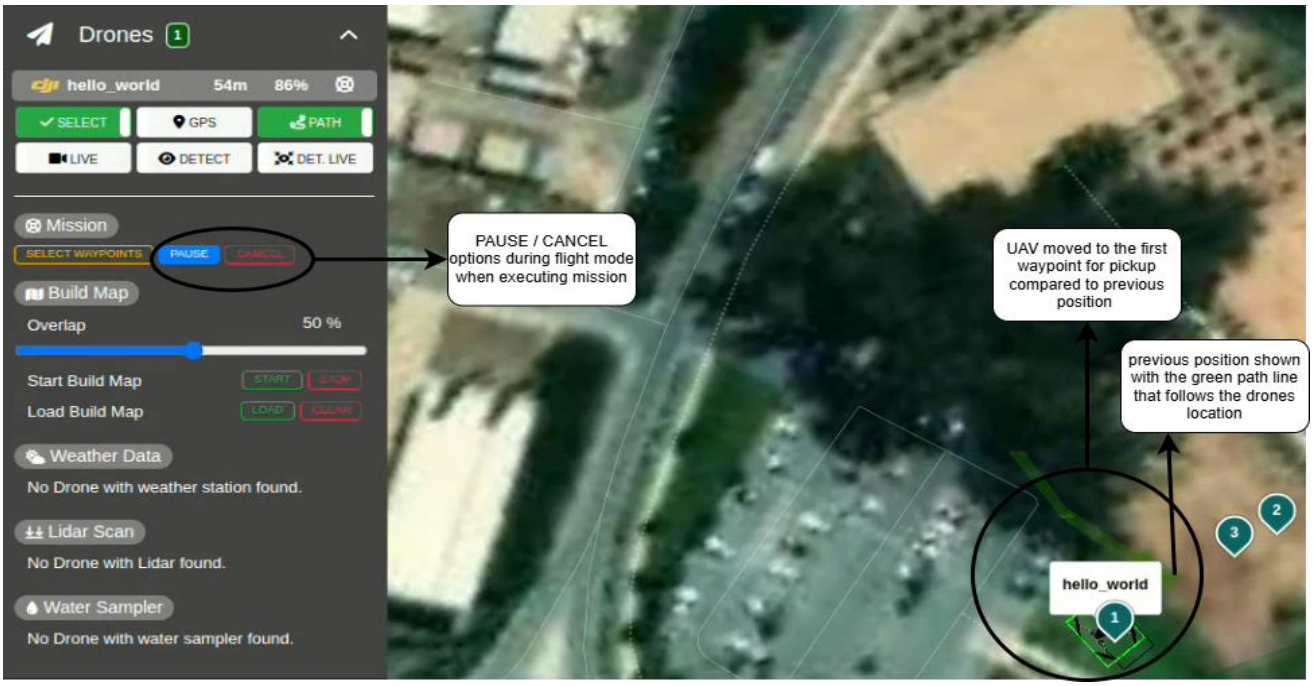

Figure 11: Showing Available PAUSE / CANCEL commands to send and also showing the first waypoint execution and the navigation path the UAV took to make it.

Figure 12: Showing landing on few waypoints and the precision landing searching for the ArUco marker that made the UAV move left and right accordingly.

5.3 Telemetry Metrics Analysis

During the live repeated delivery mission test, telemetry data were exported from the MySQL database used by AIDERS and contain 3026 records, covering approximately 18 minutes and 7 seconds of mission activity. The purpose of this analysis is to show the system records useful information during execution, including battery behavior, altitude changes, flight-state transitions, position movement, and GPS quality.

Table 8: Summary of telemetry dataset used for mission analysis.

Metric	Value
Number of telemetry records	3026
Test duration	18 min 7 s
Battery range	91% to 66%
Maximum velocity	6.26 m/s
Average velocity	0.82 m/s
Satellite count range	19 to 28
Average satellite count	26.07
Maximum displacement from start	34.68m
Mission states observed	Landed, Flying, In_Mission, Paused_Mission

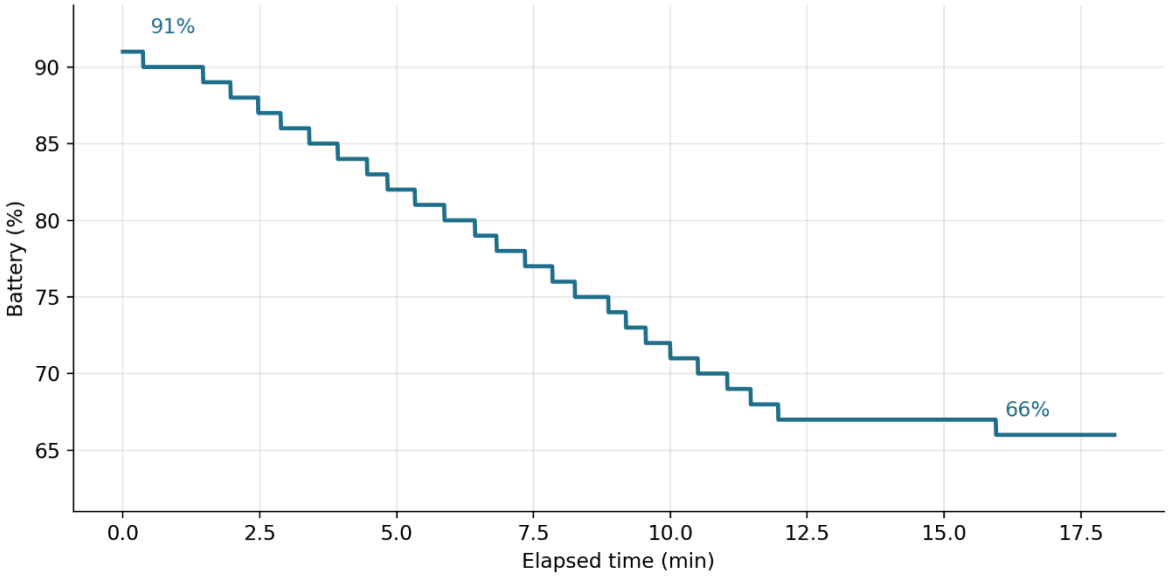

Figure 13: Battery levels during the 2-repetition mission

The above figure shows the battery percentage during the mission test. The battery decreased from 91% to 66%, which reflects the energy cost of repeated takeoff, waypoint flight, descent/landing behavior, and waiting phases.

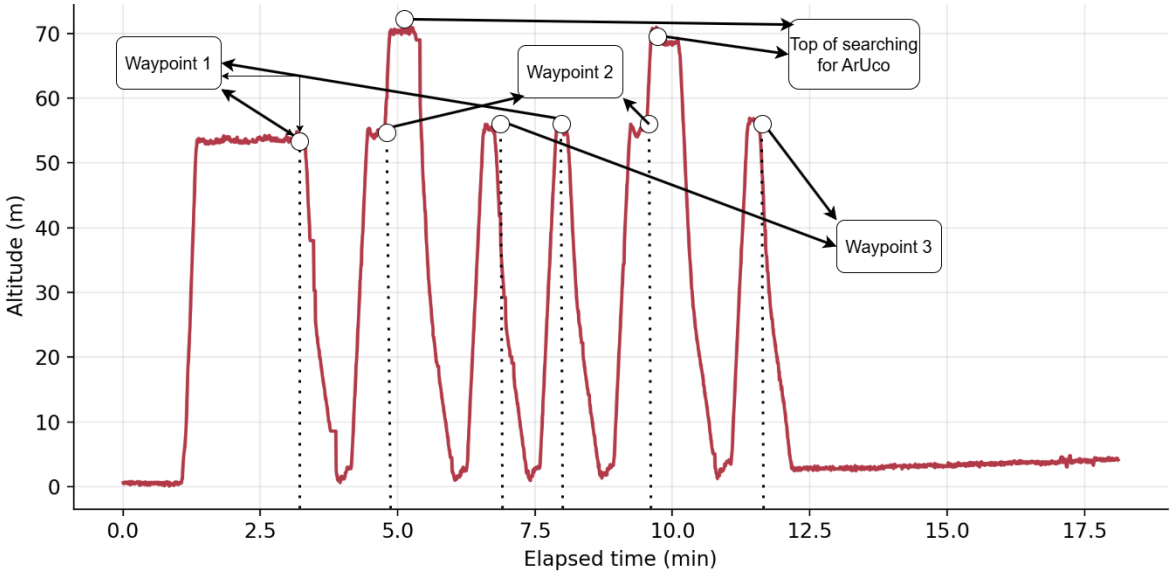

Figure 14: Altitude profile during mission

The above figure shows the altitude profile of the UAV during the delivery mission for both repetitions. During this time, the phases it clear. At start, the UAV was taken manually to the acceptable height (above 25 meters) for the AIDERS to let it accept missions. Then, the waypoints were placed and the mission started by the operator at around minute 2.5. Arrival at the first waypoint at minute 3 and then the descend until minute 4, and after that the operator pressed resume for it to ascend back to altitude 55 meters like before. At the arrival of the second waypoint at minute 4.5, the UAV ascends 15 meters before it goes into search for the ArUco marker for the detection and precision landing, then it descends while searching for the marker until minute 6. Again, after the operator's resume it goes back to 55 meters and starts navigating to the third waypoint where it does the same landing at minute 7.5. At that time the first delivery mission was over and then the second one was carried out the same way.

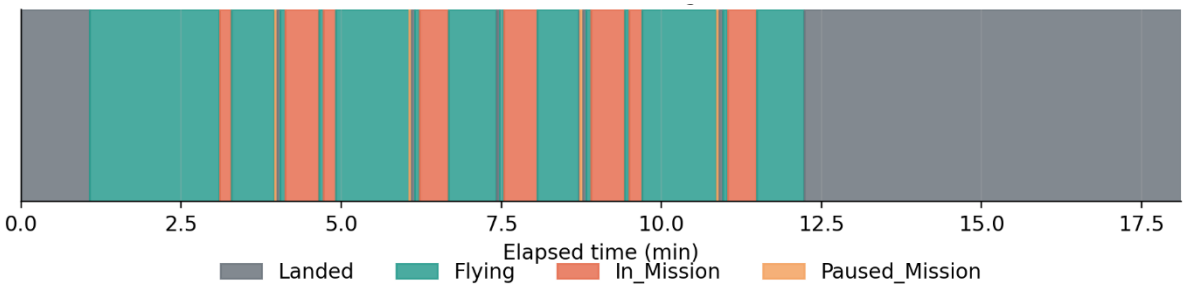

Figure 15: Drone states during mission

The above figure 5.7 helps to better understand the different UAV transitions (Landed, Flying, In_Mission, and Paused_Mission), which confirms that the mission was executed as a sequence of system states rather than as a single static action. This supports the phase-based execution model described in chapter 4. With this figure you can also compare it with the previous figure 5.6 and see for the same timestamps the altitude of the UAV and the state that it was in during that moment of action.

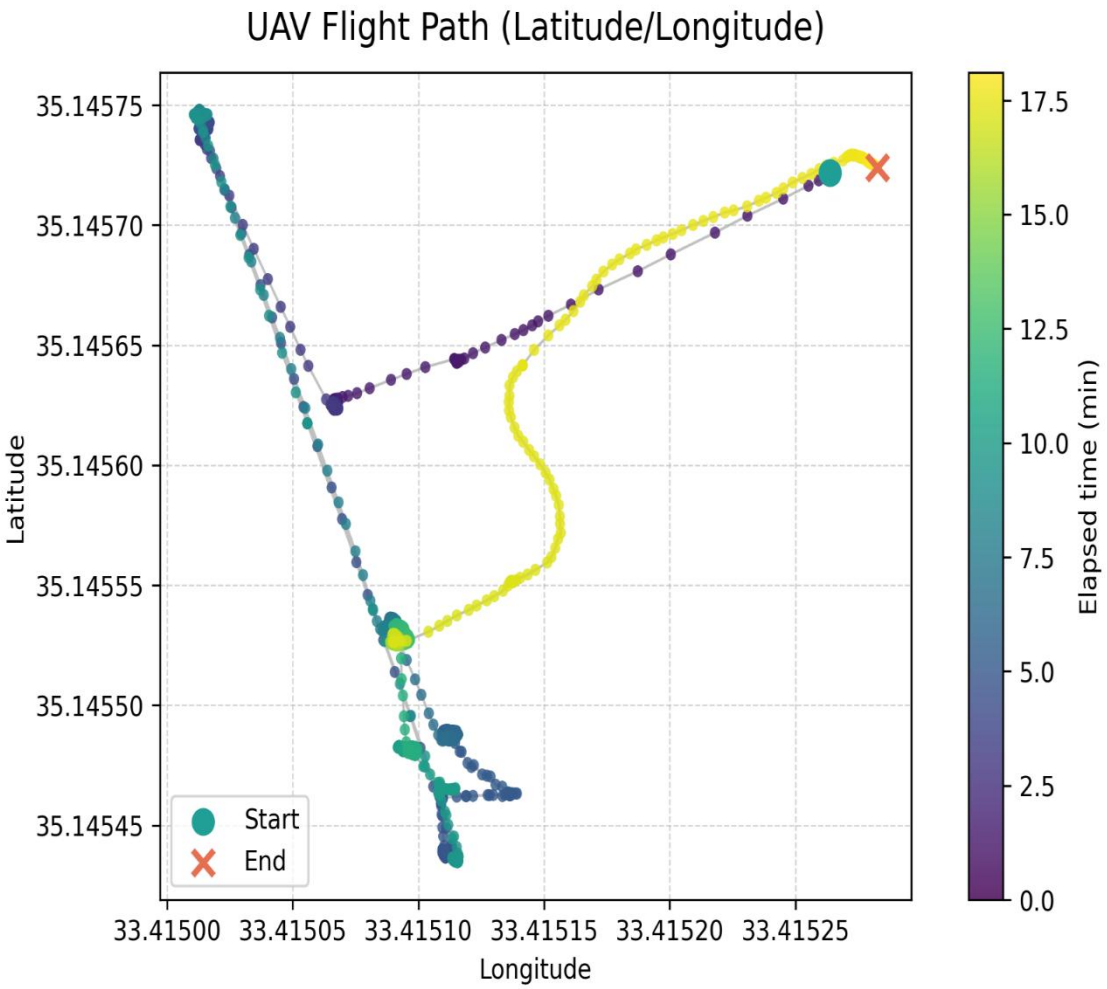

Figure 16: Shows the UAV position as local meter offsets from the starting point.

From the above figure, the UAV's position during mission is shown. The position is being shown as latitude and longitude values from the telemetry dataset. The colored points indicate elapsed mission time, which makes the repeated movement pattern easier to follow in a static figure. The path shows that the UAV moved within the outdoor test area, with a maximum displacement of approximately 34.68 m from the starting position.

5.4 ArUco Marker Detection Analysis

The ArUco marker detection experiment evaluates whether the vision module can detect the landing marker and provides marker geometry for the precision landing controller. This evaluation uses two tests. The first test is the real integrated flight test, where the onboard system performs live detection from the camera stream used during the mission. The second test is an offline recorded video, where higher quality footage is processed using the same detection logic to see how camera quality affects detection distance and stability. The live detection stream used by the onboard pipeline is resized to 420x240 before ArUco detection, and is affected by stream compression, whereas the recorded footage was captured at the same time as the real mission with the same camera in 4k and from same peak altitude to show the difference the quality makes.

Figure 17: Live onboard ArUco detection that needs to be at 6 meters due to image resizing and quality.

Figure 18: Recorder Video detection of ArUco marker (ID=14) with better quality and greater distance.

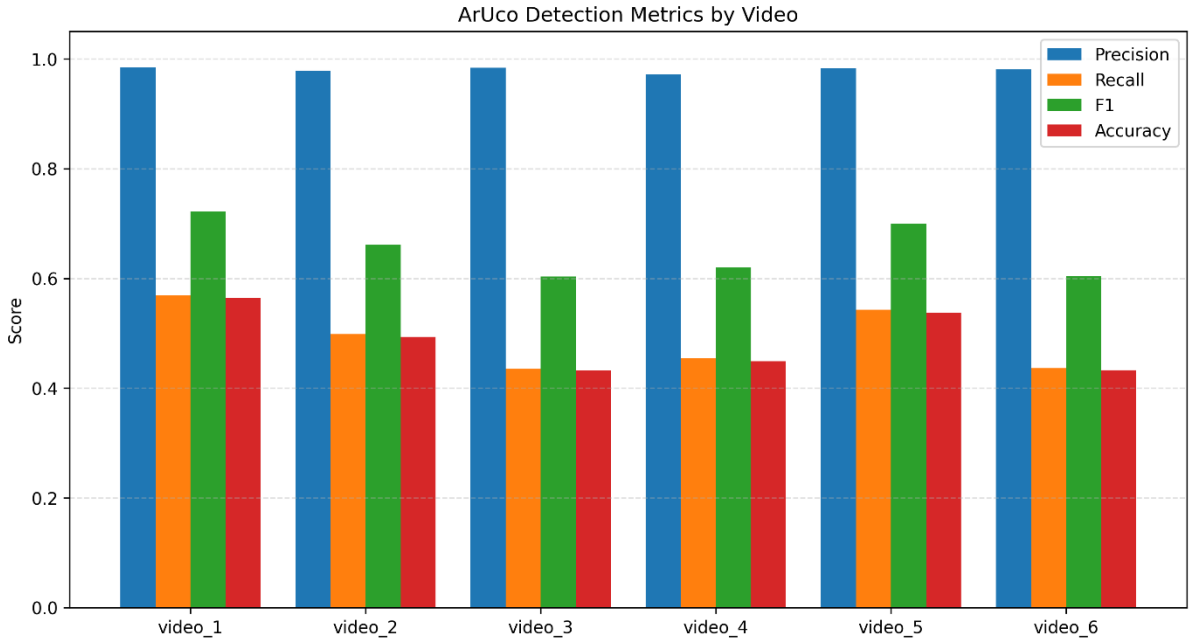

Figure 19: Plot Precision, Recall, F1, Accuracy by video dataset

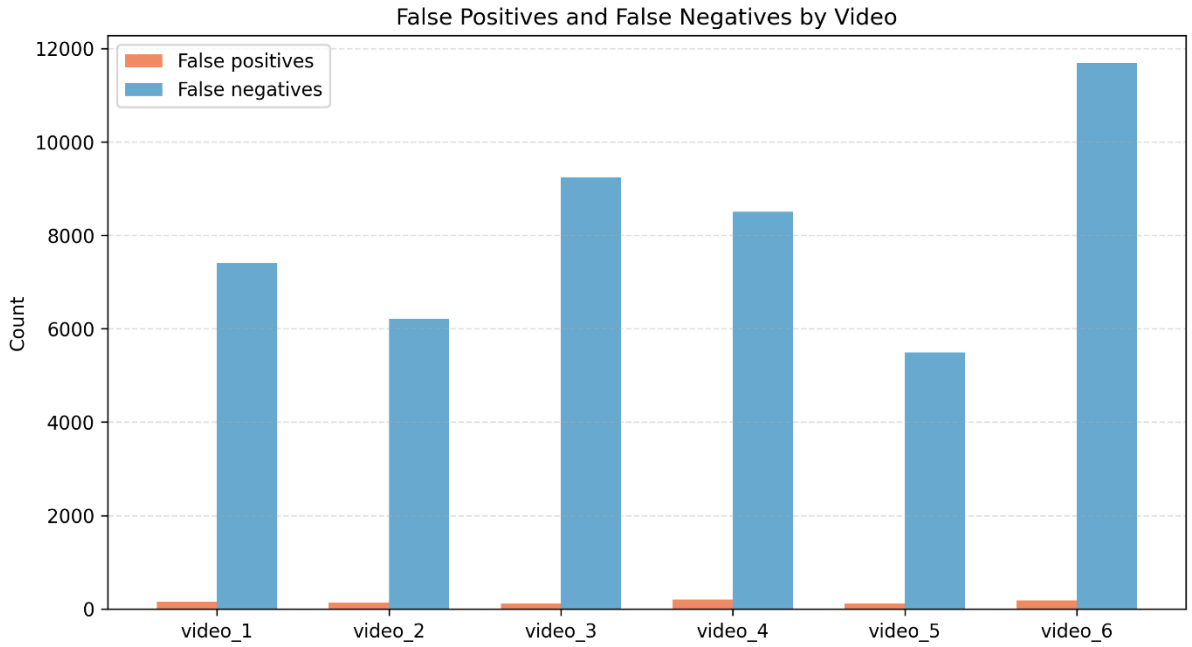

Figure 20: False positive and False Negative by Video

The above evaluations on figures 5.11 and 5.12 used the labeled marker-image dataset from the Shadow-ArUco Dataset in Zenodo.org [41]. The dataset is organized into six sequences from video_1 to video_6 and contains 8652 annotated frames in total. Each frame is stored as a PNG image with resolution 800 x 600, and each image is paired with a JSON annotation file in the corrected annotations folder. The annotation of each frame contains the marker identities and the four ground-truth corner coordinates for every visible marker. The number of labeled markers per frame ranges from 9 to 12, with an average of approximately 10.9 markers per image.

Figure 5.11 shows the detection performance per video in terms of precision, recall, F1-score, and marker-level accuracy. The most important observation is that precision remains very high in all six sequences, ranging from approximately 0.97 to 0.98. This means that when the detector reports a marker, it is usually correct. However, recall is much lower, ranging from approximately 0.43 to 0.57. As a result, the F1-score remains moderate, between approximately 0.60 and 0.72. Therefore, the detector is conservative, because it produces few false detections, but it fails to detect a considerable portion of the ground-truth markers.

Figure 5.12 presents the false positive and false negative counts for each video sequence. In all cases, false negatives are significantly higher than false positives. This shows that the detector is conservative. It rarely reports incorrect markers, but it misses many markers that are actually visible in the image. This explains why precision remains high, while recall and F1 score are lower. The large number of false negatives is mainly due to the challenging conditions of the dataset, including dark frames, reflections, perspective changes, partial visibility, and the simultaneous presence of multiple markers.

The low F1 score in figure 5.11 is mainly caused by the large number of false negatives rather than by false positives. This is consistent with visual properties of the dataset. Many frames are dark, some markers are affected by reflections and glare, several markers

appear at strong perspective angles, and some are relatively small or partly close to the image borders. Also, each frame contains many markers simultaneously, which makes the task harder than the real landing case of this thesis, where the detector mainly focuses on one target marker. The evaluation also used a strict matching criterion, requiring the correct marker identity and sufficient overlap with the annotated quadrilateral. For this reason, a marker may count as false negative not only when it is completely missed, but also when it is detected with the wrong ID or with insufficient localization overlap. Overall, the results indicate that the detector is reliable when it locks onto a marker, but its sensitivity is reduced in cluttered and visually challenging scenes.

For example, in the bellow frame, it has **12 ground truth markers** and the detector found only **4**, so it gives **8 false negatives**, it also shows the difficult environment mentioned above with the dark places, uneven illumination, projected light on wall, and some markers being smaller, weaker, farther and more distorted. This shows the failing mentioned above how it affects the false negatives and not just total fail.

Figure 21: Zenodo dataset detection results

Also, another extreme case is the that appears in video_1, where all 12 ground truth markers were missed by the detector. This is significantly darker than the partial failure

example and shows how severe illumination degradation can completely prevent marker detection.

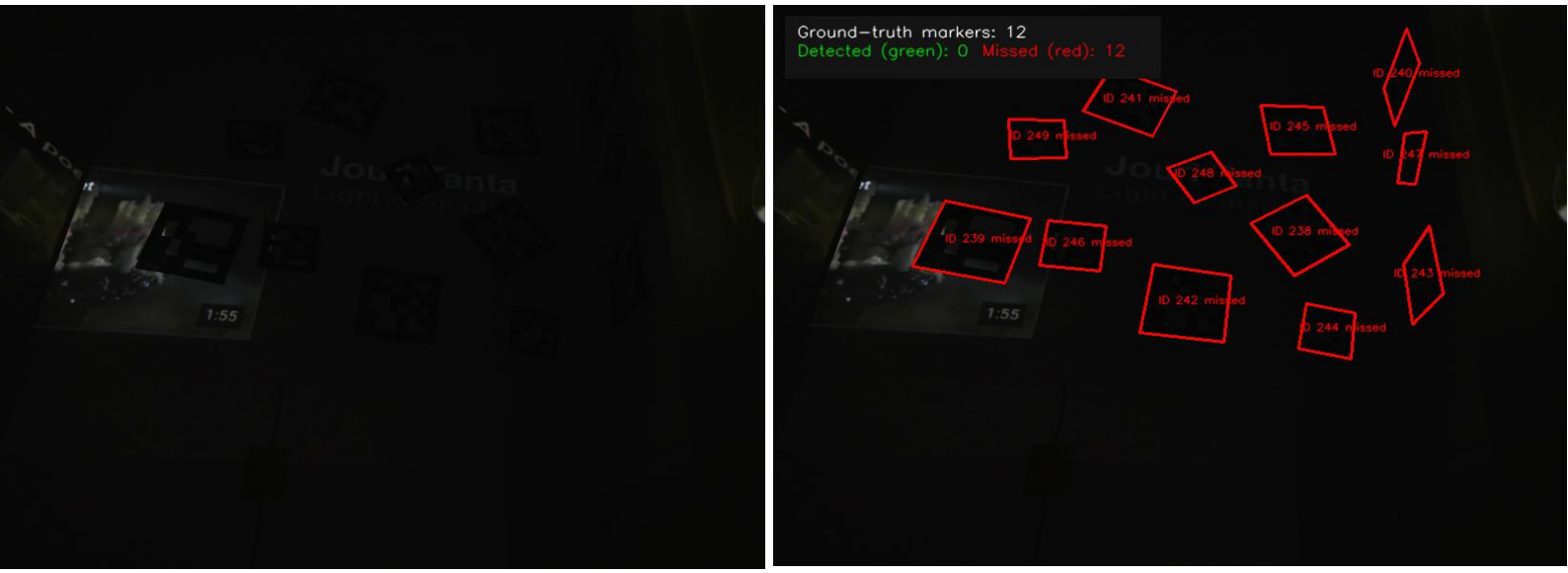

Figure 22: Zenodo dataset detection extreme case results

Table 9: Offline ArUco detection results for target marker ID 14.

Video	**Resolution**	**Processed frames**	**Raw candidate frames**	**Accepted detection frames**	**Accepted detection ratio**	**First accepted detection (sec)**	**Max marker size (px)**
Wide camera	3840 X 2160	1301	404	334	25.7%	26.83	54.3
Zoom camera	3840 X 2160	1255	798	736	58.6%	15.87	106.3

The above table shows the results of an offline recorded video clip in which the target marker was always visible during the evaluation period and was captured both in Wide camera and in Zoom camera for evaluation purposes. Both clips were exported at 3840

X 2160, and both showed stable repeated detections of the target marker. The Wide clip produced an accepted detection ratio of 25.7% and the first accepted detection occurred earlier, at 26.83 seconds, while the Zoom clip produced a higher accepted detection ratio of 58.6% and much larger maximum marker size of 106.3 px, compared to the 77.2 px of the Wide clip, with the first accepted detection at 15.87 seconds. This indicates that the Zoom view allowed the marker to occupy more image pixels, which improved marker visibility and supported stable detections once the marker became sufficiently noticeable in the frame. Also, that helped with finding the marker faster than with the Wide camera. Therefore, the comparison shows that between the 2 that Wide view finds the marker but later and with less detection ratio than the Zoom view.

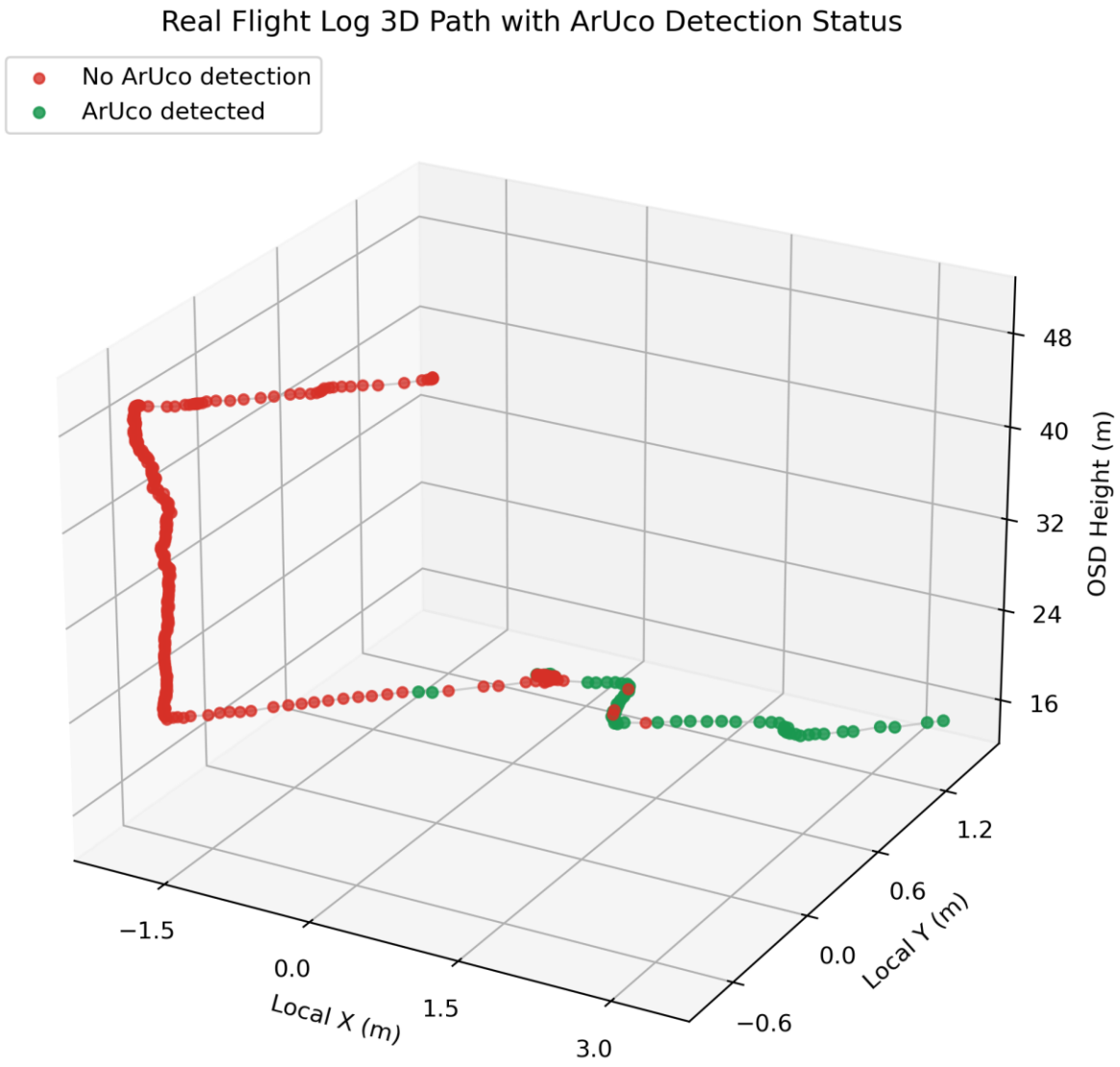

Figure 23: Offline video recorded marker detection 3D Plot

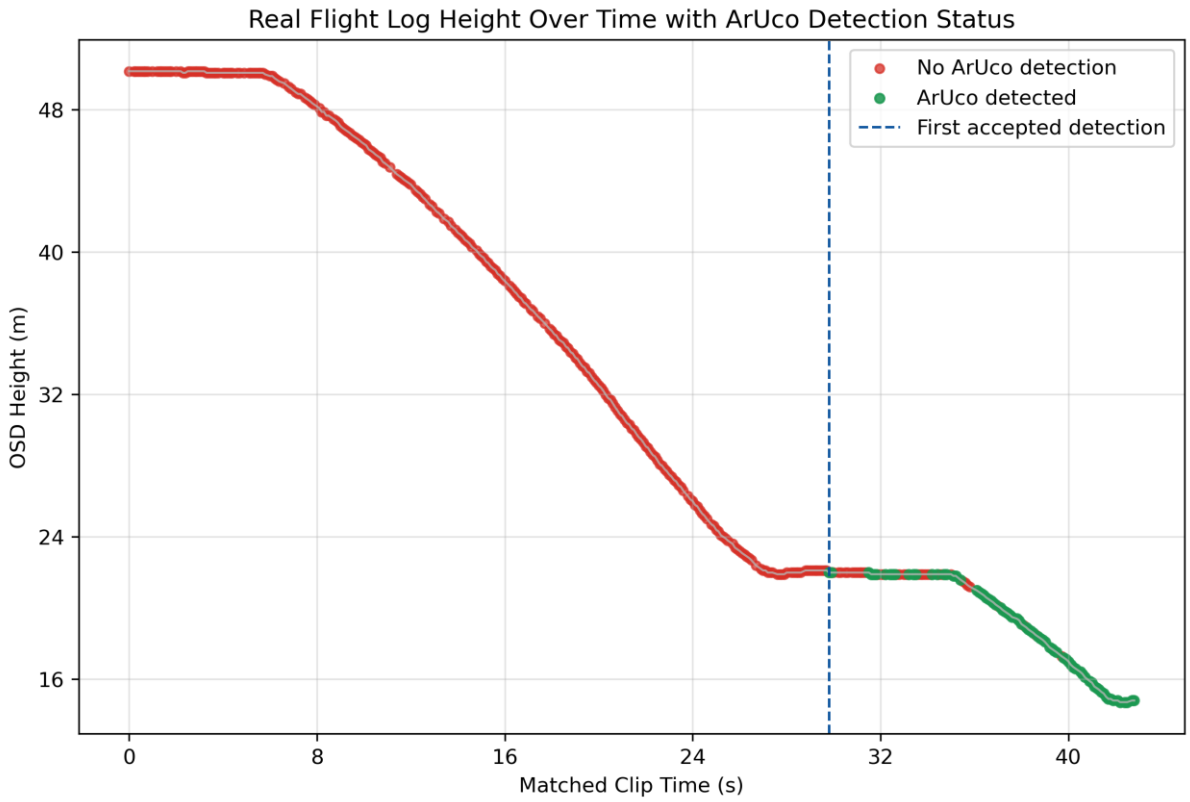

Figure 24: Offline video recorded marker detection 2D Plot in Altitude over time

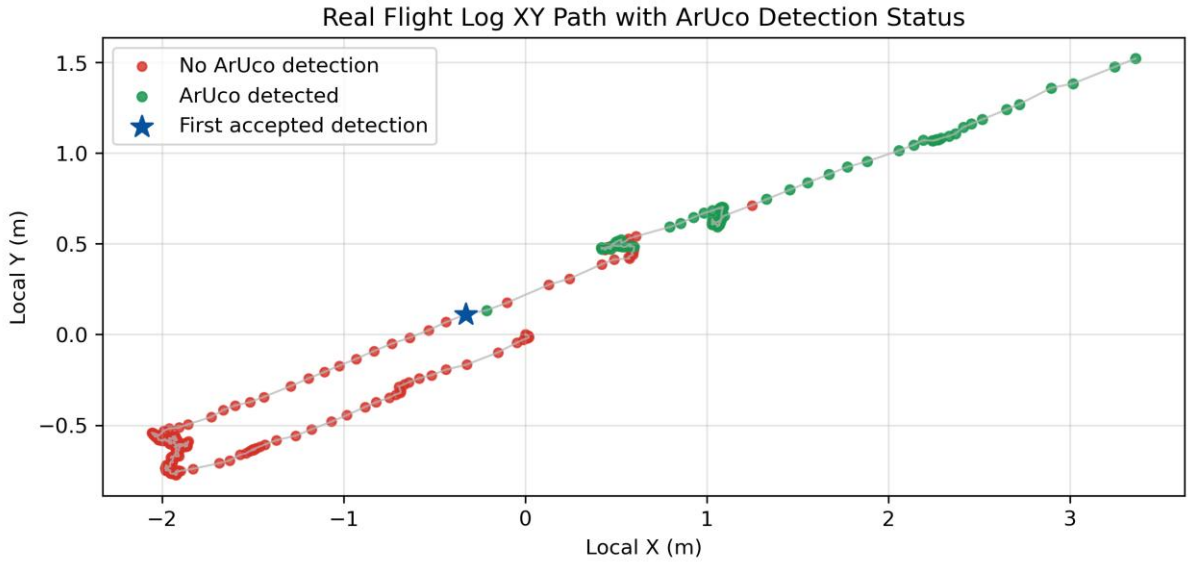

Figure 25: Flight Log XY Path with ArUco Detection Status 2D Plot

Figures 5.13, 5.14, 5.15 provide a visual representation of the offline recorded-video evaluation using the raw DJI clip with available flight metadata. In Figure 5.13, the flight path is shown in a local 3D coordinate frame, where the horizontal axes represent local position offset in meters from the starting point (that is shown as the top red part in the 3D figure), and the vertical axis represents altitude relative to the takeoff point (OSD Height) in meters. Red points indicate frames in which the ArUco marker was not accepted, while green points indicate accepted detections. The figure shows that the marker was mainly detected during the later part of the approach, when the UAV had moved closer to the target area.

Figure 5.14 presents the same sequence as an altitude-versus-time profile and highlights more clearly when the first accepted detection occurred. It can be observed that accepted detections begin after the descent has already progressed and then continue during the final approach.

Figure 5.15 presents the same sequence in 2D showing how the UAV moved during that flight in its local XY axis. It starts from the local (X= 0, Y=0) shown in the figure and moves according to the path where again the first detection is marked with the blue star and in all the path the color is either Red for no detection and Green for detection and matches the other figures.

Figure 5.14, 5.15 when put together explain and show the 3D figure (5.13) and make it easier to understand how that path was formed over time. Together, these figures show not only whether the detector worked, but also where along the recoded path and at what stage of the descent the marker became detectable.

Figure 26: Zoom camera view of offline recorded video detection

Figure 27: Wide camera view of offline recorded video detection

Chapter 6 Conclusion and Future work

6.1 Conclusion

This proof-of-concept study successfully demonstrates the design and implementation of an autonomous drone delivery system that connects AIDERS mission planning with onboard UAV execution. The system receives mission waypoints from AIDERS, interprets them as pickup, delivery, and return or charge-station points, and executes the delivery sequence through the DJI Payload SDK. The implemented workflow also includes telemetry publishing, database storage, camera streaming, ArUco marker detection, and marker-assisted precision landing support.

The main contribution of this work is the integration of these components into one practical delivery workflow. Instead of treating waypoints navigation, telemetry, communication, and marker detection as separate modules, the implemented system combines them into a complete mission structure. AIDERS is used for mission definition and supervision, the WebSocket-to-UDP bridge transfers mission data to the onboard computer, and the Payload SDK modules handle flight execution, mission-state transitions, telemetry output, and visual landing support. Therefore, the work is mainly a system-level implementation and validation of an autonomous delivery pipeline rather than a new standalone path-planning or computer-vision algorithm.

The experimental results showed that the implemented system can execute the main delivery mission flow. The mission can be dispatched from AIDERS, received by the onboard system, mapped into the required waypoint roles, and executed using phase-based control. The telemetry evaluation also showed that the system records useful mission data for analysis after the flight. In the full mission test, 3026 telemetry records were collected over approximately 18 minutes, with battery decreasing from 91% to 66%, clear altitude and state changes, visible movement through the test area, and strong satellite availability. These results show that the telemetry pipeline is useful not only for live supervision, but also for post-flight evaluation of the mission.

The ArUco detection evaluation showed that the marker-based landing reference can be detected by the implemented vision pipeline. The live onboard test demonstrated that marker detection works in the actual flight environment, while the offline analysis of higher-quality recorded video showed that the same detection logic performs better when the image has higher resolution and clearer marker edges. This comparison highlighted an important practical limitation: the live detection stream was resized to 420x240 before detection, which reduced the detection and required the UAV to approach closer to the marker. Even with this limitation, the results confirm that the detector can provide the marker ID, center position, size, and image offset values required by the landing-support controller.

Overall, the thesis paper demonstrates a working proof of concept for an AIDERS-connected autonomous drone delivery system with telemetry monitoring and visual landing support. The system validates the feasibility of connecting external mission planning with onboard UAV behavior in a campus-scale delivery scenario. However, the prototype should not be interpreted as a fully validated operational delivery product. More repeated real-flight trials, stronger live image quality, and measured landing-accuracy results are needed before the system can be considered robust for regular use.

6.2 Future Work

The first direction for future work is improving the live vision pipeline. The comparison between live detection and offline recorded-video detection showed that image quality strongly affects ArUco performance. Future versions should process higher-resolution frames, avoid aggressive resizing before detection, or use a separate high-quality detection stream. Larger landing markers, multi-scale markers, or alternative fiducial markers could also improve detection from higher altitude and reduce the need for the UAV to approach very close before stable detection.

A second improvement is to perform repeated precision-landing experiments with quantitative measurements. Future tests should record the starting altitude, initial lateral offsets, marker detection distance, landing completion status, fallback usage, and final landing error for each trial. Repeating these tests under different lighting conditions, camera angles, and marker placements would make it possible to evaluate landing

accuracy statistically instead of relying mainly on qualitative flight evidence and video analysis.

The telemetry and database pipeline can also be extended. Since the system already stores mission data in MySQL, future work could automatically generate mission reports containing battery graphs, altitude profiles, state timelines, position paths, satellite-count trends, and landing events. This would make the evaluation process more repeatable and reduce the amount of manual graph preparation needed after each test.

Finaly, the delivery workflow can be expanded toward more realistic operation. Future versions could include payload attachment and release validation, obstacle-awareness, dynamic replanning, and longer tests across different delivery points. With these improvements, the current prototype could be developed from a proof-of-concept system into a more reliable autonomous delivery platform.

OBJ

OBJ

BIBLIOGRAPHY

[1] K. AL-Dosari, Z. Hunaiti and W. Balachandran, "Systematic Review on Civilian Drones in Safety and Security Applications," *Drones,* vol. 7, p. 210, 2023.

[2] A. Mohamed and M. Mohamed, "Unmanned Aerial Vehicles in Last-Mile Parcel Delivery: A State-of-the-Art Review," *Drones,* vol. 9, no. 6, p. 413, 2025.

[3] J. Wang and E. Olson, "AprilTag 2: Efficient and Robust Fiducial Detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[4] R. M. Claro, D. B. Silva and A. M. Pinto, "ArTuga: A novel multimodal fiducial marker for aerial robotics," *Robotics and Autonomous Systems,* vol. 163, p. 104398, 2023.

[5] S. Aggarwal and N. Kumar, "Path Planning Techniques for Unmanned Aerial Vehicles: A Review, Solutions, and Challenges," *Computer Communications,* vol. 149, pp. 270-299, 2020.

[6] Y. Chang, Y. Cheng, U. Manzoor and J. Murray, "A Review of UAV Autonomous Navigation in GPS-Denied Environments," *Robotics and Autonomous Systems,* vol. 170, p. 104533, 2023.

[7] H. Mostafa and A. Abdessattar , "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences,* vol. 91, p. 99–131, 2017.

[8] O. Alena, A. Niels , C. James, G. Bruce and P. Erwin, "Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey," *Networks,* vol. 72, no. 4, pp. 411-458, 2018.

[9] M. Silva, A. Reis and S. Sargento, "A Mission Planning Framework for Fleets of Connected UAVs," *Journal of Intelligent & Robotic Systems,* vol. 108, p. Article 2, 2023.

[10] K. Dorling, J. Heinrichs, G. G.Messier and S. Magierowski, "Vehicle routing problems for drone delivery," in *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016.

[11] A. Adham , M. Somaiyeh and N.-M. Samia , "Current advancements on autonomous mission planning and management systems: An AUV and UAV perspective," *Annual Reviews in Control,* vol. 46, pp. 196-215, 2018.

[12] C. Pengyun , Z. Guobing, L. Jiacheng, C. Ze and Y. Qichen, "Path-Following Control of Small Fixed-Wing UAVs under Wind Disturbance," *Drones,* vol. 7, p. 253, 2023.

[13] P. B. Sujit, S. Saripalli and J. B. Sousa, "Unmanned Aerial Vehicle Path Following: A Survey and Analysis of Algorithms for Fixed-Wing Unmanned Aerial Vehicless," *IEEE Control Systems Magazine,* vol. 34, no. 1, pp. 42 - 59, 2014.

[14] J. A. Marshall, W. Sun and A. L'Afflitto, "A survey of guidance, navigation, and control systems for autonomous multi-rotor small unmanned aerial systems," *Annual Reviews in Control,* vol. 52, pp. 390-427, 2021.

[15] K. Harikumar, D. Sidhant, P. Jinraj V and B. M. Seetharama, "Integrated guidance and control framework for the waypoint navigation of a miniature aircraft with highly coupled longitudinal and lateral dynamics," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering,* vol. 235, no. 8, p. 949–961, 2021.

[16] T. Sarah and K. Vijay, "Autonomous Flight," *Annual Review of Control, Robotics, and Autonomous Systems,* vol. 1, pp. 29-52, 2018.

[17] Z. Yijing, Z. Zheng and L. Yang, "Survey on computational-intelligence-based UAV path planning," *Knowledge-Based Systems,* vol. 158, pp. 54-64, 2018.

[18] J. V. Shirabayashi and L. B. Ruiz, "Toward UAV path planning problem optimization considering the Internet of Drones," *IEEE Access,* vol. 11, 2023.

[19] H. Rienecker, V. Hildebrand and H. Pfifer, "Energy optimal 3D flight path planning for unmanned aerial vehicle in urban environments," *CEAS Aeronautical Journal,* vol. 14, p. 621–636, 2023.

[20] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research,* vol. 30, no. 7, pp. 846-894, 2011.

[21] Y. Jiang, X.-X. Xu, M.-Y. Zheng and Z.-H. Zhan, "Evolutionary computation for unmanned aerial vehicle path planning: a survey," *Artificial Intelligence Review,* vol. 57, 2024.

[22] A. Puente-Castro, D. Rivero, A. Pazos and E. Fernandez-Blanco, "A review of artificial intelligence applied to path planning in UAV swarms," *Neural Computing and Applications,* vol. 34, pp. 153-170, 2022.

[23] N. Bashir, S. Boudjit, G. Dauphin and S. Zeadally, "An obstacle avoidance approach for UAV path planning," *Simulation Modelling Practice and Theory,* vol. 129, 2023.

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, St. Louis, MO, USA, 1985.

[25] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011.

[26] W. Xinwei, W. Hai, W. Min and Z. Hongyun , "A mini review on UAV mission planning," *Journal of Industrial and Management Optimization,* 2022.

[27] L. Damilano, G. Guglieri, I. Sale and F. Quagliotti, "Ground Control Station Embedded Mission Planning for UAS," *Journal of Intelligent & Robotic Systems,* vol. 69, pp. 241-256, 2013.

[28] G. A. Skaltsis, H.-S. Shin and A. Tsourdos, "A Review of Task Allocation Methods for UAVs," *Journal of Intelligent & Robotic Systems,* vol. 109, no. 4, 2023.

[29] L. Ye, Y. Yang, X. Wu, W. Meng, X. Li and R. Zhu, "Offline and online task allocation algorithms for multiple UAVs in wireless sensor networks," *EURASIP Journal on Advances in Signal Processing,* vol. 2024, 2024.

[30] A. Poma, A. Sojo, I. Maza and A. Ollero, "Ground Control Station for Multi-UAV Systems in Infrastructure Inspection and Environmental Monitoring Applications," *Journal of Intelligent & Robotic Systems,* vol. 111, 2025.

[31] R.-A. Cristian and D. Camacho, "Extending QGroundControl for Automated Mission Planning of UAVs," *Sensors,* vol. 18, no. 7, 2018.

[32] H.-H. Chen, "Developing a custom communication protocol for UAVs: Ground control station and architecture design," *Internet of Things,* vol. 27, 2024.

[33] A.-M. Mai A. and M. Azab, "UAV-fleet management for extended NextG emergency support infrastructure with QoS and cost aware," *Internet of Things,* vol. 25, 2024.

[34] F. Heilemann, S. Lindner and A. Schulte, "Experimental evaluation of tasking and teaming design patterns," *Human-Intelligent Systems Integration,* 2021.

[35] S. R. Dixon, C. D. Wickens and D. Chang, "Mission Control of Multiple Unmanned Aerial Vehicles: A Workload Analysis," *Human Factors,* vol. 47, no. 3, pp. 479-487, 2005.

[36] L. Xin, Z. Tang, W. Gai and H. Liu, "Vision-Based Autonomous Landing for the UAV: A Review," *Aerospace,* vol. 9, no. 11, 2022.

[37] P. Wang, C. Wang, J. Wang and M. Q. H. Meng, "Quadrotor Autonomous Landing on Moving Platform," *Procedia Computer Science,* vol. 209, pp. 40-49, 2022.

[38] L. Mu, S. Cao, Y. Zhang, X. Zhang, N. Feng and Y. Zhang, "Autonomous Landing Guidance for Quad-UAVs Based on Visual Image and Altitude Estimation," *Drones,* vol. 9, no. 1, 2025.

[39] S. Garrido-Jurado, R. Muñoz-Salinas, M.-C. F.J. and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition,* vol. 47, no. 6, pp. 2280-2292, 2014.

[40] A. Khazetdinov, A. Zakiev, T. Tsoy, M. Svinin and E. Magid, "Embedded ArUco: a novel approach for high precision UAV landing," *2021 International Siberian Conference on Control and Communications (SIBCON),* pp. 1-6, 2021.

[41] R. Berral-Soler, H. Sarmadi, R. Muñoz-Salinas, R. Medina-Carnicer and M. J. Marín-Jiménez, "Shadow-ArUco Dataset," Zenodo, 2024.