UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Industriales

Multi UAVs Autonomous Missions. Navigation, Execution Control, and Exploration

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Pedro Arias Pérez
M.Sc. in Robotics and Automation

Madrid, 2025


UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Industriales

Doctoral Degree in Automatic Control and Robotics

Multi UAVs Autonomous Missions. Navigation, Execution Control, and Exploration

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Pedro Arias Pérez
M.Sc. in Robotics and Automation

Under the supervision of:
Dr. Pascual Campoy Cervera (Supervisor)
Dr. José María Cañas Plaza (Co-supervisor)

Madrid, 2025

Title: Multi UAVs Autonomous Missions. Navigation, Execution Control, and Exploration

Author: Pedro Arias Pérez

Doctoral Programme: Automatic Control and Robotics

Thesis Supervision:

Dr. Pascual Campoy Cervera, Full Professor, Universidad Politécnica de Madrid (Supervisor)

Dr. José María Cañas Plaza, Associate Professor, Universidad Rey Juan Carlos (Co-supervisor)

External Reviewers:

Thesis Defense Committee:

Thesis Defense Date:

A Pedro, Loli e Raquel
polo voso constante apoio
e infinito amor.

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Pascual Campoy, for his unwavering support, guidance, and trust throughout this journey. His sharp insight, patience, and encouragement have been invaluable to my development as a researcher. I am also sincerely thankful to my co-advisor, Prof. José María Cañas, whose constructive feedback, technical expertise, and motivation helped shape this thesis in meaningful ways.

I would also like to thank the Universidad Politécnica de Madrid and the Center for Automation and Robotics for providing the infrastructure and environment to pursue this work. The support from the FPU grant from the Spanish Ministry of Science, Innovation and Universities has been instrumental, not only in funding the research but also in enabling international collaboration and hands-on validation.

My heartfelt thanks go to all my colleagues and friends at the Computer Vision and Aerial Robotics Lab, especially to Miguel, David, Rafa, and Javi, for making long days of experiments, endless debugging sessions, and writing marathons more bearable and even enjoyable. I am particularly grateful for the camaraderie we shared during late-night sprints and the excitement of preparing for international competitions—we truly built something special together. It was a privilege to grow alongside such a talented and driven group. To the newer members of the lab—Guille, Rodrigo, Carmen, and Kiko—may your journey be as fulfilling and full of learning as mine has been. You are in the right place, surrounded by the right people.

A special mention goes to the amazing team at the Texas A&M Unmanned Systems Lab where I conducted my research stay. Thank you, Sri, Alvika, George, David, and Shaunak for your warm welcome, stimulating conversations, and collaborative spirit. Your fresh perspectives and support abroad enriched both my research and personal experience immensely.

On a more personal note, I owe everything to my family—Mamá, Papá, and Raquel—for their love, understanding, and unconditional support through every high and low of this journey. To my closest friends—*if you know, you know*—thank you for keeping me grounded, making me laugh, and reminding me that life exists beyond papers and deadlines. And to the person walking beside me now, thank you for your kindness, love, and all the moments that made this final stretch brighter.

This thesis is not just the result of my work, but the collective support, kindness, and wisdom of those around me. To all of you, thank you.

Abstract

Advancements in artificial intelligence, sensor fusion, and real-time decision-making are progressively increasing the autonomy of Unmanned Aerial Vehicles (UAVs), enabling them to perform complex tasks with minimal human intervention. Modern UAVs can now undertake sophisticated operations such as autonomous navigation, obstacle avoidance, and coordinated missions involving multiple agents.

Despite the promising developments, achieving full autonomy remains a significant challenge. UAVs must be capable of perceiving and interpreting their environment, adapting their decisions in real time, and executing precise control strategies to operate reliably in complex and uncertain scenarios. Many current deployments still rely on teleoperation or semi-autonomous modes, as fully autonomous systems must overcome these critical issues. The deployment of multiple UAVs further introduces challenges in coordination, communication, and distributed execution to ensure effective collaboration.

To address these issues, this thesis investigates techniques to enhance UAV autonomy by focusing on three core areas: navigation in complex environments, real-time mission adaptation, and multi-agent coordination. The aim is to reduce reliance on human operators and increase the effectiveness of UAVs in real-world scenarios, such as industrial infrastructure inspection and large-scale environment exploration.

To that end, the research is guided by three main questions: how to enable reliable autonomous navigation, how to execute high-level missions with flexibility, and how to explore unstructured environments using minimal sensing. These questions are explored through a set of experiments, real-world UAV inspections at wind farms and solar parks, as well as cooperative exploration using nano-drones equipped with only minimal onboard sensing, each validating different aspects of the proposed solutions.

This thesis advances UAV autonomy by presenting a modular and reusable software framework based on Aerostack2, an open-source aerial robotics architecture. Contributions include robust navigation components for complex environments, a dynamic task execution system for distributed multi-UAV operations, and a novel exploration strategy for minimal-sensing nano-drones. These capabilities were integrated into Aerostack2, extending its functionality and validating the system in extensive real-world and simulated experiments. The results demonstrate the framework's scalability, robustness, and suitability for a wide range of autonomous UAV applications, while also enabling future research and technology transfer through its open-source availability.

Ultimately, this work contributes toward the realization of fully autonomous UAV systems, offering a solid foundation for further advancements in multi-agent coordination, mission planning, and lightweight exploration strategies in real-world environments.

Resumen

Los avances en inteligencia artificial, fusión de sensores y toma de decisiones en tiempo real están incrementando progresivamente la autonomía de los Vehículos Aéreos No Tripulados (UAVs), permitiéndoles ejecutar tareas complejas con una intervención humana mínima. En la actualidad, los UAVs modernos son capaces de llevar a cabo operaciones sofisticadas como la navegación autónoma, la evasión de obstáculos y misiones coordinadas que involucran múltiples agentes.

A pesar de estos avances prometedores, alcanzar una autonomía completa sigue siendo un desafío significativo. Los UAVs deben ser capaces de percibir e interpretar su entorno, adaptar sus decisiones en tiempo real y ejecutar estrategias de control precisas para operar de manera fiable en escenarios complejos e inciertos. Muchas implementaciones actuales aún dependen de la teleoperación o modos semiautónomos, ya que los sistemas completamente autónomos deben superar estos problemas críticos. La implementación de sistemas multi-UAV introduce además desafíos adicionales en coordinación, comunicación y ejecución distribuida para asegurar una colaboración efectiva.

Para abordar estos retos, esta tesis investiga técnicas orientadas a mejorar la autonomía de los UAVs, centradas en tres áreas clave: navegación en entornos complejos, adaptación de misión en tiempo real y coordinación multiagente. El objetivo es reducir la dependencia de operadores humanos e incrementar la eficacia de los UAVs en escenarios reales, como la inspección de infraestructuras industriales o la exploración de grandes entornos.

En esta línea, la investigación se guía por tres preguntas principales: cómo habilitar una navegación autónoma y fiable, cómo ejecutar misiones de alto nivel con flexibilidad, y cómo explorar entornos no estructurados utilizando capacidades sensoriales mínimas. Estas cuestiones se abordan a través de un conjunto de experimentos, incluyendo inspecciones reales de parques eólicos y solares, así como exploraciones cooperativas con nano-drones equipados únicamente con sensores básicos, validando distintos aspectos de las soluciones propuestas.

Esta tesis impulsa la autonomía de los UAVs mediante la presentación de un framework software modular y reutilizable, basado en Aerostack2, una arquitectura de robótica aérea de código abierto. Las contribuciones incluyen componentes de navegación robustos para entornos complejos, un sistema dinámico de ejecución de tareas para operaciones multi-UAV distribuidas y una novedosa estrategia de exploración para nano-drones con percepción mínima. Estas capacidades se integraron en Aerostack2, ampliando su funcionalidad y validando el sistema mediante extensos experimentos reales y simulados. Los resultados demuestran la escalabilidad, robustez y aplicabilidad del framework en un amplio rango de aplicaciones autónomas con UAVs, al tiempo que se facilita la investigación futura y la transferencia tecnológica gracias a su disponibilidad como software libre.

En última instancia, este trabajo contribuye al avance hacia sistemas UAV completamente autónomos, ofreciendo una base sólida para desarrollos futuros en coordinación multiagente, planificación de misiones y estrategias de exploración ligeras en entornos reales.

Table of Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

API Application Programming Interface

BT Behavior Tree

CAC Contextual Autonomous Capability

CVAR Computer Vision and Aerial Robotics group

EC Environmental Complexity

EKF Extended Kalman Filter

FCU Flight Control Unit

FSA Finite State Automata

FSM Finite State Machine

GCS Ground Control Station

GUI Graphical User Interface

GPS GLobal Positioning System

GVD Generalized Voronoi Diagram

HFSM Hierarchical Finite State Machine

HI Human Independence

HIL Hardware-in-the-Loop

IMU Inertial Measurement Unit

IoT Internet of Things

JSON JavaScript Object Notation

LoA Levels of Autonomy

MAV Micro Air Vehicle

MC Mission Complexity

OSH Open Source Hardware

OSS Open Source Software

PN Petri Net

PV Photovoltaic

RAP Reactive Action Package

R&D&I Research, Development, and Innovation

RQ Research Question

ROS Robotic Operating System

RTL Return To Launch

SDK Software Development Kit

SITL Software-in-the-Loop

SLAM Simultaneous Localization and Mapping

TF Transformation Frame

UAV Unmanned Aerial Vehicle

UPM Universidad Politécnica de Madrid

Chapter 1

Introduction

UNMANNED Aerial Vehicles (UAVs) are flying devices that operate without an onboard pilot. In the last decades, UAVs have been widely used in both military and civilian applications due to their small size and high maneuverability [Kim et al., 2019]. In particular, their applications in the civilian sector have expanded to almost every field. In agriculture, the rapid evolution of UAVs enables precision agriculture applications, such as aerial crop monitoring and smart spraying tasks [Radoglou-Grammatikis et al., 2020]. In the industrial sector, UAV developments enhance the efficiency of missions like industrial inspection [Nooralishahi et al., 2021], logistics [Rejeb et al., 2023], and delivery tasks [Yao et al., 2019]. Additionally, UAVs are increasingly utilized in search and rescue operations [Daud et al., 2022].

Despite their exponential growth, most of these applications require low levels of autonomy for the UAV, mainly teleoperated or controlled via waypoints. Huang [Huang, 2004] defines four levels of autonomy for UAVs, ranging from full human control to complete independence. At the lowest level, *Remote Control*, the UAV is entirely dependent on a human operator who provides continuous commands based on direct observation. The UAV lacks decision-making capabilities and does not operate without external input. *Teleoperation* introduces a degree of autonomy, where the UAV responds to sensory feedback but still requires frequent human intervention to execute incremental goals. Moving further up the autonomy scale, *Semi-Autonomous* UAVs can operate independently for certain tasks, with human interactions occurring intermittently rather than continuously. This allows for a balance between automation and human oversight. Finally, at the highest level, *Fully Autonomous* UAVs can complete entire missions without human intervention, adapting dynamically to environmental changes and operational conditions.

Despite the promising advancements in UAV autonomy, achieving full autonomy remains a significant challenge in many scenarios. Operating in complex and dynamic environments requires UAVs to perceive their surroundings, make real-time decisions, and execute precise control strategies without human intervention [Floreano and Wood, 2015]. Many existing applications still rely on teleoperation or semi-autonomous control, as fully autonomous UAVs must overcome obstacle avoidance, localization uncertainty, and dynamic mission adaptation to be reliable in real-world settings [Zhilenkov and Epifantsev, 2018]. Additionally, multi-UAV

systems introduce further challenges in terms of coordination, communication, and mission execution to ensure efficient collaboration.

Given these challenges, this thesis explores methods to enhance UAV autonomy by addressing key aspects of navigation, real-time mission adaptation, and multi-agent coordination. The goal is to enable UAVs to operate with minimal human intervention, increasing their effectiveness in applications such as industrial inspections and large-scale environment exploration.

1.1 Motivation

The field of robotics and unmanned aerial vehicles has experienced significant expansion in recent years, with strong growth expectations and a high demand for qualified professionals at both the European and national levels in the coming years. By 2035, the estimated annual business volume is projected to reach €10 billion, generating 90,000 jobs (with €1.22 billion and 11,000 jobs in Spain alone). By 2050, this volume is expected to increase to €14.6 billion and 110,000 jobs (with €1.52 billion and 11,500 jobs in Spain) [Ministerio de Fomento, 2018].

UAVs are becoming increasingly autonomous, with advancements in artificial intelligence, sensor fusion, and real-time decision-making enabling them to operate with minimal human intervention. Modern UAVs can perform complex tasks such as navigation, obstacle avoidance, and coordinated multi-agent missions. However, achieving full autonomy remains a significant challenge due to unpredictable environments, dynamic obstacles, and the need for real-time adaptability. Factors such as limited onboard computational resources, communication constraints, and regulatory restrictions further complicate the deployment of fully autonomous UAV systems. Additionally, mission planning and execution in real-world scenarios require robust frameworks capable of handling uncertainty and optimizing UAV behavior in diverse applications. Overcoming these challenges is crucial to unlocking the full potential of UAV autonomy in areas like industrial inspection, search and rescue, and large-scale exploration.

To address these challenges, the robotics scientific community has been actively developing software frameworks to streamline UAV operation. Despite this trend, the lack of collaboration and standardization limits the broader applicability of these frameworks. Many existing aerial robotics stacks focus on specific aspects, such as low-level control or platform-dependent implementations, leading to fragmentation in the field. This specialization makes it difficult to integrate well-tested and robust algorithms across different applications, preventing researchers and developers from fully leveraging existing solutions. A more unified approach to UAV software frameworks would not only enhance interoperability but also allow developers to focus on innovation and customization rather than reinventing core functionalities. By addressing these limitations, UAVs could achieve greater autonomy, reliability, and scalability in real-world applications.

One notable effort in this direction is Aerostack2, a modular and open-source framework designed to support the development of autonomous aerial systems. It provides a flexible architecture with reusable components for perception, control, and decision-making, enabling developers to construct complex UAV applications with minimal overhead. However, at the time of writing, Aerostack2 still lacked mature navigation capabilities and support for high-

level mission execution, particularly in multi-UAV contexts. These gaps limit its applicability in scenarios requiring robust autonomy, such as dynamic industrial inspections or distributed exploration. This thesis builds upon Aerostack2 by addressing these limitations and extending the framework with advanced navigation, task execution, and exploration functionalities, ultimately enhancing its versatility and real-world impact.

The objectives of this thesis align with the national R&D&I Strategic Lines, specifically within the fields of Artificial Intelligence and Robotics, under the intervention areas of Digital World, Industry, Space, and Defense. These priorities are outlined in the Spanish Strategy for Science, Technology, and Innovation 2021-2027 [Ministerio de Ciencia e Innovación, 2021]. The objectives aim at increasing the aircraft's autonomy, strongly linked to sub-areas of multi-agent systems and advanced navigation and guidance. The next section outlines the specific research questions that guide this work. Afterwards, the technical objectives and contributions of the thesis are defined.

1.1.1 Research Questions

The present Doctoral Thesis aims at improving the autonomy level of multi-UAV missions. Developing fully autonomous UAVs remains very challenging due to the wide range of new applications and the various levels of the tasks' complexity. Consequently, the main research question driving this dissertation can be formulated as follows:

Main Research Question

How can we make multi-UAV systems more autonomous?

To make this question tackleable, we need to define what "autonomy" means. According to the National Institute of Standards and Technology [Huang, 2004], autonomy is defined as *the own ability of integrated sensing, perceiving, analyzing, communicating, planning, decision-making, and acting/executing, to achieve its goals as assigned by its human operator(s).* UAVs' autonomy is characterized into levels from the perspective of Human Independence (HI), and is further characterized in terms of Contextual Autonomous Capability (CAC).

The Levels of Autonomy (LoA) are defined as: fully autonomous, semi-autonomous, teleoperation, and remote control. A fully autonomous UAV accomplishes its assigned mission, within a defined scope, without human intervention while adapting to operational and environmental conditions. However, autonomy is further portrayed by the UAVs' CAC. It defines three aspects: mission complexity (MC), environmental complexity (EC), and human independence (HI). Thus, the UAVs' autonomy can be further characterized by the missions that the system is capable of performing, the environments within which the missions are performed, and human independence that can be allowed in the performance of the missions.

Our attention will now shift to the presented autonomous capabilities. Therefore, the first research question is how can the environmental complexity be increased while maintaining the UAVs fully autonomous:

RQ1: Research Question 1

How do we bring navigation through complex environments to UAV systems?

This research question focuses on sensing and planning capabilities from a robotics navigation perspective. Robotic navigation is defined by the IEEE Standard 172-1983 as *the process of determining and maintaining a course for a vehicle so as to reach the intended location.* The course following conveys moving purposefully instead of moving randomly. Moreover, collisions have to be avoided because they lead to failure or damage to the aircraft. Navigation remains challenging in high-complexity environments, such as unstructured, highly dynamic, or cluttered scenarios (e.g., urban areas, forests, disaster zones).

The second research question aims at increasing the mission complexity. MC refers to the difficulty and sophistication of the tasks that the UAV can autonomously perform. High-complexity missions include advanced tasks such as cooperative multi-UAV operations, real-time decision-making, or adaptive execution. Hence, the second research question can be formulated as follows:

RQ2: Research Question 2

How can UAVs execute flexible high-level abstraction missions?

The ability to execute flexible and high-level abstraction missions directly impacts MC. Flexible execution allows the UAVs to dynamically adapt to environmental changes. High-level abstraction tasks enable using semantic goal definitions instead of low-level commands.

If the above questions are answered, we can push the boundaries one step further to increase UAVs' autonomy. In many scenarios, UAVs have to deal with reduced capabilities due to onboard computing or limited payload. Therefore, this leads us to the last research question:

RQ3: Research Question 3

How can UAVs explore an unstructured environment using limited sensing?

Again, environmental complexity is tackled by reduced UAVs' one autonomy-enabling function, the sensing. Limited sensing increases EC since the information gathered about the environment is restricted. For this problem, exploration is considered a higher-level task than navigation, which provides targets to reach.

1.2 Objectives

The overall objective of this Doctoral Thesis is to enhance the level of autonomy in multi-UAV systems by designing and implementing modular and reusable solutions. To achieve this, the work is structured around five specific objectives that arise from the research questions introduced in the previous section:

O.1 Design robust and reusable navigation components capable of enabling UAVs to au-

tonomously traverse complex and unstructured environments. This includes mapping and planning modules that ensure safe, efficient path generation and execution, addressing challenges associated with environmental complexity (related to **RQ1**).

O.2 Design an execution control architecture that supports the flexible execution of high-level missions. This includes enabling UAVs to perform abstract tasks with minimal human intervention, dynamically adapt to runtime changes, and collaborate in multi-agent scenarios, thereby increasing mission complexity (related to **RQ2**).

O.3 Design a lightweight exploration strategy for UAVs operating with severely limited sensing capabilities. This objective targets small-scale platforms and aims to maintain autonomous behavior in highly constrained conditions, advancing capabilities in minimal-sensing exploration (related to **RQ3**).

O.4 Develop and validate the proposed methods in both simulated environments and real-world scenarios, including complex industrial settings such as wind farms and solar parks.

O.5 Contribute to the robotics research community through the development, open-source release, and documentation of reusable software. This objective facilitates reproducibility and supports future research in autonomous aerial robotics.

1.3 Contributions

The main contributions of the present Doctoral Thesis are the design, development, and testing of methods and algorithms to increase the autonomy of multi-UAV missions. To achieve the aforementioned objectives—all of which address the previously stated research questions—this thesis aims to deliver the following specific, verifiable, and testable contributions. These are as follows:

C.1 Development and standardization effort for a common aerial robotics navigation framework. Two dedicated components for mapping and planning are proposed. Its design offers reliable navigation through complex environments addressing **RQ1**.

C.2 A novel dynamic task planning execution control for high-level abstraction missions that work reliable and flexible in distributed multi-UAV systems. It answers **RQ2** to deal with higher mission complexity.

C.3 A novel exploration algorithm that effectively utilizes minimal sensing capabilities—using only four single-beams range sensor—tailored specifically for nano-UAVs is proposed. This contribution addresses **RQ3** by combining classic bug-algorithms with frontier-based exploration techniques.

C.4 Extensive experimental evaluation has been performed in simulated and real-world scenarios. Additionally, the author would like to highlight the broad experimentation performed in industrial facilities, such as wind farms and solar parks.

C.5 This dissertation, together with other colleagues work, greatly contributed to the original design, development, implementation and maintenance of Aerostack2 framework

[Fernandez-Cortizas et al., 2023]. All methods proposed in **C.1**, **C.2**, and **C.3** have been integrated in Aerostack2, enhancing its capabilities and extending the software applicability.

C.6 Open source and data availability of the development associated with previous contributions **C.1**, **C.2**, **C.3**, and **C.5**. This enables researchers to evaluate and build upon the proposed framework.

1.4 Outline

This thesis contains seven chapters; in the first chapter, the introduction and historical background are presented, alongside the thesis motivation, objectives, and contributions. The following chapters are organized as follows.

Chapter 2 will give a deeper overview of the existing literature. It is primarily focused on five topics: what is the current status of (1) aerial robotic autonomy stacks, and (2) navigation stacks, in the research community; what is the trend in (3) execution control systems for complex missions nowadays; how (4) multi-robot exploration is tackled in the state-of-the-art techniques; and how (5) nano-sized UAVs deal with minimal sensing.

Chapter 3 outlines the methodology used in the development of the thesis. This chapter targets the methodological process guiding this research, combining established systems engineering approaches with an analysis of Aerostack2's limitations. The chapter points out how the research questions are solved based on proposals to meet Aerostack2's autonomous capabilities.

Chapter 4 explains the design and implementation of new Aerostack2 components to meet the proposed research questions. First, RQ1 is addressed by describing the navigation framework improvements for reliable operation in complex environments. Then, the chapter details the execution control architecture developed to support high-level mission abstraction in multi-UAV systems, answering RQ2. Finally, it presents the design of a novel minimal-sensing exploration algorithm, tackling RQ3.

Chapter 5 presents the experimental evaluation of the proposed methods. It showcases extensive tests conducted in both simulation and real-world environments, including solar park and wind turbine inspections, as well as indoor exploration with nano-UAVs. These results validate the system's robustness, modularity, and applicability to real-world missions.

Chapter 6 provides a comprehensive discussion of the results. It analyzes the technical contributions from a system-level perspective, highlighting key aspects such as behavior composition, multi-UAV deliberation, energy efficiency, and agent workload distribution. The discussion links experimental outcomes to the thesis objectives and research questions, offering a critical evaluation of the system's performance.

Chapter 7 presents the overall conclusions of the thesis and provides an overview of future research directions. It summarizes the main contributions and discusses how the proposed advancements pave the way for more autonomous, adaptable, and scalable UAV systems.

Finally, the document includes several appendices that complement the main chapters. These appendices highlight the author's participation in international robotics competitions, which served as benchmarks to validate autonomy strategies in realistic and time-constrained scenarios, and work in several national and international R&D projects. Additionally, a summary of scientific dissemination activities—including journal and conference publications, invited talks, and manuscripts under peer review—is provided to contextualize the research impact within the aerial robotics community.

Chapter 2

State of the Art

The present chapter offers the reader a brief literature review related to this thesis. The remainder of the chapter is organized as follows: Section 2.1 reviews aerial robotic software architectures, covering both programmable flight control units and high-level frameworks. Section 2.2.1 discusses mapping approaches, followed by Section 2.2.2, which explores motion planning and navigation techniques for UAVs. Section 2.3 delves into execution control architectures for mission-level autonomy. Finally, Section 2.4 presents relevant work in multi-robot exploration under minimal sensing constraints.

2.1 Aerial Robotic Software Architectures

In order to control an UAV autonomously, there are two complementary parts: the Flight Control Unit (FCU), systems that focus on low-level control of the aircraft, allowing it to fly stably, and the high-level control frameworks that are responsible for providing further autonomy to the vehicle, making decisions, and commanding the FCU to perform a specific task. Below, we discuss the state of the art in these two areas in more detail.

2.1.1 Programmable Flight Control Unit

FCUs serve as the central computing unit responsible for managing the vehicle's flight dynamics by processing sensor data, executing low-level control algorithms, and interfacing with actuators. They operate in real-time to ensure precise control, with deterministic execution of flight-related commands. The "programmable" aspect of modern FCUs allows developers to customize and adapt its functionality for specific applications or mission requirements, or even support autonomous operation by integrating waypoint navigation and high-level decision-making logic.

FCUs can be classified into two blocks: generic or embedded, each presenting distinct advantages and limitations. Generic FCUs offer versatility, accommodating various frames and components for customizable configurations, which is beneficial for developers. However, their integration and calibration require technical expertise and time. Embedded FCUs, integrated

into complete aerial platforms, provide a user-friendly experience with pre-calibrated hardware for reliable performance. This simplicity suits users seeking ready-to-fly solutions, but may limit adaptability to new technologies or specific needs.

Table 2.1: Comparison of relevant low-level control systems.

Flight Controller	**Open Source**	**Simulation**	**Rate Input**	**Generic**
Pixhawk/PX4	OSH, OSS	SITL, HIL	✓	✓
Ardupilot	OSS	SITL, HIL	✓	✓
Paparazzi	OSH, OSS	SITL, HIL	✓	✓
Crazyflie	OSH, OSS	SITL, HIL	✓	✓
Betaflight	OSS	SITL	✓	✓
DJI Matrice	Propietary	HIL	✓	✗
Parrot	Propietary	SITL	✗	✗
Skydio	Propietary	-	✗	✗

OSH: Open Source Hardware, OSS: Open Source Software, SITL: Software In The Loop, HITL: Hardware In The Loop.

In 2018, Ebeid et al. presented a survey of open-source hardware and software, comparing their main features [Ebeid et al., 2018]. In Table 2.1, some relevant flight controller projects are listed. These projects may cover both hardware and software development of these controllers. They range from Open Source Hardware (OSH) and Open Source Software (OSS) to proprietary commercial controllers. Furthermore, most of them allow simulation of their behavior in a fully simulated way (Software-in-the-Loop, SITL) or in a Hardware-in-the-Loop (HIL) manner, where the autopilot code runs on the specific hardware of the controller. This enables more thorough validation of the systems, leading to robust and reliable solutions.

An essential feature of agile flight controllers is rate input control. This allows precise maneuvering and stabilization by translating angular velocities together with thrust inputs (*acro* commands) into accurate signals to the vehicle's motors. Effective rate input control is especially important for navigation in complex or dynamic scenarios, where rapid response and fine-tuned control are crucial for maintaining stability and executing challenging maneuvers.

Although advanced flight controllers can be programmed to execute basic autonomous tasks, such as waypoint navigation and simple obstacle avoidance, their onboard computing power is insufficient for handling more complex operations. To address this limitation, they are often paired with high-level controllers that enhance the robustness and autonomy of the vehicles, enabling them to perform sophisticated autonomous missions.

2.1.2 High-Level Frameworks for Aerial Robotics

Several high-level control systems have been published in recent years, each offering different features and capabilities to address specific challenges in aerial robotics. Table 2.2 compares the existing aerial stacks in the literature, focusing on key features such as open-source availability, framework modularity, tested scenarios, middleware usage, software maintenance (last update), multi-frame capabilities, rate output control, multi-agent support, multi-platform compatibility, and plugin-oriented architecture.

Table 2.2: Comparison of relevant open-sourced high level control systems. Flight stacks are presented in alphabetic order. Features compared are columns: Open-Source (availability under open-source license), Modular (independent components in system), Tested In (environments validated), Middleware (communication middleware used), Software last update (recency of updates, last check on 11/2024), Multi-Frame (handling of different coordinate frames), Rate Output (precise rate-based control output), Multi-Agent (support for multi-robot coordination), Multi-Platform (support for different aerial platform), and Plugin-Oriented (architecture extensibility via plugins).

Flight Stack	Open Source	Modular	Tested in	Middle-ware	Soft. last update	Multi-Frame	Rate Output	Multi-Agent	Multi-Platform	Plugin Ori-ented
Aerostack [Sanchez-Lopez et al., 2016]	✓	✓	S,RL,RO	ROS	11/2021	✗	✓	✓	✓	✗
Agilicius [Foehn et al., 2022]	✗	✓	S,RL,RO	ROS	03/2023	✗	✓	✗	✗	✗
CrazyChoir [Pichierri et al., 2023]	✓	✗	S,RL	ROS 2	09/2024	✗	✓	✓	✗	✗
CrazySwarm2 [Preiss et al., 2017]	✓	✓	S,RL	ROS 2	11/2024	✗	✓	✓	✗	✓
KumarRobotics [Mohta et al., 2018]	✓	✗	S,RL,RO	ROS	08/2023	✗	✓	✗	✓	✗
LARICS-uavrosstack [Markovic et al., 2023]	✓	✓	S,RL	ROS	10/2024	✗	✓	✗	✗	✗
MRS UAV System [Baca et al., 2021]	✓	✓	S,RL,RO	ROS	11/2024	✓	✓	✓	✗	✓
RotorS [Furrer et al., 2016]	✓	✓	S	ROS	07/2021	✗	✓	✗	✗	✗
UAL [Real et al., 2020]	✓	✗	S,RL,RO	ROS	12/2022	✓	✗	✗	✓	✗
XTDrone [Xiao et al., 2020]	✓	✓	S	ROS	05/2024	✗	✓	✗	✗	✗
Aerostack2 [Fernandez-Cortizas et al., 2023]	✓	✓	S,RL,RO	ROS 2	11/2024	✓	✓	✓	✓	✓

S: Simulation, RL: real experiments in the lab, RO: real experiments outside the lab.

Aerostack [Sanchez-Lopez et al., 2016] is a software framework that helps developers design and build the complete control architecture of aerial robotic systems, integrating multiple heterogeneous computational solutions (e.g., computer vision algorithms, motion controllers, self-localization and mapping methods, motion planning algorithms, etc.). Aerostack was developed in the past in our research laboratory using ROS. The experience in its use and the development of successive versions has been an important antecedent for the creation of the new framework based on ROS 2.

Agilicious [Foehn et al., 2022] is a co-designed hardware and software framework tailored to autonomous and agile quadrotor flight, which has been developed and used since 2016 at the Robotics and Perception Group (RPG) of the University of Zurich. The stack uses a custom license but is free to use for academics after registration. It supports both model-based and neural network-based controllers. Also, it provides high thrust-to-weight and torque-to-inertia ratios for agility, onboard vision sensors, GPU-accelerated compute hardware for real-time perception and neural network inference, a real-time flight controller, and a versatile software stack.

CrazyChoir [Pichierri et al., 2023] is a ROS 2 toolbox that allows users to run simulations and experiments on swarms of Crazyflie nano-quadrotors. CrazyChoir implements several

tools to model swarms of Crazyflie nano-quadrotors and to run distributed, complex tasks both in simulation and real experiments.

Similarly, Crazyswarm2 Preiss et al., 2017, is a toolbox designed for a large swarm of Crazyflies flying in dense indoor formations, obtaining accurate flights for up to 50 drones. The system fully utilizes the vehicles' onboard computation, allowing for robustness against unreliable communication and a rich set of trajectory planning methods requiring little radio bandwidth.

KumarRobotics flight stack [Mohta et al., 2018] allows a quadrotor to navigate autonomously in cluttered and GPS-denied environments. It consists of a set of modules that work together to allow fast autonomous navigation of an aerial robot through an unknown environment. The system has been designed so that all sensing and computation occur onboard the robot. Once the robot has been launched, there is no human interaction necessary for the robot to navigate to the goal.

LARICS UAV ROS stack [Markovic et al., 2023] attempts to distinguish and standardize both the hardware and software modular systems of an aerial vehicle capable of autonomous operations. The stack was tested in the form of a firefighting UAV competition organized as part of ICUAS'22, obtaining great participation. In total, 125 people across 16 teams attempted the Gazebo simulation qualifiers, while 5 teams qualified for the finals, successfully deploying their solution using the presented UAV platform.

MRS UAV System [Baca et al., 2021] is an aerial system built using ROS Noetic and meant to be executed entirely onboard. It can be deployed on any multi-rotor vehicle, given it is equipped with a PX4-compatible flight controller, for both indoor and outdoor use. It supports multi-robot experiments using Nimbro network communication and provides both agile flying and robust control.

RotorS [Furrer et al., 2016] is a modular Micro Aerial Vehicle (MAV) simulation framework, which allows a quick start to perform research on MAVs. The simulator was designed in a modular way so that different controllers and state estimators can be used interchangeably, while incorporating new MAVs is reduced to a few steps. The provided controllers can be adapted to a custom vehicle by simply changing a parameter file. Different controllers and state estimators can be compared with the provided evaluation framework. All components were designed to be analogous to their real-world counterparts. This allows the usage of the same controllers and state estimators, including their parameters, in the simulation as on the real MAV.

UAV Abstraction Layer (UAL) [Real et al., 2020] is a software layer to abstract users of unmanned aerial vehicles from the specific hardware of the platform and the autopilot interfaces. Its main objective is to simplify the development and testing of higher-level algorithms in aerial robotics by trying to standardize and simplify the interfaces with unmanned aerial vehicles. Unmanned aerial vehicle abstraction layer supports operation with PX4 and DJI autopilots (among others), which are current leading manufacturers. Besides, unmanned aerial vehicle abstraction layer can work seamlessly with simulated or real platforms and provides calls to issue standard commands such as taking off, landing, or pose and velocity controls.

XTDrone [Xiao et al., 2020] is a UAV simulation platform based on PX4, ROS, and Gazebo. XTDrone supports multirotors (including quadrotors and hexarotors), fixed wings, VTOLs (including quadplanes, tailsitters, and tiltrotors), and other unmanned systems (such as UGVs, USVs, and robotic arms). It is convenient to deploy the algorithm to real UAVs after testing and debugging on the simulation platform.

Aerostack2 [Fernandez-Cortizas et al., 2023], recently developed by our research group CVAR, is a modular and flexible framework specifically designed for aerial robotics. It supports multi-agent and multi-platform operations, making it highly adaptable. Its plugin-oriented architecture facilitates the easy addition of new components and enables dynamic system reconfiguration, making it easy to maintain and suitable for both research and industrial applications. A deep overview of Aerostack2 will be given in Section 3.2.

The comparison shows that almost all frameworks are open-source, while only half are actively maintained with an update in the current year. Most of the flight stacks are modular, but with a limited number supporting ROS 2 as middleware, relying instead on ROS. Approximately half of the frameworks support multi-agent operations, while multi-frame and multi-platform capabilities remain relatively uncommon. Nearly all frameworks provide rate output control, enabling advanced flight maneuvers. However, only a few adopt a plugin-oriented design. Additionally, the testing environments differ significantly; some frameworks have been evaluated exclusively in simulations, while others have undergone validation in real-world laboratory settings and industrial cases.

This detailed comparison highlights how the Aerostack2 framework builds upon and extends the capabilities of existing systems to provide a more comprehensive and adaptable solution for aerial robotics.

2.2 Navigation, Mapping, and Motion Planning

After a thorough analysis of the current state of the art in existing flight stacks, it becomes evident that there is a significant gap in their mapping and planning capabilities. These functionalities are often delegated to external frameworks rather than being integrated within the stacks themselves. To address this limitation comprehensively, we also review related work specifically focused on mapping and planning in aerial robotics.

2.2.1 Mapping

Regarding mapping, there are different algorithms depending on three main issues: sensors, data processing, and map representation [Saeedi et al., 2016]. The selection of the appropriate combination of sensors, processing methods, and map representation depends on the vehicle type, the complexity of the environment, and the intended application.

The choice of sensors directly impacts the mapping process, as they define how the robot perceives its surroundings. Common sensors include LiDAR for precise range measurements, RGB-D cameras for capturing depth and color information, ultrasonic sensors for detecting obstacles at shorter ranges, and RADAR for long-range detections. The suitability of a

sensor depends on the environment, the required resolution, and the available computational resources. For instance, LiDAR is preferred for high-precision 3D mapping, while RGB-D cameras are more practical for close-range indoor mapping.

Data processing involves the algorithms and techniques used to convert raw sensor data into meaningful maps. Data processing includes sensor fusion, filtering to reduce noise, and techniques like SLAM (Simultaneous Localization and Mapping) to estimate the robot's position while building a map. The complexity of processing varies depending on the environment's dynamics, the type of data, and whether the mapping is done in real-time or offline. Effective processing is critical to ensure accuracy and consistency, particularly in environments with moving objects or changing conditions.

Map representation refers to how the environment is modeled and stored. Six types of maps commonly found in the literature include: grid maps, feature maps, topological maps, semantic maps, appearance maps, and hybrid maps [Saeedi et al., 2016]. Each type has its strengths and trade-offs. The choice of representation depends on the specific application, such as navigation, manipulation, or environment monitoring, and the computational constraints of the robotic platform.

For navigation tasks, grid maps and topological maps are among the most commonly used representations. Grid maps, particularly occupancy grids, are highly effective for low-level path planning and obstacle avoidance because they provide a detailed and quantitative representation of free and occupied spaces in the environment. Frameworks like GMapping [Grisetti et al., 2007] and Cartographer [Hess et al., 2016] shine at generating occupancy grids in real-time, making them ideal for robots navigating indoor or moderately unstructured environments. These algorithms process sensor data such as LiDAR or depth cameras to produce detailed grid maps optimized for SLAM (Simultaneous Localization and Mapping).

In 3D scenarios, three-dimensional grid maps, also known as volumetric pixel (voxel) maps, are used; frameworks like OctoMap [Hornung et al., 2013] and Voxgraph [Reijgwart et al., 2020] are the best choice. OctoMap employs an octree-based representation to model the environment in 3D, efficiently storing free, occupied, and unknown spaces, which is particularly useful for robots requiring volumetric awareness. Voxgraph, with its emphasis on multi-session mapping and pose-graph optimization, extends these capabilities to dynamic and large-scale scenarios, supporting both exploration and navigation.

Topological maps, on the other hand, are well-suited for high-level navigation in large-scale or complex environments, as they represent the environment as a graph of nodes (key locations) and edges (paths connecting them). RTAB-Map (Real-Time Appearance-Based Mapping) [Labbé and Michaud, 2019] is a versatile framework for a 2D and 3D graph-based SLAM approach based on an incremental appearance-based loop closure detector. It uses visual odometry and integrates data from RGB-D cameras, LiDAR, and other sensors.

In more advanced applications involving semantic understanding or multi-modal representations, hybrid maps that combine the strengths of grid and topological maps are increasingly being used. Frameworks like Maplab [Cramariuc et al., 2023], Kimera [Tian et al., 2022] and MRPT [Blanco et al., 2006] further provide tools to integrate non-visual key points and semantic labels into maps. These maps enable robots to make higher-level decisions, such as

Table 2.3: Comparison of different mapping frameworks based on their capabilities.

Framework	Sensors	Data Processing	Map Representation
maplab 2.0 [Cramariuc et al., 2023]	IMU, LiDAR, Cameras, GPS	Multi-modal factor graph optimization	6DoF factor graph
Kimera [Tian et al., 2022]	IMU, Cameras	Distributed multi-robot dense metric-semantic SLAM	3D mesh pose-nodes graph
Voxgraph [Reijgwart et al., 2020]	LiDAR, RGB-D Cameras, IMU	Pose-graph optimization for multi-session maps	3D Voxel Grid
RTAB-Map [Labbé and Michaud, 2019]	LiDAR, Cameras	Incremental appearance-based loop closure detector	Topological node graph
Cartographer [Hess et al., 2016]	IMU, LiDAR, RGB Cameras	Graph-based SLAM with loop closure	2D/3D Occupancy Grid
OctoMap [Hornung et al., 2013]	LiDAR, RGB-D Cameras	Octree-based volumetric mapping	3D Grid Map (Octree)
GMapping [Grisetti et al., 2007]	LiDAR	Particle filter-based SLAM	2D Occupancy Grid
MRPT [Blanco et al., 2006]	IMU, LiDAR, RGB-D Cameras, GPS	Multi-modal SLAM with hybrid optimization	Hybrid Maps (Grid + Topological)

prioritizing specific paths or identifying contextual landmarks for navigation. By leveraging the strengths of these frameworks, developers can select the most appropriate mapping approach based on their application's requirements, from fine-grained obstacle avoidance to large-scale exploration and semantic navigation.

Table 2.3 links the frameworks to the main issues identified by [Saeedi et al., 2016], allowing for easy comparison and selection based on specific use cases in robotic navigation and mapping.

2.2.2 Motion Planning

Robotic navigation can be categorized into global navigation and local navigation, which correspond to different levels of planning and execution. Global navigation involves high-level and long-term planning to compute a path from a starting position to a goal location over long distances. Global navigation often relies on a map of the environment and uses path-planning algorithms to generate a feasible path. Local navigation, on the other hand, operates at a smaller scale, focusing on the immediate surroundings of the robot. It involves real-time adjustments to follow the global path while avoiding dynamic obstacles, adapting to changes in the environment, and ensuring smooth and safe motion. Local navigation relies on sensors for obstacle detection and uses trajectory planning and control algorithms for execution.

In recent years, advancements in navigation and planning for autonomous systems have focused on improving efficiency, adaptability, and robustness. Key trends include optimizing

real-time performance for dynamic environments [Hüppi et al., 2022; Romero et al., 2022; Tordesillas and How, 2021], high-speed navigation [Penicka et al., 2022; Teissing et al., 2024; Tordesillas et al., 2022], enhancing multi-robot coordination [Kondo et al., 2023], and integrating perception and planning to ensure safety [Song et al., 2023; Tordesillas and How, 2022, 2023].

Navigation frameworks integrate state-of-the-art trends alongside classic approaches, ensuring they remain versatile for diverse applications. As discussed earlier, global navigation focuses on finding optimal paths across the environment, typically relying on algorithms like A*, Dijkstra's algorithm, or sampling-based planners like RRT and PRM. Local navigation, in contrast, emphasizes real-time adaptability to dynamic environments, improving techniques such as the Dynamic Window Approach (DWA). Classic methods provide reliable groundwork that has been tested extensively, while modern trends integrate innovative techniques, such as optimization-based planning and machine learning enhancements. This dual approach ensures that these frameworks remain versatile and adaptable to diverse robotic applications. Below are some examples of frameworks that embody this integration, while Table 2.4 showcases their main attributes.

The classic ROS Navigation Stack[1] has long been a staple for autonomous navigation, offering robust implementations of algorithms like A* and Dynamic Window Approach (DWA) for global and local planning, respectively. Recent updates and plugins have introduced support for more advanced planners, such as Timed-Elastic Band (TEB) [Rösmann et al., 2017], integrating optimization-based methods that align with modern trends.

Primarily designed for robotic manipulation, MoveIt [Coleman et al., 2014] has been extended to support motion planning for mobile robots. By incorporating sampling-based planners like RRT* and hybrid optimization methods, MoveIt bridges classic path planning with trajectory refinement. These methods are essential for motion planning in sparsely constrained environments, where computational efficiency is a priority.

OMPL (Open Motion Planning Library) [Şucan et al., 2012] provides a suite of motion planning algorithms, including RRT* and PRM, while actively integrating modern advancements such as asymptotically optimal planners. Its modularity allows researchers to experiment with combining classical approaches and the latest trends, such as perception-aware planning.

An evolution of the ROS Navigation Stack, Navigation2 (Nav2) [Macenski et al., 2020] includes support for advanced path and trajectory planners, such as SmacPlanner [Macenski et al., 2024] and improved TEB. It also integrates hybrid approaches that combine grid-based global planning with kino-dynamic local planning, reflecting the current emphasis on adaptability and performance.

Designed specifically for autonomous vehicles, Autoware [Kato et al., 2018] incorporates classical grid-based approaches and modern optimization techniques for global navigation in large-scale environments. Autoware's ability to integrate with sensors and maps makes it an ideal choice for urban navigation and autonomous driving.

Despite the robustness and versatility of existing navigation frameworks, these solutions

[1] https://github.com/ros-planning/navigation

Table 2.4: Comparison of different navigation frameworks.

Framework	Path Planning	Trajectory Planning	Applications
ROS Navigation stack	Dijkstra-based and sampling planners	Limited trajectory generation	2D navigation for mobile robots
MoveIt! [Coleman et al., 2014]	Supports global planners (RRT, PRM, etc.)	Time-parameterized trajectories	Robotic arms, mobile manipulation
OMPL [Şucan et al., 2012]	Extensive planner support (RRT, PRM, FMT*)	Minimal direct trajectory support	Path planning for versatile robots
Navigation2 [Macenski et al., 2020]	Extensive global planner support (A*, RRT)	Trajectory smoothing and obstacle avoidance	Mobile robot navigation, advanced successor to ROS Navigation
Autoaware [Kato et al., 2018]	Lane/path planners	Polynomial-based trajectory generation	Autonomous vehicle navigation

primarily apply to ground robots or specific use cases, such as robotic manipulation or autonomous vehicles. While some aspects of these frameworks can be adapted for aerial robotics, several critical limitations highlight the need for a dedicated framework for aerial systems. Aerial robotics poses unique challenges, including operating in dynamic 3D environments, requiring real-time obstacle avoidance, optimizing energy consumption, and managing limited sensor payloads. Additionally, aerial platforms often rely on lightweight, computationally efficient modules and require robust multi-agent coordination for tasks like swarming and collaborative mapping.

These limitations highlight the need for a specialized framework designed specifically for aerial robotics. Such a framework would prioritize 3D environments, energy optimization, swarm collaboration, and address the unique constraints of drones, enabling reliable and efficient operation in complex and dynamic scenarios.

2.3 Execution Control Architectures

Deliberation has been one of the most studied fields in robotics in the last decades. Nowadays, it still is as it is considered to be the intersection of Robotics and Artificial Intelligence [Ingrand and Ghallab, 2017]. Deliberation refers to purposeful, chosen, or planned actions, carried out in order to achieve some objectives [Fox et al., 2014]. While many robotic applications perform effectively without requiring deliberation (e.g., stationary robots used in manufacturing and other structured environments, single-task devices such as vacuum cleaners, or tele-operated systems like surgical robots), the ability to deliberate becomes essential for robots operating autonomously in complex environments (e.g., industrial inspections or multi-robot exploration). Deliberation allows autonomous robots to reason about their actions before executing them, enabling goal-directed behavior in dynamic environments. Unlike reactive systems that respond purely based on current stimuli, deliberative systems can evaluate multiple courses of action, predict their outcomes, and select the most appropriate one based on long-term objectives.

The deliberative capabilities are often defined in the literature in three tiers: *planning*, *acting*, and *monitoring* [Bonasso and Kortenkamp, 1995; Kortenkamp et al., 2016; Volpe et al., 2001]. These layers have different names in the bibliography, such as decision layer, executive layer, and behavioral control layer [Molina et al., 2019]. Newer perspectives include one new tier, *learning*, which allows the robot to acquire, adapt, and improve through experience the models needed for deliberation [Ingrand and Ghallab, 2017].

Autonomous robots require switching between different tasks[2] to perform complex missions [Colledanchise and Ögren, 2018]. There are several ways to orchestrate the execution control. A well-known solution is Finite State Automata (FSA), or Finite State Machines (FSMs), first introduced in the late 1950s [Myhill, 1957; Rabin and Scott, 1959]. In the 1960s, Petri Nets (PNs) were proposed [Petri, 1962; Murata, 1989] including concurrent handling of transitions. Later, Firby presented in the late 1980s the idea of reactive action packages (RAPs) [Firby, 1987; Bonasso and Kortenkamp, 1995], a skill-based robot control offering better adaptability to the environment. Around the same time, Hierarchical FSMs (HSFMs or Statecharts) were introduced by Harel [Harel, 1987]. Statecharts offer hierarchical structuring, concurrency, and event-driven transitions compared to conventional FSMs. Finally, Behavior Trees (BTs) were first introduced in the gaming industry around the early 2000s as an alternative to FSMs for controlling non-player characters. One of the earliest notable uses was in the game Halo 2 (2004) by Bungie [Colledanchise and Ögren, 2018], where they replaced FSMs to improve modularity and scalability in AI behaviors. Afterward, BTs gained popularity in robotics [Marzinotto et al., 2014], which formalized their use in robotic decision-making. Newer approaches try to incorporate more flexible and richer knowledge representation techniques to facilitate better interaction between planning and execution [Cashmore et al., 2015; Tenorth and Beetz, 2013].

In the ROS ecosystem, there are several frameworks of the previous execution control systems. Table 2.5 summarizes the frameworks discussed in this section. SMACH [Bohren and Cousins, 2010] is a well-known implementation using representations based on Statecharts, but it scales quickly in complexity and does not foresee a mechanism to react to external events. Other subsequent HFSM systems appeared after SMACH. Reduced Finite State Machine (rFSM) focused on a model with the smallest number of primitives necessary for humans to construct practical coordination Statecharts [Klotzbücher and Bruyninckx, 2012]. ROSco built several robot manipulation behaviors with an underlying structure of HFSMs, showing robust and versatile results in realistic home scenarios. More recent works proposed improvements in HFSMs, as concurrency was introduced in RAFCON [Brunner et al., 2016] together with a graphical tool to facilitate the creation of concurrent flow controls, while FlexBE [Schillinger et al., 2016] incorporates flexible operator collaboration for changing the structure of the mission on the fly during runtime.

The main drawback of HFSMs is that the transitions between states must be specified manually and remain fixed, limiting the reuse of behaviors across different scenarios. Similarly, FSM suffers from scalability issues, making it difficult to manage complex robotic behaviors as the number of states increases. Moreover, HFSMs lack the flexibility of more advanced

[2]the term *task* used here is also denoted in the literature as actions or control modes, assuming that an activity can somehow be broken down into reusable sub-activities.

frameworks like BTs or PNs. As a result, the static composition of behaviors in HFSMs leads to limited capabilities for situation-dependent task execution or optimization, restricting adaptability in dynamic environments.

Behavior Trees can be structured similarly to Statecharts but provide a more expressive model. One of the first ROS implementations for executing plans is *behavior_trees* [Marzinotto et al., 2014]. Authors claim to present a more accurate and compact framework for robotic tasks. Later, an evolution of the software was proposed, ROS-BehaviorTree [Colledanchise and Natale, 2021; Colledanchise and Ögren, 2016], which rapidly became the de facto standard[3] in the ROS community, thanks to its BT editor called Groot[4]. Other systems appeared after, e.g. *py_trees*[5], but with limited impact and acceptance.

Despite their flexibility and modularity, Behavior Trees (BTs) have several limitations. BTs demand a stricter adherence to their framework, due to tick-based evaluation, hard-coded logic, and tree structure, unlike other approaches like Statecharts or RAPs which support orthogonality and transitions across different levels. Moreover, as the number of nodes increases, managing and designing large BTs becomes challenging, making them harder to debug and maintain due to complex nesting and reduced readability. Additionally, BTs lack adaptability, as they rely on predefined structures and do not allow modification during runtime.

The authors of [Ziparo et al., 2011] presented a framework based on Petri Nets that allows for intuitive and effective behavior design of multi-robot systems in dynamic environments named Petri Net Plans. Later, generalized Petri nets with rewards (GSPNRs) were introduced [Azevedo et al., 2021]. Their execution algorithm uses a predefined policy to coordinate a multi-robot system. A model-driven approach was addressed in DiNeROS [Ebert et al., 2024]. Their method claims to improve the development of safe and reliable ROS applications due to the integration of verification activities and model-driven design with code generation combined with runtime models.

Despite their strengths in modeling concurrency and synchronization, Petri Nets face several limitations that prevent their scalability and adaptability. Similarly to HFSMs, the number of possible states grows exponentially with system complexity. Additionally, the lack of hierarchical structure in basic Petri Nets limits modularity compared to other solutions like HFSMs or BTs. Furthermore, their execution control limitations restrict the ability to handle priorities, interrupts, and dynamic decision-making efficiently, necessitating additional mechanisms to adapt to real-world, evolving conditions.

Modern RAP approaches, such as CRAM [Beetz et al., 2010] and SkiROS [Rovida et al., 2017], often incorporate knowledge representation and symbolic planning within their architectures. The CRAM kernel includes the CPL plan language, based on Lisp, alongside the KnowRob knowledge processing system [Tenorth and Beetz, 2013]. Unlike CRAM, where the planning domain must be manually defined, SkiROS dynamically derives the planning domain at runtime based on the available skill set. Additionally, SkiROS does not introduce a domain-

[3] https://github.com/BehaviorTree/BehaviorTree.CPP
[4] https://www.behaviortree.dev/groot/
[5] https://github.com/splintered-reality/py_trees

System	Method	Robotic domain	Reference
SMACH	HFSM	Service robots	[Bohren and Cousins, 2010]
rFSM	HFSM	Manipulation and Mobile robots	[Klotzbücher and Bruyninckx, 2012]
ROSco	HFSM	Manipulation	[Nguyen et al., 2013]
RAFCON	HFSM	Multidisciplinary	[Brunner et al., 2016]
FlexBE	HFSM	Manipulation	[Schillinger et al., 2016]
CRAM	RAPs	Manipulation	[Beetz et al., 2010]
SkiROS	RAPs	Manipulation	[Rovida et al., 2017]
DiNeROS	PNs	Manipulation	[Ebert et al., 2024]
GSPNRs	PNs	Mobile robots	[Azevedo et al., 2021]
PNP	PNs	Multidisciplinary	[Ziparo et al., 2011]
behavior_trees	BTs	Humanoid Manipulation	[Marzinotto et al., 2014]
ROS-Behavior-Tree	BTs	Multidisciplinary	[Colledanchise and Ögren, 2016]

Table 2.5: Execution control frameworks in ROS.

specific language like CPL for low-level behavior design; instead, it relies directly on C++ code. Another distinction is that CRAM employs a proprietary planner, whereas SkiROS utilizes a modular design that supports integration with any PDDL-compatible planner.

Although modern RAP approaches offer high-level abstraction and context awareness, they have yet to achieve widespread application in real-world industrial tasks. This limitation arises partly from their reliance on a predefined world model, which tends to be either overly restrictive or too loosely defined. Additionally, their execution follows a rigid procedural structure, making it difficult to adapt dynamically to changing conditions during runtime. Furthermore, RAPs typically require significantly more computational resources than alternative methods, posing challenges for efficient deployment.

2.4 Multi-robot Minimal Sensing Exploration

The following review examines state-of-the-art research in multi-robot exploration and the development of navigation strategies for nano-drones with minimal sensing. It highlights the interplay between hardware constraints, algorithmic design, and real-world applications, with an emphasis on cooperative strategies, lightweight mapping frameworks, and robust coordination mechanisms.

Autonomous exploration of unknown environments has been a common robotics research topic over the last two decades. One of the most popular approaches is a frontier-based exploration technique, first introduced in [Yamauchi, 1997]. The main concept behind frontier-based exploration is moving to the boundary where explored space meets uncharted territory to maximize new information about the world. By constantly moving to these new frontiers, a robot can expand its map into unexplored areas until the entire environment is fully mapped. This technique is specially useful for exploring unstructured environments due to the lack of morphology that structured scenarios provide and other exploration techniques exploit.

Burgard et al., 2005 present one of the first frontier-based exploration solutions for a team of robots. Their approach chooses the appropriate target points for the individual robots so that it simultaneously takes into account the cost of reaching a target point and its utility,

which is reduced whenever the target point is assigned to a robot. In Mannucci et al., 2018 propose a 3D exploration strategy based on a combination of local and global information for a team of flying robots with constrained payload. Motivated by the insufficient exploration rate, B. Zhou et al., 2021 provide FUEL, a hierarchical framework that can support fast drone exploration in complex unknown environments. Later, they extend their prior work for a fleet of decentralized drones, suggesting a new system named RACER [B. Zhou et al., 2023]. They decompose the unknown space into hgrid and distribute the task units among quadrotors by a pairwise interaction, assuming asynchronous and unreliable communication. Hui et al., 2024 consider a limited communication bandwidth between the multi-drone system. Their decentralized exploration planning framework, named DDPM, uses a lightweight information structure which enables exploration with little overlap. Each drone maintains a topological graph to save the information structure, serving as a high-level understanding and abstraction of the environment. Bartolomei et al., 2023 suggest a higher-level reasoning for multi-robot missions. Each individual drone has the ability to alternate between various execution modes. This balance involves a cautious exploration of yet completely unknown regions and more aggressive exploration of smaller, unfamiliar areas within the space.

Nevertheless, despite the progress in the topic in recent years, there are big limitations to apply these solutions on nano-drones. All systems previously cited require heavy sensors, such as depth cameras or 3D LiDAR, and high computational power of their algorithms. Resource restrictions on nano-drones rule out the currently available solutions for frontier-based exploration.

Recently, several studies have shown enormous improvement in autonomous navigation and mapping [Niculescu et al., 2024; H. Zhou et al., 2022], both key components to fulfill an exploration. As a result of the extreme payload constraints of nano-sized drones, previous works have sought alternative exploration strategies.

A promising bug exploration algorithm was introduced by [McGuire et al., 2019]. This algorithm empowered a swarm of nano quadcopters to effectively navigate through unfamiliar and cluttered structured environments, ultimately returning to their initial starting location. In [Duisterhof et al., 2021] proposes Sniffy Bug, a bug-inspired autonomous swarm to seek gas leaks in unknown environments. Their exploration strategy relies on the observed gas concentrations by the swarm, which indicates search directions, while the bug navigation is performed in two stages: line following and wall following. In [Lamberti et al., 2023] presents a bio-inspired autonomous exploration for object detection. The authors compare four different navigation policies, from random to scan spinning, showing the pseudo-random policy the best result. [Pourjabar et al., 2023] suggests a system design to achieve robust autonomous exploration capabilities for a swarm of nano-drones. Their multi-sensory approach combines lightweight single-beam laser ranging, low-resolution camera, and an ultra-wide-band-based ranging module.

Chapter 3

Materials and Methods

The development of autonomous UAV systems requires a structured approach to ensure reliability, adaptability, and robustness in real-world applications. This chapter presents the methodology followed in this research, combining a formal systems engineering process with a targeted analysis of existing UAV frameworks.

First, in Section 3.1 we describe a structured development methodology, considering established approaches such as the V-model [Forsberg and and, 1992] and the SIMILAR process [Bahill and Gissing, 1998]. These models provide a systematic framework for iterative development, validation, and refinement of UAV autonomy strategies.

Second, we delve into the architecture and components of Aerostack2 in Section 3.2. The section aims to provide the necessary context for understanding the principles and capabilities of the framework.

Third, in Section 3.3 we analyze how this thesis builds upon and extends the Aerostack2 framework, identifying gaps in its current capabilities and addressing them through novel contributions. This includes enhancements in navigation, dynamic task execution, and cooperative exploration strategies.

3.1 Methodology Process

This research follows a methodology combining elements from the V-Model [Forsberg and and, 1992] and the SIMILAR process [Bahill and Gissing, 1998], two well-established systems engineering frameworks that guide the iterative design and validation of complex systems. The process unfolds through a series of interdependent steps, ensuring that the research remains aligned with both theoretical foundations and practical applications. The methodology is structured into six key steps:

1. *Analysis of the problem*: The process begins with an initial analysis of the problem domain, identifying the requirements for UAV autonomy in complex environments and complex missions. This step involves understanding the operational context and the

limitations of existing frameworks, particularly Aerostack2. By reviewing state-of-the-art approaches, this phase establishes the research objectives and defines the functional and performance requirements that the proposed methodology must address.

2. *System and software design*: Once the requirements are defined, the system is conceptualized through a structured design phase. This step includes high-level system modeling that describes the UAV's autonomy architecture, incorporating new components that meet the defined requirements.

3. *Implementation and Integration*: Once the design is established, the implementation and integration phase develops the proposed algorithms and system functionalities. This step involves translating conceptual designs into software components, ensuring that they align with the predefined architecture. The implementation follows an iterative approach, where individual components are integrated and tested incrementally.

4. *Testing and Validation*: The testing and validation phase evaluates the performance of the developed system under simulated and real-world conditions. Validation is performed at multiple levels, following the V-Model's principle of continuous verification. Individual modules are tested in isolation before being integrated into a complete system.

5. *Deployment and Iteration*: After successful validation, the research moves to the deployment and iteration phase. The system is tested in increasingly complex scenarios, refining its performance based on empirical results. Unexpected challenges encountered in real-world deployments provide insights that drive improvements, following the adaptive principles of the SIMILAR process. The iterative cycle ensures that the system remains responsive to changing requirements and practical constraints, gradually enhancing its operational effectiveness.

6. *Dissemination*: Finally, the dissemination phase ensures that the research findings contribute to the broader robotics and UAV community. The developed methodologies, software, and experimental results are documented and shared through academic publications, open-source contributions, and collaborative projects. This step ensures that the work not only advances the field of UAV autonomy but also provides valuable resources for future research and development in aerial robotics.

By structuring the research into these six phases, this methodology ensures a systematic and iterative approach to developing autonomous UAV systems. It balances theoretical advancements with practical validation, guaranteeing that the proposed solutions are both innovative and applicable to real-world scenarios.

3.2 Aerial Robotic Framework on Target: Aerostack2

Aerostack2 serves as the foundation for the implementation and validation of the methodology outlined in this section. The structured approach, based on the V-Model and SIMILAR process, provides a systematic way to analyze, design, implement, and validate autonomous aerial systems, aligning with the modular and behavior-based principles of Aerostack2.

As briefly introduced previously in chapter 2, Aerostack2 represents a comprehensive solution

for addressing the challenges of aerial robotics. This section provides an in-depth overview of the framework, detailing its architecture, key features, and core functionalities, as well as its design principles and implementation in version 1.0.9[1]. This specific version reflects the latest stable release at the time of writing, incorporating all implemented features and improvements up to this point. All the information gathered here, including figures and tables, is taken from the cited Aerostack2 paper [Fernandez-Cortizas et al., 2023] and from the official documentation website[2].

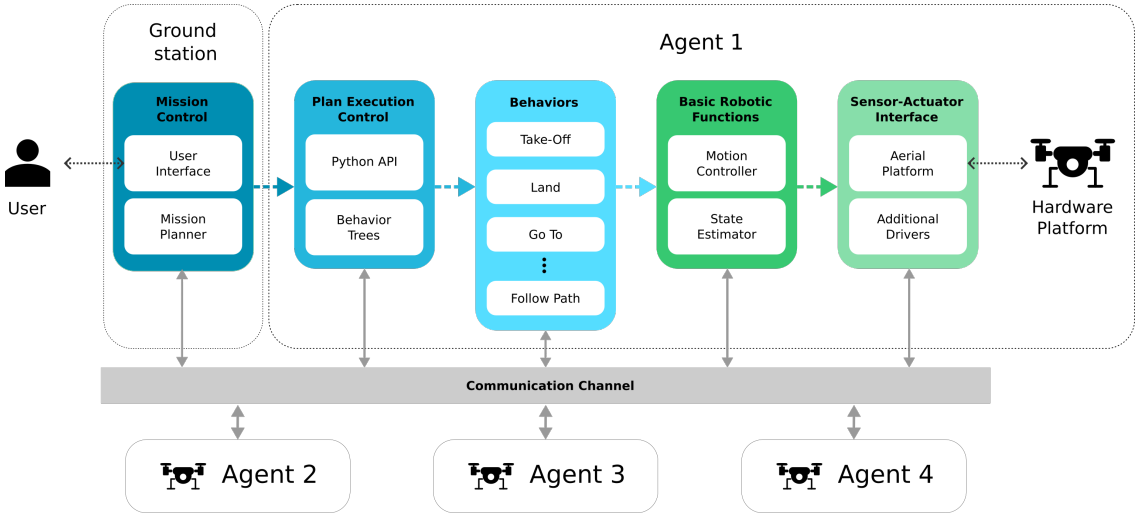


Figure 3.1: Overview of Aerostack2 framework structure. Each hierarchical layer are represented by blocks in colors. White blocks constitute components, ROS 2 nodes or processes, from each layer. Big arrows in colors pointing right show relationships between layers where *commands* travel from left to right. Information flow between layers is bidirectional through communication channel, which also serves to communicate different agents.

Aerostack2 is built on top of ROS 2, taking advantage of its improved communication middleware and multi-robot capabilities. It introduces a custom node architecture where each node, referred to as a component, has well-defined interfaces based on ROS 2 messages, topics, services, and actions. This design ensures consistent communication patterns and facilitates interoperability between components. By enforcing fixed interfaces, Aerostack2 promotes modularity, making it easier to integrate new functionalities and enabling more reliable and scalable development of autonomous aerial systems.

Aerostack2's structure is organized in different hierarchical layers, as can be seen in Figure 3.1. Each layer is composed of one or more components (ROS 2 nodes or processes). Following a logical top-down layered architectural pattern (in Figure left-to-right), a component at one layer can *command* one or more components of the subsequent lower layer or its own layer. However, they cannot use components of the higher layers. Relation is strictly unidirectional, but *commanding* only the layer immediately below does not occur sometimes in practice, lessening the structural restriction. Communication between components is explained later in

[1] https://github.com/aerostack2/aerostack2/tree/1.0.9
[2] https://aerostack2.github.io/

the section, although information can flow bidirectionally through the communication channel. Layers from right to left (or bottom to top) are:

- *Sensor-actuator interface*. Bottom level abstraction layer. Components in this layer serve as interfaces with aerial platforms and sensors. Aerostack2 defines standard interfaces via abstract classes that allow operating with physical and simulated platforms indistinctly. Pixhawk, DJI OSDK and Crazyflie are some of the physical platforms available, while the simulated platforms supported are Gazebo (former Ignition Gazebo) and Gazebo classic (gazebo11). Besides, many different sensor interfaces are provided, e.g., USB cameras, RealSense depth cameras, among others.
- *Basic robotic functions*. This layer contains a set of components that are essential for autonomous operation of aerial robots. There are two components that implement robotics-specialized algorithms such as motion control and state estimation.
- *Behaviors*. This layer includes a set of components corresponding to different robot behaviors for autonomous operation. Each behavior component encapsulates the implementation a particular skill (e.g., take off, generate trajectory, take picture, etc.) together with mechanisms for execution monitoring and self-supervision. Compared to directly use state estimators and motion controllers, behaviors provide an accessible and uniform method to perform autonomous operations.
- *Plan execution control*. This layer offers mechanisms for defining and supervising mission plans, enabling developers to specify complex tasks by activating and deactivating robot behaviors, thus supporting the autonomous execution of complex missions.
- *Mission control*. Top level layer provides user-friendly applications for newcomers, operators and developers. These components support users with tools for mission planning, system monitoring, and debugging, ensuring an accessible and efficient experience.
- *Communication channel*. Sidecar layer. It contains common utilities, such as message definitions and shared interfaces, for information exchange along all layers in the stack. Notice that through this layer only flows information bidirectionally, while commands follow the layered pattern design explained before.

The motivation behind adopting a layered architectural model is modularity, modifiability, and usability. A key advantage of this modular approach is its flexibility: developers can choose to use the framework in its entirety or selectively utilize individual layers or components, such as the state estimator, to build applications for specific needs. This design provides low-level abstraction for robotics developers while simultaneously offering high-level simplicity and accessibility for users and operators. The following sections detail each layer, highlighting their distinct functionalities and contributions to the overall framework.

3.2.1 Middleware and Component Communication

Aerostack2 utilizes a standardized data channel that manages the concurrent operation of multiple processes within the robot architecture, facilitating seamless interaction among its components. This communication framework adheres to conventions and standards relevant

to aerial robotics, ensuring compatibility and efficiency. While some of these standards align with those established by the ROS 2 community, Aerostack2 also introduces its own specific conventions, including custom message types and naming schemes for ROS 2 topics, services, and actions. These carefully defined standards enhance process interoperability and system integration.

In the robot control loop, five communication interface groups are defined: motion references, actuator commands, sensor measurements, self-localization, and emergency channel. The content of each channel group is described below:

- *Sensor measurements* are values corresponding to direct measurements recorded by sensors. This includes, for example, images from cameras, data from IMU, data from GPS, etc. Platform and external sensor usually publish these messages, while the state estimator use the sensor measurements to fuse them and estimate the robot state. Also, external devices such as a motion capture system utilize this channel to send the robot state to the state estimator.
- *Actuator commands* correspond to commands that are directly understandable by aerial platforms, such as thrust, velocity or position. The different control modes available are defined by a combination of the previous cited actuator commands. With them, each controller mode information is sent to the platform.
- *Self localization* values correspond to the robot localization (position and orientation) in the environment together with kinematic values (e.g., speed) as they are believed by the robot. These values may be obtained by the state estimator, for example, by fusing sensor measurements with the help of extended Kalman filters (EKF).
- *Motion reference* messages are motion values considered as goals by motion controllers. This includes, for instance, desired values for pose, speed, trajectory, among others.
- *Emergency* channel to publish alerts and warnings. All components can publish in this channel, and each one must know how to properly act based on the received messages. For that, a severity level is used to indicate the importance of the message and which component must react to it.

Table 3.1 presents examples of ROS topics used in Aerostack2, along with their corresponding standard names and message types.

3.2.2 Platforms and Sensors Interfaces

A key feature of the Aerostack2 framework is its independence from the specific physical platform used. To enable the seamless implementation of autonomous aerial systems across different deployment platforms - provided that sensory inputs and actuators are compatible - Aerostack2 incorporates an abstract class called `AerialPlatform`. This class is responsible for managing the integration of various aerial platforms into the framework. By abstracting platform-specific details, it simplifies the addition of new platforms by providing guidance on integration steps, overloading functions of the Platform class, and maintaining full compatibility with the framework's components. The AerialPlatform interface is responsible

ROS topic name	Message type	Description
`sensor_measurement/camera`	`sensor_msgs/Image`	Raw image data received from camera.
`sensor_measurement/imu`	`sensor_msgs/Imu`	Inertial Measurement Unit data.
`actuator_command/twist`	`geometry_msgs/TwistStamped`	Actuator command for the multirotor specifying vx, vy and vz (m/s: meters per seconds) and roll rate, pitch rate and yaw rate (rad/s: radians per seconds).
`actuator_command/thrust`	`as2_msgs/Thrust`	Actuator command for the multirotor specifying thrust (N: Newtons).
`self_localization/pose`	`geometry_msgs/PoseStamped`	Localization of the multirotor specifying x, y and z (m: meters) and roll, pitch and yaw (rad: radians).
`self_localization/twist`	`geometry_msgs/TwistStamped`	Localization of the multirotor specifying vx, vy and vz (m/s: meters per seconds) and roll rate, pitch rate and yaw rate (rad/s: radians per seconds).
`motion_reference/pose`	`geometry_msgs/PoseStamped`	Reference for the motion controller specifying x, y, and z (m: meters) and roll, pitch and yaw (rad: radians).
`motion_reference/trajectory`	`as2_msgs/TrajectoryPoint`	Reference trajectory point specifying x, y and z (m: meters), vx, vy and vz (m/s: meters per seconds), ax, ay, and az (m/s^2: meters per seconds squared) and yaw angle (rad: radians).
`alert_event`	`as2_msgs/AlertEvent`	. Alert message specifying severity level.

Table 3.1: Example ROS topics and messages used by Aerostack2 for interprocess communication.

for collecting sensory data from the aircraft and distributing it to the rest of the system. Additionally, it handles actuator commands and other requests from Aerostack2's layers, ensuring they are transmitted to the aircraft in a manner tailored to its specific requirements. This design guarantees flexibility and scalability for diverse aerial robotics applications. Figure 3.2 shows a diagram of the components of the layer.

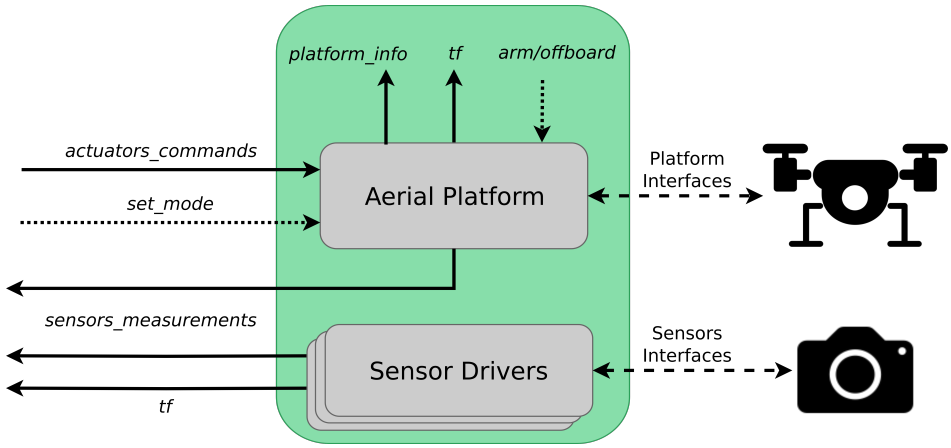


Figure 3.2: Sensors and actuators interfaces diagram. Grey blocks represent ROS 2 nodes. Dashed lines stand for generic ROS 2 interfaces, while solid lines are topics and dotted lines are services.

The proposed framework's interaction with the platform interface facilitates the integration of both physical and simulated interfaces, without requiring the rest of the framework to distinguish between the two. This aerial platform interface eases the transition from simulation to real-world deployment since the logic modules remain agnostic to whether the system is operating on a real platform or in simulation. In addition, it also simplifies the implementation of heterogeneous aerial systems using different platforms.

In some applications, additional sensors or actuators may be required to perform specific tasks, such as controlling a mounted arm or utilizing specialized sensors. To accommodate such needs, Aerostack2 includes the `Sensor` abstraction class, which streamlines the management and integration of external sensors. Furthermore, the framework is fully compatible with all ROS 2 drivers, ensuring seamless integration with existing solutions developed by the broader ROS 2 community. This compatibility allows developers to leverage prior efforts while expanding the system's capabilities for specialized applications.

3.2.3 Basic Robotic Functions

The Aerostack2 basic robotic functions comprise a suite of software components designed to execute fundamental algorithms essential to aerial robotics. These components perform critical tasks such as motion control and state estimation, ensuring the autonomous operation of various types of aerial robots.

Aerostack2 components are built to be general and reusable, supporting alternative algorithms implemented as plug-ins. This modular design allows the selection of specific plug-ins based on the requirements of each application or scenario, offering flexibility and adaptability. This approach is particularly meaningful when applied to basic robotic functions, as these modules are vital to the overall operation of the system.

Within this architecture, basic functions are managed by a function manager, which oversees the loading and integration of plug-ins and coordinates their interaction with other framework components. The function manager also supports meta-control features like plug-in replacement or plug-in bypassing, enabling direct input-output linkage when necessary. Input and output adapters further enhance system flexibility by aligning the data flow between function plug-ins and the broader Aerostack2 framework.

Among these, two core functions form the foundation of the framework, with their connection depicted in Figure 3.3. Both components are described in the following subsections.

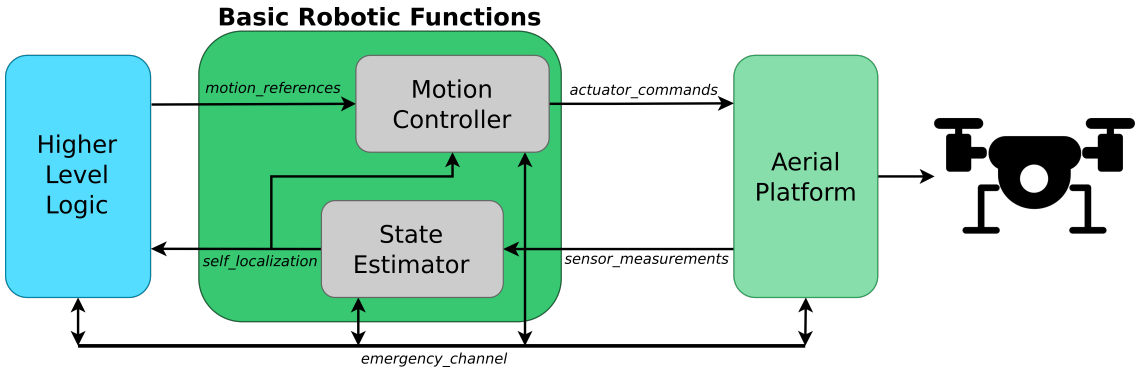


Figure 3.3: Basic robotics functions connection scheme in Aerostack2. Grey blocks are ROS 2 nodes and solid lines are topics interfaces.

Motion Controller

The motion controller component is responsible for handling motion reference commands from the upper layers and converting them into actuator command signals that the aerial platform can execute. Within this component, controllers process motion references to generate the necessary control signals for the platform.

The control mode negotiator plays a central role in ensuring proper integration and processing of motion references. It manages the list of possible input-output control mode pairs and handles requests to change the existing control mode. The input-output adaptation modifies the references to align with the active controller's requirements, such as ensuring they are expressed in the appropriate reference frame. Additionally, it adjusts and post-processes the resulting actuator command signals to match the desired format for the aerial platform.

To enhance flexibility, the plugin selector within this component can load multiple controllers implemented as plug-ins. It monitors the performance and operational state of each controller, enabling dynamic selection and seamless switching between controllers as needed. Current plugin implementations are PID or Differential Flatness (DF), while a Model Predictive Controller (MPC) is now under development. This modular approach ensures adaptability across various scenarios and aerial platforms.

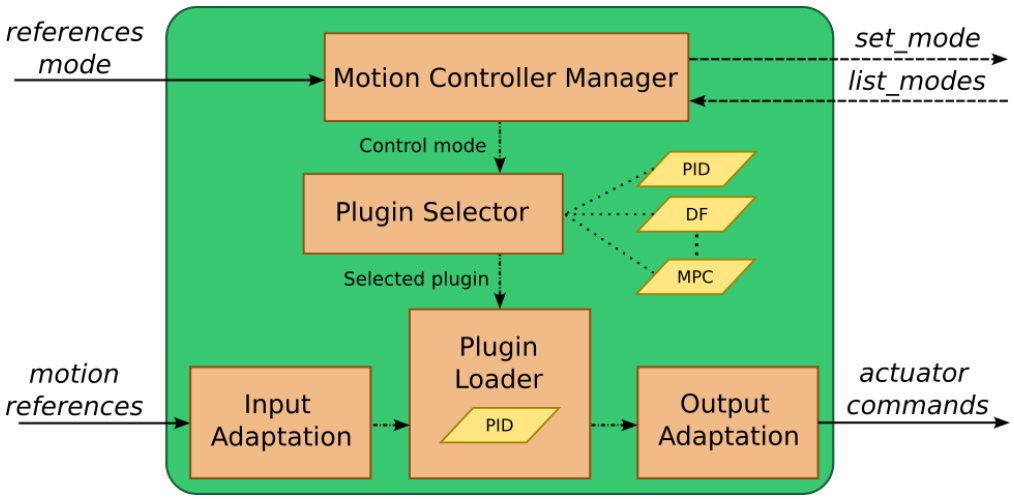


Figure 3.4: Motion controller component diagram. Orange boxes represent the motion controller key elements and how they interact internally. Plugins are represented with yellow trapezoids.

The internal architecture of the motion controller component is illustrated in Figure 3.4, showing its key elements:

- *Motion Controller Manager*: Manages the list of available control modes and handles the negotiation to set a specific control mode.
- *Plugin Selector*: Chooses the most appropriate controller plug-in based on the current control mode and operational requirements.
- *Plugin Loader*: Loads and executes the selected control plug-in, such as PID, DF, or

MPC controllers.

- *Input Adaptation*: Prepares motion reference inputs to match the selected plug-in's expected format.
- *Output Adaptation*: Processes control outputs from the plug-in to ensure compatibility with the aerial platform's actuator commands.

This architecture, depicted in Figure 3.4, highlights the system's modularity and extensibility for managing diverse control strategies.

State Estimator

The state estimator component is designed to combine data from various sensors to estimate the aircraft's state, which includes its position, orientation, and speed over time. This component supports multiple state estimation algorithms implemented as plugins, offering flexibility for different scenarios and aerial platforms. Existing implementations range from pure odometry estimation to reliance on external localization systems such as motion capture systems or devices like Intel RealSense T265.

At the core of this component is the state estimator manager, which plays a pivotal role in generating transformation frame trees (TF trees) that are utilized throughout the Aerostack2 framework. These trees provide the spatial relationships between different reference frames, ensuring consistent and accurate navigation and control. Furthermore, the manager is responsible for monitoring the available state estimation plugins and selecting the most appropriate one based on the operational environment or task requirements.

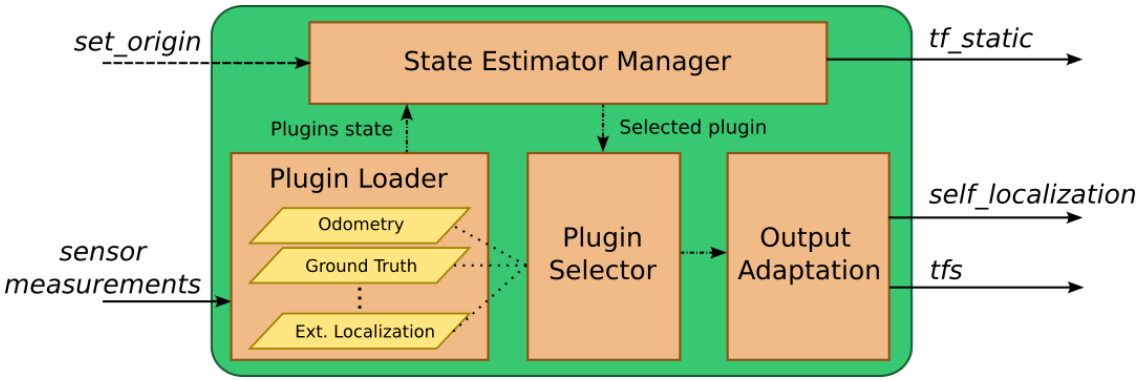


Figure 3.5: State estimator component diagram. Orange boxes represent the estimator key elements and how they interact internally. Plugins are represented with yellow trapezoids.

The modular design of the state estimator also allows for seamless adaptation of input and output data. Specifically:

- *Manager*: Builds TF trees. When operating outdoors, it is in charge of storing the geodesic operation origin, allowing the system to be able to work in the Euclidean space and translate geodetic coordinates.
- *Plugin Loader*: Enables the integration of various estimation algorithms, such as

odometry, ground truth, or external localization systems.

- *Plugin Selector*: Chooses the algorithm plugin based on the user pre-sets or decided in runtime using predefined rules, mission context and sensor availability.
- *Output Adaptation*: Processes the state estimation results to provide the necessary outputs, including estimated position, speed, geodetic data, and static transformation frames.

This architecture, as illustrated in Figure 3.5, highlights the flexibility and robustness of the state estimator component, making it suitable for a wide range of aerial robotic applications.

3.2.4 Behaviors

Aerostack2 introduces a specialized component known as a *behavior*, which provides a logical abstraction layer to simplify the formulation of mission plans. This abstraction allows developers to work at a higher level of logic, avoiding the complexity of directly interacting with state estimators or actuator controllers. Each behavior corresponds to a specific robotic skill, such as flight-related motions (e.g., takeoff, landing, hovering, or path-following) or other capabilities like video recording or inter-agent communication. Table 3.2 lists the collection of behaviors available in Aerostack2.

Table 3.2: List of behaviors available in Aerostack2 grouped by task type.

Group	Behavior	Description
Motion	`TAKEOFF`	Change UAV state from landed to flying. Aircraft must be armed and in offboard mode.
	`GO_TO`	Move the UAV to a given position in the desired frame.
	`FOLLOW_PATH`	Move the UAV along a given path in the desired frame.
	`FOLLOW_REF`	Keep the UAV at a given pose reference in the desired frame.
	`LAND`	Change UAV state from flying to landed.
Perception	`DETECT_ARUCO`	Detect ArUco markers from image and publish the result.
	`POINT_GIMBAL`	Orientate gimbal to aim to point.
`GENERATE_POLYNOMIAL_TRAJECTORY`		Convert a list of waypoints to a trajectory.

The concept of behaviors in robotics has its roots in the behavior-based paradigm [Brooks, 1986, Arkin, 1998], which originally emphasized reactive behaviors. Over time, this idea has evolved into behavior-based systems [Michaud and Nicolescu, 2016], encompassing both reactive and deliberative behaviors. In Aerostack2, behaviors span a wide range: from simple reactive actions, such as maintaining a hover, to more complex skills that involve higher-level reasoning, such as generating a trajectory to reach a specific destination. This versatility makes behaviors a powerful tool for creating autonomous aerial systems with varying levels of complexity and intelligence.

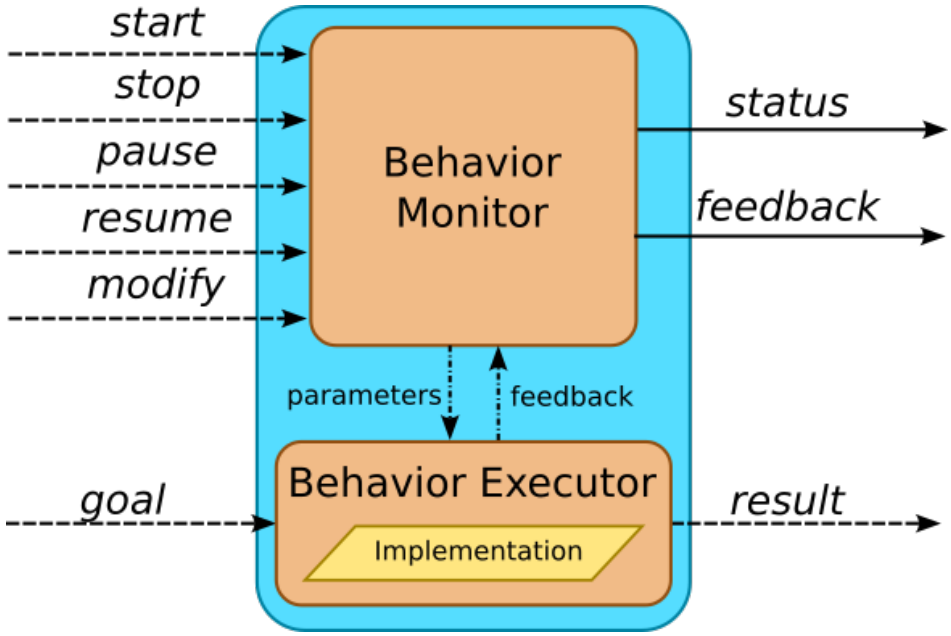


Figure 3.6: Characteristics of the robot behavior used in Aerostack2 to implement robot skills following a distributed and modular approach. Orange boxes are internal behavior elements, while the implementation (plugins) are represented with yellow trapezoids.

Aerostack2 incorporates a behavior template architecture that facilitates the creation and management of behaviors by separating execution and monitoring into two distinct parts in the component (see Figure 3.6). The behavior executor is responsible for performing the specific algorithm associated with each behavior, while the behavior monitor ensures proper operation by interacting with the executor before, during, and after execution. This interaction enhances adaptability and robustness by enabling dynamic adjustments and having different implementations plug-ins for the same behavior. For instance, during a landing maneuver, the behavior monitor might adapt the execution to either descend until contact with the ground or use vision to align the drone with a landing target, depending on the environmental conditions.

The behavior executor is responsible for executing tasks using the specific algorithm for each scenario. Employing various plugins when a task can be accomplished through multiple methods, each linked to distinct algorithms and implementations, is recommended. For instance, a land behavior can be executed by reducing altitude until vertical speed is zero or by using visual aids. Both cases represent the same skill, while the implementation is wrapped in the plugin.

Unlike centralized approaches, Aerostack2 employs distributed execution and monitoring, where each behavior independently runs and verifies the environmental conditions and detects operational issues. For example, a visual marker recognition behavior may check lighting conditions to ensure proper functioning, while a path-following behavior can identify failures, such as not reaching the destination within a predefined time or moving in an incorrect direction. It is worth mentioning that more than one behavior can run in parallel depending

on the mission requirements. This distributed approach improves system robustness and facilitates error detection and recovery during mission execution.

The behavior components in Aerostack2 provide a standardized module for robust and flexible plan execution. They encapsulate the algorithmic details, offering a uniform interface for all behaviors, which simplifies mission planning. Using behaviors, mission plans are formulated as controlled sequences of activations and deactivations of concurrent behaviors, with outcomes reported as success or failure. This enables dynamic decision-making for the next steps in the mission.

The uniform interface for controlling behavior execution includes basic services such as `start`, `pause`, `resume`, and `stop`. Additionally, the `modify` service allows parameter adjustments during execution without interruption. Behaviors also provide information about their execution states (e.g., idle, running, or paused) and periodic feedback with detailed updates about the ongoing behavior. Notably, Aerostack2 behaviors are implemented as extended ROS 2 actions, fully compatible with the standard, but enhanced with additional functionalities such as pausing, resuming, or modifying goals during execution. This implementation aligns with ROS 2 conventions while offering advanced features made for aerial robotics applications.

3.2.5 Plan Execution Control

Aerostack2 offers robust mechanisms for specifying and supervising mission plans, enabling developers to design and execute tasks that drones perform autonomously. Mission plans are formulated by specifying the activation and deactivation of robot behaviors, which allows for the combination of individual tasks to create complex, autonomous missions. To accommodate diverse user needs and levels of expertise, Aerostack2 provides two main approaches for generating and executing mission plans:

The Python API provides a flexible and powerful interface for interacting with the Aerostack2 framework. Through the `DroneInterface` class, developers can send commands to drones and retrieve meaningful status information. This method is especially useful for creating plans with intricate control regimes or for situations where dynamic interactions are required. The API includes functions for behavior activation and deactivation, as well as direct commands through motion reference messages, offering full control over the drone's operations. Code Snippet 3.1 shows a mission example built with the Python API.

Behavior Trees are one of the most widely used tools for describing and managing plans in robotics [Colledanchise and Ögren, 2018]. Their modularity and hierarchical structure simplify both the design and execution of mission plans. With Aerostack2, robot behaviors are fully compatible with the BehaviorTree.CPP library[3], through their ROS 2 action interface. This compatibility provides graphical monitoring during execution, making it easier to visualize and debug the mission's progress. Figure 3.7 shows a mission sketched with Behavior Trees.

These two mechanisms provide flexibility and scalability, allowing users to choose the most suitable method based on their application needs, whether they require detailed programming through Python, or modular visual structures using Behavior Trees. This versatility ensures

[3] https://github.com/BehaviorTree/BehaviorTree.CPP/tree/v3.8

that Aerostack2 can supply a wide range of users, from beginners to advanced developers, and support diverse aerial robotics applications.

```python
import rclpy
from as2_python_api.drone_interface import
    DroneInterface

# Initialize drone interface
rclpy.init()
drone = DroneInterface("drone_sim_0")

# Arm and change into offboard mode
drone.arm()
drone.offboard()

# Take Off
drone.takeoff(height=3.0, speed=1.0)

# Go To
drone.go_to(0, 5, 3, speed=1.0)

# Follow Path
drone.follow_path([[5, 0, 3], [5, 5, 3], [0,
    0, 3]], speed=1.0)

# Land
drone.land(speed=0.5)

# Exit
drone.shutdown()
rclpy.shutdown()
```

Code Snippet 3.1: Example of a mission plan programmed with Aerostack2 python API.

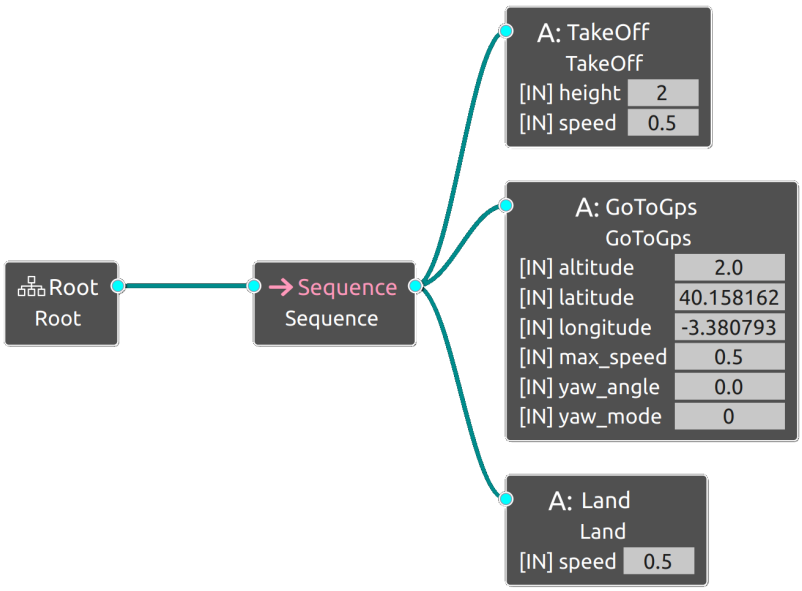


Figure 3.7: Mission plan defined via a Behavior Tree.

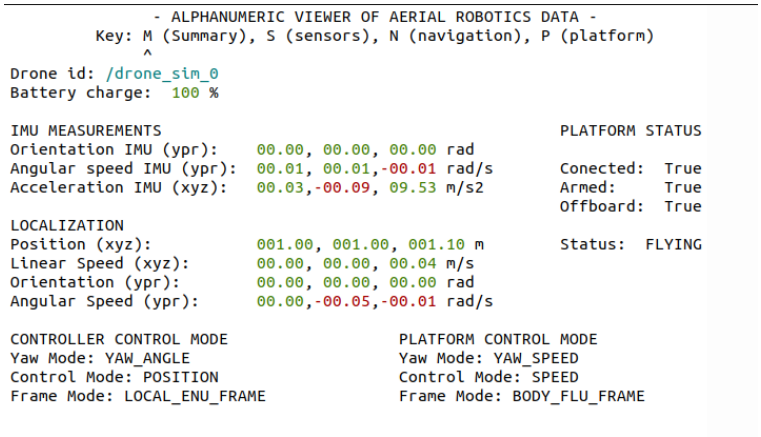
3.2.6 Mission Control and Supervision

The top layer of the Aerostack2 architecture is primarily dedicated to mission control and supervision through user interfaces. This layer not only facilitates the specification of mission plans but also offers tools to monitor and control their execution. Its components are designed to provide an intuitive interface for human operators, allowing them to define missions and supervise system behavior effectively. Depending on the user's level of expertise, these tools can be divided into two main categories: mission definition tools and mission supervision tools.

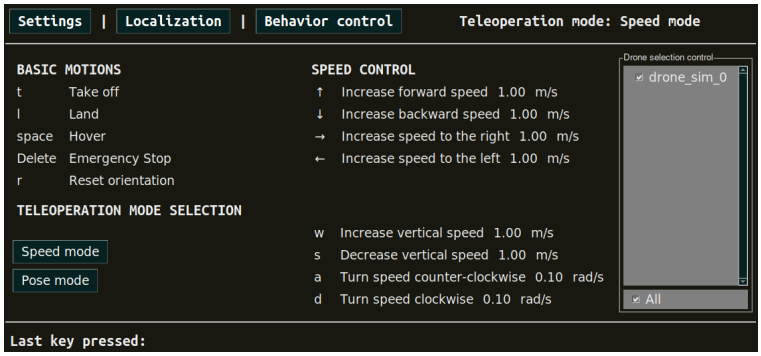
Mission Supervision Tools

Mission supervision tools provide detailed insights into the internal state of the system, making it easier for operators to understand and monitor what is happening during mission execution and take control if needed.

- *Alphanumeric Viewer*: This component displays a real-time overview of the system's operation by monitoring specific system variables, such as sensor measurements, state estimation values, and controller references. The information is organized into multiple panes, facilitating the quick identification of relevant data (see Figure 3.8a). This feature is particularly useful for debugging or understanding the behavior of complex aerial systems.
- *Keyboard Teleoperation*: This tool enables operators to manipulate the drone swarm directly by sending position and speed commands. It provides a simple and efficient way to test system behavior or take manual control in cases where the autonomous logic fails (see Figure 3.8b). Designed with ease of use in mind, this tool allows for rapid and secure testing and debugging of aerial systems.



(a) Alphanumeric view summary window.



(b) Keyboard teleoperation view.

Figure 3.8: Mission supervision tools available in Aerostack2.

Mission Definition Tools

Mission definition tools are designed to present information in a graphical, user-friendly manner, enabling non-developers to interact with and define aerial system missions with minimal technical expertise.

- *Graphical User Interface (GUI)*: Aerostack2 provides a web-based GUI[4] that simplifies interaction with the software framework. This tool is particularly valuable for rapidly planning missions for one or more drones. Users can generate automated paths to cover areas, use graphical elements such as geographic maps, and define graphical references. Mission planning can be conducted either online or offline, ensuring repeatability and flexibility. Figure 3.9 shows the tool working.

In addition to mission definition, the GUI enables real-time monitoring and modification of mission execution, displaying mission details graphically for easier supervision. The interface supports multiple users simultaneously, allowing operators to connect from various devices over a network. This architecture enhances the system's robustness and accessibility, as the web-based nature of the GUI ensures compatibility across a wide range of devices.

By providing intuitive mission definition and supervision tools, Aerostack2 bridges the gap between technical developers and end users, ensuring that both groups can interact effectively with the aerial robotics framework. This combination of flexibility and usability makes Aerostack2 a versatile solution for a wide range of mission scenarios.

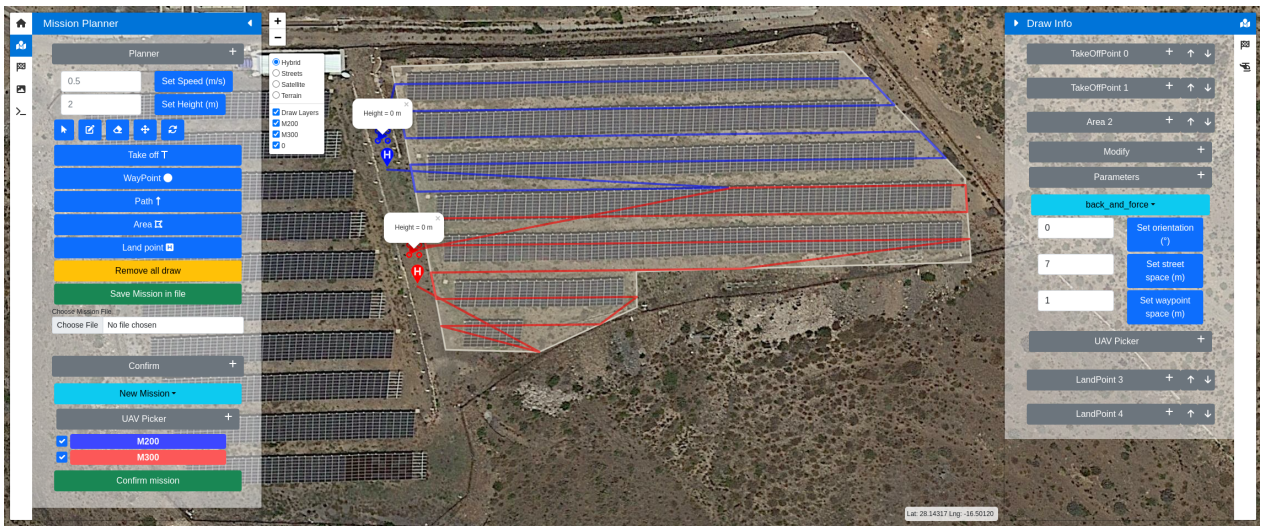


Figure 3.9: Aerostack2 Web-GUI mission planning view.

[4] https://github.com/aerostack2/as2_web_gui

3.3 Limitations in Aerostack2

In addition to following these methodological principles, this research directly addresses gaps in the Aerostack2 framework. While Aerostack2 provides a strong foundation for UAV autonomy, it presents limitations in key areas such as dynamic task planning execution, real-time navigation strategies, and exploration with minimal sensing. The research conducted in this thesis extends the framework by enhancing these capabilities, ensuring that UAVs can operate in complex, real-world environments with increased autonomy.

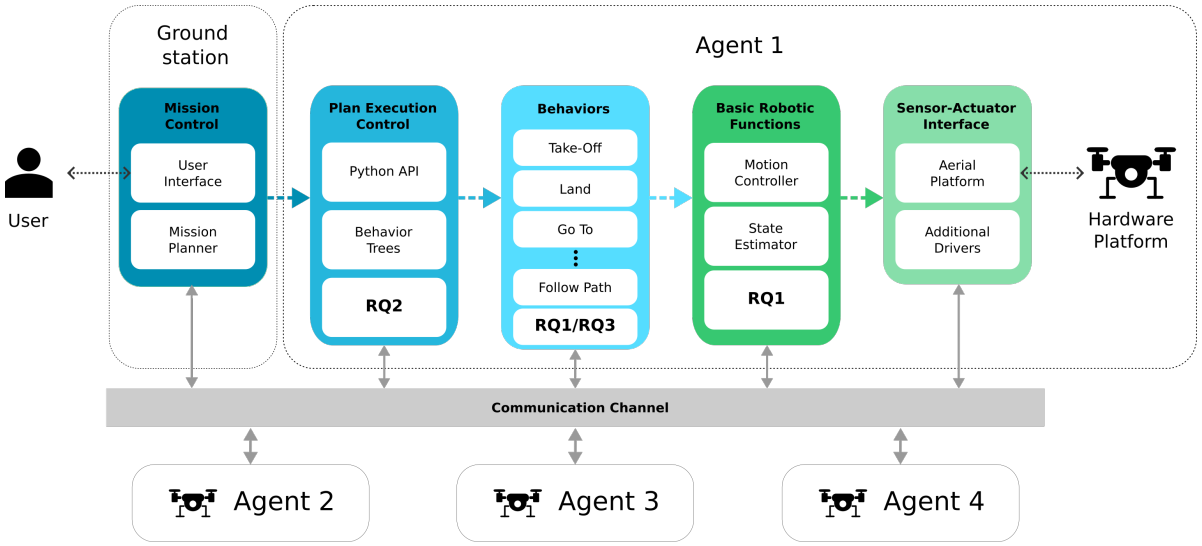


Figure 3.10: Proposal of contributions and integration in Aerostack2 design.

Figure 3.10 illustrates the Aerostack2 architecture, highlighting the integration of new components developed in this thesis to address its limitations. Each RQ is linked to specific contributions that extend the framework with additional functionalities:

RQ1 *Extending navigation capabilities*: This research question is addressed through the development of a new Mapping Component, integrated into the Basic Robotic Functions layer. This component enables UAVs to construct and maintain an internal representation of their environment, facilitating improved state estimation and navigation. Additionally, the new Path Planner Behavior, embedded within the Behaviors layer, enhances UAV path generation, allowing for more efficient and adaptive navigation strategies. These additions significantly improve UAV autonomy by providing robust motion planning and environmental awareness.

RQ2 *Dynamic task planning execution control for multi-UAV systems*: The new Plan Interpreter extends the Plan Execution Control module, enhancing how UAVs manage and execute complex missions. This component improves the interpretation of high-level plans, allowing for more dynamic and flexible task allocation among multiple UAVs. By integrating this capability, the system can adapt to environmental changes and mission updates in real time, ensuring a more robust execution strategy for cooperative UAV teams.

RQ3 *Exploration using minimal sensing on cooperative nano-drones*: To address this research question, a new Explorer Behavior has been introduced within the Behaviors layer. This component is specifically designed to enable UAVs to explore unknown environments using minimal sensing, improving cooperative strategies among multiple drones. The integration of this behavior, combined with the mapping and path planning enhancements, allows UAVs to efficiently navigate and explore unstructured spaces, even with limited onboard sensing capabilities.

By incorporating these new components, this thesis enhances Aerostack2's UAV autonomy in navigation, task execution, and exploration. These contributions make Aerostack2 more capable of handling real-world applications that demand adaptive multi-UAV cooperation.

Chapter 4

Multi UAVs Autonomous Missions

This chapter presents the core algorithmic and architectural contributions developed to enhance UAV autonomy in the areas of navigation, execution control, and exploration. It introduces the proposed components designed to enable reliable navigation in complex environments (Section 4.1), flexible execution of high-level missions (Section 4.2), and coordinated exploration using minimal sensing (Section 4.3). Each section details the rationale, system overview, and integration strategy of these components, focusing on how they address the research questions posed in this thesis. The solutions are implemented within the Aerostack2 framework and validated in real-world and simulated scenarios during the next chapter.

4.1 Design and Integration of a Navigation Component

Navigation is a fundamental capability for aerial robots, enabling them to traverse complex environments efficiently and safely while responding to dynamic changes. The existing limitations become particularly critical in scenarios that demand precise path planning, obstacle avoidance, or multi-robot coordination in unstructured environments. While Aerostack2 provides a comprehensive framework for aerial robotics, it currently lacks a dedicated component for autonomous navigation.

To address this gap, this work introduces new navigation components that integrate seamlessly into the Aerostack2 framework. These components are designed to provide advanced navigation functionalities, including (e.g., real-time path planning, obstacle detection and avoidance, and goal-directed movement), all while keeping Aerostack2's modular and plug-in-based architecture. By adhering to the framework's design principles, the new navigation components ensure compatibility with existing modules such as motion control and state estimation, while significantly enhancing the overall system's capabilities.

Recent Aerostack2 releases (v1.0.0 and above) already include these navigation components, making them readily available for use and further customization by developers. This addition not only fills a critical gap in Aerostack2's functionality but also demonstrates the framework's adaptability and potential for ongoing enhancement.

4.1.1 System Overview

The navigation capabilities proposed in this thesis are structured into two main subsystems: the mapping and the path planning subsystem. These two components operate in close coordination to enable navigation. Their design follows the modular philosophy of Aerostack2 and integrates seamlessly into its architecture.

The mapping subsystem is responsible for incrementally constructing a representation of the environment using onboard sensors. It processes sensor data—such as LiDAR or stereo cameras—to generate an occupancy grid or voxel-based map, which serves as the environmental model used by the planner. The subsystem maintains and updates this map in real time, ensuring that the navigation stack always has access to an up-to-date view of the surroundings.

The path planning subsystem interacts with the mapping subsystem to compute paths from the UAV's current pose to the target location. The system utilizes the updated map to avoid obstacles and navigate through the environment. It provides high-level planning capabilities to various Aerostack2 elements, including task planners or exploration algorithms, which request navigation commands.

These two subsystems are integrated into Aerostack2 through its modular structure. The path planner subsystem is designed as a new behavior, while the mapping subsystem fits as a new component for the basic robotic function layer. The planner behavior triggers other behaviors to correctly move to the generated waypoints, which provide low-level trajectory commands to the flight control stack. Meanwhile, the mapping subsystem feeds information to both the planner and other high-level modules, enabling safe navigation in unknown environments.

4.1.2 Map Server

The map server is a newly introduced component in Aerostack2, designed to provide aerial robots with the ability to generate, manage, and utilize maps for navigation and situational awareness. This component plays a key role in the navigation module by allowing drones to perceive and interact with their environment effectively.

All the component software explained below was developed to fulfill this thesis' purposes. The map server is mostly coded in C++, and it is available at the Aerostack2 repository[1] under the BSD 3-clause license.

The component is conceived as a new basic robotic function component, since a map is required to navigate through environments with obstacles, adhering seamlessly to Aerostack2 modular architecture. The map server is structured into several key blocks, as illustrated in the Figure 4.1. As their component siblings do, it follows a plugin architecture, which ensures seamless integration with existing mapping frameworks or allows own mapping implementations for custom applications. The component scheme follows a similar approach to other basic robotic function components:

- *Input Adaptation*: This block processes incoming sensor measurements, adapting them to a standardized format that can be used by the map server. Examples of input data

[1] https://github.com/aerostack2/aerostack2/tree/main/as2_map_server

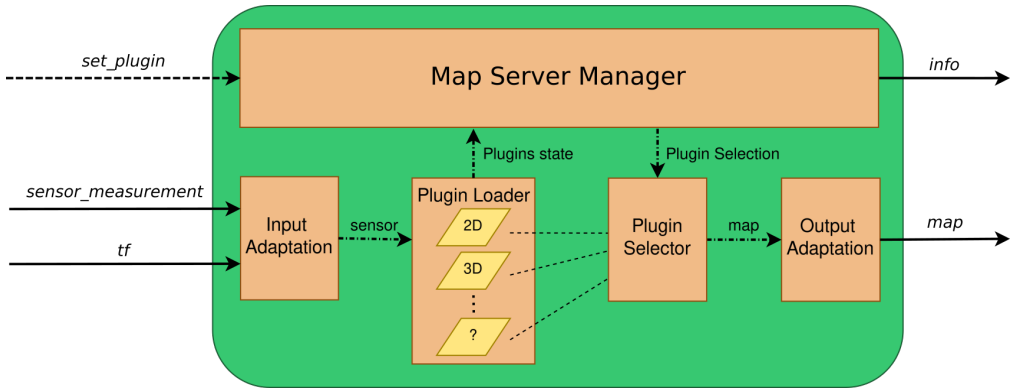


Figure 4.1: Map server component architecture diagram. Orange boxes represent the mapper key elements and how they interact internally. Plugins are represented with yellow trapezoids.

include LiDAR scans, depth images, or stereo camera data, which are transformed into usable sensor data.

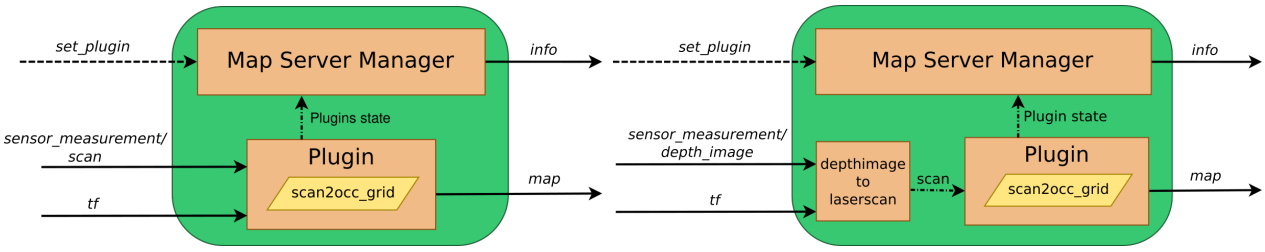
- *Plugin Loader*: The modularity of the map server is achieved through the Plugin Loader, which supports multiple mapping plugins. Plugins depend on the application needs and can vary from geometric representations (two or three-dimensional grids) to topological maps for simplifying complex volumetric outdoor spaces, or even custom representations for specialized mapping algorithms. This design ensures flexibility and adaptability for different applications.
- *Plugin Selector*: This block manages the selection of the appropriate mapping plugin based on the operational requirements and environmental context. It uses a plugin state manager to monitor the status and compatibility of each plugin during runtime.
- *Map Server Manager*: The central logic responsible for generating, storing, and updating the map. It interacts with the Plugin Selector to retrieve mapping outputs and maintain a consistent map representation.
- *Output Adaptation*: This block formats the generated maps into ROS 2-compatible messages and topics for integration with other Aerostack2 components. It also provides meta-information about the map server, including status, current plugin loaded, and sensor integration details.

The map server currently supports two primary kinds of plugins, geometric and topological maps. Figure 4.2 depicts the existing plugins for generating occupancy grid maps:

- `scan2occ_grid` Plugin: This plugin takes in laser scan data from the sensor measurement and processes it to generate a 2D occupancy grid map, which is then published. It is ideal for applications using LiDAR sensors to obtain precise distance measurements and create a grid-based representation of the environment. Extended details will be explained later in the section.
- `depth2occ_grid` Plugin: This plugin is designed to work with depth cameras (RGB-D cameras). It takes depth images as input from the sensor measurement topic and

converts the depth data into a laser scan format using the `depthimage_to_laserscan` standard package[2] from the ROS 2 perception. The resulting laser scan data is then processed by the `scan2occ_grid` plugin to produce the final map.

- `pointcloud2occ_grid` Plugin: This plugin works with 3D LiDAR sensors. It converts point clouds from the input sensor measurement topic to laser scans using the `pointcloud_to_laserscan` standard package[3] from the ROS 2 perception. Similar to the previous plugin, the laser scan output is forwarded to the `scan2occ_grid` plugin to generate the map. These two implementations ensure compatibility with both LiDAR-based and vision-based systems, making the map server adaptable to various sensor configurations, without re-implementing the mapping algorithm.



(a) scan2occ_grid plugin. (b) depth2occ_grid plugin.

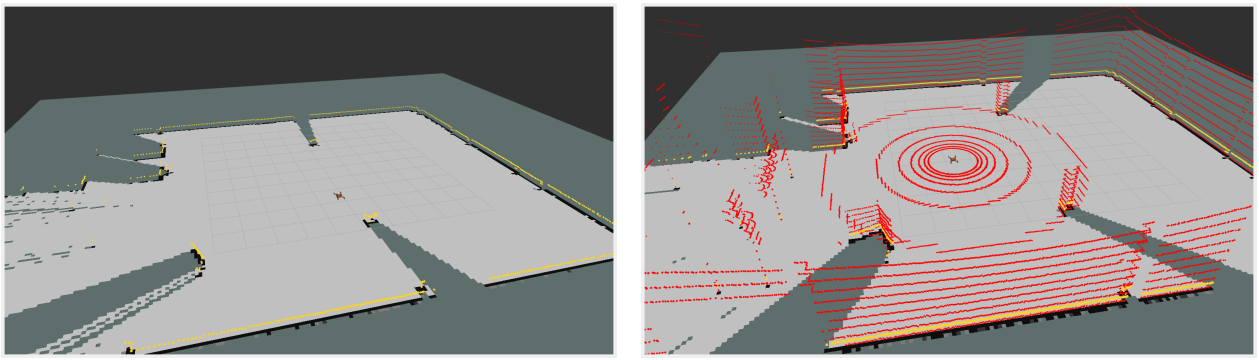
Figure 4.2: Map server plugins example. On the right, input sensor measurement adaptation from depth image to laser scan.

The design of these plugins emphasizes modularity, enabling the integration of additional sensor types or mapping algorithms in the future. Each plugin focuses on a specific combination of input-output data, ensuring that the map server can process sensor measurements efficiently and suit different hardware and operational requirements.

The `scan2occ_grid` plugin processes laser scan data to generate 2D occupancy grid maps, which are essential for navigation in environments where obstacle detection is critical. The plugin uses the Bresenham line algorithm to efficiently project sensor rays onto grid cells, marking free and occupied spaces based on measured distances. It employs a probabilistic approach to update occupancy values, balancing accuracy and computational efficiency. Each hit sums a probability for the cell to be occupied, while every miss subtracts another value and unknown cells remain unmodified. Every sensor update changes the map representation at the plugin based on the hits and misses, after clipping occupancy probability between 0 and 100.

To ensure compatibility across different UAVs, the plugin integrates seamlessly with the ROS 2 ecosystem. It can be customized via ROS 2 parameters, such as grid resolution, sensor range limits, and update hit/miss confidence, to suit specific applications. For instance, a higher resolution map is advantageous for indoor navigation, while a lower resolution map is preferable for outdoor exploration to reduce memory overhead. Figure 4.3 shows the plugin working for two different sensor inputs.

[2] https://github.com/ros-perception/depthimage_to_laserscan/tree/ros2
[3] https://github.com/ros-perception/pointcloud_to_laserscan



(a) Map generated using a laser scan sensor. (b) Map generated using a 3D LiDAR sensor.

Figure 4.3: Map server output examples.

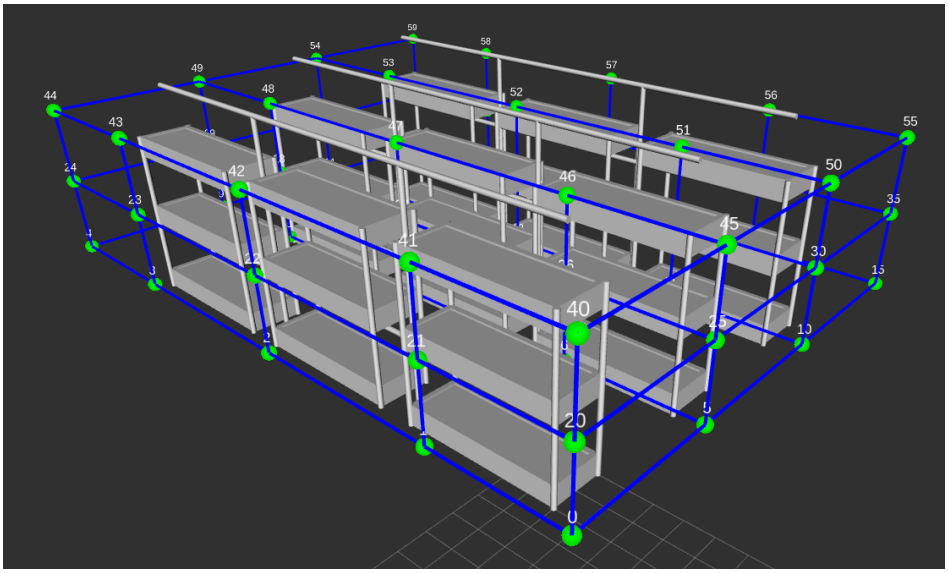


Figure 4.4: Topological map for an indoor farm environment.

A topological map plugin is also available with the `indoor_farm2graph` plugin. The plugin creates a graph composed of nodes and edges. Nodes represent landmarks or points of interest in the environment, while edges serve as routes of how these nodes are connected. The plugin is designed for indoor farm monitoring. Thus, nodes are located at the plant beds, where the vegetables can be inspected. The graph is built from several farm parameters, such as row count, plant bed dimensions, or plant bed spacing. Figure 4.4 illustrates the graph built by the plugin in a simulated indoor farm scenario.

The map server integrates smoothly into Aerostack2 data flow communication. It subscribes to sensor topics, processes the data, and publishes maps that can be used by other navigation-related modules, such as path planning and obstacle avoidance. It interacts with the state estimation component for real-time updates about the UAV's position relative to the map using the TF tree.

The map server is fully compatible with Aerostack2's behavior system. The following section

deepens into a new behavior called `Path Planning` that leverages the maps generated by this component for planning and execution.

4.1.3 Path Planning Behavior

The path planner behavior is a key component in Aerostack2, responsible for generating efficient and collision-free paths for UAVs based on the map of the environment and the UAV's localization data. This behavior integrates consistently into the Aerostack2 behavior-based architecture, making it adaptable and easy to use within complex mission plans.

The software component detailed in this section was created specifically for the aims of this thesis. The newly developed functionalities were primarily implemented in C++ and can be accessed in the Aerostack2 repository[4], distributed under the BSD 3-clause license.

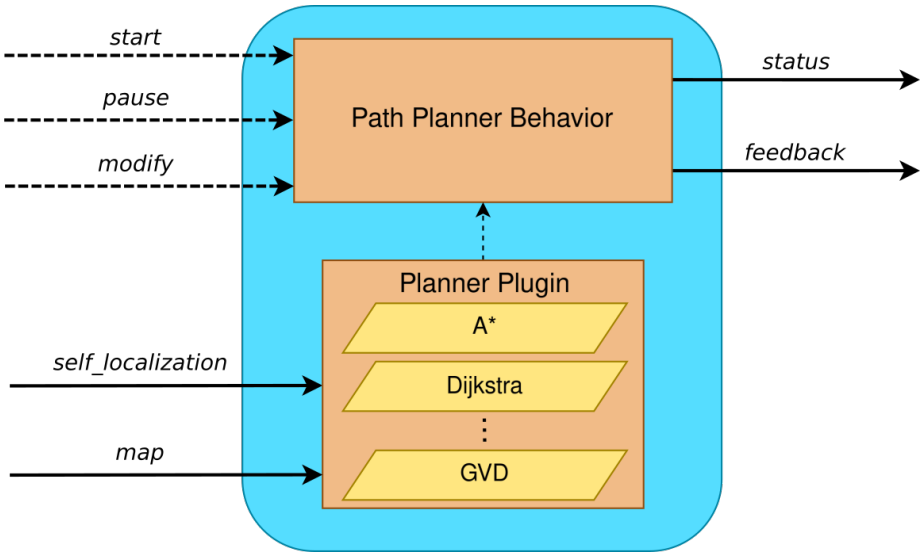
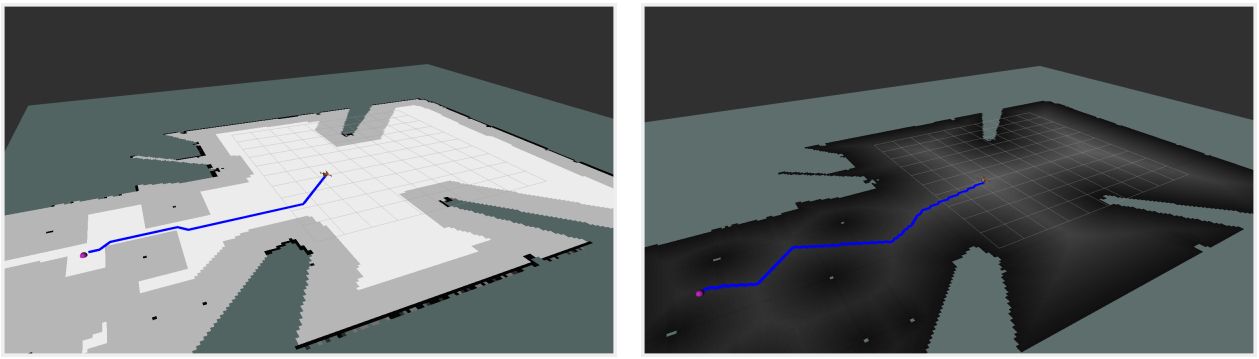


Figure 4.5: Path planning behavior architecture diagram. Orange boxes are internal behavior elements, while the implementation (plugins) are represented with yellow trapezoids.

As every behavior in the framework, the design of the path planner behavior consists of two main parts, as detailed in Figure 4.5. The first part is the behavior monitor, which provides a standardized control mechanism with services to start, pause, resume, and modify the path planning process. It also generates real-time updates on the execution status and provides detailed feedback on the progress or potential failures during path generation. This design ensures compatibility with Aerostack2's modular framework and facilitates its integration with other system components. The second part is the behavior executor, which implements the core functionality of the behavior and supports a variety of path planning algorithms in the form of plugins. These plugins include graph-based methods such as A* or Dijkstra, for shortest and cost-optimal paths, and sampling-based methods like GVD (Generalized Voronoi Diagram), which computes paths by maximizing clearance from obstacles, making it highly suitable for narrow environments. The modular plugin architecture enables flexibility and

[4] https://github.com/aerostack2/aerostack2/tree/main/as2_behaviors/as2_behaviors_path_planning

allows developers to extend the framework with additional planning algorithms. Figure 4.6 shows the path generated by A* and GVD plugins.



(a) A* plugin path planner example output. (b) GVD plugin path planner example output.

Figure 4.6: Path planning behavior examples for grid map representation.

Path planner plugins are strongly linked to the map representation chosen by the map server. However, plugins can be used for more than one representation. For instance, A* plugin can be used for grid maps and graph maps, as shown in Figure 4.7.

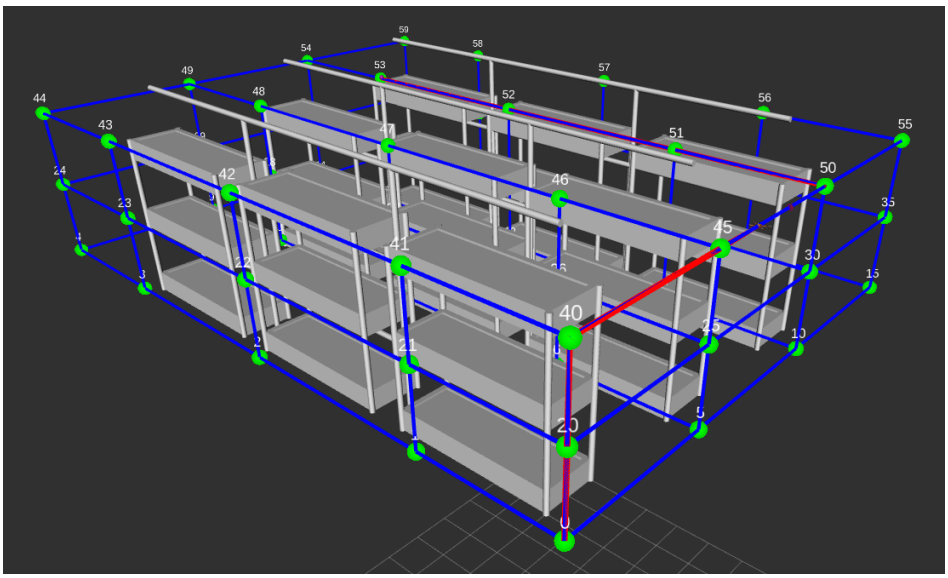


Figure 4.7: A* path planning plugin working in a topological map.

The path planner behavior relies on two primary inputs: real-time localization data, which provides the UAV's current position and orientation, and the map of the environment, which serves as the basis for calculating a feasible path. The output of this component is not limited to status updates and feedback; it also generates a path that is commanded to the `FollowPath` behavior. This subsequent behavior converts the planned path into a trajectory, via the `TrajectoryGenerator` behavior (see Figure 4.8), and produces motion references for the controller, ensuring the UAV can execute the path efficiently and accurately. This seamless

communication between components highlights the strength of Aerostack2's modularity and distributed design.

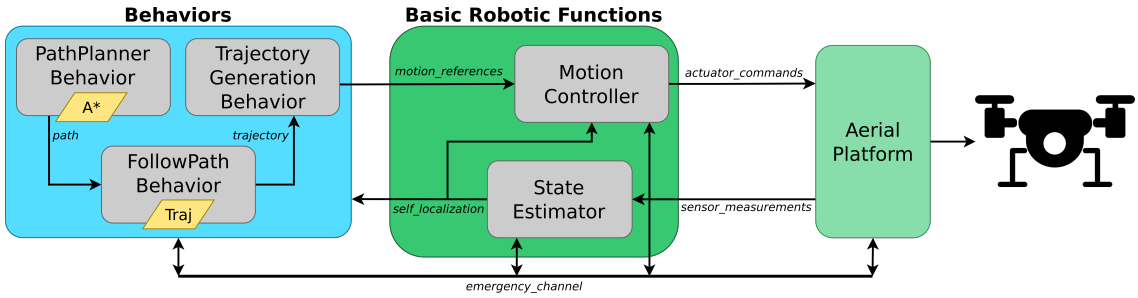


Figure 4.8: Path planning behavior cascade activation. Grey blocks are ROS 2 nodes and solid lines are topics interfaces. Yellow trapezoids represent the active plugin.

The path planner behavior is critical for achieving autonomy in tasks such as waypoint navigation, obstacle avoidance, and autonomous exploration. Its support for multiple planning algorithms makes it adaptable to both structured indoor settings and unstructured outdoor environments. By combining flexibility, modularity, and a standardized interface, the path planner behavior enhances Aerostack2's capabilities for autonomous navigation.

4.2 Task Planning Execution Control System

This section presents a new deliberation architecture, focusing on a new method to control the execution of a plan. This layer, as many others in the literature, is composed of three tiers: *planning*, *acting*, and *monitoring*. Figure 4.9 shows the three-tier architecture interacting with above and below layers. The human operator defines the mission through a ground control station (GCS) and sends it to the planning sub-system. The plan is then converted into some language that the acting sub-system can understand to perform the mission. Plan execution data is gathered by the monitoring sub-system, and useful information about the mission execution is sent to the GCS. The operator can decide to modify, pause, or stop the mission at any time based on the mission control information received.

By defining only missions, the plan execution control layer frees the developer from needing to specify intricate details on task execution management. Notice the difference in notation between *mission* and *plan*. *Mission* refers to user-defined assignment to the multi-robot system, while *plan* indicates the sequence of actions that a UAV has to perform to complete a mission. The mission is shared by every agent, but plans can differ. The translation from mission to plan is done by the planning sub-system.

Aerostack2 relies on the behavior-based paradigm [Arkin, 1998; Brooks, 1986], as first introduced in Section 3.2. In our work, behaviors correspond to a generalization of the conventional concept of reactive behaviors (e.g., for motion control) in order to consider other kinds of robot skills or tasks (e.g., visual recognition, simultaneous localization and mapping, etc.) [Michaud and Nicolescu, 2016]. According to the behavior-based paradigm, the global control is divided into a set of behavior controllers, and each one is in charge of a specific

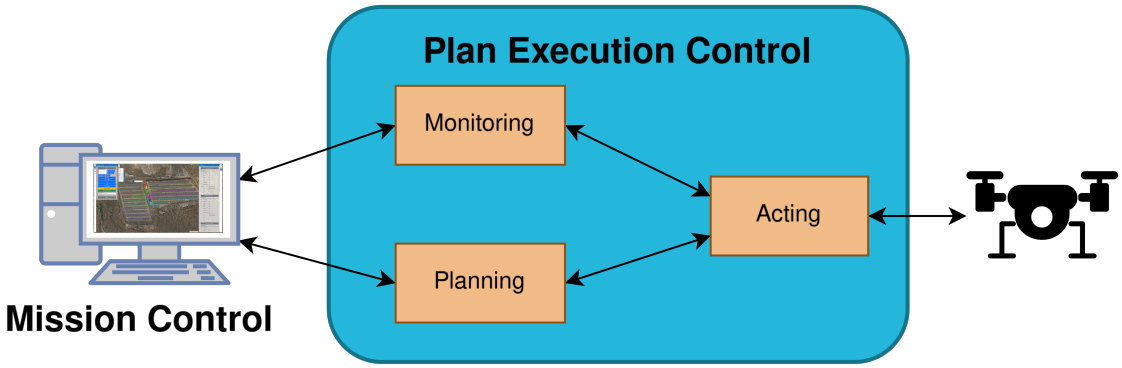


Figure 4.9: Three-tier architecture for Plan Execution Control. Orange boxes are internal behavior elements from each system.

control aspect separately from the other behavior controllers [Molina et al., 2020]. A behavior controller typically functions in a loop, processing input data from sensors or other controllers and producing output that can be utilized by actuators or additional behavior controllers. This forms the behavioral control tier in the deliberative architecture.

Based on the analysis proposed in Chapter 2, we have selected Reactive Action Packages (RAPs) as a starting point for developing a minimal coordination mechanism suitable for the robotics domain. This decision is driven by several key needs: a high-level abstraction centered on robotic skills, enabling modular and reusable plan composition; flexibility for run-time plan modifications, allowing adaptation to unforeseen circumstances; a distributed architecture to support heterogeneous multi-robot systems operating in dynamic environments; a safe and reliable execution framework suitable for real-world industrial applications; and dynamic resource configuration, ensuring seamless adaptation to the capabilities and constraints of UAVs in operation.

The section focuses on the acting sub-system, while planning and monitoring sub-systems are out of the scope of this work. Here, a new dynamic task planning execution control named *Mission Executor* is proposed. Before this system was implemented, there existed two alternatives for executing plans in Aerostack2: Python API and Behavior Trees. Both were detailed explained in Section 3.2.5. A third method is proposed since neither of the two solutions fulfilled the requirements introduced at the end of the previous section. The existing Behavior Trees, based on BehaviorTree.CPP, executor lack the flexibility to modify the mission online without deep modifications on the framework. Whereas, the Python API allows creating intricate missions but has to be programmed prior to operation.

Compared with the existing alternatives, the new method is designed to satisfy the following quality attribute requirements derived from the previously defined needs:

- **Behavior-based.** The atomic elements of mission planning are Aerostack2 behaviors, each representing a robotic skill, adding expressiveness to mission execution.
- **Modularity.** A reusable structure for behaviors that enables flexible composition for complex mission definitions.

- **Adaptability.** Flexibility for run-time plan modifications and dynamic resource configuration to accommodate UAVs' capabilities and environmental changes.
- **Distributed.** Designed to support heterogeneous multi-robot systems, ensuring decentralized operation.
- **Reliability.** Ensures safe execution, making it suitable for real-world industrial applications.

The system proposed here works similarly to the existing methods in Aerostack2, by specifying the activation and deactivation of robot behaviors, which allows for the combination of individual tasks to create complex, autonomous missions. Rather than using Python or XML languages for defining the mission plan, as the Python API and BTs do, the JSON format is chosen for the Mission Executor. Code Snippet 4.1 illustrates a takeoff-go-land plan defined using the JSON format. This choice responds to the distributed requirements previously explained. JSON format facilitates mission sharing, as the plan can be communicated in a standardized text format and autonomously interpreted by each UAV.

```
{
    "target": "drone_0",
    "plan": [
        {
            "behavior": "takeoff",
            "args": {
                "height": 3.0,
                "speed": 1.0
            }
        },
        {
            "behavior": "go_to",
            "method": "go_to_path_facing",
            "args": {
                "x": 0.0,
                "y": 5.0,
                "z": 3.0,
                "speed": 1.0
            }
        },
        {
            "behavior": "land",
            "args": {
                "speed": 0.5
            }
        }
    ]
}
```

Code Snippet 4.1: Example of a mission plan in JSON to be used with Aerostack2 mission executor.

4.2.1 System Overview

The Mission Executor is composed of three components: an adapter, an interpreter, and a drone interface. The core component is the interpreter, which processes the plan and translates the JSON descriptions into Python API commands. Indeed, the executor is coupled with the Python API to trigger behaviors through the drone interface. These commands are executed in a different thread using the Drone Interface class from the Python API. On top of the interpreter, an adapter offers two communication interfaces to interact with the executor. Those two channels are one input topic, *mission_update*, and one output topic, *mission_status*. Figure 4.10 displays a whole picture of the Mission Executor.

The interpreter is not coupled with one adapter. The adapter depends on the communication domain used. Currently, there is only one adapter implemented to perform the communication through ROS 2, but the architecture allows implementing custom proprietary protocols, such

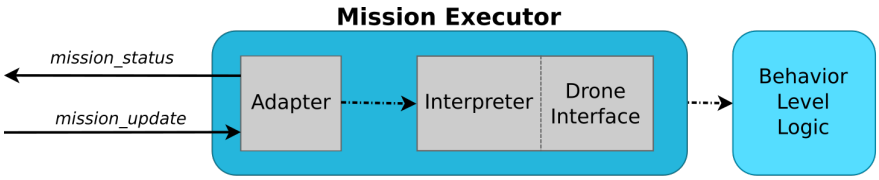


Figure 4.10: Components and communication interfaces part of the Mission Executor. Grey blocks are ROS 2 nodes and solid lines are topics interfaces.

as the DJI MOP channel[5] or other specific protocols.

It is worth mentioning that the interpreter is ROS-agnostic, since the Drone Interface manages all interactions with the UAV and its behaviors. In the current scenario, where the existing adapter implementation also uses ROS for the communication, there are two ROS nodes working in the same process, adapter and drone interface.

The input topic receives commands, or mission updates, and triggers certain methods from the interpreter. The output topic sends periodic information about the status of the interpreter. On one hand, *MissionUpdate* message is formed by a drone target ID, mission ID, item ID, action, and mission defined as a JSON string. A further description of message fields is explained in Table 4.1. This JSON mission corresponds to a sequence of behaviors with their activation methods and arguments, as shown in Code Snippet 4.1. On the other hand, *MissionStatus* messages are formed by a unique string field. It contains a JSON formatted message with the execution state, pending, done, and current items, and the feedback on the current behavior being executed. Execution state field is an integer enumeration where: 0 is *idle*, 1 is *running*, and 2 is *paused*. Code Snippet 4.2 shows a status message during the execution of the previously presented plan.

Message field	Field type	Description
drone_id	int32	UAV identifier.
mission_id	int32	Mission identifier.
item_id	int32	Item identifier.
action	uint8	Action to perform by the interpreter.
mission	string	JSON formatted mission.

Table 4.1: *MissionUpdate* message description.

4.2.2 Mission Interpreter

Translation from *MissionUpdate* messages to behavior activations and deactivations occurs in the Mission Interpreter. This core component manages the mission execution flow, keeping a history of what is done and what is left to do. Additionally, it interfaces with Aerostack2 lower-level layers from behaviors to the platform, exposing them to the adapter.

The interpreter supports a pool of available actions. Currently supported actions are mainly for two tasks: changing the plan (load, insert, modify, remove, or reset) and modifying its

[5] https://developer.dji.com/doc/payload-sdk-api-reference/en/practice/mop-channel.html

```
{
  "state": 1,
  "pending_items": 1,
  "done_items": 1,
  "current_item": {
    "behavior": "go_to",
    "method": "go_to_path_facing",
    "args": {
      "x": 0.0,
      "y": 5.0,
      "z": 3.0,
      "speed": 1.0
    }
  },
  "feedback_current": {
    "actual_speed": 0.683,
    "actual_distance_to_goal": 4.563
  }
}
```

Code Snippet 4.2: Mission status message during a go_to execution.

execution (start, pause, resume, stop, jump_to_item, next_item). Table 4.2 gathers the pool of actions with a brief description for each command. This set of actions allows for supporting conditional logic to guide mission execution, enabling UAVs to adapt to different scenarios during the mission.

ID	Action	Description
0	EXECUTE	Execute the *mission* into the interpreter. Same result to load and start.
1	LOAD	Load the *mission* linked to *mission_id* into the interpreter.
2	START	Start the execution of *mission_id*.
3	PAUSE	Pause the execution of *mission_id* by pausing the running behavior.
4	RESUME	Resume the execution of *mission_id* by resuming the running behavior.
5	STOP	Stop the execution of *mission_id*.
6	NEXT_ITEM	Stop the execution of a behavior and start the next behavior in queue.
7	JUMP_TO	Stop the execution of a behavior and start behavior *item_id* from mission.
8	INSERT	Insert an item(s) in the mission after *item_id* position.
9	MODIFY	Modify *item_id* element from *mission_id*.
10	REMOVE	Remove *item_id* from *mission_id*.
11	RESET	Reset the interpreter. Stop mission in execution and unload *mission_id*.

Table 4.2: Available actions in *MissionUpdate* message.

Currently, the interpreter supports handling several plans, via `load` action, but only one can be executed at a time. This grants fast dynamic changes in the plan under execution. This enables reliable safety mission activation, or different UAV roles based on plan switching.

The mission plan can be modified in two different ways, changing the list of behavior items and changing the pointer to the current behavior. New item(s) can be inserted by using `insert` action, and can be deleted with `remove` action. Executing behavior can be changed by `jump_to_item`, which allows going front and back in the plan. `next_item` simply advances one item in the plan, which is helpful with behaviors that never end. Moreover, there are other actions to modify a behavior execution, such as `start`, `pause`, `resume`, and `stop`.

Additionally, one key benefit of behaviors over standard ROS 2 actions is the ability to modify the goal in progress. The Mission Executor integrates this capability using the `modify` action, which changes the behavior goal during runtime. This behavior modification allows for more flexible plan adaptations to unforeseen circumstances. For instance, a two-UAV possible collision can be avoided by modifying one `go_to` goal, changing its speed or height, instead of pausing the behavior or changing the plan.

4.2.3 Behavior Modules

Loading a plan not only means to save it in memory, but also to create an instance of a Drone Interface to later execute the plan. If a Drone Interface is already created, the interpreter reconfigures its behavior modules to only load the ones needed to successfully run the plan. This dynamic reconfiguration ensures minimal computational resource usage based on the mission definition.

A behavior module acts as a wrapper around the BehaviorClient implementation from the Python API. The wrapper includes basic behavior execution control (`start`, `resume`, `resume`, and `stop`), but also custom specialized methods accessible from the plan description (e.g. `go_to_path_facing` and `go_to_with_yaw` from the `go_to` behavior). Moreover, it incorporates common functionalities for module finding, loading, and dependency handling. For better understanding, a class diagram of Behavior Modules is displayed in Figure 4.11.

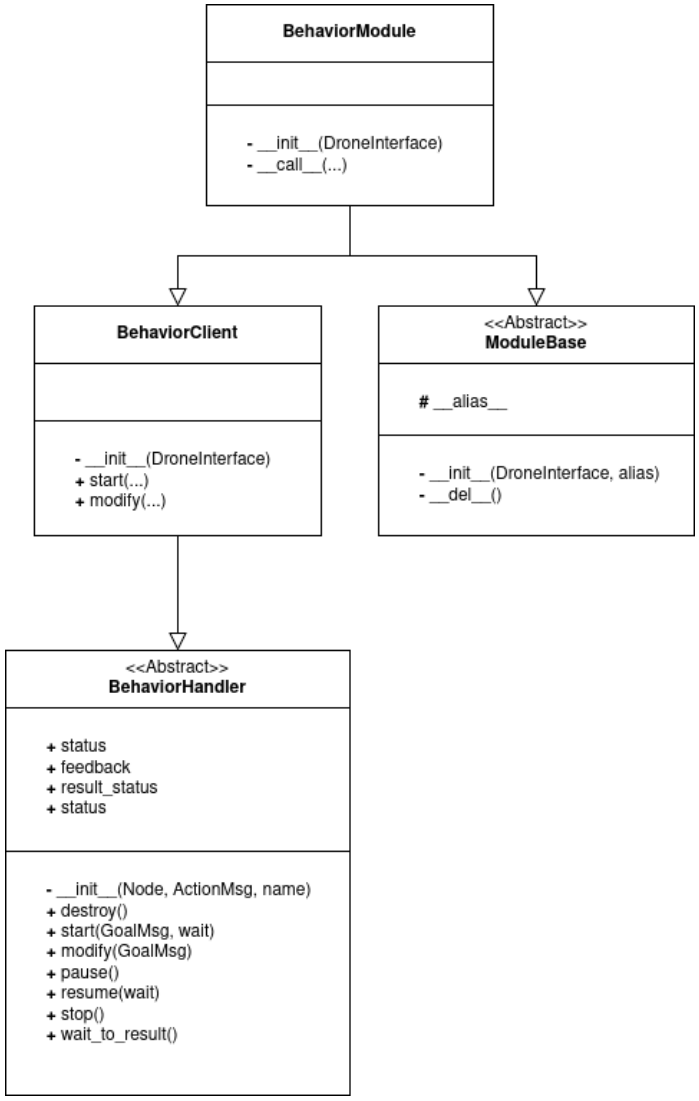


Figure 4.11: UML class diagram for the Behavior Module.

Furthermore, modules allow composition and parallel execution of behaviors. First, modules can include one or more behaviors (or other modules). This modular approach grants the reuse of currently implemented skills, increasing reliability and repeatability. For instance, RTL module includes `go_to` and `land` modules as dependencies, while its `start` method calls a `go_to` to the origin of the UAV's map frame at a desired height, followed by a land call. Second, parallel execution is reached by including a `wait` argument to every `__call__` module method. This argument indicates to the interpreter whether to wait until the behavior ends execution or to activate the next behavior without halting.

4.2.4 Multi-UAV approach

The presented executive layer supports centralized and decentralized multi-robot system approaches. To understand the possible schemes, the different existing monitoring systems have to be explained first. The system proposes a three-level monitoring: mission, plan, and behavior. Supervision at the mission level applies to reaching the global mission goals, while plan-level goals refer to each UAV's individual goals. Behavior monitoring ensures that the robotic skill is working under certain minimal criteria. In a multi-UAV area coverage application, the mission goal might be covering the total area, the plan goal might be scanning a sub-area, and the behavior goal might be following a path with a certain error threshold and a certain velocity.

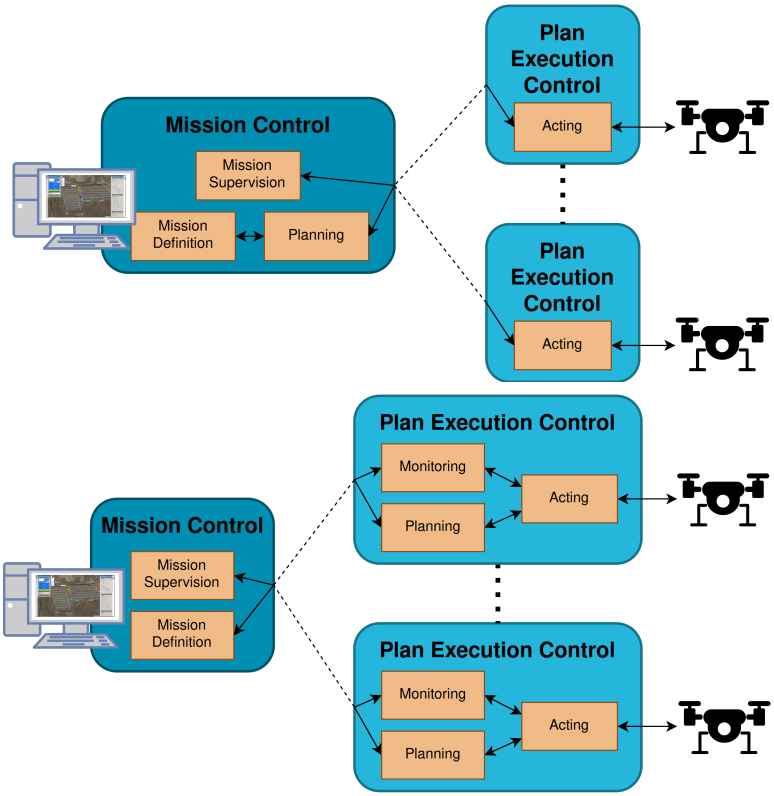


Figure 4.12: Centralized (top) and decentralized (bottom) multi-robot schemes. Orange boxes are internal behavior elements from each system.

Mission monitoring and behavior are likely to happen most of the time, but plan monitoring might be omitted in some cases. In a single UAV scenario, mission and plan monitoring can be joined, since mission goals and plan goals can be identical. In a centralized approach, mission and plan monitoring can be kept for better re-planning management and adaptability, as suggested by [Luna et al., 2024], or even tied in a fully centralized scheme. On the contrary, decentralized schemes need three-level monitoring to maintain the system working.

In any case, plan execution is distributed. The same applies to behavior monitoring, which is inherent to the Aerostack2 behavior definition. Figure 4.12 illustrates a comparison between centralized and decentralized multi-robot schemes. Centralization affects only the mission-to-plan translation and plan-level monitoring, while the mission executor adapts to both arrangements. The suitability of the scheme used depends strongly on the application. During the experimental validation section, different schemes and applications are described.

4.3 Exploration using Minimal Sensing on Cooperative Nano-UAVs

Exploration and mapping of unknown unstructured environments is a crucial area of research in the field of mobile robotics. As the technology of nano-sized UAVs advances [Bonato et al., 2023; Jaisinghani et al., 2023; Liu et al., 2021], their utility in both indoor and outdoor autonomous tasks has gained considerable prominence. A nano-sized UAV typically weighs ∼50 g or less and is the size of a palm. These UAVs are particularly well-suited for potentially hazardous indoor exploration missions due to their small size, which ensures safety around humans, and their agility and speed, which are essential for effective exploration. Moreover, their low cost contributes to their feasibility for widespread use as they are easily replaceable.

Building on the capabilities of individual nano-UAVs, the use of UAV swarms enhances these operations significantly. Swarms employ the collective speed and maneuverability of multiple UAVs, which are confirmed to accomplish tasks more efficiently and rapidly than a single UAV by theoretical and empirical research [Guzzoni et al., 1997]. Moreover, the strategic deployment of multiple UAVs introduces a level of redundancy, thereby enhancing the overall system's fault tolerance compared to a single UAV. Additionally, cooperating robots can significantly reduce sensor uncertainty thanks to the information overlap between different robots.

Several authors in the literature have shown promising progress in robotic exploration [Burgard et al., 2005; Oleynikova et al., 2018; Zhu et al., 2015] as cited in Section 2.4. Available algorithms strongly rely on complex sensors, such as depth cameras or 3D lidars, for obstacle avoidance and mapping. However, current nano-UAVs cannot adopt these algorithms due to the restricted payload resources for on-board sensors and computational power. The available payload in nano-UAVs is on the order of grams. Therefore, it is only possible to mount light and low-power sensors that can provide only sparse and noisy measurements. Consequently, the minimal sensing capacity introduces a challenging exploration problem.

In this section, we present a novel coordination algorithm specifically designed for nano-UAVs to achieve efficient, full-environment exploration using minimal sensing. Contrary to the

latest contributions in the field, which prioritize exploration time and speed, our method takes a fundamentally different direction: we focus on growing smaller, not on complexity. Our aim is to push the boundaries of existing techniques by adapting them to smaller, resource-constrained platforms and rigorously evaluating the trade-offs between performance and minimal sensor input. Our method combines classic bug-algorithms with frontier-based exploration techniques. Frontier-based is effective in 2D environments [Thrun et al., 2004] and viable for nano-UAVs' onboard sensors. The central idea behind frontier-based exploration is to gain the most new information about the world, moving to the boundary between open space and uncharted territory [Yamauchi, 1997]. Frontiers are regions on the boundary between open space and unexplored space.

Therefore, the primary objective is to achieve complete coverage of the area while efficiently coordinating multiple UAVs with minimal onboard sensing and processing capabilities. Given the physical constraints of nano-UAVs, such as their small size and extremely limited payload capacity, equipping them with advanced sensors commonly used in robotic exploration (e.g., LiDARs or depth cameras) is unfeasible. As a result, the available sensory input is highly sparse, making the exploration and mapping process significantly more challenging.

To tackle this problem, we propose an exploration pipeline specifically designed to operate under these minimal sensing conditions. The strategy ensures that nano-UAVs can navigate and fully explore the environment despite their constrained perception. Each UAV in our system is equipped with only four single-beam range sensors, oriented in the front, back, right, and left directions, with a maximum sensing range of 4 meters. This minimal sensor configuration imposes a critical limitation: the UAVs can only perceive obstacles in a narrow field of view and lack direct height or depth information. Consequently, the exploration task is treated as a two-dimensional problem, meaning that all UAVs maintain a constant altitude during the entire mapping and exploration process.

In addition to the sensing constraints, we make two key assumptions to enable efficient multi-agent coordination. First, all UAVs can localize themselves within a shared global reference frame, ensuring a common understanding of their positions relative to the environment. Second, we assume that agents communicate seamlessly through an unlimited-range data-sharing channel, allowing real-time exchange of information about the explored areas, detected obstacles, and swarm coordination strategies.

4.3.1 System Overview

The proposed system is structured into three primary components, each responsible for a distinct phase of the exploration mission: the mapping sub-system, the exploration sub-system, and the navigator sub-system. These components work together in a sequential and complementary manner to achieve efficient and autonomous navigation through unknown environments. The mapping sub-system focuses on constructing and maintaining an up-to-date representation of the environment, which serves as the foundation for subsequent planning tasks. The exploration sub-system utilizes this map to identify unexplored regions, prioritize exploration targets, and make decisions regarding the next areas to investigate. Finally, the navigator sub-system takes these selected targets and computes safe and efficient paths to

reach them, ensuring smooth and reliable navigation. The coordination strategy between agents in the swarm follows a centralized approach for mapping and exploring, while navigating depends only on each individual UAV. Figure 4.13 illustrates the overall architecture and the interaction between these core sub-systems, providing a visual representation of the mission execution flow.

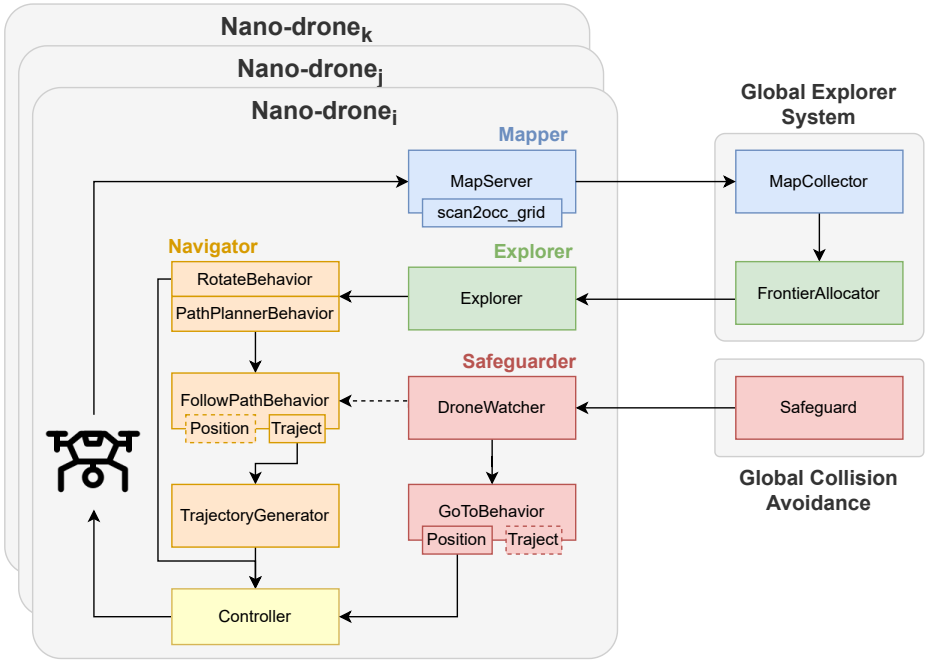


Figure 4.13: Overview of the pipeline. Box color indicates the sub-system each component belong to: Mapping (blue), exploration (green), navigation (orange), and intra-swarm collision avoidance (red).

During the mapping phase, each agent in the swarm utilizes distance measurements obtained from the multi-ranger sensor. These measurements are shared among all agents to collaboratively build a common representation of the environment. Since the sensor provides only two-dimensional distance information, the environment is encoded as an occupancy grid map, which effectively captures the presence of obstacles and free space in the horizontal plane. To create a comprehensive and unified map, each agent generates a local occupancy grid using its onboard map server. These local maps are then transmitted to a central map collector component, which merges them into a global map. The global map is stored as a 2.5-dimensional grid map, following the representation described by [Fankhauser and Hutter, 2016]. This map structure employs multiple layers of occupancy grids, with each layer corresponding to the mapping contributions from individual agents.

The exploration phase is composed of three steps: exploration strategy, frontier generation, and frontier allocation. The exploration strategy implemented in this system is based on a hybrid frontier range-bug algorithm. Initially, the system analyzes the global map to identify

potential frontiers—regions that lie at the boundary between explored and unexplored areas. Once these candidate frontiers are generated, the system proceeds to allocate them among the available agents. The allocation process considers various factors based on the swarm's state, which ensures balanced workload distribution among agents and maximizes exploration efficiency.

The final phase, navigation, is responsible for guiding each UAV from its current position to the selected goal point. The navigation process relies on motion references sent to the controller, which executes the necessary commands to move the UAV. The primary component of this phase is the path planner behavior provided by Aerostack2, previously introduced in Section 4.1. This behavior computes safe and efficient paths based on the global map and the assigned goal locations. Additionally, the rotate behavior is utilized to scan the local surroundings. Aerostack2's flexible behavior-based architecture allows operators to initiate, pause, or stop these navigation behaviors on demand. This dynamic behavior control applies not only to the path planner but also to other essential actions such as go to waypoint, hover, or take-off.

Complementing these core components, an intra-swarm collision avoidance sub-system operates continuously throughout the mission. This safeguard mechanism monitors the real-time positions of all UAVs within the swarm to detect potential collision scenarios. Upon identifying an imminent collision, the Drone Watcher module intervenes by temporarily halting the navigator of one of the involved UAVs. It then triggers a predefined safety maneuver to increase the separation distance and prevent the collision. Once the safe distance is reestablished, the Drone Watcher resumes normal navigation operations, ensuring that the exploration mission continues safely and without significant interruptions.

4.3.2 Explore Bug

The exploring strategy operates as a FSM, depicted in Figure 4.14, where each UAV begins at the "Init" state. The approach is similar to *CautiousBug* [Magid and Rivlin, 2004] or *Rotate-and-measure* [Lamberti et al., 2023], where a conservative spiral search is performed before going to a specific location. This ensures a safer exploration strategy by thoroughly scanning the environment before navigation. During the "Sense" phase, the UAV activates the rotate behavior and continuously collects data from the surroundings at a rate of 20 Hz.

Frontiers, which represent potential next points for navigation, are identified over the generated map, leading to the "Navigate to Frontier" state (path planner behavior activation). It should be noted that the term navigation implies the need for a map and can only be performed on the already explored surroundings. Frontier generation and allocation to UAVs are described in subsequent subsections in detail. Upon reaching the goal ($d_{front} < d_{th}$), the robot returns to the "Sense" state.

Due to the symmetry of the range-sensor, spinning consists of a $\frac{\pi}{2}$ radians turn only $\psi_{goal} = \psi_0 + \frac{\pi}{2}$. Rotating just occurs during the "Sense" state since navigation can be performed without yaw control. Our cautious algorithm ensures to fly under an already known environment only, so the UAV will not fly over an obstacle.

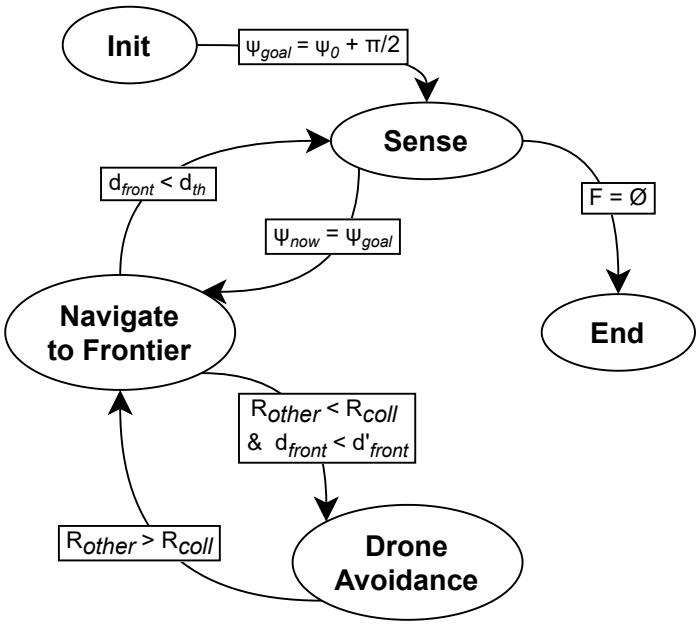


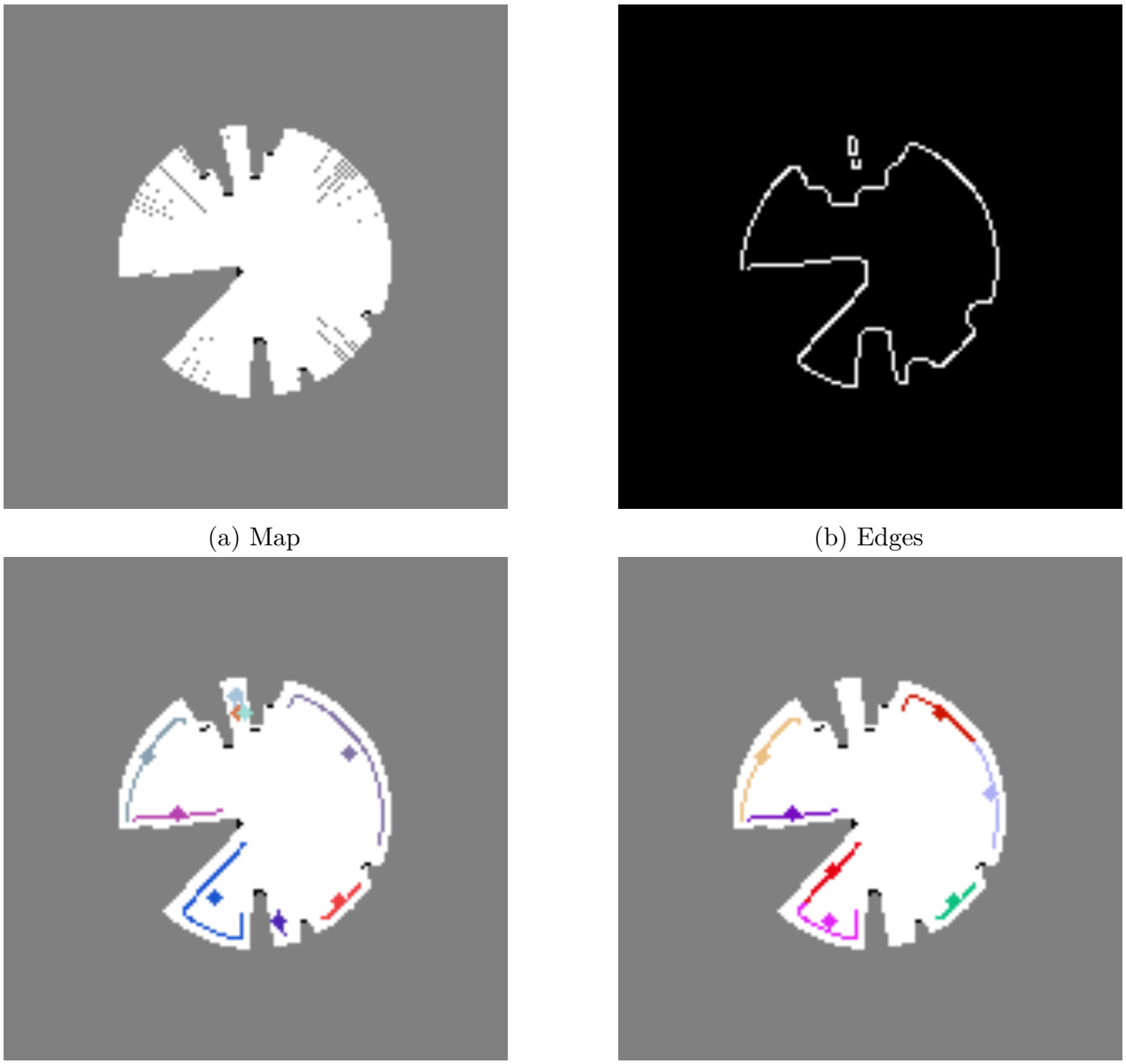
Figure 4.14: FSM of the exploration algorithm. Ellipses show states, while boxes illustrate conditions to change between states.

During the navigation state, if two or more UAVs are within a safety margin distance $R_{other} < R_{coll}$, the one that's closest to its goal ($d_{front} < d'_{front}$) transitions into the "Drone Avoidance" state, while the other UAV continues to fly without any avoidance maneuver. This maneuver consists of pausing the path planner behavior, followed by a change in altitude, via the activation of go-to behavior, and hover. This ensures a quicker return to the navigation state once the safety distance is restored, through returning to the original position and resuming the path planner behavior. Finally, when no frontiers are left, the exploration is considered finished and the algorithm enters into "End" state, which is accessible from any other state within the FSM.

4.3.3 Frontier Generation

Given the sensor's nature, occupancy grid maps are selected to represent the environment, thus the plugin selected for the map server is `scan2occ_grid`. These maps maintain the probability of each grid cell being empty or occupied. Occupancy maps have been proven to be an efficient and powerful description of the surroundings, incrementally updating the map based on the sensor readings [Moravec, 1988].

Local occupancy grids are received from each UAV at the map collector and stored in a cell grid central map M. Cells can be unknown or have a certainty to be occupied/free $M = M_{free} \cup M_{obs} \cup M_{unk}$. This occupation probability is updated based on the hit or miss of the sensor readings, as explained in `scan2occ_grid` description. The central map is used to



(a) Map

(b) Edges

(c) Frontiers clustered

(d) Frontiers filtered

Figure 4.15: Frontier generation pipeline: (a) map, (b) edges extraction, (c) frontiers clustered and (d) frontiers filtered. M_{free} are represented in white, M_{obs} in black and M_{unk} in gray. Each color pictures pixels and centroid from every frontier.

obtain a set of frontiers F, where $F \subseteq M$. A frontier f_i is formed by a group of consecutive cells in the grid, and it is determined by a unique frontier descriptor. A descriptor is formed by the centroid μ_i, orientation ψ_i, and area A_i (or number of cells) of the frontier.

$$\forall f_i \in F,\ f_i : \{\mu_i, \psi_i, A_i\} \tag{4.1}$$

Frontiers F are obtained from the relative complement of $\bar{M}_{obs}$ in the edges detected over

$\bar{M}_{free}$ (Eq. 4.2). In turn, edges are calculated by applying a Canny filter over $\bar{M}_{free}$, previously Gaussian filtered to reduce noise in the map.

$$F = Canny(\bar{M}_{free}) \cap \bar{M}_{obs} \tag{4.2}$$

After the extraction, frontiers are clustered based on pixel connectivity and filtered depending on their area. If a frontier is smaller than a lower threshold, it is discarded; whether it is bigger than an upper threshold, it is split. Both thresholds are chosen based on the laser scan sensor range and the map resolution. The Algorithm 1 describes frontier generation.

Algorithm 1 Frontier generation

Require: M: current grid map
  $\bar{M}_{free} = M_{obs} \cup M_{unk}$
  $\bar{M}_{obs} = M_{free} \cup M_{unk}$
  $E = \text{Canny}(\bar{M}_{free})$
  $F = \bar{M}_{obs} \cap E$
  for all $f_i \in F$ **do**
    if $A_i < lower\_threshold$ **then**
      remove f_i from F
    else if $A_i > upper\_threshold$ **then**
      split f_i ▷ By pixel continuity
    end if
  end for

Figure 4.15 shows the frontier generation pipeline, from the global map M (Figure 4.15a) to the frontiers extracted F (Figure 4.15d). Edges detected over the map $canny(M_{free})$ are presented in Figure 4.15b. Finally, Figure 4.15c exhibits the frontiers clustered prior to the filtering.

4.3.4 Frontier Allocator

Goal choosing relies on the frontier candidates and the UAV swarm. The optimization problem is defined in Equation 4.3. The cost function depends on two terms. The first minimizes the path to the next frontier. We believe that the exploration rate is maximized during the "Sense" phase and not during the "Navigation" state since navigation occurs over an already mapped area. The second term minimizes the overlap between agents, maximizing the distance between them.

$$G^* = \underset{G}{\text{argmax}} \quad \frac{1}{X}\sum_{i}^{X}\sum_{j}^{F} d_{ij} - 2\sum_{i}^{F} c_i \tag{4.3}$$

where G^* is the optimal frontier, X is the set of frontiers allocated to UAVs, F is the set of frontiers, d_{ij} is the distance between the frontier allocated to UAV i and frontier j, and c_i is the distance between frontier i and the current UAV.

The proposed heuristic is independent of any specific motion constraints. The proposed exploration algorithm trades safety against speed. Since each nano-UAV comes to a stop upon reaching a goal frontier to perform a scanning action, motion speed is not a factor in the frontier allocation process. Additionally, heading adjustments during navigation are unnecessary, as the paths traverse known space and leverage the symmetrical sensor configuration. To assess the effectiveness of this approach across different conditions, we carried out a series of experiments, which are detailed in the following section.

Chapter 5

Experimental Validation and Results

This chapter presents a set of experimental scenarios designed to evaluate the proposed system in diverse and realistic applications. The selected use cases demonstrate the system's versatility and ability to operate in heterogeneous conditions, from structured industrial environments to unstructured and partially known spaces. First, in Section 5.1 we consider wind turbine inspection missions, where the UAV must synchronize its actions with the rotating nacelle, performing both frontal and oblique inspections without interrupting the turbine's operation. Second, we address solar park inspection, a domain characterized by large, repetitive structures distributed over wide areas in Section 5.2. These missions require efficient coordination between multiple UAVs and run-time plan modifications to maximize inspection performance. Finally, in Section 5.3 we explore a more challenging scenario involving nano-UAVs operating in unknown, unstructured environments, using minimal onboard sensing to cooperatively map and explore indoor spaces. These experiments validate different aspects of the system, including high-level behavior execution, real-time task adaptation, scalability to multiple agents, and robustness in the presence of environmental uncertainty.

5.1 Windmill Inspection

In this experiment, we tackled windmill inspections. The innovation offered against previous methods is the non-operational stop of the wind turbine during the checkup. Not halting the wind turbine operation supposes important savings and an increment in productivity compared to other intrusive methods. The project emerges as a partnership between the company Aeromedia S.L. [1] and the Computer Vision and Aerial Robotics group.

We performed most of the experiments using a simulated wind turbine due to the difficulty of testing our system in an industrial facility. The procedure consisted of doing real-world flights in a safe environment, e.g. test field free of obstacles, where the wind turbine is simulated. Further details of the methodology followed are explained in [Perez-Segui, Arias-Perez, et al., 2023].

[1] https://aeromedia.es/

The experiments presented here correspond to flights performed in March of 2023 at the Las Águilas aerodrome, and later in June of 2023 at a flight field in the south of Madrid. Afterward, we performed the industrial environment validation by inspecting several wind turbines in the experimental wind farm of Sotavento in Lugo, Spain. However, in order to preserve the intellectual property of the partnered companies, those experiments are not shared in this thesis.

Figure 5.1: Autonomous wind turbine inspection at Sotavento, Lugo.

5.1.1 Experimental Setup

Our setup proposes to mount onboard the three components of the deliberation systems: planning, monitoring, and acting. The design follows a decentralized approach, see Figure 5.2, where the GCS is in charge of defining the mission and supervising its execution. The UAV arranges the plan from the received mission and runs it, ensuring its completion while fulfilling certain constraints.

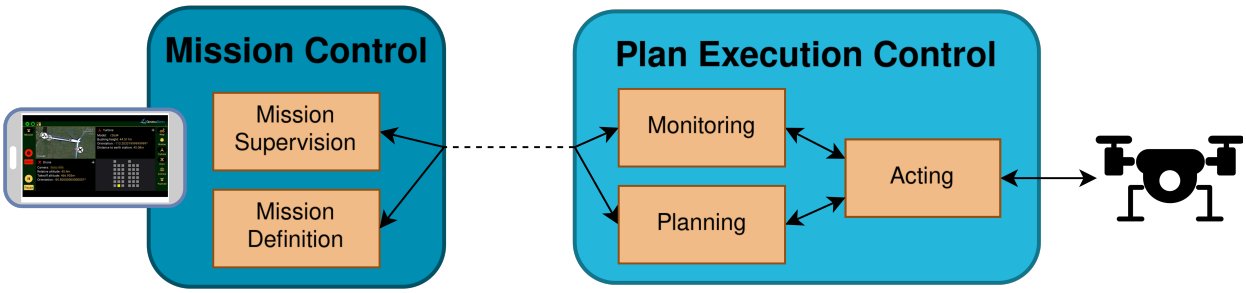


Figure 5.2: Decentralized deliberation scheme for wind farm inspection missions.

The UAV used in the experiments was a DJI M300. It carried an NVIDIA Xavier AGX board with Aerostack2 software running onboard. The payload is formed by an intelligent capturing system that takes pictures of the blades on its pass.

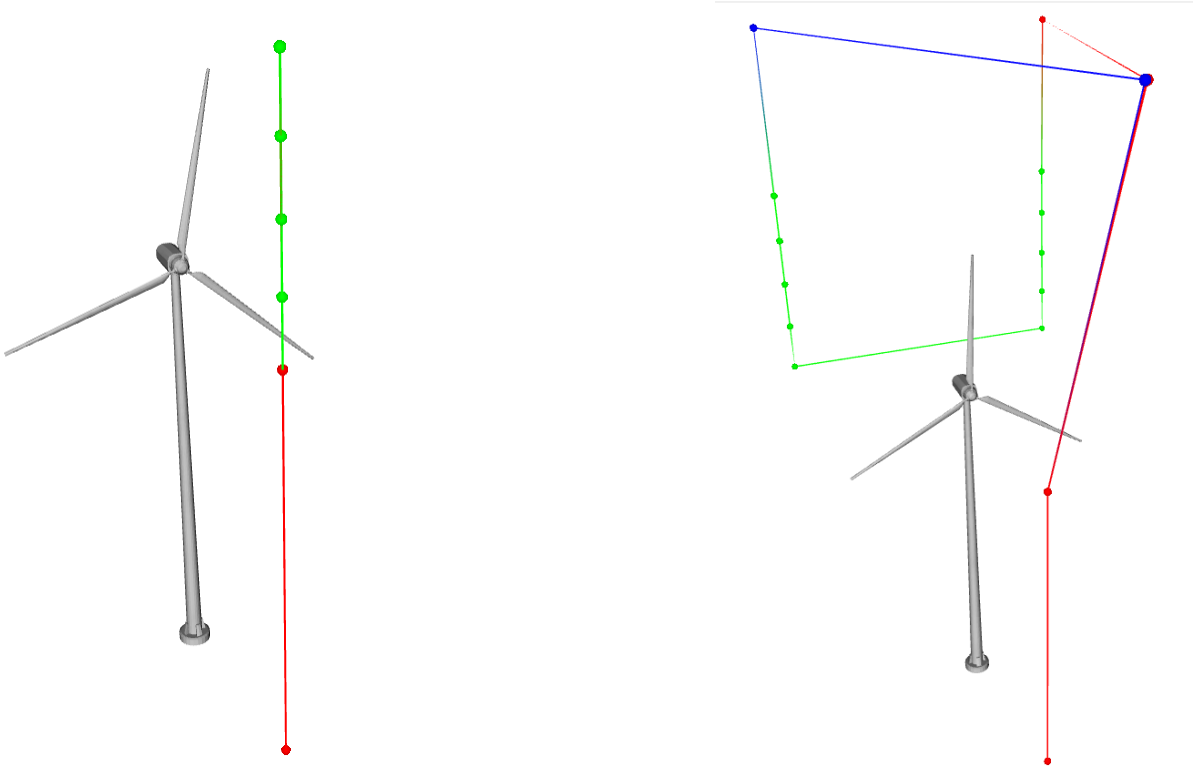


Figure 5.3: Frontal (right) and back oblique (left) inspection missions. Waypoints are classified as inspection (green), pre-inspection (blue) and post-inspection (red) points.

The GCS is an Android app built using the DJI Mobile SDK. Communication between the UAV and GCS goes through the DJI MOP channel[2]. Messages are codified in JSON format to exchange commands and monitoring information between ground and air sides.

Multiple inspection missions can be performed depending on the wind turbine model. Mission is defined by the human operator by setting parameters such as inspection angles, number of pictures to take, camera specifications, wind turbine dimensions, and other safety parameters. We describe two experiments in this thesis. First, a frontal inspection is examined, and later a back oblique inspection is detailed. Figure 5.3 illustrates the mission plan for the two experiments.

Both experiments use similar mission plans. As explained, our mission executor utilizes behavior as planning primitives. The behaviors used are the same for every inspection plan possible. The list of behaviors managed is in Table 5.1.

The plan consists of reaching the first inspection point, following the nacelle pan to align facing the wind turbine blades, and when in position activating the shooting system. Then, repeat it for every point in the inspection angle and go to the next angle. When all inspection angles are covered, return to the landing point. To safely cross from front to back, and vice-versa, a crossing maneuver is defined. Code Snippet 5.1 exemplifies the frontal inspection plan in JSON format.

[2] https://developer.dji.com/doc/payload-sdk-api-reference/en/practice/mop-channel.html

Table 5.1: List of behaviors used in the wind farms inspections.

Behavior	Description	Parameters
`TAKEOFF`	Change UAV state from landed to flying. Aircraft must be armed and in offboard mode.	*Height* [m]: desired takeoff height from the starting point; *Speed* [m/s]: desired takeoff speed.
`GO_TO`	Move the UAV to a given position in the desired frame.	*Target position* [m]: desired goal position; *Speed* [m/s]: desired speed; *Yaw*: desired mode; *Frame ID*: reference frame.
`FOLLOW_REF`	Keep the UAV at a given pose reference in the desired frame.	*Pose* [m]: desired position and orientation to frame; *Speed* [m/s]: desired maximum speed; *Yaw*: desired mode; *Frame ID*: reference frame.
`TAKE_PHOTO`	Activate capturing system to take photos.	*N*: number of photos to take.
`LAND`	Change UAV state from flying to landed.	*Speed* [m/s]: desired landing speed.

```
{"takeoff": {"height": initial_height}},
{"go_to": {"z": nacelle_height, "frame_id": "earth"}},
{"go_to": {"x": inspection_distance, "frame_id": "wind_turbine/nacelle"}},
{"follow_reference": {"x": inspection_distance, "z": 0.0, "frame_id": "wind_turbine/nacelle"
    }},
{"follow_reference": {"x": inspection_distance, "z": blade_length/steps, "frame_id": "
    wind_turbine/nacelle"}},
            ...
{"follow_reference": {"_x": inspection_distance, "z": blade_length, "frame_id": "
    wind_turbine/nacelle"}},
{"go_to": {"x": 0.0, "y": 0.0, "z": current_height, "frame_id": "earth"}},
{"go_to": {"z": initial_height, "frame_id": "earth"}},
{"land": {"speed": speed}}
```

Code Snippet 5.1: Frontal inspection mission description.

5.1.2 Frontal Inspection Mission

We examined the sequences of behavior activations during experimental flights to evaluate the execution control system. This analysis aimed to verify that the system correctly interpreted the mission plan, ensuring a smooth transition from planned tasks to executed behaviors.

Figure 5.4 shows the behavior activation sequence during one frontal inspection. The figure on the left discriminates behaviors by type and UAV, while the figure on the right splits them by UAV only. Besides, right figure displays mission update messages from the GCS, where load and start commands are indicated as “X” markers. Shooting system activation is marked as black dashed lines.

A strong relation between the sequence of activations and the plan described in Code Snippet 5.1 can be observed. It is worth mentioning that the shooting system is triggered by the plan

monitoring subsystem. The subsystem tracks the position of the UAV, and when in position, it sends a trigger signal to the camera.

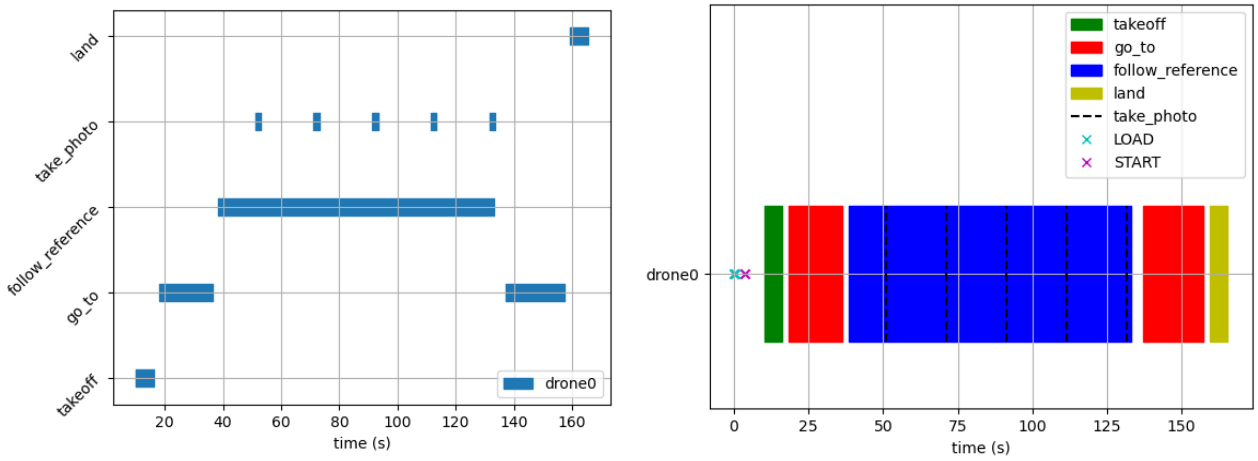


Figure 5.4: Sequence of behavior activations during the frontal inspection of a wind turbine.

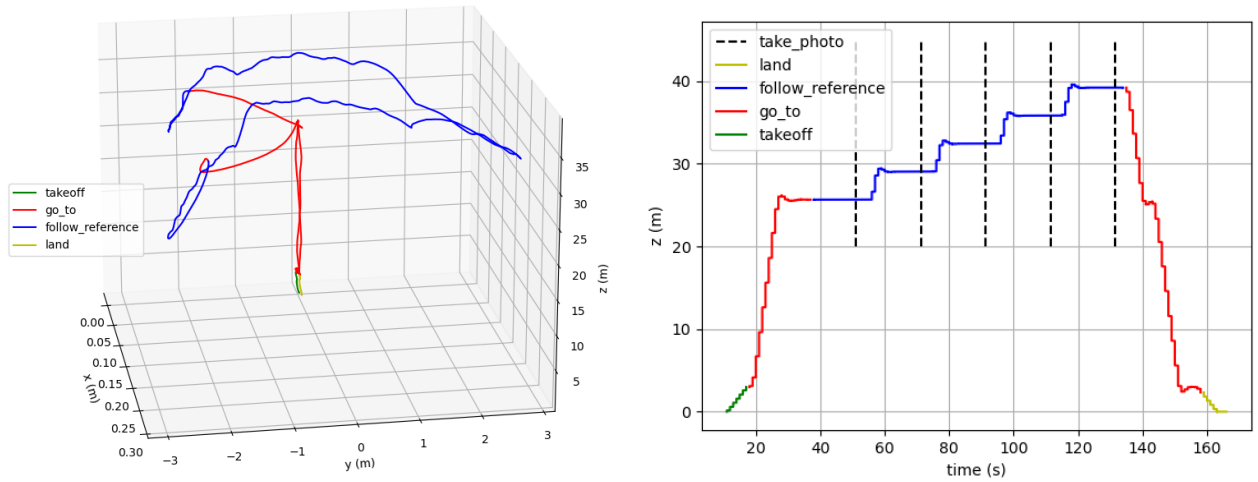


Figure 5.5: Trajectory followed by the UAV during the frontal inspection mission. Colors represent the behavior active during those timestamps.

Figure 5.5 plots the 3D trajectory followed by the UAV during the frontal inspection. The path is colorized according to active behavior. It can be noticed how the UAV moves on the XY plane in order to follow the wind turbine panning. A summary video of the experiment is available at https://vimeo.com/825790099/c5e64a026a.

The evaluation method confirmed that the sequence of behavior activations accurately followed the mission plan. The UAV successfully synchronized its movements with the nacelle's rotation, maintaining the correct positioning throughout the inspection. As a result, it was able to capture the required images at the designated locations without disrupting the wind turbine operation. These findings demonstrate the system's capability to execute dynamic inspections in real-world conditions.

5.1.3 Back-oblique Inspection Mission

During the oblique back inspection, pictures are taken from two perspectives: 135º and -135º from the front of the windmill nacelle. The UAV starts in the front, so it has to travel to the back and return by flying over the windmill.

We perform the same analysis for this experiment as for the frontal inspection. Figure 5.6 shows the behavior activation sequence during the experiment. On the left, behaviors are displayed by type and UAV. On the right, behaviors are plotted by UAV only, alongside the mission update messages from the GCS. Load and start commands are indicated as "X" markers, and photograph activation is marked as black dashed lines.

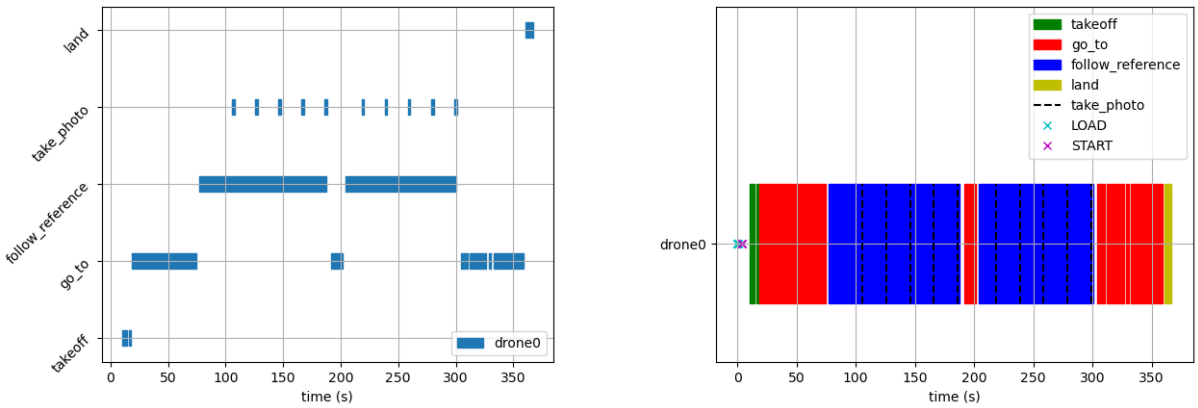


Figure 5.6: Sequence of behavior activations during the back-oblique inspection experiment.

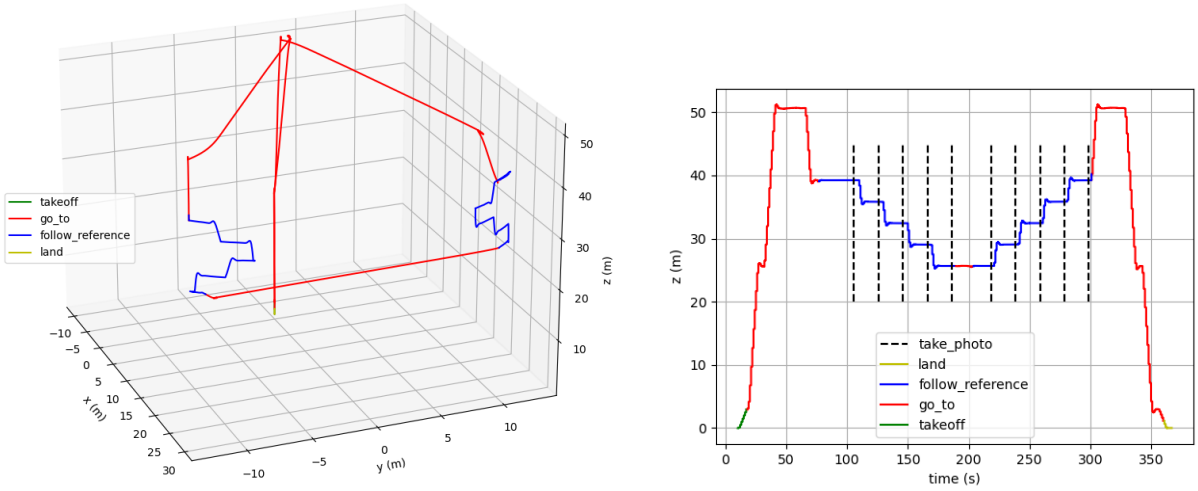


Figure 5.7: Trajectory followed by the UAV during the back-oblique inspection experiment. Colors represent the behavior active during those timestamps.

Figure 5.7 plots the 3D trajectory followed by the UAV during the inspection experiment. On the left, the 3D trajectory is plotted, while on the right, the Z coordinate against time is presented. The path is colorized according to active behavior.

The evaluation method confirmed that the sequence of behavior activations aligned with the mission objectives. The UAV successfully navigated back and forth between the starting position and the designated inspection angles while maintaining safe operation. Additionally, it activated the imaging system precisely at the correct positions, synchronizing with the nacelle's rotation to capture the required oblique-view images. The mission was executed smoothly, demonstrating the system's ability to perform dynamic inspections with accuracy and reliability.

5.2 Multi-UAV PV-plant Inspection

The problem addressed in this scenario is the autonomous multi-UAV inspection of a PV-panel solar park. We performed two different experiments. First, a back-and-forth multi-UAV area coverage was done to inspect the whole solar park. Second, an on-demand panel inspection was implemented, based on online requests to inspect single panels.

Figure 5.8: DJI M300 inspecting a PV plant at the Repsol Technology Lab in Móstoles, Madrid.

Experiments were performed during March of 2024 in the PV plant of the Repsol Technology Lab in Móstoles, Madrid. The solar park inspected has an extension of around 5000 square meters. Figure 5.8 shows one of the UAVs during an inspection mission.

5.2.1 Experimental Setup

Both experiments share the same experimental setup. The pipeline of the automated inspection is composed of three elements, detailed in Figure 5.9. These components are an intelligent PV plant monitoring system, a ground control station, and a swarm of UAVs equipped with RGB-IR sensors.

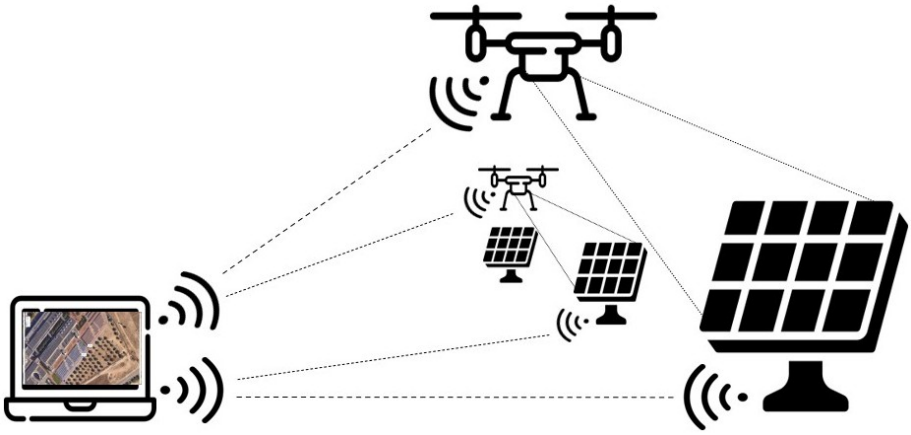


Figure 5.9: Overview of the inspection system components for the multi-UAV solar park inspection.

The intelligent PV plant monitoring is composed of IoT modules that have been wired to the electrical system of the panels, along with self-diagnosing tools capable of generating different types of alerts. The system is further explained in [Melero-Deza et al., 2024; Tradacete-Ágreda et al., 2025].

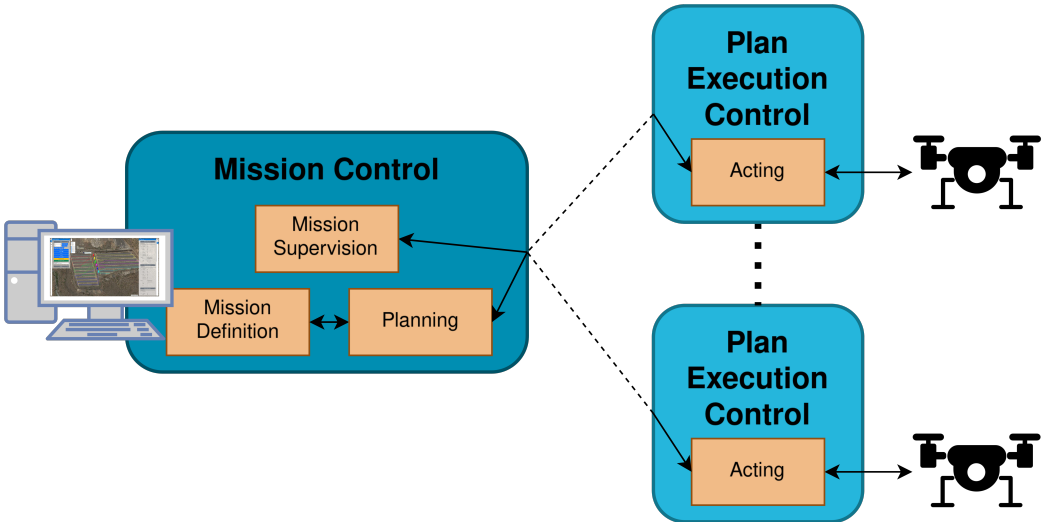


Figure 5.10: Centralized deliberation scheme for solar park inspection missions.

The ground control station serves as the planner and mission supervision unit. The GCS selected was the Aerostack2 webGUI[3]. The approach followed is centralized, as depicted in Figure 5.10, where the planning system is integrated together with the mission definition, and the plan monitoring is omitted. Moreover, the GCS integrates a communication system to receive alerts from the IoT modules. These alerts are listed to the operator, who manually confirms sending a request to one of the UAVs in the swarm.

[3]https://github.com/aerostack2/as2_web_gui

The swarm of UAVs is capable of capturing and processing thermal images of the PV panels. Every UAV carries the same payload formed by a gimbal and a hybrid RGB-IR camera. However, the swarm is heterogeneous, formed by a DJI M300 and DJI M350. Both carry an NVIDIA Xavier AGX board with computer vision and deep learning algorithms, and Aerostack2 software running onboard.

5.2.2 Area Coverage Inspection Mission

The planning method used is a coverage path planning algorithm. The method takes several areas to inspect, calculates the paths to perform the coverage using a back-and-forth algorithm, and allocates them evenly to the available UAVs in the swarm. Further details of the algorithm are detailed in [Luna et al., 2024]. Figure 5.11 shows a coverage mission defined using the Aerostack2 webGUI.

Figure 5.11: Area coverage mission definition using Aerostack2 webGUI.

Utilizing behaviors as fundamental planning primitives simplifies the complexity of plan definition. By incorporating semantic elements, high-expressivity plans minimize planning effort and enhance clarity in mission specification. Mainly, two behaviors were needed for creating the coverage inspection mission: `FOLLOW_PATH` and `POINT_GIMBAL`. Alongside, three other basic behaviors are used to completely define the plan. Table 5.2 lists the whole set of behaviors used with a brief description and mandatory parameters of each. During this experimental scenario, we were capturing the camera images in a rosbag, through a sensor measurement topic. The `TAKE_PHOTO` behavior is listed in the table, even though it was not needed for capturing the images.

Basically, the plan consists of reaching the starting point (first waypoint from the back-and-forth path), pointing the gimbal to the appropriate orientation to correctly capture the panels, and following the path of waypoints. After finishing the path, the recording is stopped, and a return-to-land maneuver is executed. Code Snippet 5.2 exemplifies the plan in JSON format.

Table 5.2: List of behaviors used in PV-plant inspections.

Behavior	Description	Parameters
`TAKEOFF`	Change UAV state from landed to flying. Aircraft must be armed and in offboard mode.	*Height* [m]: desired takeoff height from the starting point; *Speed* [m/s]: desired takeoff speed.
`GO_TO`	Move the UAV to a given position in the desired frame.	*Target position* [m]: desired goal position; *Speed* [m/s]: desired speed; *Yaw*: desired mode; *Frame ID*: reference frame.
`FOLLOW_PATH`	Move the UAV along a given path in the desired frame.	*Path* [m]: desired path with a list of coordinates; *Speed* [m/s]: desired speed; *Yaw*: desired mode; *Frame ID*: reference frame.
`POINT_GIMBAL`	Orientate gimbal to aim to point.	*Target point* [m]: coordinate to point gimbal to; *Frame ID*: reference frame.
`TAKE_PHOTO`	Capture image from camera.	-
`LAND`	Change UAV state from flying to landed.	*Speed* [m/s]: desired landing speed.
`RTL`	Composition of `GO_TO` takeoff location and `LAND` behaviors.	*Height* [m]: desired go_to height; *Speed* [m/s]: desired landing speed.

```
{"takeoff": {"z": inspection_height}},
{"go_to": {"z": inspection_height, "frame_id": "earth"}},
{"point_gimbal": {"frame_id": "panel_x"}},
{"go_to": {"position": waypoint_0, "frame_id": "earth"}},
{"follow_path": {"path": [waypoints], "speed": speed, "frame_id": "panel_x"}}
{"rtl": {"z": safety_height, "speed": speed}},
```

Code Snippet 5.2: Coverage inspection mission plan description.

To assess the execution control system, we conducted an analysis of behavior activation sequences and corresponding process executions observed during experimental flights. The goal was to ensure that the system accurately translated the plan into coherent behavior activations.

We applied this method to the coverage inspections executed. From them, we extracted the temporal sequence of behavior activations and the 3D trajectory followed by the UAVs. Figure 5.12 shows the behavior activation sequence during one of the coverage inspections. The figure on the left discriminates behaviors by type and UAV, while the figure on the right splits them by UAV only. Besides, right figure displays mission update messages from the GCS, where load and start commands are indicated as “X” markers.

The outcome sequence corresponds to the plan generated by the GCS. First message update is received around second 200; the UAVs were ready, waiting to receive the plan while the human operator was defining the mission in the GCS. Mission starts after second 250, when the operator hits the “fly” button. For this experiment, there were no behaviors active in parallel. The long-duration execution of the `FOLLOW_PATH` behavior, covering approximately 70% of the mission time, indicates that most of the operation was spent efficiently navigating through the coverage area, as intended.

Figure 5.13 displays the 3D trajectory followed by both UAVs during the area coverage mission.

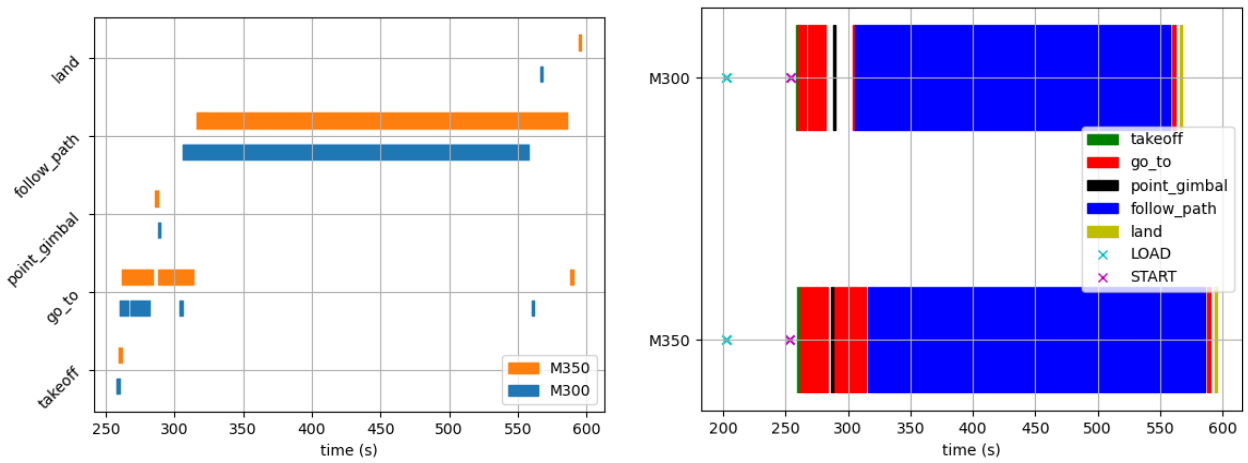


Figure 5.12: Sequence of behavior activations during the area coverage mission.

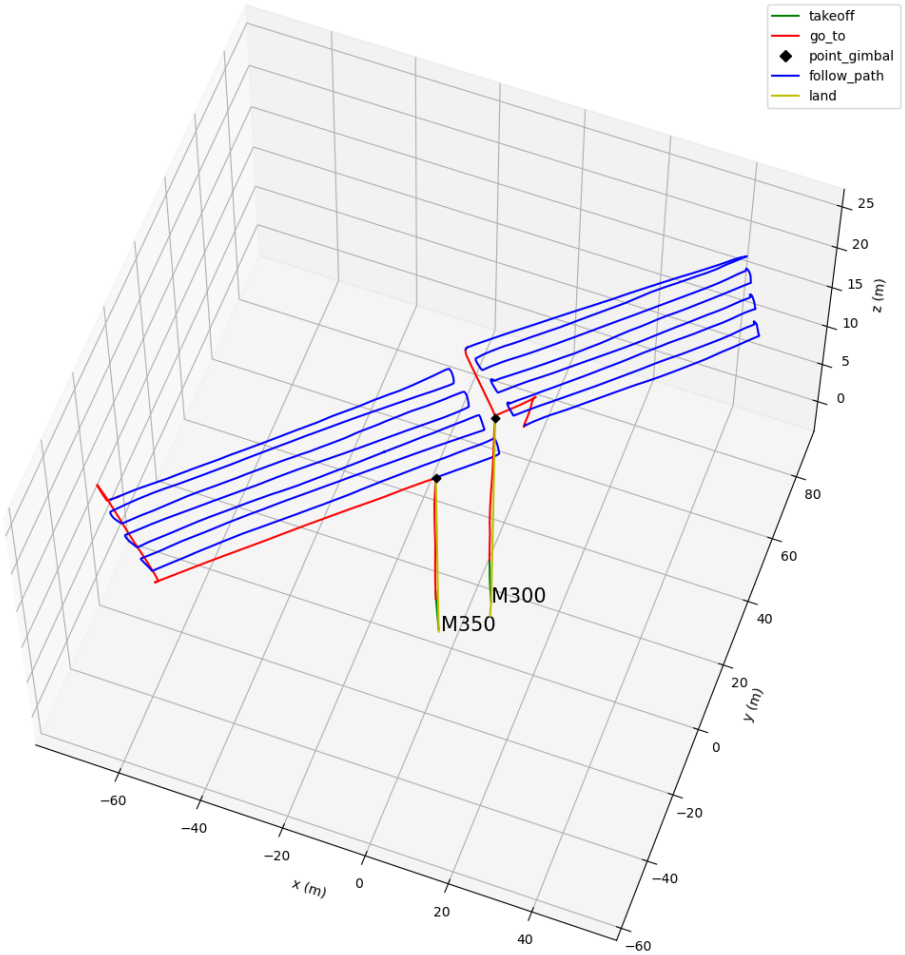


Figure 5.13: 3D trajectory followed by the UAVs during the coverage mission. Colors represent the behavior active during those timestamps.

The path is colorized according to active behavior. Additionally, the 3D flight paths in Figure 5.13 show a strong correlation between planned and executed trajectories. The comparison between the flown path and the original mission design in the GCS (Figure 5.11) verifies that the UAVs executed the mission objectives without significant deviations. A video summarizing the experiment and confirming these observations is available at https://vimeo.com/1002821624.

The results of the area coverage inspection experiments confirm that the execution control system effectively translates planned missions into coherent and consistent behavior activations. The extracted sequences of behavior activations and 3D UAV trajectories demonstrate that the system reliably follows the intended plan, ensuring precise execution of coverage tasks. Furthermore, the successful execution of multiple UAVs operating simultaneously without interference highlights the scalability and robustness of the system.

5.2.3 On-demand Panel Inspection Mission

During an on-demand panel inspection, plans are generated and loaded into the mission executor during run-time. UAVs can be landed or already performing a mission. In the second case, the plan being executed is halted and replaced by the new plan.

The plan is formed by approaching the alert panel location, vertically descending to the inspection height, correcting gimbal orientation to center the panel in the image, and performing the panel diagnostic analysis, as depicted in Code Snippet 5.3. The plan is built relative to the panel coordinated frame, so the onboard system does not implicitly need to know where the panel is located in the world. The GCS handles transformation tree creation based on the receiving alerts, which contain location information of the panel to inspect.

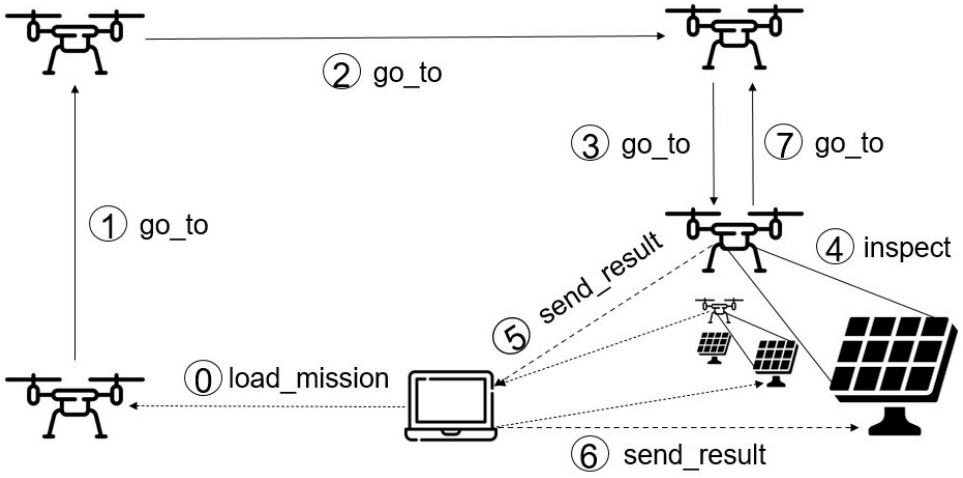


Figure 5.14: Visual representation of the on-demand panel inspection mission.

The result of the diagnosis is sent to the ground control station, where the human operator decides whether to repeat the inspection, start another mission, or return to the landing point. Figure 5.14 shows a visual representation of the generated mission.

Behaviors used in this scenario are the ones used for the coverage inspection mission, plus `PANEL_DIAGNOSE` behavior in charge of capturing the image, detecting the panel in the image,

```
{"takeoff": {"z": approach_height}},
{"go_to": {"position": [current_x, current_y, approach_height], "frame_id": "earth"}},
{"go_to": {"position": [0.0, 0.0, approach_height], "frame_id": "panel_x"}}
{"go_to": {"position": [0.0, 0.0, capture_distance], "frame_id": "panel_x"}},
{"trigger_inference": {"frame_id": "panel_x"}},
{"go_to": {"position": [0.0, 0.0, approach_height], "frame_id": "panel_x"}}
```

Code Snippet 5.3: On-demand panel inspection mission description.

and performing a temperature analysis. It works similarly to `TAKE_PHOTO` behavior, but it needs the frame ID to correctly point to the panel before being activated.

To enhance clarity, we present two experiments illustrating this scenario. In the first experiment, a single UAV is tasked with handling inspection requests for two panels, demonstrating the system's ability to manage sequential operations efficiently. In the second experiment, two UAVs work collaboratively to inspect three panels, showcasing the scalability of the approach and its capability to coordinate multiple agents for improved efficiency.

Single UAV on-call inspection experiment

We conducted the same analysis for this scenario as for the coverage mission. Figure 5.15 shows the behavior activation sequence. Three plans are created and loaded during run-time, two correspond to on-call inspection and one to return-to-land. Each start command involves a sequence of behavior activations.

Figure 5.16 illustrates the trajectory followed by the M300 UAV. The figure on the left plots the 3D trajectory, while the one on the right plots coordinate z only. The path is colorized according to the active behavior. Two rows of PV panels, highlighting the ones that send the inspection request, are also plotted.

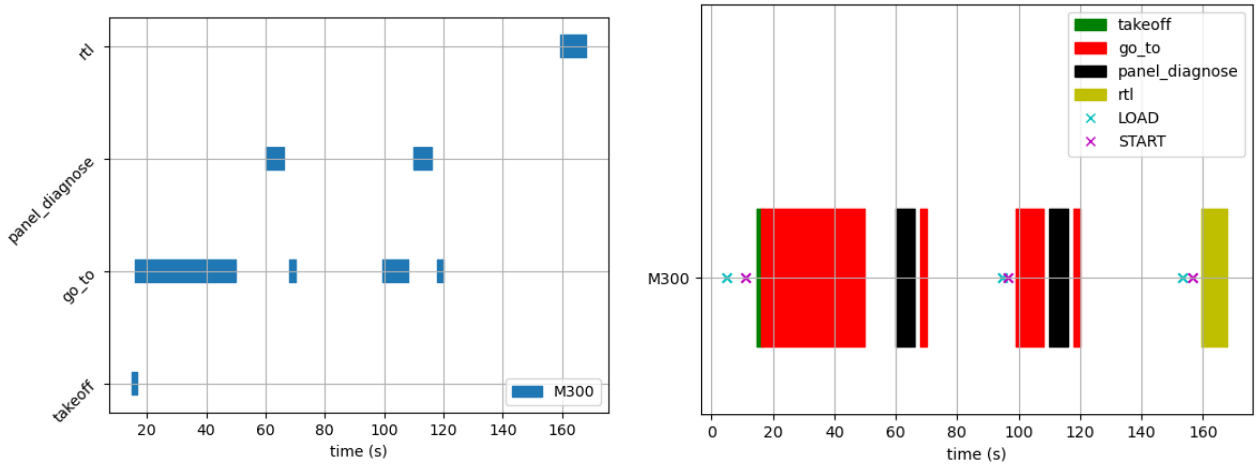


Figure 5.15: Sequence of behavior activations during the on-call inspection mission.

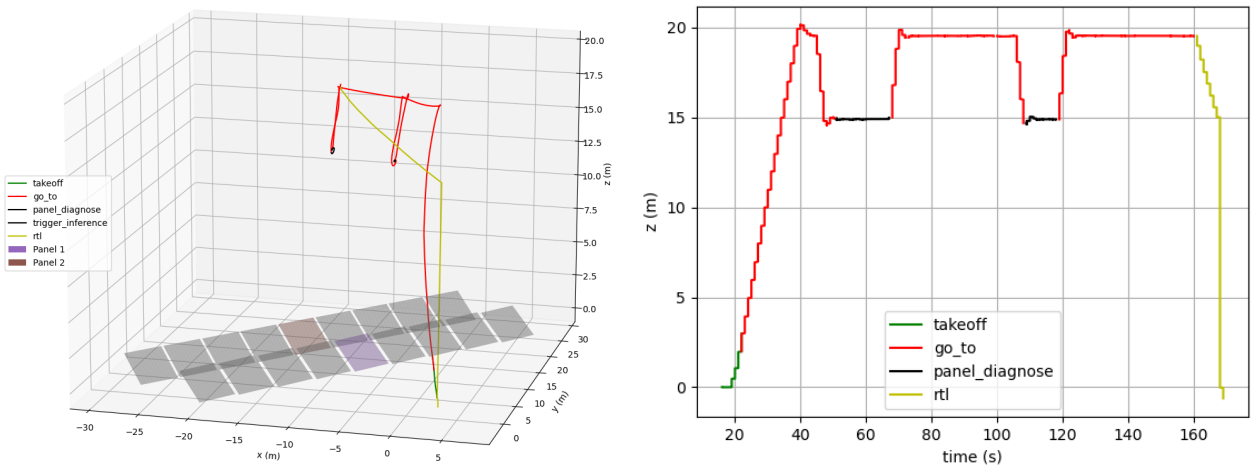


Figure 5.16: Trajectory followed by the M300 during the on-call inspection mission. Colors represent the behavior active during those timestamps. The array of panels are plotted as colored squares.

Multi-UAV on-call inspection experiment

Second experiment includes the M350 UAV and another panel to inspect. Figure 5.17 presents the activation sequence of behaviors for both UAVs. The M300 handles requests for panels 1 and 2, while the M350 handles one request for panel 3.

Figure 5.18 showcases the trajectories followed for both UAVs. The figure on the left plots the 3D trajectory, while the one on the right plots coordinate z only. Paths are colorized according to the active behavior. Panels that generated requests are also plotted in colorized squares.

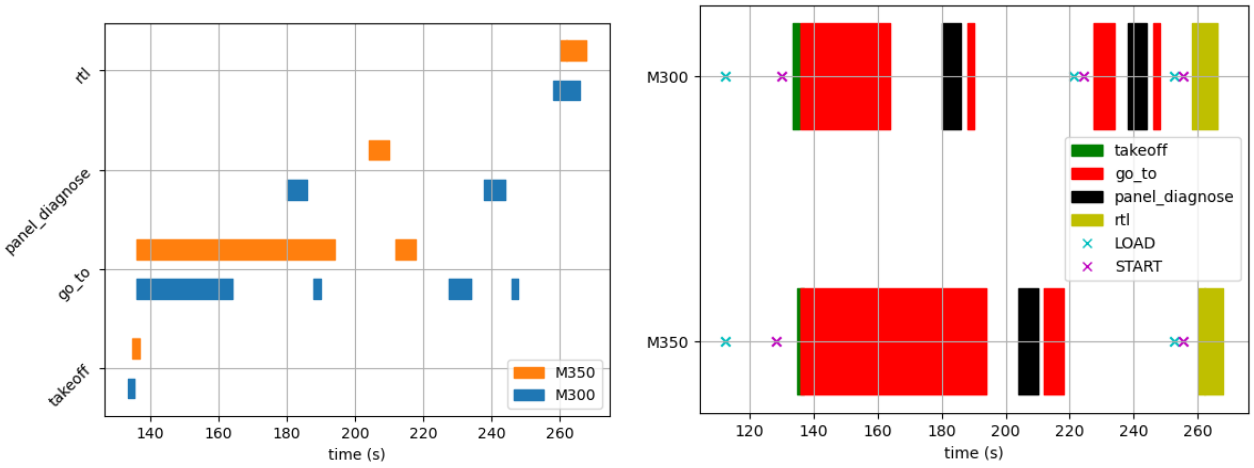


Figure 5.17: Sequence of behavior activations during the on-call inspection mission with 2 UAVs swarm.

The results from the analysis of the two experiments demonstrated that all behaviors were consistently executed according to the mission plan, ensuring coherence between planned

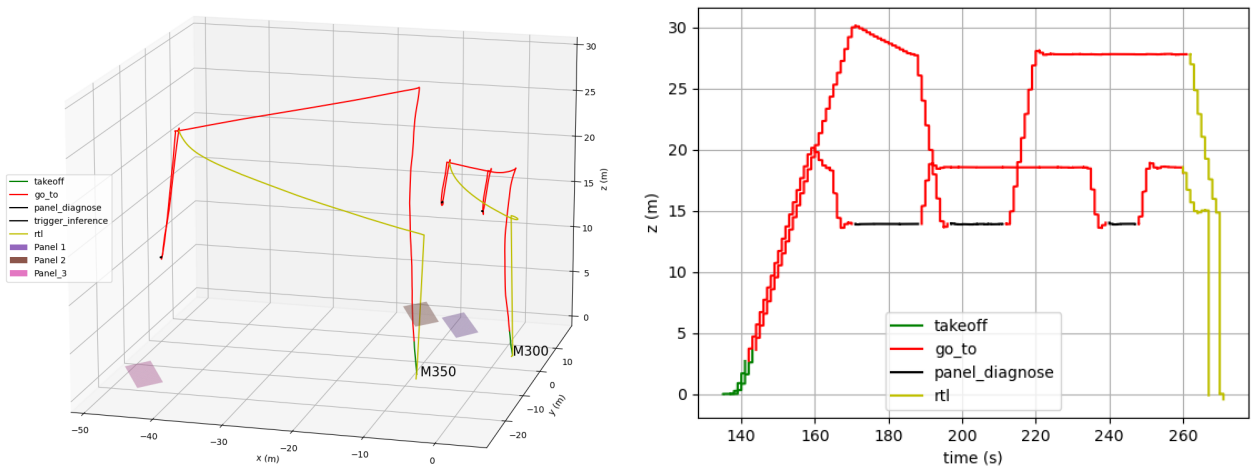


Figure 5.18: Trajectory followed by the UAVs during the on-call inspection mission. Colors represent the behavior active during those timestamps. Panels are plotted as colored squares.

and actual execution. Moreover, the outcome indicates that the executor is reliable for real-time planning applications, where new plans must be created or modified dynamically during run-time. The system demonstrated efficient behavior transitions with minimal latency, successfully adapting to changing conditions without interrupting the mission flow. Additionally, the experiments confirmed that the approach scales effectively with multiple UAVs, maintaining synchronization and avoiding conflicts, further validating its robustness in multi-agent scenarios for real-world industrial applications.

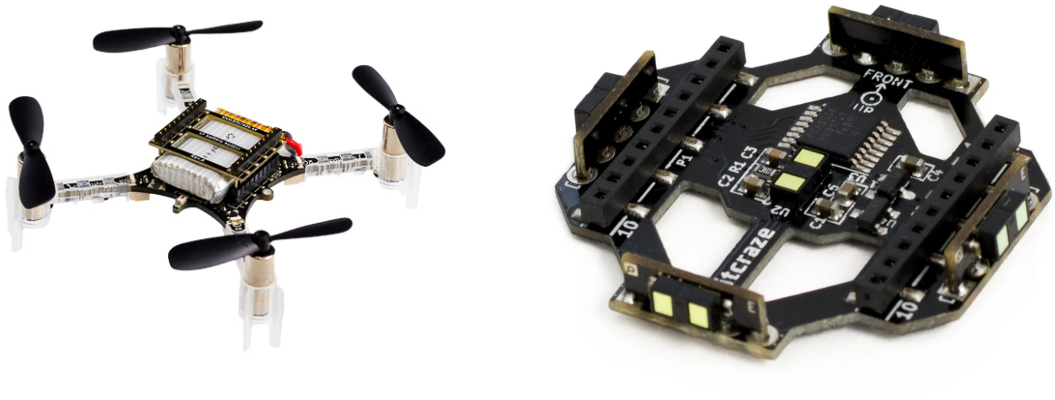
5.3 Minimal Sensing Exploration of Unstructured Environments

To evaluate the performance of the proposed algorithm in Section 4.3, we conducted a series of experiments under varying conditions, including different world sizes, obstacle densities, and swarm sizes. The validation process was carried out in both simulated and real-world environments to assess the algorithm's effectiveness and robustness across diverse scenarios.

5.3.1 Experimental Setup

The nano-UAVs used to form the swarm in our experiments are Bitcraze Crazyflie 2.1 quadcopters, each weighing 27 grams with a diameter of 10 cm (see Figure 5.19a). These UAVs are equipped with a multi-ranger deck featuring five VL53L1x Time-of-Flight (ToF) sensors (see Figure 5.19b), capable of measuring distances up to 4 meters with millimeter precision at a frequency of 20 Hz. Additionally, each UAV carries a motion capture marker deck, with the total UAV payload, including the battery, being under 10 grams.

Our experiment scenarios replicated non-structured environments, using cylindrical poles as obstacles that imitate forest trees for both simulated and real environments (see Figure 5.20). The variety in our experiments stemmed from different parameters such as the density of



(a) Bitcraze Crazyflie 2.1 quadcopter. (b) Multi-ranger deck.

Figure 5.19: Hardware setup used in real environments.

these obstacles, the size of the UAV swarm, and the initial placements of UAVs and obstacles' locations within the testing area.

Figure 5.20: Nano-UAV performing a minimal sensing exploration of a real-world unknown unstructured environment.

Simulations are conducted using Gazebo replicating Crazyflie characteristics. The quadcopter

dynamics are simulated based on RotorS [Furrer et al., 2016] using the ground truth of the UAV. A multi-ranger deck imitation is placed onboard as the only payload.

The exploration algorithm utilizes Aerostack2 [Fernandez-Cortizas et al., 2023] for navigation and control, enabling the use of a consistent algorithm across both simulated and real-world setups. However, there are slight variations in the Aerostack2 components used in each system. Specifically, the interfaces connecting the platforms to Aerostack2 differ, and the state estimator plugins vary. Aerostack2 functions uniformly across both environments, providing a seamless transition from simulation to actual deployment.

5.3.2 Exploration in Simulated Scenarios

To evaluate the performance of the proposed exploration strategy, we conducted a series of benchmark tests in a Gazebo simulation environment. These tests assessed the algorithm's effectiveness under various conditions by varying swarm size, initial UAV positions, obstacle density, and world size. The primary objective was to analyze how different configurations impact exploration efficiency and overall system behavior. For all experiments, we set a navigation speed of 0.75 m/s, a rotational speed of 0.15 rad/s, and a sensor range of 4 meters. Additionally, we defined safety distances of 0.4 meters to obstacles and 2.0 meters to other UAVs to prevent collisions during autonomous exploration. Notably, these values differ slightly from those used in real-world experiments due to differences in the physical dimensions of the UAVs. Simulated UAVs are slightly larger than their real-world counterparts, necessitating an increase in both sensor range and safety distances to maintain equivalent operational conditions. The complete set of simulation parameters is summarized in Table 5.3.

Parameter	Value
Map resolution	0.1
Sensor range	4
Safety distance to obstacles	0.4
Safety distance to other UAVs	2.0
Navigation speed	0.75
Reached distance threshold	0.2
Spin speed	0.15
Yaw threshold	0.15
Minimum frontier size	1.5
Maximum frontier size	3.5

Table 5.3: Parameter configuration for simulation experiments.

To simulate diverse exploration scenarios, we designed three distinct virtual environments with varying obstacle densities. Each scenario consists of a 50m $\times$ 50m area populated with obstacles at densities of 0.05, 0.1, and 0.2 obstacles per square meter (from now on obs/m^2), respectively. These environments, illustrated in Figure 5.21, represent different levels of navigational complexity, ranging from relatively open spaces to highly cluttered terrains. For each environment configuration, we conducted experiments using swarms of one, two,

three, five, and seven UAVs, allowing us to analyze how swarm size influences exploration performance under different environmental constraints.

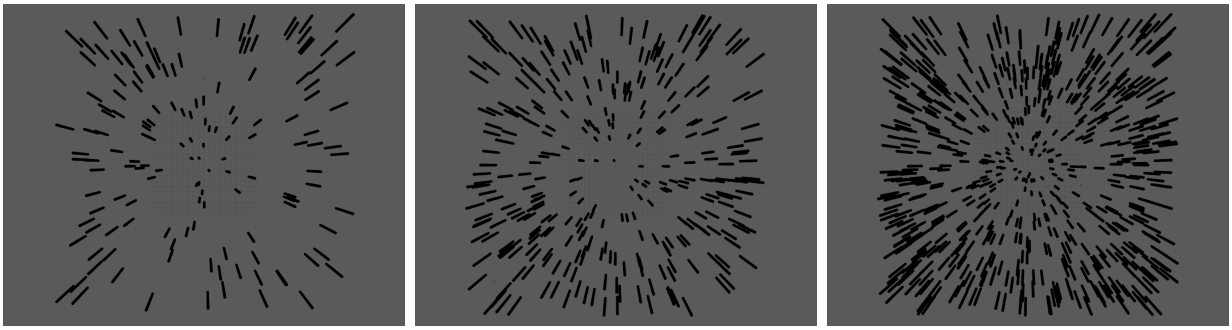


Figure 5.21: Environments used in the simulated experiments. From left to right, obstacle density is 0.05, 0.1, and 0.2 obs/m^2.

Figure 5.22 provides an example of exploration progress in the low-obstacle-density scenario with a swarm of three UAVs, showcasing how the team systematically expands the mapped area over time. Additionally, Figure 5.24 presents a top-down view of the tested swarm sizes actively exploring a low-obstacle-density environment.

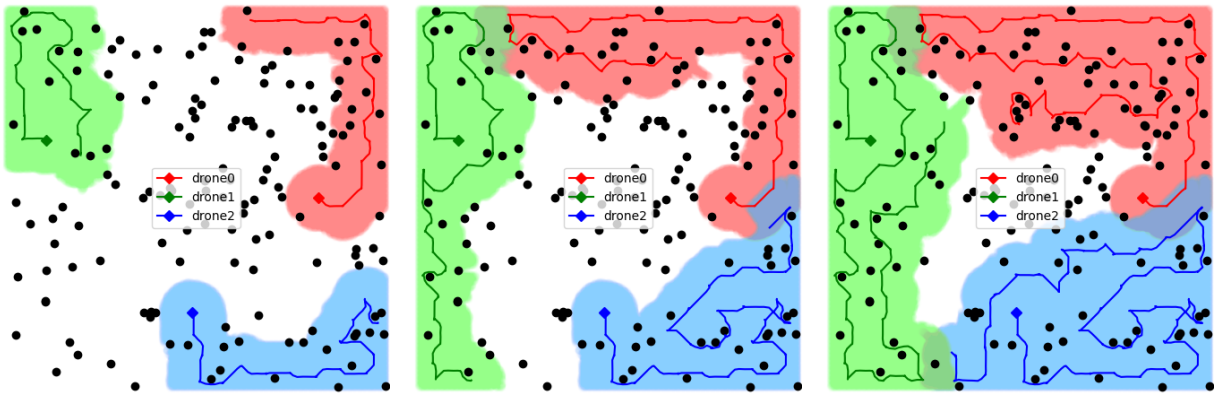
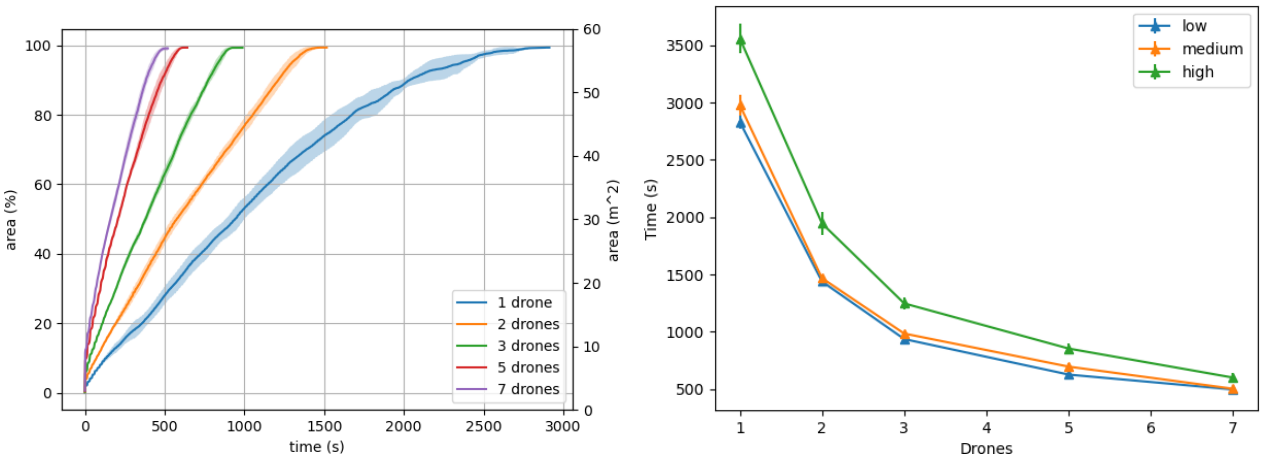


Figure 5.22: Exploration progress at 25%, 50% and 75% of the area mapped with three UAVs in the low obstacle density environment. Each colored line represents the trajectory followed by each UAV, while the shaded areas symbolizes the explored regions. Colored diamonds shows UAVs starting points and black dots represent the obstacles in the environment.

We observed that the exploration rate, which quantifies how quickly new areas are discovered over time, increased consistently across all three experimental scenarios as we added more UAVs to the exploration algorithm. This trend suggests that a larger swarm size leads to a more efficient distribution of exploration tasks, enabling the environment to be mapped at a faster pace. Figure 5.23a illustrates the exploration performance in the low obstacle density environment, where UAVs were able to navigate with minimal obstructions and cover large portions of the map efficiently. However, when we increased the number of obstacles, we observed a noticeable decline in the exploration rate, as depicted in Figure 5.23b. This drop occurs because UAVs require additional maneuvering to avoid obstacles, leading to longer paths and more complex trajectories.



(a) Average exploration rate for the 0.05 obs/m^2 density scenario. The shaded area shows the standard deviation.

(b) Exploration time per swarm size depending on the obstacle density. Low stands for 0.05 obs/m^2, medium for 0.1 obs/m^2 and high for 0.2 obs/m^2.

Figure 5.23: Simulation experiment results up to seven UAVs.

The top-view visualization shown in Figure 5.24 highlights how the frontier allocation heuristic effectively distributes exploration tasks across the swarm, ensuring that each UAV is assigned to a distinct region of the environment. In contrast, alternative heuristics struggled to maintain a balanced task allocation, often leading to inefficient movement patterns or redundant coverage of already-explored areas.

Obstacle density [obs/m^2]	# of UAVs	Time [s]	Area [%]	Path Length [m]	Overlap [%]
0.05	1	2831.79 ± 59.62	99.37 ± 0.01	986.66 ± 24.51	-
	2	1438.00 ± 45.98	99.37 ± 0.02	972.4 ± 34.08	25.79 ± 8.67
	3	935.40 ± 24.79	99.32 ± 0.01	**913.10 ± 23.65**	**16.88 ± 3.27**
	5	624.98 ± 21.77	**99.37 ± 0.02**	920.92 ± 26.87	31.32 ± 5.17
	7	**494.99 ± 22.17**	99.14 ± 0.66	1037.30 ± 46.89	35.26 ± 5.06
0.1	1	2979.00 ± 91.46	98.56 ± 0.06	1008.97 ± 37.59	-
	2	1467.00 ± 29.25	98.62 ± 0.04	998.47 ± 18.87	24.0 ± 6.16
	3	983.39 ± 23.43	98.62 ± 0.03	**941.17 ± 22.53**	**17.85 ± 2.52**
	5	694.60 ± 27.77	98.63 ± 0.02	1008.51 ± 39.45	30.29 ± 5.45
	7	**500.80 ± 20.02**	**98.67 ± 0.02**	1126.36 ± 34.72	36.06 ± 3.44
0.2	1	3560.80 ± 125.83	95.19 ± 1.13	1153.33 ± 19.85	-
	2	1944.99 ± 99.69	96.26 ± 0.62	1214.82 ± 86.44	26.42 ± 8.81
	3	1246.79 ± 49.86	95.02 ± 0.98	**1153.14 ± 64.81**	**25.18 ± 8.35**
	5	852.38 ± 49.70	96.28 ± 1.02	1265.07 ± 69.12	27.73 ± 3.01
	7	**598.61 ± 37.37**	**96.54 ± 0.43**	1288.79 ± 91.09	35.71 ± 3.99

Table 5.4: Results of the simulation experiments. We report the average and standard deviation for the completion time, percentage of area explored and total path traveled for the swarm. Best results are highlighted in bold.

A comprehensive summary of the quantitative results obtained from our experiments is provided in Table 5.4. In our study, we define an exploration experiment as completed when no unexplored frontiers remain within the environment, meaning that all reachable areas have been mapped. The experiment timer starts when the UAVs initiate takeoff and stops when all UAVs have successfully landed after completing their assigned tasks. Several key performance metrics were evaluated:

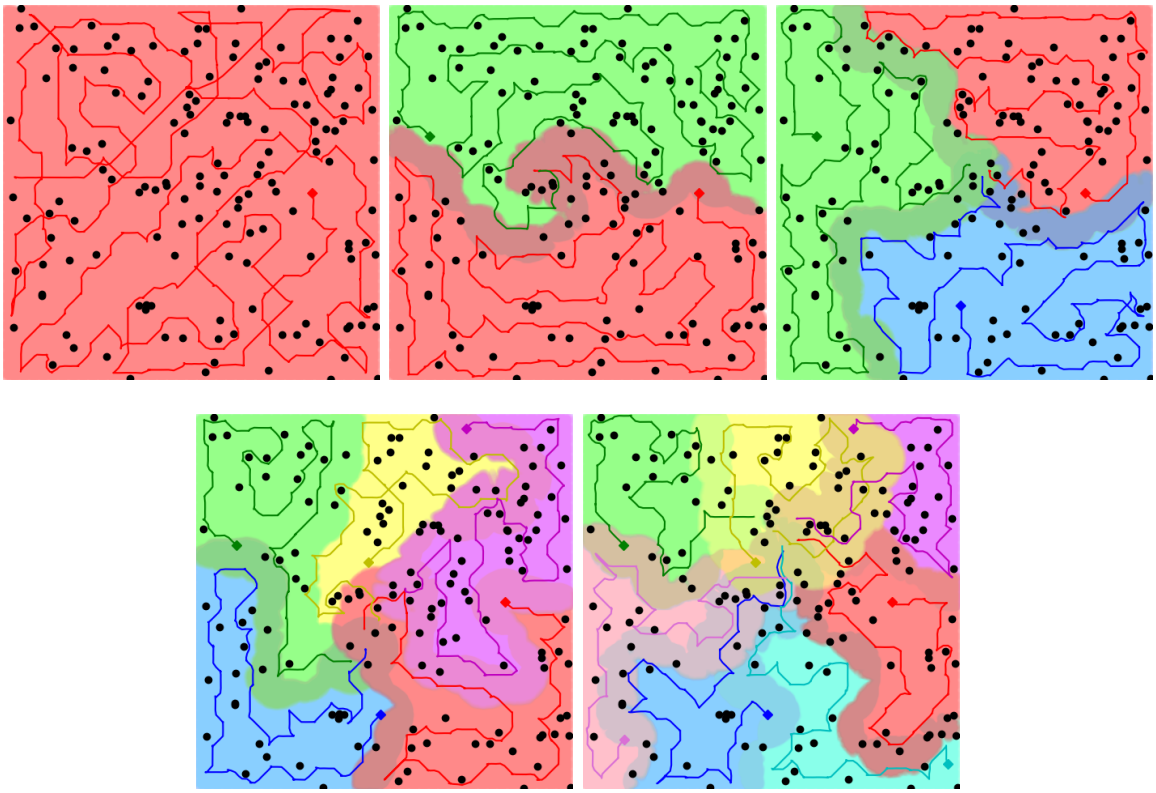


Figure 5.24: Explorations from one to seven UAVs in the low obstacle density environment. Each colored line represents the trajectory followed by each UAV, while the shaded areas symbolizes the explored regions. Colored diamonds shows UAVs starting points and black dots represent the obstacles in the environment.

- The area explored column measures the fraction of the total environment that was successfully mapped, expressed as the ratio between the known and full map size.
- Path length is computed as the cumulative distance traveled by all UAVs during the experiment, providing insight into the efficiency of motion planning.
- Overlap quantifies the redundant exploration performed by the swarm, calculated as the intersection of all UAV-explored regions.

From the analysis of these results, we observed several important trends:

1. Exploration time decreases as the number of UAVs increases, meaning that adding more agents leads to faster completion of the mapping task. However, increasing the density of obstacles extends the exploration time, as UAVs require more time to navigate through constrained spaces.
2. The total area explored remains relatively constant regardless of the number of UAVs, indicating that the entire reachable environment can be fully mapped with different swarm sizes.
3. Path length exhibits stability with respect to swarm size, meaning that adding more

UAVs does not necessarily lead to excessive travel distance. However, as obstacle density increases, the path length also increases, reflecting the need for longer detours and obstacle avoidance maneuvers. Additionally, the lowest observed path lengths tend to align with experiments where overlap is minimized, meaning that efficient task distribution reduces redundant travel.

4. Overlap increases with swarm size, as more UAVs in a confined space naturally lead to greater redundancy in coverage. However, obstacle density has little effect on overlap, suggesting that frontier allocation heuristics distribute tasks effectively regardless of environmental complexity.

These findings reinforce the effectiveness of using larger UAV swarms for rapid exploration while also highlighting the trade-offs associated with increasing obstacle density. The results further demonstrate the importance of designing robust allocation heuristics to balance the workload among agents, ensuring minimal path overlap while maintaining high exploration efficiency.

5.3.3 Real-World Experiments

To validate the proposed exploration strategy in real-world conditions, we conducted a series of experiments using a physical swarm of nano-UAVs. These tests aimed to assess the algorithm's performance when subjected to real-world uncertainties, such as sensor noise, communication delays, and dynamic environmental conditions. To systematically analyze the effects of different configurations, we varied swarm size, initial UAV positions, and obstacle placements across multiple trials.

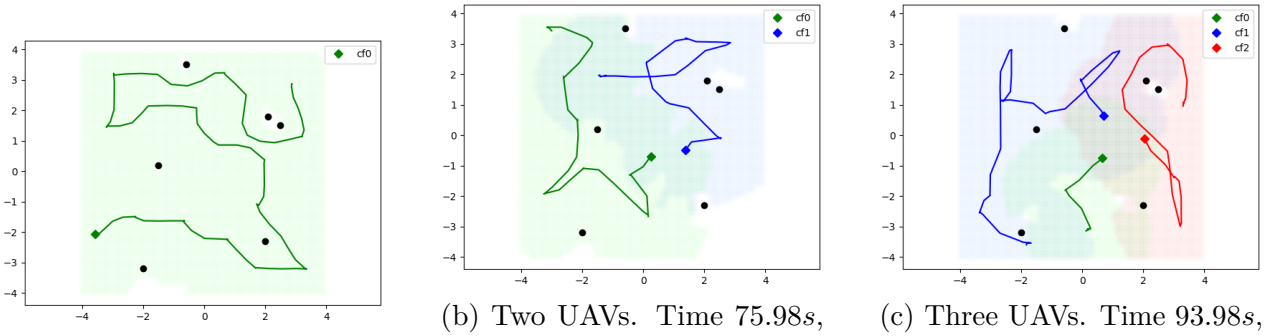
All real-world experiments were carried out within a constrained 8m $\times$ 8m indoor testbed, with an obstacle density of approximately 0.05 obs/m^2. Due to hardware and spatial limitations, we restricted swarm size to a maximum of three UAVs. The UAVs operated at a navigation speed of 0.75 m/s and a rotational speed of 0.1 rad/s, ensuring safe and controlled movements within the confined testing environment. Compared to the simulation setup, we reduced safety distances to account for the smaller Crazyflie UAV platform, setting values of 0.25 meters to obstacles and 1.0 meter to neighboring UAVs. Additionally, to accommodate the test area's limited dimensions, we clipped the multi-ranger deck's maximum sensing range to 2 meters, preventing excessive sensor readings beyond the available workspace. The complete set of experimental parameters is detailed in Table 5.5.

Figure 5.25 presents a top-down perspective of three real-world exploration experiments, each conducted with a different number of UAVs. The exploration algorithm can adapt to various swarm sizes, successfully completing the mission. Despite the physical constraints of the test area, the algorithm successfully enables UAVs to navigate and explore efficiently, completing the mission in all tested configurations.

Figure 5.26 depicts the evolution of the explored area over time for different swarm sizes. As expected, the exploration rate improves as more UAVs are introduced, reducing the time required to cover the environment. This trend aligns with observations from simulation experiments, reinforcing the algorithm's consistency across both simulated and real-world

Parameter	Value
Map resolution	0.1
Sensor range	2
Safety distance to obstacles	0.25
Safety distance to other UAVs	1.0
Navigation speed	0.75
Reached distance threshold	0.2
Spin speed	0.1
Yaw threshold	0.15
Minimum frontier size	1.5
Maximum frontier size	3.5

Table 5.5: Parameter configuration for real-world experiments.



(a) One UAV. Time 194.0s and path length 40.64m.

(b) Two UAVs. Time 75.98s, path length 37.38m and overlap 28.16%.

(c) Three UAVs. Time 93.98s, path length 47.22m and overlap 37.27%.

Figure 5.25: Real flight exploration experiments with different number of UAVs. Each colored line represents the trajectory followed by each UAV, while the shaded areas symbolizes the explored regions. Colored diamonds shows UAVs starting points and black dots represent the obstacles in the environment.

scenarios.

A detailed summary of the real-world experimental results is provided in Table 5.6, which includes key performance metrics such as the number of UAVs, exploration time, explored area, total path length, and overlap ratio. These metrics are defined similarly to those in the simulation results, allowing for a direct comparison between simulated and real-world performance.

The results obtained in real-world experiments exhibit a high degree of similarity to those observed in simulation, reinforcing the validity of our approach across different testing conditions. Notably, the fastest exploration time, shortest path length, and lowest overlap were recorded in the two-UAV experiments, indicating an optimal balance between collaboration and efficiency within the given environment.

However, in the three-UAV experiments, we observed a notable increase in exploration overlap, which impacted overall efficiency. The limited 8×8 m^2 test area imposed spatial constraints

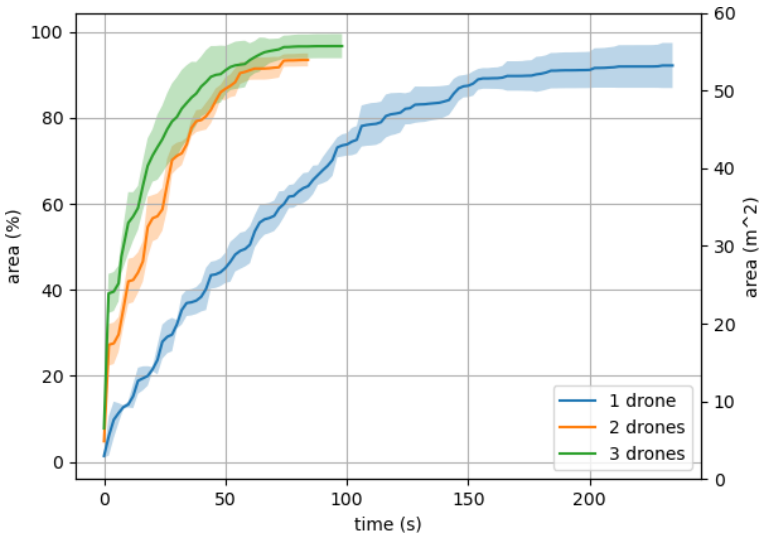


Figure 5.26: Average exploration rate for the real world experiments. The shaded area shows the standard deviation.

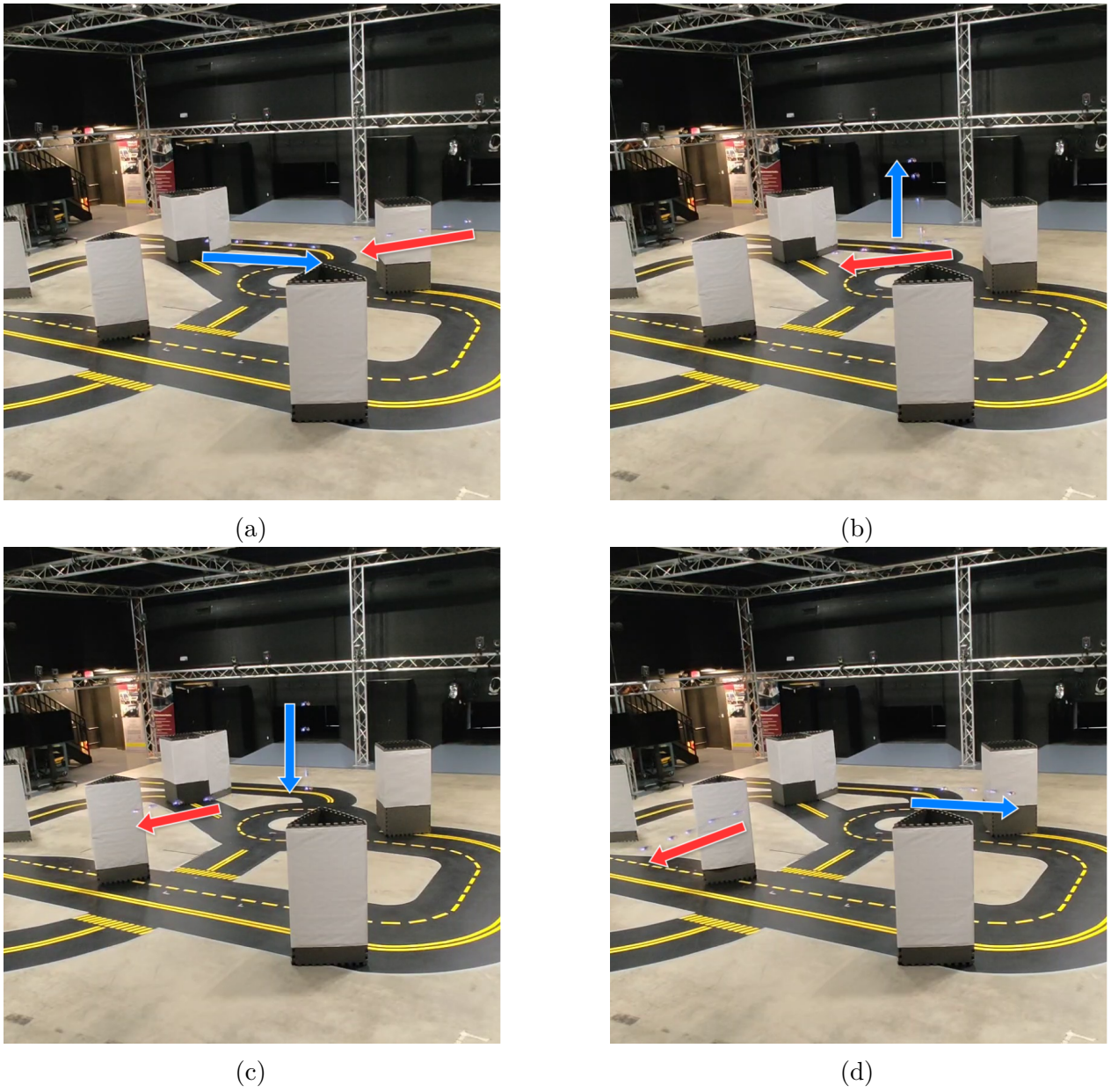
#	Time [s]	Area [%]	Path Length [m]	Overlap [%]
1	176.0 ± 31.13	92.16 ± 5.2	39.29 ± 8.48	-
2	**78.63 ± 3.77**	93.45 ± 1.5	**36.9 ± 2.42**	**30.69 ± 2.38**
3	79.25 ± 13.46	**96.67 ± 2.8**	47.55 ± 11.5	39.23 ± 11.0

Table 5.6: Results of the real world experiments according to the number of UAVs (#). We report the average and standard deviation for the completion time, percentage of area explored and total path traveled for the swarm. Best results are highlighted in bold.

that forced UAVs into redundant coverage, reducing the expected benefits of adding an additional agent. In a larger environment, our method would likely yield more significant improvements in both time efficiency and path length, as UAVs would have greater space to distribute their exploration efforts without excessive overlap.

During the real-world experiments, we also validated the proposed collision avoidance mechanism. Figure 5.27 illustrates a sequence of intra-swarm collision avoidance in action. As two UAVs approach each other, the system detects a potential collision risk (Figure 5.27a). In response, the navigation system of the left UAV is momentarily paused, and a safety maneuver is triggered. The UAV executes an altitude adjustment to avoid crossing paths with the second UAV (Figure 5.27b). Once a safe distance is reestablished, the avoidance maneuver concludes (Figure 5.27c), and the paused UAV resumes its navigation without compromising the exploration mission (Figure 5.27d). This mechanism ensures that multi-UAV exploration proceeds smoothly and safely, minimizing potential disruptions due to intra-swarm interactions.

These real-world experiments validate the feasibility and robustness of our exploration strategy in practical conditions. The results align closely with simulation outcomes, demonstrating the effectiveness of the approach across different swarm sizes. Despite the spatial limitations of the test environment, the system successfully managed navigation, obstacle avoidance, and



(a) (b)

(c) (d)

Figure 5.27: Intra-swarm collision avoidance sequence. Arrows represent each UAV motion. A potential crash sequence follows as: (a) A potential crash is detected; (b) drone navigator at the left is paused and the safety maneuver is started (blue), while the other UAV continues its mission (red); (c) safety maneuver ends and (d) the navigator resumes its operation.

intra-swarm coordination. The observed increase in overlap for larger swarms highlights the impact of constrained environments on multi-UAV exploration efficiency. However, these experiments confirm that our method scales well and is capable of operating reliably in real-world scenarios, paving the way for future deployments in larger and more complex environments.

5.4 Results Summary

The experimental results across the three scenarios validate the robustness, flexibility, and applicability of the proposed system in real-world missions. In the wind turbine inspection tasks, the UAV successfully executed dynamic maneuvers synchronized with the turbine's movement, demonstrating the system's ability to handle time-sensitive, safety-critical operations without human intervention. The solar park inspection experiments highlighted the scalability of the approach, with multiple UAVs executing coordinated behaviors in large structured environments, achieving consistent task execution and reliable plan adaptation at run-time. Lastly, the nano-UAV exploration scenario showcased the system's capacity to operate under strict hardware constraints. Our strategy allows complete exploration using just four single-beam range sensors per UAV. Additionally, safety is ensured via a collision avoidance algorithm that prevents the UAVs from colliding with each other by coordinating their behavior execution.

Together, these results confirm that the proposed execution and planning architecture effectively supports autonomous UAV operations in diverse and complex environments. The execution control system effectively translates mission plans into precise and reliable UAV behavior activations across different scenarios. The navigation system allows flying through cluttered, unstructured environments, while the exploration algorithm proposes a working methodology for minimal sensing capabilities. The system demonstrated high levels of autonomy, behavior modularity, and adaptability, meeting key quality attributes such as reliability, scalability, and reusability across different mission types and platforms.

Chapter 6

Discussion

Inspection and exploration missions may differ in their nature, but both presented experimental domains serve as rigorous validations of the proposed autonomy framework under distinct operational paradigms. The inspection tasks—focused on solar parks and wind turbines—demanded precise execution of pre-planned behavior sequences, synchronization with dynamic environmental elements (e.g., rotating nacelles), and strict safety margins in proximity to infrastructure. These scenarios validated the system's capacity for deterministic behavior execution, accurate trajectory tracking, and real-time adaptation to dynamic mission parameters.

Conversely, the exploration experiments—designed around swarms of nano-UAVs with minimal sensing capabilities—posed challenges centered on real-time environmental interaction, decentralized coordination, and reactive behavior planning. Here, the system's modular behavior-based architecture enabled dynamic plan generation and intra-swarm conflict resolution without centralized control, demonstrating autonomy in fully unknown environments.

The navigation components developed in this thesis were successfully tested across all presented experiments, demonstrating their reliability in real-world scenarios. It is important to highlight that exploration represents a higher-level task that builds upon the navigation layer by providing dynamically generated targets based on environmental perception. The fact that exploration tasks—particularly under minimal sensing conditions—were completed successfully confirms the navigation system's ability to handle dynamic target updates, obstacle avoidance, and trajectory planning with high reliability.

Crucially, across both mission types, the execution control system consistently ensured reliable activation of behaviors in alignment with high-level mission plans. The unified behavior interface supported flexibility in task definition and reusability across scenarios. The experiments also highlighted the system's adaptability, both in terms of responding to dynamic elements during execution and handling heterogeneous UAV configurations. These results affirm the framework's suitability for real-world autonomous missions, capable of supporting both deliberative and reactive planning under a common autonomy model.

While the previous analysis focused on the overall validation of the proposed autonomy

system across diverse mission profiles, several technical aspects merit further discussion. Each experimental scenario presents unique challenges and opportunities to analyze specific components of the architecture in greater depth.

The inspection experiments provide a suitable context to discuss behavior composition strategies and the multi-UAV deliberation scheme, particularly regarding synchronization, coordination, and flexibility in real-time mission execution. In contrast, the exploration experiments — especially those involving nano-UAVs with minimal sensing — highlight important insights into energy efficiency, agent workload distribution, and the effectiveness of the proposed frontier allocation heuristic.

The following sections explore these key aspects, drawing from selected experimental results to illustrate the system's design trade-offs, performance under constraints, and implications for future extensions.

The following sections explore these key aspects. Specifically, Section 6.1 discusses the advantages of modular behavior composition and its impact on development efficiency and mission reliability. Section 6.2 examines the deliberation schemes used for coordinating multi-UAV missions. Section 6.3 addresses optimal swarm sizes that contribute to more sustainable UAV operations during the exploration. Section 6.4 analyzes workload balancing among agents, and Section 6.5 presents the frontier allocation heuristic developed to optimize swarm exploration using minimal sensing.

6.1 Behavior Composition

The experiments presented in Chapter 5 validate the advantages of adopting a behavior-based architecture that supports hierarchical behavior composition. This design allows complex mission functionalities to be constructed by combining simpler, well-tested behavioral primitives.

For example, in inspection missions, the `RTL` behavior encapsulates a sequence of lower-level behaviors, including `GO_TO` and `LAND`. This abstraction improves mission plan readability and reduces operator cognitive load during mission supervision and validation. Similarly, the `PANEL_DIAGNOSE` behavior integrates more complex capabilities, such as `POINT_GIMBAL`, `TAKE_PHOTO`, and `TRIGGER_INFERENCE`. This composition not only encapsulates the required inspection workflow but also minimizes human errors in mission design and promotes execution consistency.

From a software engineering perspective, behavior composition significantly reduces the integration and testing effort. Reusable behaviors, once validated, can be seamlessly incorporated into new composed behaviors or repurposed for different mission contexts, such as transitioning between wind turbine and solar park inspections. This promotes code reuse, reduces redundancy, and accelerates development cycles by allowing developers to focus on higher-level coordination logic instead of reimplementing low-level functionalities. Moreover, highly tested behaviors increase reliability and ensure the safe execution of the missions.

Behavior composition also played a key role in the exploration experiments with nano-UAVs.

Although the behaviors in this scenario were simpler due to hardware constraints, the mission logic still benefited from composition. This layered structure allowed the swarm to maintain robust execution even in the presence of limited sensing capabilities.

In summary, behavior composition improves system modularity, simplifies mission planning, enhances reusability across domains, and reduces programming effort. Its application across both inspection and exploration missions demonstrates its flexibility and effectiveness as a foundational design principle for autonomous UAV systems.

6.2 Multi-UAV Deliberation Scheme

The scenarios presented two different deliberation schemes. First, during the wind turbine inspection, a decentralized approach was implemented. Mission planning and plan monitoring were distributed and executed onboard. Having plan monitoring onboard allowed triggering the shooting system precisely at the correct position and synchronized with the nacelle's rotation. However, only one UAV was utilized, limiting its decentralized applicability.

The second scenario proposed a centralized scheme for various UAVs. The centralized scheme was chosen to ease coordination by a human operator as a safety requirement. The deliberation system proved the execution of multiple UAVs synchronously. Moreover, the deliberation scheme worked reliably for real-time applications, even with the communication between GCS and UAVs.

The plan execution system was also utilized in other works in a different architecture. In [Luna et al., 2024], we presented a distributed-centralized architecture that performs the planning/re-planning and decision-making tasks during the control of the mission execution. Mission planning and supervision are centralized, but the plan is created and monitored distributively. The system was tested in a laboratory environment, showing promising results. A summary video of the work is available at https://vimeo.com/825491989.

6.3 Energy Efficiency

The results from both simulation and real-world experiments demonstrate the impact of swarm size on exploration performance. Notably, the largest swarm size achieved the shortest exploration times, highlighting the effectiveness of our proposed method in distributing the exploration workload. However, despite the advantage in time, larger swarms do not correspond to shorter individual path lengths. This discrepancy arises from path overlaps, where multiple UAVs unintentionally explore the same areas, leading to redundant coverage and increased travel distances.

Further analysis, considering the energy efficiency of the swarm, reveals that the optimal number of UAVs is influenced by the environment's characteristics, particularly its size and obstacle density. We estimate energy efficiency based on the total path length traveled by the swarm, as longer paths correlate with greater energy consumption. In simulation experiments, which involved larger and more complex environments with a higher density of obstacles, the analysis indicated that using three UAVs yielded the best balance between exploration

time and energy efficiency. Conversely, in real-world flight experiments—conducted in a comparatively smaller and less obstacle-dense environment—the optimal swarm size was found to be two UAVs. In this scenario, adding more UAVs did not produce meaningful time savings and instead led to inefficient coverage patterns and increased path overlap.

To validate this hypothesis on swarm efficiency, we conducted additional simulation experiments within a smaller $20 \times 20\ m^2$ environment with an obstacle density of $0.05\ obs/m^2$. These experiments aimed to further explore how swarm size affects both exploration time and energy consumption when the search area is more confined. By reducing the operational space, we intended to observe whether the patterns identified in larger environments, such as diminishing returns from increasing the swarm size, would persist under more constrained conditions. The simulation setup replicated the same exploration strategy and swarm coordination methods used in previous experiments to ensure consistency and comparability of results. Results obtained are listed in Table 6.1.

#	Time [s]	Area [%]	Path Length [m]	Overlap [%]
1	523.0 ± 43.65	**99.24 ± 0.02**	175.48 ± 13.35	-
2	226.34 ± 5.14	99.22 ± 0.01	**145.51 ± 1.72**	**24.91 ± 3.03**
3	169.15 ± 20.27	99.15 ± 0.12	150.16 ± 15.01	36.29 ± 6.43
5	**120.35 ± 13.95**	99.11 ± 0.2	161.63 ± 16.74	42.01 ± 5.19

Table 6.1: Results of the simulation experiments for a smaller environment. We report the average and standard deviation for the completion time, percentage of area explored and total path traveled for the swarm. Best results are highlighted in bold.

The results revealed that using two UAVs achieved the most favorable balance between exploration time and energy efficiency. Although larger swarms still completed the exploration faster, the marginal gains in speed were offset by a significant increase in overlapping trajectories, leading to longer average path lengths per UAV. These findings reinforce the conclusion that the optimal swarm size depends on the spatial characteristics of the environment. Specifically, as the environment becomes smaller, smaller swarms maintain better energy efficiency while achieving reasonably fast exploration times.

This experiment further substantiates our hypothesis: while larger swarms benefit expansive environments, smaller swarms are more effective in confined spaces. The results emphasize that the relationship between swarm size and performance is highly context-dependent, reinforcing the need for adaptive exploration strategies that consider environment dimensions when deploying multi-robot systems.

6.4 Agent Workload

The proposed heuristic effectively balances the workload among all agents in the swarm, ensuring an even distribution of both the area explored and the distance traveled, as demonstrated in Figure 5.24. This equalization of effort among agents is essential for optimizing the exploration rate since minimizing overlap between explored areas reduces redundant coverage and accelerates overall mission completion.

A detailed breakdown of each agent's contribution to the exploration process is presented in Table 6.2. Unlike previous tables, which reported global system metrics, these results focus on individual agent performance, providing insight into the workload distribution within the swarm. The values listed correspond to the 0.05 obs/m^2 experiment.

# of UAVs	Area / UAV [%]	Path Length / UAV [m]
1	98.98 ± 0.01	986.66 ± 24.51
2	62.37 ± 4.43	486.20 ± 17.56
3	39.13 ± 2.59	304.37 ± 9.43
5	28.23 ± 3.07	184.18 ± 6.51
7	**22.17 ± 2.64**	**148.19 ± 10.97**

Table 6.2: Impact of each UAV area explored and path traveled, mean and standard deviation, on the global algorithm for the 0.05 obs/m^2 density simulated scenario. Best results are highlighted in bold.

The results confirm that the proposed heuristic effectively addresses one of the key challenges in multi-agent exploration: maintaining both spatial and temporal workload parity. By achieving a low standard deviation in area coverage and path length, it indicates a near-uniform distribution of workload among all UAVs.

6.5 Frontier Allocation Heuristic

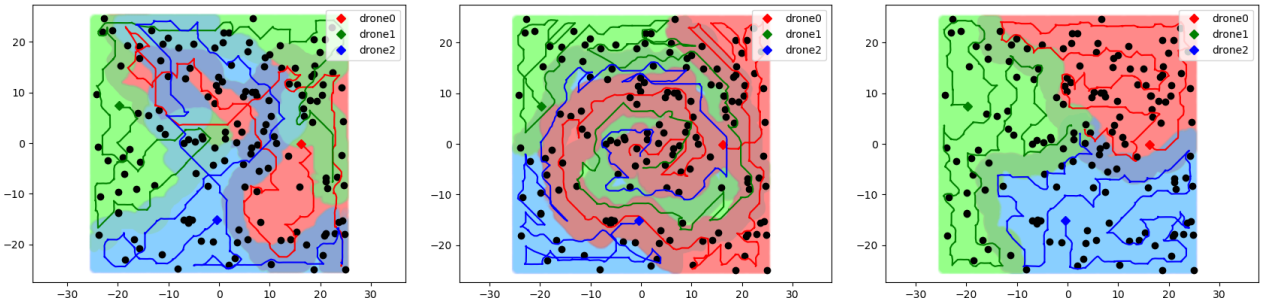
The impact of the proposed frontier allocation heuristic was analyzed by isolating the effect of each term in the optimization problem (Eq. 4.2). This analysis provided insights into the roles of the individual terms and their influence on the swarm's exploration behavior.

In the first scenario, the second term of the optimization problem—which promotes agent dispersion by maximizing the distance between agents—was removed. As a result, each UAV selected frontiers purely based on proximity, following a greedy approach that prioritized the nearest unexplored areas. This behavior led to inefficient exploration patterns, where agents frequently overlapped in their search areas, causing redundant coverage and reducing the overall exploration rate. The swarm exhibited erratic trajectories with no coordination, as each agent operated independently without considering the positions of its peers.

In contrast, the second scenario involved canceling the first term, which encourages agents to navigate toward the nearest frontiers. With this constraint removed, agents instead selected the most distant frontiers, maximizing their dispersion. Although this approach initially resulted in efficient area coverage due to the broad spread of agents, it ultimately caused an unintended pattern: as the mission progressed, agents gradually converged towards the map's center, forming a spiral-like trajectory.

Figure 6.1 visually compares these two behaviors, showing a top-down view of exploration patterns for each scenario. The figure highlights how the absence of each heuristic term affects swarm coordination and coverage efficiency. The results underline the importance of balancing both terms within the heuristic. The first term ensures that each agent efficiently covers

unexplored areas by selecting nearby frontiers, while the second term prevents excessive overlap by dispersing agents across the environment. Together, these terms create a cooperative behavior that combines local efficiency with global coverage, resulting in a more effective exploration strategy.



(a) Nearest frontier. Time 1030s, path length 1026.1m and overlap 34.48%.

(b) Maximum distance between UAVs. Time 1562s, path length 1329.73m, overlap 46.4%.

(c) Ours. Time 942s, path length 917.06m and overlap 14.18%.

Figure 6.1: Exploration experiments with different frontier allocation heuristic. Each colored line represents the trajectory followed by each UAV, while the shaded areas symbolizes the explored regions. Colored diamonds shows UAVs starting points and black dots represent the obstacles in the environment.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This dissertation set out to investigate how UAVs can operate more autonomously in complex, real-world environments. To that end, we addressed three main research questions focused on enabling autonomous navigation in complex environments (RQ1), high-level mission execution (RQ2), and exploration with limited sensing (RQ3). The results obtained through the various experiments—solar park inspections, wind turbine inspections, and minimal-sensing exploration—demonstrate clear progress toward these objectives and validate the contributions made throughout the thesis.

Regarding RQ1, the implementation of robust navigation capabilities through mapping and path planning components within a standardized aerial framework has proven effective. UAVs operated reliably in challenging environments, including cluttered unstructured scenarios and industrial facilities. These experiments validate Contribution C.1, as the system demonstrated adaptability to diverse scenarios, confirming the relevance of a unified and reusable navigation infrastructure.

RQ2 was addressed through the development of a dynamic execution control mechanism capable of handling complex missions composed of flexible, high-level behaviors. This was particularly evident in inspection scenarios, where tasks like diagnosing solar panels required tight coordination of actions such as pointing sensors, triggering inference modules, and reacting to dynamic mission conditions. The successful execution and adaptation to unforeseen changes confirm the value of Contribution C.2, enabling the system to perform reliably across multiple UAVs in distributed settings.

In response to RQ3, we proposed and evaluated a minimal-sensing exploration strategy tailored to nano-UAVs. This approach demonstrated effective frontier-based navigation using only four single-beam range sensors per UAV. Results across different simulation and real-world experiments confirm the effectiveness and scalability of this strategy, supporting Contribution C.3. Notably, the swarm maintained safe operation and efficient area coverage without relying on rich sensing, which is critical for lightweight, power-constrained platforms.

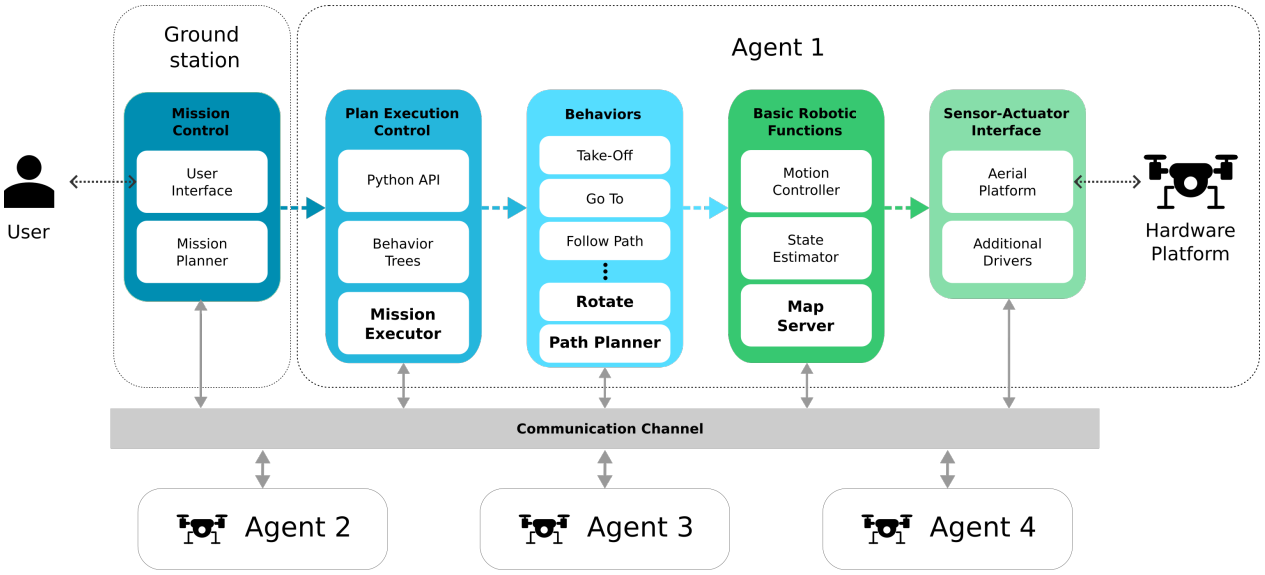


Figure 7.1: Aerostack2 structure with the new components highlighted in bold.

Across all three domains, the system was validated under both simulation and real-world conditions, contributing to C.4. Importantly, the execution of missions in industrial-grade settings such as solar farms and wind turbine fields demonstrates not only the robustness but also the practical applicability of the proposed methods.

All core contributions (C.1, C.2, and C.3) have been fully integrated into the Aerostack2 framework (C.5), a modular open-source platform that has benefited from this thesis's architectural decisions, implementation efforts, and long-term maintenance. Figure 7.1 illustrates Aerostack2's structure highlighting the contributions of this thesis. This integration extends Aerostack2's capabilities toward real-world autonomy and enables future research to build upon a solid and tested foundation.

Finally, through the open-source release of software components, datasets, and experiment configurations (C.6), this dissertation supports reproducibility and encourages adoption by both academic and industrial researchers. These contributions aim to strengthen the collaborative development of aerial robotics systems and move the field toward higher levels of autonomy, reliability, and practical relevance.

7.2 Future Work

While this dissertation presents significant advances toward autonomous UAV systems, several promising directions remain for future work, aligned with the core contributions of this research.

For C.1, which introduced a standardized framework for UAV navigation, future efforts will focus on expanding support for 3D planning and real-time obstacle avoidance in highly dynamic environments. While the current system handles structured and semi-structured environments effectively, more work is needed to integrate advanced perception capabilities

and probabilistic mapping to cope with uncertainty and incomplete information. Another potential improvement involves the use of semantic maps, allowing UAVs to reason about their environment with higher-level information, ultimately enabling more intelligent and context-aware navigation. Additionally, benchmarking the navigation stack under standardized testing conditions and integrating alternative planning paradigms (e.g., sampling-based, learning-based) could further validate and extend its robustness.

Building upon Contribution C.2, future work will integrate the mission execution system with symbolic planners such as ROSPlan [Cashmore et al., 2015] to enable fully automated high-level planning and reactive re-planning. This will elevate the system's autonomy by allowing UAVs to reason over mission goals and dynamically generate plans based on environmental feedback and task requirements. Incorporating ontology-based knowledge representation and reasoning modules, such as those inspired by the belief management in Aerostack1, will further enhance semantic understanding, context-awareness, and decision-making. These capabilities are essential for operating in more uncertain and unstructured scenarios, where real-time adaptability is key. In addition, tighter integration with multi-agent task allocation strategies will be explored to optimize collaborative planning in heterogeneous UAV teams.

As for C.3, which addressed exploration using minimal sensing on nano-UAVs, future work will aim at decentralizing the current algorithm to support fully distributed swarm behavior. Decentralization will require improved environmental representations, crucial for sharing information among the swarm while minimizing data transmission to reduce memory footprint, enabling the algorithm to run onboard. Exploration strategies will be further optimized to reduce redundant paths and overlap among UAVs, improving area coverage and efficiency. To improve real-world deployment readiness, upcoming work will extend the experimental validation to larger swarms and more diverse settings, particularly under noisy localization derived from onboard sensors rather than external motion capture systems.

It is our hope that the developments and open-source contributions presented in this work, along with these future extensions, will serve as a stepping stone for the broader aerial robotics community, accelerating progress toward truly autonomous, intelligent, and reliable UAV systems.

Appendices

Appendix A

International Competitions

During the course of this PhD, participation in several international robotics competitions provided valuable opportunities to validate the proposed algorithms and system architectures in realistic, high-pressure scenarios. These events posed complex challenges requiring autonomy, real-time decision-making, and multi-agent coordination. The following sections summarize each competition, the nature of its challenge, the technical approach taken, and the results achieved.

A.1 1st Place – ICUAS UAS Competition 2023, Poland

- **Challenge:** Develop a fully autonomous UAV system to execute an infrastructure inspection mission under time and operational constraints. The three benchmarks were requested, 3D exploration, defect detection, and onboard pose estimation, with performance assessed across simulation and live trials.
- **Approach:** A modular mission execution system was designed using a framework similar to Aerostack2, but in ROS 1. Three behaviors were implemented: Go Across, Search Target and Detect Cracks, which commanded a speed-to-position PID control.
- **Outcome:** Achieved 1st[1] place for delivering a reliable and high-performing system. The UAV executed the inspection task with strong autonomy and resilience to dynamic conditions.

A.2 Semifinalist (Accesit) – MBZIRC 2022, UAE

- **Challenge:** The MBZIRC 2022 competition focused on advancing collaborative aerial robotics through three main tasks: autonomous multi-UAV inspection of large structures, intra-swarm communication with decentralized decision-making, and cooperative transport of bulky objects. Teams were challenged to develop scalable, fully autonomous solutions integrating perception, coordination, and control under realistic outdoor GNSS-denied conditions.
- **Approach:** Our approach was built on top of Aerostack2, developing robust multi-UAV behaviors. We implemented distributed coordination strategies for collective inspection and transport, enabling each UAV to reason independently while maintaining swarm-level coherence. Aerostack2's integration with ROS 2 and its support for multi-agent mission planning allowed us to rapidly prototype, test, and deploy scalable solutions aligned with the MBZIRC 2023 challenge objectives.
- **Outcome:** Reached the semifinals[2] (Accesit) in a highly competitive international field. The system demonstrated effective multi-agent coordination and robustness in simulation.

[1] https://www.uasconferences.com/2023_icuas/uav-competition-final-results/
[2] https://www.mbzirc.com/grand-challenge#loop-9

A.3 Best Gates Passing Award – Nanocopter AI Challenge, IMAV 2022, Netherlands

- **Challenge:** Vision-based autonomous flight using Crazyflie nano-drones equipped with a single camera and onboard neural processing. Teams had to navigate unpredictable obstacle courses and pass through orange gates to maximize distance and score, all within a constrained arena and limited time.
- **Approach:** A lightweight perception and control pipeline using monocular vision and IMU data was implemented using Aerostack2. Real-time performance and accuracy were prioritized to ensure consistent gate transitions.
- **Outcome:** Received the "Best Gates Passing" award[3] for the most precise gate traversing among competitors.

A.4 1st Place – ICUAS UAS Competition 2022, Croatia

- **Challenge:** The competition simulated an urban firefighting mission where a UAV had to autonomously navigate through obstacle-dense environments, detect a fire target, and accurately deploy an extinguishing device. The challenge tested perception, navigation, and precision delivery under onboard computation constraints, using only an RGB-D sensor and relying on an external localization system.
- **Approach:** A modular mission execution system was designed using a framework similar to Aerostack2, but in ROS 1. Three behaviors were implemented: Go Across, Search Target and Throw Ball, which commanded a speed-to-position PID control.
- **Outcome:** Secured 1st place[4] by completing all mission objectives and deploying the extinguishing device three times. A summary video of our approach is available at https://vimeo.com/776655431.

[3] https://cvar-upm.github.io/misc/2022/09/20/imav22.html
[4] https://www.uasconferences.com/2022_icuas/winners-of-the-uav-competition/

Appendix B

Participation in R&D Projects

During the development of this PhD, the author has actively participated in several national and international research and development (R&D) projects. These projects have provided not only valuable technical and experimental frameworks for testing the methods proposed in this thesis but also contributed significantly to the alignment of academic research with real-world industrial and societal needs. Below is a summary of the most relevant initiatives:

B.1 SHEREC: Safe, healthy and environmental ship recycling - Horizon Europe

- **Role:** Team member
- **Funding Body:** Horizon Europe (European Commission)
- **ID / Ref. Number:** 10.3030/101136056
- **Duration:** 01/01/2024 - 31/12/2027
- **Total Budget:** 9.723.810 €

B.2 AEROGENIA: Sistema de Mantenimiento Integral de Aerogeneradores Mediante Gemelo Digital Apoyado Por Drones Autónomos e Imagen Multiespectral - National Plan

- **Role:** Team member
- **Funding Body:** Plan Estatal - Proyectos de colaboración público-privada (Ministerio de Ciencia e Innovación)
- **ID / Ref. Number:** CPP2022-009932
- **Duration:** 01/11/2023 - 31/10/2026
- **Total Budget:** 709.164 €

B.3 GENERADRON: Investigación avanzada de UAS en el ámbito de la categoría específica - CDTI

- **Role:** Team member
- **Funding Body:** Programa Tecnológico Aeronáutico (CDTI)
- **ID / Ref. Number:** PTAG-20221006
- **Duration:** 15/03/2023 - 31/12/2025
- **Total Budget:** 799.843 €

B.4 INSERTION: UAV percepción, control y operación en ambientes hostiles - National Plan

- **Role:** Team member
- **Funding Body:** Plan Estatal - Proyectos de Generación de Conocimiento (Ministerio de Ciencia e Innovación)
- **ID / Ref. Number:** PID2021-127648OB-C32
- **Duration:** 01/09/2022 - 31/08/2025
- **Total Budget:** 171.457 €

B.5 RATEC: Robot Aéreo+Terrestre conectado por cable para tareas de inspección y mantenimiento - National Plan

- **Role:** Team member
- **Funding Body:** Plan Estatal - Proyectos de Pruebas de Concepto (Ministerio de Ciencia e Innovación)
- **ID / Ref. Number:** PDC2022-133643-C21
- **Duration:** 01/12/2022 - 31/05/2025
- **Total Budget:** 69.000 €

B.6 COPILOT: Control, supervisión y operación optimizada de plantas fotovoltaicas mediante integración sinérgica de drones, IoT y tecnologías avanzadas de comunicaciones - Comunidad de Madrid

- **Role:** Team member
- **Funding Body:** Proyectos Sinérgicos de I+D en nuevas y emergentes áreas científicas (Comunidad de Madrid)
- **ID / Ref. Number:** Y2020/EMT-6368
- **Duration:** 01/07/2021 - 31/12/2024
- **Total Budget:** 584.430 €

Appendix C

Scientific Dissemination

C.1 Journal Publications

2024

Arias-Perez, P., Gautam, A., Fernandez-Cortizas, M., Perez-Saura, D., Saripalli, S., & Campoy, P. (2024). Exploring Unstructured Environments using Minimal Sensing on Cooperative Nano-Drones. *IEEE Robotics and Automation Letters*.
https://doi.org/10.1109/LRA.2024.3486212 [Arias-Perez et al., 2024]

Luna, M. A., Molina, M., Da-Silva-Gomez, R., Melero-Deza, J., **Arias-Perez, P.**, & Campoy, P. (2024). A multi-UAV system for coverage path planning applications with in-flight replanning capabilities. *Journal of Field Robotics*, 41(5), 1480-1497.
https://doi.org/10.1002/rob.22342 [Luna et al., 2024]

2023

Perez-Segui, R., **Arias-Perez, P.**, Melero-Deza, J., Fernandez-Cortizas, M., Perez-Saura, D., & Campoy, P. (2023). Bridging the gap between simulation and real autonomous UAV flights in industrial applications. *Aerospace*, 10(9), 814.
https://doi.org/10.3390/aerospace10090814 [Perez-Segui, Arias-Perez, et al., 2023]

Rutinowski, J., Schuster, D., Kauffmann, S., **Arias-Pérez, P.**, Polachowski, F., Granero, M., Roidl, M., & Campoy, P. (2023). Exploring the Re-Identification of Industrial Entities on Autonomous Guided Vehicles. *Logistics Journal: referierte Veröffentlichungen*, 2023(1).
https://doi.org/10.2195/lj_proc_rutinowski_en_202310_01 [Rutinowski et al., 2023]

Perez-Saura, D., Fernandez-Cortizas, M., Perez-Segui, R., **Arias-Perez, P.**, & Campoy, P. (2023). Urban firefighting drones: Precise throwing from UAV. *Journal of Intelligent & Robotic Systems*, 108(4), 66.
https://doi.org/10.1007/s10846-023-01883-6 [Perez-Saura et al., 2023]

Awasthi, S., Fernandez-Cortizas, M., Reining, C., **Arias-Perez, P.**, Luna, M. A., Perez-Saura, D., Roidl, M., Gramse, N., Klokowski, P., & Campoy, P. (2023). Micro UAV swarm for industrial applications in indoor environment: a systematic literature review. *Logistics Research*, 16(1), 1-43.
https://doi.org/10.23773/2023_11 [Awasthi et al., 2023]

2022

Arias-Perez, P., Fernández-Conde, J., Martin Gomez, D., Cañas, J. M., & Campoy, P. (2022). A middleware infrastructure for programming vision-based applications in uavs. *Drones*, 6(11), 369.
https://doi.org/10.3390/drones6110369 [Arias-Perez et al., 2022]

C.2 Peer-reviewed Conference Publications

2024

Mejias, L., **Arias-Perez, P.**, Melero-Deza, J., & Campoy, P. (2024, June). A Virtual Spring-Damper Approach for UAV Swarm Formation and Decentralised Collision Avoidance. *In 2024*

International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 579-585). IEEE. https://doi.org/10.1109/ICUAS60882.2024.10556825 [Mejias et al., 2024]

Melero-Deza, J., Perez-Segui, R., **Arias-Perez, P.**, Fernandez-Cortizas, M., Perez-Saura, D., Guillermo, G. L., Tradacete-Agreda, M., Pérez, C. S., Sánchez, F. J. R., & Campoy, P. (2024) Photovoltaic plant monitoring and inspection through synergic integration of UAVs and IoT. International Micro Air Vehicle Conference, 184–191. https://www.imavs.org/papers/2024/22.pdf [Melero-Deza et al., 2024]

2023

Perez-Segui R., Melero-Deza J., **Arias-Perez P.**, Perez-Saura D., Fernandez-Cortizas M., Campoy P. (2023). Pruebas con simulación guiada para robots aéreos autónomos. *Jornadas Nacionales de Robótica y Bioingeniería 2023*, 179-184. https://doi.org/10.20868/UPM.book.74896 [Perez-Segui, Melero-Deza, et al., 2023]

C.3 Manuscripts Under Peer-Review (April 2025)

2025

Melero-Deza, J., **Arias-Perez, P.**, GP-Lenza, G., Molina, M., & Campoy, P. (2025) Learning Efficient Frontier Selection with Cross-Attention Policy for Exploring Unstructured Environments *2025 Conference on Robot Learning.*

DR.Pita-Romero, C., **Arias-Perez, P.**, Fernandez-Cortizas, M., Perez-Segui, R., & Campoy P. (2025). Flocking behavior for dynamic and complex swarm structures *In 2025 International Conference on Unmanned Aircraft Systems (ICUAS).* IEEE.

C.4 Preprints

2023

Fernandez-Cortizas, M., Molina, M., **Arias-Perez, P.**, Perez-Segui, R., Perez-Saura, D., & Campoy, P. (2023). Aerostack2: A software framework for developing multi-robot aerial systems. *arXiv preprint* arXiv:2303.18237. [Fernandez-Cortizas et al., 2023]

C.5 Workshop Publications

2023

Perez-Segui, R., **Arias-Perez, P.**, Melero-Deza, J., Perez-Saura, D., Fernandez-Cortizas, M., & Campoy, P. Simulation-Guided Testing for Autonomous Aerial Robotics Applications. *The Role of Robotics Simulators for Unmanned Aerial Vehicles, ICRA 2023* https://imrclab.github.io/workshop-uav-sims-icra2023/papers/RS4UAVs_paper_13.pdf

C.6 Other Communication Activities

2024

Arias-Perez, P. (2024) Exploración de entornos no estructurados usando sensores mínimos en nano-drones cooperativos. *ROSConference Spain 2024*. Open Source Robotics Foundation. Oral presentation.
https://vimeo.com/1029492555

Arias-Perez, P.; Perez-Segui, R. (2024) Aerostack2, desarrolla tu enjambre de drones desde simulación a real. *ROSConference Spain 2024*. Open Source Robotics Foundation. Workshop.
https://github.com/aerostack2/demo_ROSConES24

Arias-Perez, P.; Perez-Segui, R. (2024) From Gazebo simulation to wind turbine inspections powered by Aerostack2. *Gazebo Community Meeting*. Open Source Robotics Foundation. Online oral presentation.
https://vimeo.com/908361057

Fernandez-Cortizas, M.; **Arias-Perez, P.**; Pérez-Segui, R.; Perez-Saura, D.; Molina, M.; Campoy, P. (2024) Aerostack2: A framework for developing Multi-Robot Aerial Systems. *ROS Conference*. Open Source Robotics Foundation. Oral presentation.
https://vimeo.com/879000655

Bibliography

Arias-Perez, P., Fernández-Conde, J., Martin Gomez, D., Cañas, J. M., & Campoy, P. (2022). A middleware infrastructure for programming vision-based applications in uavs. *Drones*, *6*(11), 369.

Arias-Perez, P., Gautam, A., Fernandez-Cortizas, M., Perez-Saura, D., Saripalli, S., & Campoy, P. (2024). Exploring unstructured environments using minimal sensing on cooperative nano-drones. *IEEE Robotics and Automation Letters*.

Arkin, R. C. (1998). *Behavior-based robotics*. MIT press.

Awasthi, S., Fernandez-Cortizas, M., Reining, C., Arias-Perez, P., Luna, M. A., Perez-Saura, D., Roidl, M., Gramse, N., Klokowski, P., & Campoy, P. (2023). Micro uav swarm for industrial applications in indoor environment: A systematic literature review. *Logistics Research*, *16*(1), 1–43.

Azevedo, C., Matos, A., Lima, P. U., & Avendaño, J. (2021). Petri net toolbox for multi-robot planning under uncertainty. *Applied Sciences*, *11*(24). https://doi.org/10.3390/app112412087

Baca, T., Petrlik, M., Vrba, M., Spurny, V., Penicka, R., Hert, D., & Saska, M. (2021). The mrs uav system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles. *Journal of Intelligent & Robotic Systems*, *102*(1), 26.

Bahill, A., & Gissing, B. (1998). Re-evaluating systems engineering concepts using systems thinking. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *28*(4), 516–527. https://doi.org/10.1109/5326.725338

Bartolomei, L., Teixeira, L., & Chli, M. (2023). Fast multi-uav decentralized exploration of forests. *IEEE Robotics and Automation Letters*, *8*(9), 5576–5583. https://doi.org/10.1109/LRA.2023.3296037

Beetz, M., Mösenlechner, L., & Tenorth, M. (2010). Cram—a cognitive robot abstract machine for everyday manipulation in human environments. *2010 IEEE/RSJ international conference on intelligent robots and systems*, 1012–1017.

Blanco, J., Gonzalez, J., & Fernandez-Madrigal, J. (2006). Consistent observation grouping for generating metric-topological maps that improves robot localization. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 818–823. https://doi.org/10.1109/ROBOT.2006.1641810

Bohren, J., & Cousins, S. (2010). The smach high-level executive [ros news]. *IEEE Robotics & Automation Magazine*, *17*(4), 18–20. https://doi.org/10.1109/MRA.2010.938836

Bonasso, R. P., & Kortenkamp, D. (1995). Characterizing an architecture for intelligent, reactive agents. *Reactive Agents, AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*.

Bonato, S., Lambertenghi, S. C., Cereda, E., Giusti, A., & Palossi, D. (2023). Ultra-low power deep learning-based monocular relative localization onboard nano-quadrotors. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 3411–3417. https://doi.org/10.1109/ICRA48891.2023.10161127

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, *2*(1), 14–23.

Brunner, S. G., Steinmetz, F., Belder, R., & Dömel, A. (2016). Rafcon: A graphical tool for engineering complex, robotic tasks. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3283–3290.

Burgard, W., Moors, M., Stachniss, C., & Schneider, F. (2005). Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, *21*(3), 376–386. https://doi.org/10.1109/TRO.2004.839232

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., & Carreras, M. (2015). Rosplan: Planning in the robot operating system. *Proceedings of the international conference on automated planning and scheduling*, *25*, 333–341.

Coleman, D., Sucan, I., Chitta, S., & Correll, N. (2014). Reducing the barrier to entry of complex robotic software: A moveit! case study. *arXiv preprint arXiv:1404.3785*.

Colledanchise, M., & Natale, L. (2021). On the implementation of behavior trees in robotics. *IEEE Robotics and Automation Letters*, *6*(3), 5929–5936. https://doi.org/10.1109/LRA.2021.3087442

Colledanchise, M., & Ögren, P. (2016). How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics*, *33*(2), 372–389.

Colledanchise, M., & Ögren, P. (2018). *Behavior trees in robotics and AI: An introduction*. CRC Press. https://doi.org/10.1201/9780429489105

Cramariuc, A., Bernreiter, L., Tschopp, F., Fehr, M., Reijgwart, V., Nieto, J., Siegwart, R., & Cadena, C. (2023). maplab 2.0 – A Modular and Multi-Modal Mapping Framework. *IEEE Robotics and Automation Letters*, *8*(2), 520–527. https://doi.org/10.1109/LRA.2022.3227865

Daud, S. M. S. M., Yusof, M. Y. P. M., Heo, C. C., Khoo, L. S., Singh, M. K. C., Mahmood, M. S., & Nawawi, H. (2022). Applications of drone in disaster management: A scoping review. *Science & Justice*, *62*(1), 30–42.

Duisterhof, B. P., Li, S., Burgués, J., Reddi, V. J., & de Croon, G. C. (2021). Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9099–9106. https://doi.org/10.1109/IROS51168.2021.9636217

Ebeid, E., Skriver, M., Terkildsen, K. H., Jensen, K., & Schultz, U. P. (2018). A survey of open-source uav flight controllers and flight simulators. *Microprocessors and Microsystems*, *61*, 11–20.

Ebert, S., Mey, J., Schöne, R., Götz, S., & Aßmann, U. (2024). Distributed petri nets for model-driven verifiable robotic applications in ros. *Innovations in Systems and Software Engineering*, *20*(4), 531–557.

Fankhauser, P., & Hutter, M. (2016). A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation. In A. Koubaa (Ed.), *Robot operating system (ros) – the complete reference (volume 1)*. Springer. https://doi.org/10.1007/978-3-319-26054-9{_}5

Fernandez-Cortizas, M., Molina, M., Arias-Perez, P., Perez-Segui, R., Perez-Saura, D., & Campoy, P. (2023). Aerostack2: A software framework for developing multi-robot aerial systems. *arXiv preprint arXiv:2303.18237*.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. *AAAI*, *87*, 202–206.

Floreano, D., & Wood, R. J. (2015). Science, technology and the future of small autonomous drones. *nature*, *521*(7553), 460–466.

Foehn, P., Kaufmann, E., Romero, A., Penicka, R., Sun, S., Bauersfeld, L., Laengle, T., Cioffi, G., Song, Y., Loquercio, A., et al. (2022). Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Science Robotics*, *7*(67), eabl6259.

Forsberg, K., & and, H. M. (1992). The relationship of systems engineering to the project cycle. *Engineering Management Journal*, *4*(3), 36–43. https://doi.org/10.1080/10429247.1992.11414684

Fox, M., De Raedt, L., Ingrand, F., & Ghallab, M. (2014). Robotics and artificial intelligence: A perspective on deliberation functions. *AI communications*, *27*(1), 63–80.

Furrer, F., Burri, M., Achtelik, M., & Siegwart, R. (2016). Robot operating system (ros): The complete reference (volume 1). In A. Koubaa (Ed.). Springer International Publishing. https://doi.org/10.1007/978-3-319-26054-9_23

Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, *23*(1), 34–46.

Guzzoni, D., Cheyer, A., Julia, L., & Konolige, K. (1997). Many robots make short work: Report of the sri international mobile robot team. *AI Magazine*, *18*(1), 55. https://doi.org/10.1609/aimag.v18i1.1274

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, *8*(3), 231–274.

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-time loop closure in 2d lidar slam. *2016 IEEE international conference on robotics and automation (ICRA)*, 1271–1278.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees [Software available at https://octomap.github.io]. *Autonomous Robots*. https://doi.org/10.1007/s10514-012-9321-0

Huang, H.-M. (2004, September). Autonomy levels for unmanned systems (alfus) framework volume i: Terminology version 2.0. https://doi.org/https://doi.org/10.6028/NIST.sp.1011-I-2.0

Hui, Y., Zhang, X., Shen, H., Lu, H., & Tian, B. (2024). Dppm: Decentralized exploration planning for multi-uav systems using lightweight information structure. *IEEE Transactions on Intelligent Vehicles*, *9*(1), 613–625. https://doi.org/10.1109/TIV.2023.3322705

Hüppi, M., Bartolomei, L., Mascaro, R., & Chli, M. (2022). T-prm: Temporal probabilistic roadmap for path planning in dynamic environments. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10320–10327. https://doi.org/10.1109/IROS47612.2022.9981739

Ingrand, F., & Ghallab, M. (2017). Deliberation for autonomous robots: A survey. *Artificial Intelligence*, *247*, 10–44.

Jaisinghani, D., Rehman, T. U., Mulkey, R., & Berns, A. (2023). Iot in the air: Thread-enabled flying iot network for indoor environments. *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 142–147. https://doi.org/10.1109/PerComWorkshops56833.2023.10150308

Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y., & Azumi, T. (2018). Autoware on board: Enabling autonomous vehicles with embedded systems. *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 287–296.

Kim, J., Kim, S., Ju, C., & Son, H. I. (2019). Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. *Ieee Access*, *7*, 105100–105115.

Klotzbücher, M., & Bruyninckx, H. (2012). Coordinating robotic tasks and systems with rfsm statecharts. *Journal of Software Engineering for Robotics*, *3*(1), 28–56.

Kondo, K., Figueroa, R., Rached, J., Tordesillas, J., Lusk, P. C., & How, J. P. (2023). Robust mader: Decentralized multiagent trajectory planner robust to communication delay in dynamic environments. *IEEE Robotics and Automation Letters*.

Kortenkamp, D., Simmons, R., & Brugali, D. (2016). Robotic systems architectures and programming. *Springer handbook of robotics*, 283–306.

Labbé, M., & Michaud, F. (2019). Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, *36*(2), 416–446. https://doi.org/https://doi.org/10.1002/rob.21831

Lamberti, L., Bompani, L., Kartsch, V. J., Rusci, M., Palossi, D., & Benini, L. (2023). Bio-inspired autonomous exploration policies with cnn-based object detection on nano-drones. *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–6. https://doi.org/10.23919/DATE56975.2023.10137154

Liu, Y., Meng, Z., Zou, Y., & Cao, M. (2021). Visual object tracking and servoing control of a nano-scale quadrotor: System, algorithms, and experiments. *IEEE/CAA Journal of Automatica Sinica*, *8*(2), 344–360. https://doi.org/10.1109/JAS.2020.1003530

Luna, M. A., Molina, M., Da-Silva-Gomez, R., Melero-Deza, J., Arias-Perez, P., & Campoy, P. (2024). A multi-uav system for coverage path planning applications with in-flight re-planning capabilities. *Journal of Field Robotics*, *41*(5), 1480–1497.

Macenski, S., Booker, M., & Wallace, J. (2024). Open-source, cost-aware kinematically feasible planning for mobile and surface robotics. *Arxiv*.

Macenski, S., Martín, F., White, R., & Ginés Clavero, J. (2020). The marathon 2: A navigation system. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. https://github.com/ros-planning/navigation2

Magid, E., & Rivlin, E. (2004). Cautiousbug: A competitive algorithm for sensory-based robot navigation. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, *3*, 2757–2762. https://doi.org/10.1109/IROS.2004.1389826

Mannucci, A., Nardi, S., & Pallottino, L. (2018). Autonomous 3d exploration of large areas: A cooperative frontier-based approach. *Modelling and Simulation for Autonomous Systems*, *10756*, 18–39.

Markovic, L., Petric, F., Ivanovic, A., Goricanec, J., Car, M., Orsag, M., & Bogdan, S. (2023). Towards a standardized aerial platform: Icuas'22 firefighting competition. *Journal of intelligent & robotic systems*, *108*(3), 52.

Marzinotto, A., Colledanchise, M., Smith, C., & Ögren, P. (2014). Towards a unified behavior trees framework for robot control. *2014 IEEE international conference on robotics and automation (ICRA)*, 5420–5427.

McGuire, K. N., Wagter, C. D., Tuyls, K., Kappen, H. J., & de Croon, G. C. (2019). Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, *4*(35), eaaw9710. https://doi.org/10.1126/scirobotics.aaw9710

Mejias, L., Arias-Perez, P., Melero-Deza, J., & Campoy, P. (2024). A virtual spring-damper approach for uav swarm formation and decentralised collision avoidance. *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, 579–585.

Melero-Deza, J., Perez-Segui, R., Arias-Perez, P., Fernandez-Cortizas, M., Perez-Saura, D., Guillermo, G.-L., Tradacete-Agreda, M., Pérez, C. S., Sánchez, F. J. R., & Campoy, P. (2024). Photovoltaic plant monitoring and inspection through synergic integration of uavs and iot. *International Micro Air Vehicle Conference*, 184–191.

Michaud, F., & Nicolescu, M. (2016). Behavior-based systems. *Springer handbook of robotics*, 307–328.

Ministerio de Ciencia e Innovación. (2021). Spanish science, technology and innovation strategy 2021-2027 [Online; accessed 2 april 2025]. https://www.ciencia.gob.es/InfoGeneralPortal/documento/1f4e85ac-9e50-49b4-a978-6c15a5195c88

Ministerio de Fomento. (2018). Plan estratégico para el desarrollo del sector civil de los drones en españa 2018-2021 [Online; accessed 2 april 2025]. https://www.transportes.gob.es/recursos_mfom/paginabasica/recursos/plan_estrategico_drones_2018-2021_0.pdf

Mohta, K., Watterson, M., Mulgaonkar, Y., Liu, S., Qu, C., Makineni, A., Saulnier, K., Sun, K., Zhu, A., Delmerico, J., Thakur, D., Karydis, K., Atanasov, N., Loianno, G., Scaramuzza, D., Daniilidis, K., Taylor, C. J., & Kumar, V. (2018). Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics*, *35*(1), 101–120.

Molina, M., Camporredondo, A., Bavle, H., Rodriguez-Ramos, A., & Campoy, P. (2019). An execution control method for the aerostack aerial robotics framework. *Frontiers of Information Technology & Electronic Engineering*, *20*(1), 60–75.

Molina, M., Carrera, A., Camporredondo, A., Bavle, H., Rodriguez-Ramos, A., & Campoy, P. (2020). Building the executive system of autonomous aerial robots using the aerostack open-source framework. *International Journal of Advanced Robotic Systems*, *17*(3), 1729881420925000.

Moravec, H. P. (1988). Sensor fusion in certainty grids for mobile robots. *AI Magazine*, *9*(2), 61. https://doi.org/10.1609/aimag.v9i2.676

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Myhill, J. (1957). Finite automata and the representation of events. *WADD Technical Report*, *57*, 112–137.

Nguyen, H., Ciocarlie, M., Hsiao, K., & Kemp, C. C. (2013). Ros commander (rosco): Behavior creation for home robots. *2013 IEEE International Conference on Robotics and Automation*, 467–474.

Niculescu, V., Polonelli, T., Magno, M., & Benini, L. (2024). Nanoslam: Enabling fully onboard slam for tiny robots. *IEEE Internet of Things Journal*, *11*(8), 13584–13607. https://doi.org/10.1109/JIOT.2023.3339254

Nooralishahi, P., Ibarra-Castanedo, C., Deane, S., López, F., Pant, S., Genest, M., Avdelidis, N. P., & Maldague, X. P. (2021). Drone-based non-destructive inspection of industrial sites: A review and case studies. *Drones*, *5*(4), 106.

Oleynikova, H., Taylor, Z., Siegwart, R., & Nieto, J. (2018). Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robotics and Automation Letters*, *3*(3), 1474–1481. https://doi.org/10.1109/LRA.2018.2800109

Penicka, R., Song, Y., Kaufmann, E., & Scaramuzza, D. (2022). Learning minimum-time flight in cluttered environments. *IEEE Robotics and Automation Letters*, *7*(3), 7209–7216. https://doi.org/10.1109/LRA.2022.3181755

Perez-Saura, D., Fernandez-Cortizas, M., Perez-Segui, R., Arias-Perez, P., & Campoy, P. (2023). Urban firefighting drones: Precise throwing from uav. *Journal of Intelligent & Robotic Systems*, *108*(4), 66.

Perez-Segui, R., Arias-Perez, P., Melero-Deza, J., Fernandez-Cortizas, M., Perez-Saura, D., & Campoy, P. (2023). Bridging the gap between simulation and real autonomous uav flights in industrial applications. *Aerospace*, *10*(9), 814.

Perez-Segui, R., Melero-Deza, J., Arias-Perez, P., Perez-Saura, D., Fernandez-Cortizas, M., & Campoy, P. (2023). Pruebas con simulación guiada para robots aéreos autónomos. *Jornadas Nacionales de Robótica y Bioingeniería 2023*, 179–184. https://doi.org/10.20868/UPM.book.74896

Petri, C. A. (1962). *Kommunikation mit automaten* [Ph.D. Thesis]. University of Bonn.

Pichierri, L., Testa, A., & Notarstefano, G. (2023). Crazychoir: Flying swarms of crazyflie quadrotors in ros 2. *IEEE Robotics and Automation Letters*, *8*(8), 4713–4720.

Pourjabar, M., Rusci, M., Bompani, L., Lamberti, L., Niculescu, V., Palossi, D., & Benini, L. (2023). Multi-sensory anti-collision design for autonomous nano-swarm exploration. *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 1–5. https://doi.org/10.1109/ICECS58634.2023.10382769

Preiss, J. A., Honig, W., Sukhatme, G. S., & Ayanian, N. (2017). Crazyswarm: A large nano-quadcopter swarm. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 3299–3304.

Rabin, M. O., & Scott, D. (1959). Finite automata and their decision problems. *IBM journal of research and development*, *3*(2), 114–125.

Radoglou-Grammatikis, P., Sarigiannidis, P., Lagkas, T., & Moscholios, I. (2020). A compilation of uav applications for precision agriculture. *Computer Networks*, *172*, 107148.

Real, F., Torres-González, A., Soria, P. R., Capitán, J., & Ollero, A. (2020). Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles. *International Journal of Advanced Robotic Systems*, *17*(4), 1–13. https://doi.org/10.1177/1729881420925011

Reijgwart, V., Millane, A., Oleynikova, H., Siegwart, R., Cadena, C., & Nieto, J. (2020). Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps. *IEEE Robotics and Automation Letters*.

Rejeb, A., Rejeb, K., Simske, S. J., & Treiblmaier, H. (2023). Drones for supply chain management and logistics: A review and research agenda. *International Journal of Logistics Research and Applications*, *26*(6), 708–731.

Romero, A., Penicka, R., & Scaramuzza, D. (2022). Time-optimal online replanning for agile quadrotor flight. *IEEE Robotics and Automation Letters*, *7*(3), 7730–7737. https://doi.org/10.1109/LRA.2022.3185772

Rösmann, C., Hoffmann, F., & Bertram, T. (2017). Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, *88*, 142–153. https://doi.org/https://doi.org/10.1016/j.robot.2016.11.007

Rovida, F., Crosby, M., Holz, D., Polydoros, A. S., Großmann, B., Petrick, R. P., & Krüger, V. (2017). Skiros—a skill-based robot control platform on top of ros. In *Robot operating system (ros) the complete reference (volume 2)* (pp. 121–160). Springer.

Rutinowski, J., Schuster, D., Kauffmann, S., Arias-Pérez, P., Polachowski, F., Granero, M., Roidl, M., & Campoy, P. (2023). Exploring the re-identification of industrial entities on autonomous guided vehicles. *Logistics Journal: referierte Veröffentlichungen*, *2023*(1).

Saeedi, S., Trentini, M., Seto, M., & Li, H. (2016). Multiple-robot simultaneous localization and mapping: A review. *Journal of Field Robotics*, *33*(1), 3–46.

Sanchez-Lopez, J. L., Fernández, R. A. S., Bavle, H., Sampedro, C., Molina, M., Pestana, J., & Campoy, P. (2016). Aerostack: An architecture and open-source software framework for aerial robotics. *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 332–341.

Schillinger, P., Kohlbrecher, S., & Von Stryk, O. (2016). Human-robot collaborative high-level control with application to rescue robotics. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2796–2802.

Song, Y., Shi, K., Penicka, R., & Scaramuzza, D. (2023). Learning perception-aware agile flight in cluttered environments. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 1989–1995. https://doi.org/10.1109/ICRA48891.2023.10160563

Şucan, I. A., Moll, M., & Kavraki, L. E. (2012). The Open Motion Planning Library [https://ompl.kavrakilab.org]. *IEEE Robotics & Automation Magazine*, *19*(4), 72–82. https://doi.org/10.1109/MRA.2012.2205651

Teissing, K., Novosad, M., Penicka, R., & Saska, M. (2024). Real-time planning of minimum-time trajectories for agile uav flight. *IEEE Robotics and Automation Letters*.

Tenorth, M., & Beetz, M. (2013). Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research*, *32*(5), 566–590.

Thrun, S., Thayer, S., Whittaker, W., Baker, C., Burgard, W., Ferguson, D., Hahnel, D., Montemerlo, D., Morris, A., Omohundro, Z., Reverte, C., & W, W. (2004). Autonomous exploration and mapping of abandoned mines. *IEEE Robotics and Automation Magazine*, *11*(4), 79–91. https://doi.org/10.1109/MRA.2004.1371614

Tian, Y., Chang, Y., Herrera Arias, F., Nieto-Granda, C., How, J. P., & Carlone, L. (2022). Kimera-multi: Robust, distributed, dense metric-semantic slam for multi-robot systems.

IEEE Transactions on Robotics, *38*(4), 2022–2038. https://doi.org/10.1109/TRO.2021.3137751

Tordesillas, J., & How, J. P. (2021). Mader: Trajectory planner in multiagent and dynamic environments. *IEEE Transactions on Robotics*, *38*(1), 463–476.

Tordesillas, J., & How, J. P. (2022). Panther: Perception-aware trajectory planner in dynamic environments. *IEEE Access*, *10*, 22662–22677. https://doi.org/10.1109/ACCESS.2022.3154037

Tordesillas, J., & How, J. P. (2023). Deep-panther: Learning-based perception-aware trajectory planner in dynamic environments. *IEEE Robotics and Automation Letters*, *8*(3), 1399–1406. https://doi.org/10.1109/LRA.2023.3235678

Tordesillas, J., Lopez, B. T., Everett, M., & How, J. P. (2022). Faster: Fast and safe trajectory planner for navigation in unknown environments. *IEEE Transactions on Robotics*, *38*(2), 922–938. https://doi.org/10.1109/TRO.2021.3100142

Tradacete-Ágreda, M., Santos-Pérez, C., Hueros-Barrios, P. J., Rodríguez-Sánchez, F. J., Pérez-Seguí, R., Melero-Deza, J., & Campoy, P. (2025). Framework for autonomous inspection of pv plants using iot electronics on each pv panel and uav collaboration. *Energy Conversion and Management: X*, 100878.

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., & Das, H. (2001). The claraty architecture for robotic autonomy. *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, *1*, 1–121.

Xiao, K., Tan, S., Wang, G., An, X., Wang, X., & Wang, X. (2020). Xtdrone: A customizable multi-rotor uavs simulation platform. *2020 4th International Conference on Robotics and Automation Sciences (ICRAS)*, 55–61.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 146–151. https://doi.org/10.1109/CIRA.1997.613851

Yao, H., Qin, R., & Chen, X. (2019). Unmanned aerial vehicle for remote sensing applications—a review. *Remote sensing*, *11*(12), 1443.

Zhilenkov, A. A., & Epifantsev, I. R. (2018). System of autonomous navigation of the drone in difficult conditions of the forest trails. *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 1036–1039.

Zhou, B., Xu, H., & Shen, S. (2023). Racer: Rapid collaborative exploration with a decentralized multi-uav system. *IEEE Transactions on Robotics*, *39*(3), 1816–1835. https://doi.org/10.1109/TRO.2023.3236945

Zhou, B., Zhang, Y., Chen, X., & Shen, S. (2021). Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters*, *6*(2), 779–786. https://doi.org/10.1109/LRA.2021.3051563

Zhou, H., Hu, Z., Liu, S., & Khan, S. (2022). Efficient 2d graph slam for sparse sensing. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6404–6411. https://doi.org/10.1109/IROS47612.2022.9981200

Zhu, C., Ding, R., Lin, M., & Wu, Y. (2015). A 3d frontier-based exploration tool for mavs. *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, 348–352. https://doi.org/10.1109/ICTAI.2015.60

Ziparo, V. A., Iocchi, L., Lima, P. U., Nardi, D., & Palamara, P. F. (2011). Petri net plans: A framework for collaboration and coordination in multi-robot systems. *Autonomous Agents and Multi-Agent Systems*, *23*, 344–383.