The Hashemite University

Faculty of Prince Al-Hussein Bin Abdullah II
for Information Technology
Information Technology Department

Reinforcement Learning for Drone Swarm Control

A project submitted
in partial fulfillment of the requirements for the
B.Sc. Degree in Data Science and Artificial Intelligence

By
BASHAR AMEEN MOHAMMED HLAIL (2136887)
EMAD ISSA MOHAMMAD MAKLOUF (2140641)
MARAM SAMEER MOHAMMAD ALMASARWEH (2144526)

Supervised by
Mo'taz Abdul Aziz Al-hami

2024

CERTIFICATE

It is hereby certified that the project titled **Reinforcement Learning for Drone Swarm Control** submitted by undersigned, in partial fulfillment of the award of the degree of "Bachelor in Data Science and Artificial Intelligence" embodies original work done by them under my supervision.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Emad Issa Makhlouf Signature _____
(2140641)
Bashar Ameen Hlail Signature _____
(2136887)
Maram Sameer Almasarweg Signature _____
(2144526)

Abstract

This study conducts a comprehensive analysis of the utilization of drones in disaster management scenarios, aiming to enhance response efficiency and effectiveness in dealing with such emergency situations. Drawing upon a thorough review of available scientific and technological literature in this field, analytical examination of several real-world cases of drone usage in disaster-stricken areas, and the utilization of simulation models to evaluate the impact of this technology on response operations, this study provides a comprehensive and detailed insight into the benefits and challenges of employing drones in this context.

The findings of this study suggest that drones serve as valuable tools for enhancing disaster preparedness and response capabilities, as they can play a significant role in various aspects of disaster management, including rapid damage assessment, effective execution of search and rescue operations, and timely provision of essential supplies and assistance to affected areas.

In light of these results, the study concludes that integrating drones into disaster response strategies can lead to quicker and more targeted interventions, ultimately resulting in reduced loss of life and property, and increased efficiency and effectiveness of relief and recovery efforts.

The potential impact of this research lies in its ability to guide decision-makers and stakeholders in the field of emergency management towards the adoption of modern technologies such as drones to improve disaster response systems, thereby contributing to a positive shift towards more resilient and adaptive disaster management systems and ensuring the safety and security of affected communities.

Acknowledgements

We would like to express our deepest gratitude to Professor Motaz Al-Hami, our supervisor, for his guidance, support, and encouragement throughout this project. His expertise have been instrumental in the successful completion of this work.

We also extend our heartfelt thanks to Professors Majdi Maabreh and Ahmad Al-Oqaily for their support and assistance in overcoming various challenges during the course of this project. Their help and encouragement were greatly appreciated.

Contents

List of Figures

List of Tables

3 — Introduction

3.1 What are Drones?

Generally, a **drone** refers to an increasingly autonomous machine. Mostly they can fly, but they might move by walking or rolling on the ground. They frequently have parts and technologies similar to those found in robotics. They tend to include a range of sensors that monitor the state of the environment and localize drones in the environment. They also have systems, often based on machine learning algorithms or other optimizing approaches, that allow them either to plan and execute paths or control forces, torques, and velocities. They sometimes must identify objects and inform planners of changes in state, as well as remain within defined specifications. A drone's sensing systems generally consist of a variety of sensors. These sensors can include inertial measurement units (IMUs), which contain accelerometers and gyroscopes; global positioning systems (GPS) to determine location; ultrasonic, laser distance or time-of-flight cameras to sense distance from the ground or other objects or drones; monocular or stereo cameras to estimate distance and inform planning through other image-based information; and radio frequency ID (RFID) sensors, which determine, through various configurations of readers and tags, positions and types of objects in space – sometimes in environments where radio-frequency (RF) waves vary too much in their patterns as they propagate through the space.[25][18]

Generally, when you think of a drone, your mind probably conjures an image of a small unmanned aircraft, either fixed wing or multirotor. But the term goes well beyond just that hardware. Drones with a range of sizes and applications have become common. They can feature onboard technologies ranging from television-quality cameras to LIDAR to radio-frequency identification scanners and heat sensors. And in some settings, they can even use machine learning to identify objects within data that they collect. Today, most engineering and computer science departments at universities have begun to build and study drones. As a result, if they have participated in high school science fairs or art contests, high school students should be familiar with drones as well.

3.2 Applications of Drones

Drones, also known as Unmanned Aerial Vehicles (UAVs), have revolutionized various industries by enhancing operational efficiency, reducing costs, and accessing difficult or remote areas. The UAV market is experiencing significant growth, with projections suggesting an increase from USD 27.43 billion in 2022 to USD 91.23 billion by 2030, reflecting a compound annual growth rate (CAGR) of 16.3%. This surge is attributed to the expanding commercial applications of drones in sectors like delivery, agriculture, and media, amidst growing technological advancements and increasing governmental investments.[1]

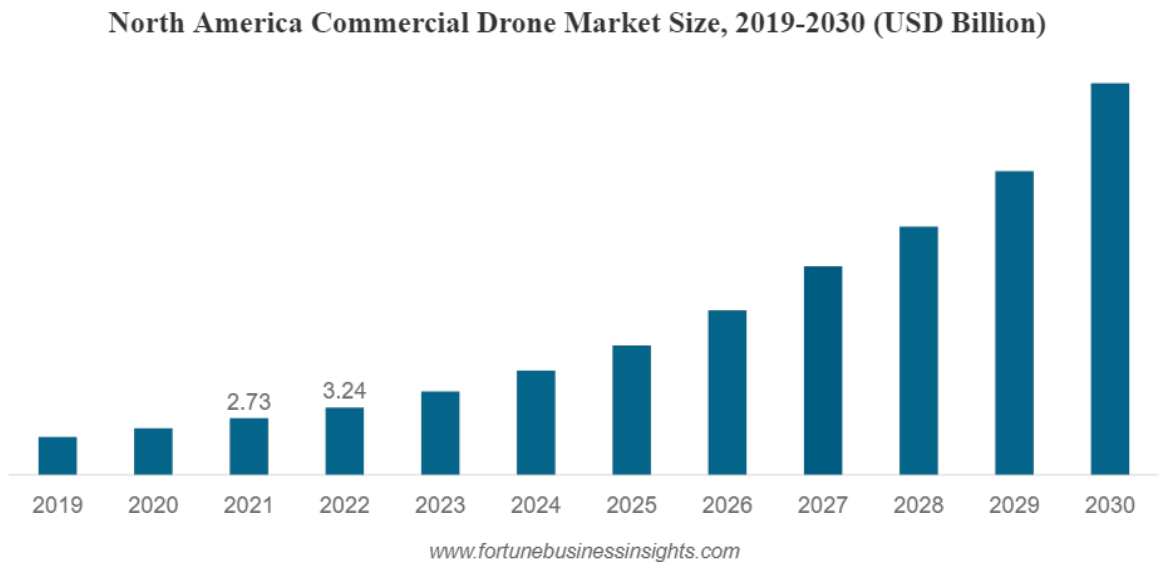


Figure 3.1: The yearly ongoing growth in UAV market.[1]

3.2.1 Aerial Photography and Videography

Drones have become indispensable for the field of aerial photography and filmmaking, providing unique perspectives and high quality images at significantly reduced costs compared to traditional methods. They are simple to operate and can be purchased for less than USD 500. The filming and photography segment is one of the largest in the UAV market, and it has accounted for the largest market share in 2022[1]. Drone photographers use both, vertical and oblique photographs for planning land-use projects, movie production, environmental studies, archaeology, power line inspection, oil and gas surveying, surveillance, commercial advertising, and even artistic projects[2].

3.2.2 Delivery and Logistics

The delivery and logistics sector has seen transformative changes with the introduction of drones, which offer speedy and cost-effective solutions for package delivery. This segment is anticipated to grow rapidly, supported by the increasing use of drones by e-commerce giants like Amazon for delivering goods directly to consumers. Drones are used to deliver medical supplies in difficult terrains. For instance, Zipline (US) used its drones to carry blood samples for COVID-19 testing and supply vaccines to countries in Africa that lack proper healthcare infrastructures. In these areas, flying drones is more effective than driving and can be a valuable substitute for high-priced solutions such as helicopters.[2]

3.2.3 Agriculture and Farming

Drones in agriculture are being used for a range of applications from crop monitoring and health assessment to planting and pesticide spraying. The technology allows for precise agriculture, enhancing productivity while minimizing waste and environmental impact. Companies like Bayer are leveraging drones to improve crop protection strategies across vast agricultural lands, highlighting the role of UAVs in sustainable farming practices.[1]

3.2.4 Other Notable Uses

In addition to the primary uses listed above, drones are also employed in various other roles in different sectors due to their low costs and increased versatility. In the recent years we've seen increased usage of drones by governmental agencies and private enterprise alike. (Table 1.1)

Table 3.1: Summary of Sector Applications [11]

Government • Law enforcement (Police, civil security) • Border security • Coastguard	**Fire Fighting** • Forest fires • Other major incidents • Emergency rescue (i.e. Mountain rescue)	**Energy Sector** • Oil and gas industry distribution infrastructure • Electric grids / distribution networks
Agriculture, Forest and Fishery • Environment monitoring • Crop dusting • Optimizing use of resources	**Earth Observation and Remote Sensing** • Climate monitoring • Seismic events • Major incidents and pollution monitoring	**Communication and Broadcasting** • VHALE platforms as proxy satellites • MALE/SMUAC as short-term, local communication coverage

3.3 Types of UAVs

UAV drones come in a variety of shapes and sizes, and can be categorized based on many different aspects such as design, price range, size, power source, or use case. In this section, we'll discuss the 4 main types of drones.

1. **Multi-rotor Drones**
   Multi-rotor drones feature a structure that extends to accommodate several propellers, with variations from tricopters, which have three propellers, to octocopters, equipped with eight propellers. These drones are categorized as rotary-wing drones, which achieve lift and take off vertically by rotating blades that force air downwards.

   Applications of these drones include aerial photography and videography, mapping and surveying from the air, inspecting assets, monitoring agricultural fields, and delivering products over short distances.

2. **Single-rotor Drones**
   Single-rotor drones look similar to small helicopters, featuring a main rotor that sustains the drone's body. Like their multi-rotor counterparts, they fall under the category of rotary-wing drones.

   Their applications include aerial mapping and surveying, aerial surveillance and patrolling, delivering heavy payloads, and conducting search and rescue missions.

3. **Fixed-wing Drones**
   Much like an airplane, a fixed-wing drone uses wings for lift instead of rotors. These drones are generally large, fuel-powered, and predominantly used by the military, necessitating a runway for takeoff and landing.

   Their applications include aerial mapping and surveying, inspecting assets, delivering payloads over long distances, and unmanned aerial refueling.

4. **Vertical Takeoff and Landing Fixed-Wing (VTOL FW) Drones**
   VTOL (Vertical Take-Off and Landing) fixed-wing hybrid drones combine the best features of rotary-wing and fixed-wing designs, allowing them to lift off vertically like a helicopter and transition to horizontal flight like an airplane. These drones are versatile, eliminating the need for a runway and enabling efficient long-range flight.

   Their applications include rapid aerial surveys, precision agriculture, emergency medical deliveries, and extended surveillance missions.

Table 3.2: Comparison of Different UAV Designs [11]

Types	**Advantages**	**Disadvantages**	**Example**
Fixed wing	Long range Endurance	Horizontal take-off, requiring substantial space or support	
Tilt wing	Combination of fixed wing and VTOL advantages	Expensive Technology complex	
Single-rotor	VTOL Maneuverability High payloads possible	Expensive Comparably high maintenance requirements	
Multi-rotor	Inexpensive, Low weight Easy to launch	Limited payloads Susceptible to wind due to low weight	

3.4 Regulations and Safety Considerations

In Jordan, the regulation of unmanned aircraft systems (UAS) or drones is governed by the Jordanian Civil Aviation Regulation (JCAR) established under the authority of the Jordanian Civil Aviation Law 41/2007 and its amendments.[8]

3.4.1 Drone Registration and licensing

Operators need to apply for a UA Operator Authorization (UOA) from the Civil Aviation Regulatory Commission (CARC). This process includes providing documentation demonstrating compliance with JCAR and includes security clearances, operational descriptions, and a list of unmanned aircraft with details like serial number, color, and mass. The UOA is valid for one year, subject to conditions such as compliance with the JCAR, maintaining valid CARC security clearance, and allowing CARC access to the operator's facilities and records.

3.4.2 No-Fly Zones and Restricted Areas

Specific airspaces are designated where flight is restricted or prohibited. Restricted areas are those where flights are allowed under specific conditions, while prohibited areas are completely off-limits for UA flights. Operations near airports, heliports, helicopter landing sites, or airfields are strictly regulated, with no flights allowed within an 8 km perimeter unless specifically approved.

3.4.3 Privacy and security concerns

Operators must obtain security clearances for the organization, and any use of aerial photographic equipment on UAs requires prior authorization by the Jordanian Security Sector. Operators are required to ensure that flights do not intentionally or unintentionally invade privacy. Procedures must be established to maintain a minimum safety distance from people not involved in the operation and from fixed or mobile objects, ensuring no less than 50 meters unless prior authorization has been obtained from CARC. Operators are also required to report any incidents that might endanger safety, including any that compromise privacy or security.

3.5 What are Drone Swarms?

In nature, many biological organisms compensate for the limitations of individual members by forming groups or clusters that communicate and coordinate their actions. This phenomenon is exemplified by wolves, which optimize their hunting strategies through coordinated efforts; birds, which reduce energy expenditure by flying in flocks during migration; bees, which exhibit sophisticated swarming behaviors when searching for new nesting sites and foraging; and ants, which demonstrate emergent intelligence capable of solving complex problems, such as identifying the shortest path to food

sources[15]. Similarly, drone technology has been increasingly modeled on these natural behaviors to overcome the constraints faced by individual drones. Drone swarm technology, in particular, has become a hot field of research in recent years, leading to numerous practical applications. Such as surveillance[21], search and rescue[9], payload transportation[14], reconnaissance and mapping[23], and public communication[20]. Which will be discussed in further detail in the following chapters.

3.6 Motivation

Drones have rapidly evolved from niche gadgets to pivotal tools across various industries. Coupled with advancements in swarm intelligence algorithms, graph traversal algorithms, and artificial intelligence, specifically reinforcement learning, drone swarm technology is set to revolutionize tasks requiring autonomy and precision. Reinforcement learning, a type of machine learning where agents learn and make decisions based on their interaction with the environment, is particularly useful for managing the complexities of drone swarm coordination in dynamic settings.

While individual drone operations have been extensively studied, the coordinated efforts of swarms, especially using reinforcement learning, remains underexplored. This project aims to bridge the gap by developing algorithms that improve autonomy, reduce communication needs, and enhance decision making capabilities in uncertain dynamic environments.

The implications of successfully coordinating drone swarm through reinforcement learning extends well beyond the immediate applications. Economically, is could reduce the costs and time needed for operations like search and rescue or disaster recovery. Societally, it could enhance safety and effectiveness, potentially saving lives during critical missions. Technologically, it could pave the way for the development of more robust autonomous systems capable of complex, independent operation without human control.

3.7 Problem Statement

3.7.1 Specific challenges in drone swarm coordination

The primary objective of this project is to address several critical challenges in drone swarm coordination that have been identified as major hurdles in deploying these technologies effectively in real-world applications. These challenges include:

- Scalability of Control: Current drone coordination systems struggle to maintain efficiency and effectiveness as the number of drones increases. The complexity of managing large swarms in unison without exponential increases in computational and communication resources remains a significant technical challenge[4].
- Intelligent Path-Finding: Drone swarms should find optimal paths to maximize coverage and minimize needless energy consumption. Current path-finding algorithms often rely on simple optimization heuristics, which can compromise

control and communication between drones, leading to inefficient operations and increased energy use.[26].

- Real-time Adaptability: Drone swarms must be able to respond immediately to changing environmental conditions and unforeseen obstacles. Current models lack the necessary flexibility, often resulting in delayed responses or suboptimal decision-making in fast-paced scenarios[4].
- Robustness Against Environmental Uncertainties: Operating in diverse and unpredictable environments, such as those encountered during natural disasters, poses a severe challenge for drone swarms. The ability to maintain operational integrity and swarm resilience under such conditions is not yet reliable[24].
- Communication and Coordination: Effective communication and coordination among drones are crucial for cohesive swarm behavior. Current systems often suffer from latency, bandwidth limitations, and communication failures, which can lead to disorganized movements and collisions. Ensuring reliable, low-latency communication in dynamic and potentially congested environments is a significant challenge[24].

3.7.2 Relevance to Real-World Scenarios

- Search and Rescue Operations: In situations where time is of the essence, the inability of drones to scale effectively and find optimal paths to insure maximum coverage could be the difference between life and death.
- Disaster Recovery Efforts: The robustness of drone swarms in unpredictable environments directly affects their utility in assessing damage and aiding in disaster recovery, where conditions can severely limit the availability of human-led efforts.
- Public Communication: In rural areas where public communication may not be available, drones must be able to communicate to coordinate themselves effectively in order to insure maximum coverage of wireless connectivity[20].

3.7.3 Research Questions

Based on the challenges we've presented, this project aims to explore the following research questions:

1. *How can reinforcement learning be used to enhance the scalability of drone swarm control without compromising operational efficiency?*
2. *Can reinforcement learning find optimal polices that contribute to the robustness of drone swarms, enabling them to maintain functionality in diverse and dynamic environmental conditions?*
3. *Can reinforcement learning be used to optimize the path-finding process to maximize coverage and minimize energy usage?*

4 — Literature Review

4.1 Existing Systems

In recent years, the field of drone swarm control and coordination has seen significant advancements with the development of various sophisticated systems. These systems aim to enhance the autonomy and capabilities of unmanned aerial vehicles (UAVs) in performing complex missions with minimal human intervention. The primary focus of these systems is to manage large-scale UAV swarms efficiently, ensuring robust performance in dynamic and uncertain environments. This section provides an overview of some of the prominent architectures developed for drone swarm control and coordination. Each architecture presents unique approaches and solutions to the challenges associated with UAV swarm operations.

4.1.1 Collaborative Mission Planning & Autonomous Control Technology (CoMPACT) System

The CoMPACT system is an advanced hierarchical control architecture designed for managing large-scale swarms of UAVs. Its primary objective is to execute complex military missions such as Intelligence, Surveillance, and Reconnaissance (ISR), Suppression of Enemy Aerial Defenses (SEAD), and Destruction of Enemy Aerial Defenses (DEAD). The system is structured into six coordinated hierarchical layers that encompass tasks and objectives from the mission planning stage to UAV task execution[7]. These layers are show in figure 4.1.

1. **Mission Planning Layer**: Located at the command center, this layer involves mission preplanning and high-level objective setting. Evolutionary algorithms are used for mission planning, generating 4D trajectories for mission execution.

2. **Mission Execution Layer**: Also at the command center, this layer executes and synchronizes tasks in line with the pre-specified timeline, utilizing Finite State Machines (FSM) for managing sequences of functional tasks.

3. **Function Execution Layer**: Functions like ISR, SEAD, and DEAD are managed by semi-global leaders who respond to environmental changes and team capabilities.

4. **Team Task Execution Layer**: Teams are coordinated by local team leaders who manage sequences of tasks and interact with other teams as needed.

5. **Platoon Task Execution Layer**: Each platoon is led by a platoon leader and consists of UAV tasks and trajectories executed using FSM, reactive assignment, and reactive path planning.

6. **Vehicle Task Execution Layer**: Located at each UAV, this layer includes dynamic path re-planning, collision avoidance, and local task execution.

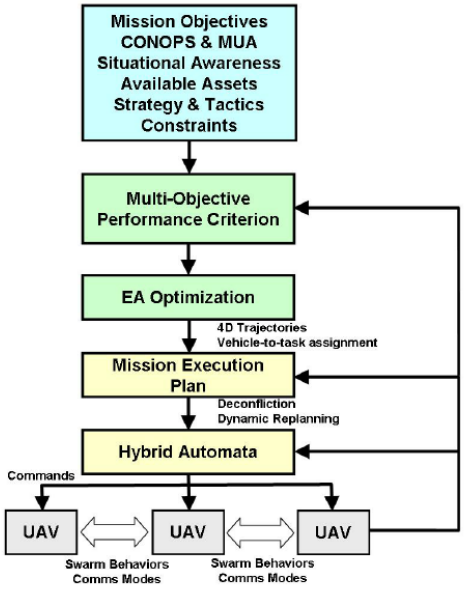


Figure 4.1: The 6 layer structure of CoMPACT systems

The CoMPACT system employs a combination of evolutionary algorithms for mission planning, hybrid automata-based task execution, and biologically-inspired emergent swarm behaviors. This combination enables the system to be scalable, robust to intermittent communication losses and delays, and adaptable to dynamic changes in the environment. Through simulation and experimentation, the CoMPACT system has demonstrated effectiveness in performing ISR, SEAD, and DEAD missions under conditions of uncertainty and vehicular losses.

4.1.2 Aerostack2

Aerostack2 is an advanced framework developed to address the fragmentation and lack of standardization in aerial robotics. Built on ROS 2 middleware, Aerostack2 offers a modular software architecture that supports a wide range of capabilities for autonomous drone operations. Key features include platform independence, versatility in handling different types of drones, and a logical level for easy mission specification. Aerostack2 also provides pre-programmed components that simplify the engineering process and is open-source, which democratizes access to its technology[12].

Aerostack2's components are organized hierarchically across several layers, each responsible for different aspects of drone operation as shown in figure 4.2.

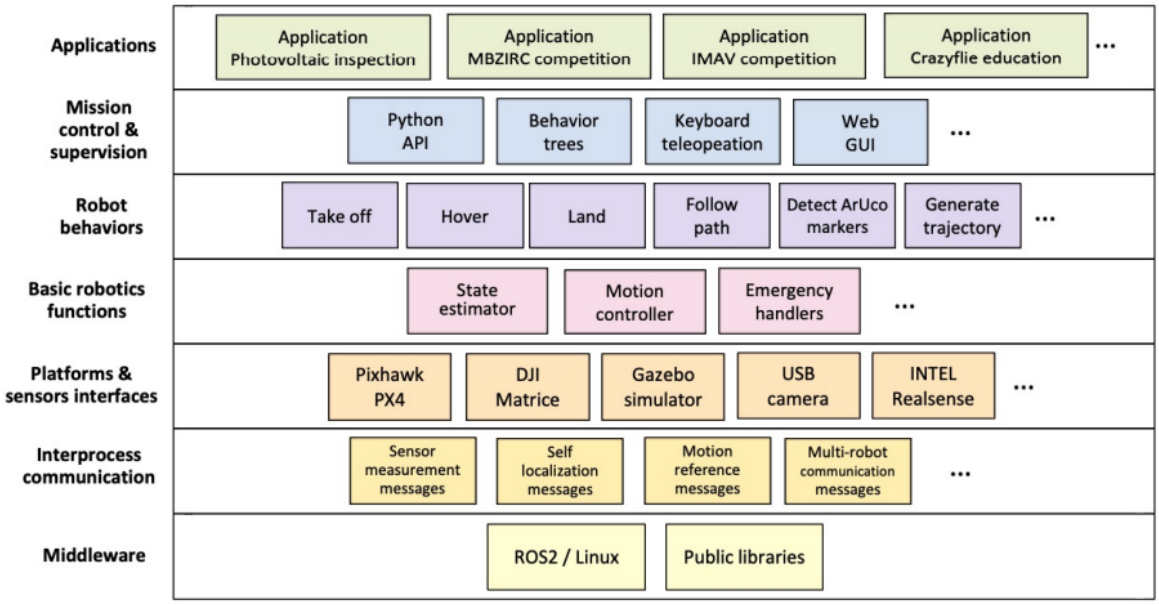


Figure 4.2: Overview of the software components provided by the Aerostack2 environment[12].

1. **Middleware Layer**: This foundational layer includes the Linux operating system, ROS 2, and general software libraries like OpenCV.
2. **Inter-process Communication Layer**: This layer provides components for facilitating communication between concurrently operating processes, using standardized data structures and communication mechanisms defined by ROS 2 and specific to aerial robotics.
3. **Interfaces with Platforms and Sensors**: Aerostack2 offers interfaces to connect various aerial platforms and sensors with the framework. These interfaces ensure compatibility with both physical and simulated platforms, supporting a broad range of hardware configurations.
4. **Basic Robotics Functions**: This layer includes components implementing essential functions such as state estimation, motion control, and emergency handling. These components are designed to be general and reusable, with alternative algorithms available as plugins.
5. **Behaviors Layer**: The behaviors layer includes components that encapsulate specific robot skills, such as taking off, landing, and following a path. Each behavior supports robust execution monitoring and task refinement, facilitating the implementation of complex mission plans.

6. **Mission Control Layer**: This top layer provides tools for specifying and supervising mission plans. Aerostack2 offers a Python API and behavior trees for flexible mission planning, along with user interfaces for real-time monitoring and control.

4.1.3 AerialCore

AerialCore is an aerial system developed with ROS Noetic, designed for full onboard execution. It can be deployed on any multi-rotor vehicle equipped with a PX4-compatible flight controller, suitable for both indoor and outdoor environments. The system supports multi-robot experiments via Nimbro network communication and offers both agile flying capabilities and robust control[6].

4.2 Limitations in Existing Systems

Despite the advancements in aerial robotics frameworks, several limitations persist, hindering the full potential of drone swarm operations.

- **Coordination and Scalability**: Effective coordination and scalability of drone swarms are still significant challenges. While some frameworks support multi-agent systems, the complexity of managing large swarms in dynamic environments requires more sophisticated coordination algorithms and better handling of communication delays and failures.
- **Incompatibility and Integration Issues**: The diversity of hardware platforms and sensors poses integration challenges. While frameworks like Aerostack2 strive for platform independence, systems like AerialCore still face difficulties in interfacing with different hardware configurations, which can hinder their widespread adoption and ease of use.
- **Narrow Focus**: Many existing frameworks have a narrow focus, often limited to low-level control. This specialization makes them less useful for comprehensive applications that require integration of multiple functionalities, such as mission planning, state estimation, and autonomous control.
- **Lack of Standardization**: One of the primary challenges in aerial robotics is the lack of standardization. While ground robotics has seen widespread adoption of standardized frameworks like Navigation2 and MoveIt, aerial robotics remains fragmented. Different research groups develop their own frameworks, leading to isolated efforts that are difficult to integrate.

5 — Preparing the Environment

In any complex engineering project, setting up the working environment is a crucial initial step. For this project, which focuses on drone swarm control and coordination using reinforcement learning, the preparation of a robust and comprehensive environment is essential. This environment includes the installation of several key components: a flight stack, a ground control station, a simulation environment, and ROS2 (Robot Operating System 2). Each of these components plays a vital role in ensuring the smooth operation and testing of the UAV swarm, facilitating effective development, and allowing for thorough experimentation.

The flight stack is the core software that controls the flight operations of the drones. It provides the necessary algorithms and interfaces for drone navigation, stabilization, and mission execution. Installing a reliable flight stack ensures that the drones can perform the required maneuvers accurately and respond to commands efficiently.

The ground control station (GCS) is the interface through which operators can monitor and control the UAVs. It provides real-time telemetry data, mission planning tools, and a user-friendly interface for sending commands to the drones. A well-configured GCS is crucial for effective mission management and for ensuring that operators can intervene when necessary.

The simulation environment is another critical component, providing a safe and controlled space to test and validate the UAV swarm algorithms. Simulation allows for extensive testing of different scenarios, which might be too risky or impractical to perform in real life. It helps in identifying potential issues and optimizing performance without the risk of damaging the actual drones.

ROS2 is a middleware framework that facilitates communication between the various components of the drone system. It supports the integration of sensors, actuators, and algorithms, enabling seamless data exchange and coordination. ROS2's modularity and extensive libraries make it an invaluable tool for developing complex robotic systems like drone swarms.

The preparation of the environment in this project can be compared to the data preprocessing workflow in machine learning. Just as data preprocessing involves cleaning, transforming, and organizing data to ensure that machine learning models can learn effectively, preparing the drone environment involves setting up the necessary software and tools to ensure that the UAVs can operate and be controlled effectively. Both processes are foundational steps that significantly impact the success of the project. In machine learning, proper preprocessing leads to better model performance and more accurate predictions. Similarly, in drone swarm control, a well-prepared environment

leads to more reliable operations, better coordination, and ultimately, more successful missions.

By meticulously setting up the flight stack, ground control station, simulation environment, and ROS2, we lay a solid foundation for the subsequent development and testing phases of this project. This preparation ensures that we can efficiently implement, test, and refine the reinforcement learning algorithms needed for effective drone swarm coordination.

5.1 Drone System Overview

A typical intelligent drone system consists of multiple components that work together to effectively control the drone. Figure 5.1 showcases these different components.[17]

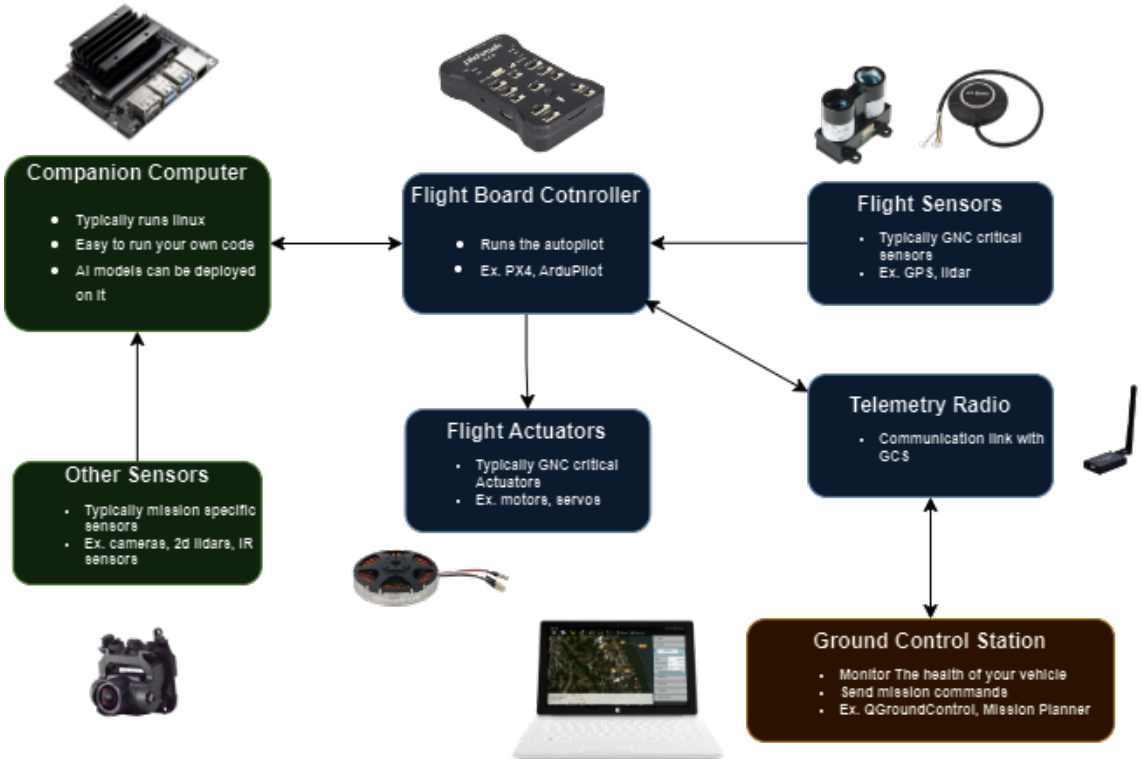


Figure 5.1: Intelligent drone system diagram.[17]

1. **Flight Board Controller**: At the core of the system is the flight controller, which acts as the brain of the drone. The flight controller contains embedded sensors such as IMUs, accelerometers, and compasses. These embedded sensors are crucial for maintaining stability and providing real-time data on the drone's orientation, acceleration, and heading.

   In addition to the hardware components, the flight controller includes a software aspect through the installation of autopilot flight stacks. These flight stacks, such

as ArduPilot or PX4, provide the essential algorithms and protocols that enable autonomous flight, navigation, and control. They handle tasks like stabilizing the drone, executing predefined flight paths, and responding to commands from the ground control station.

2. **Flight Sensors**:The next crucial component of drones is the flight sensors, which provide accurate and reliable position estimates for the drone. These sensors are essential for navigation, stability, and obstacle avoidance.

   GPS is one of the most common flight sensors, providing precise information about the drone's location and enabling accurate positioning and waypoint navigation.

   LiDAR is another important sensor, used primarily for altitude estimation and obstacle detection. By emitting laser pulses and measuring the time it takes for them to bounce back from surfaces, LiDAR helps the drone to create detailed 3D maps of its surroundings and maintain a safe altitude.

   Other common sensors include barometers for measuring atmospheric pressure to determine altitude, optical flow sensors for detecting ground motion, and ultrasonic sensors for short-range distance measurement.

3. **Telemetry Radio**: The next crucial component is the telemetry radio, which allows the drone to communicate with other devices via radio signals. This communication link is essential for transmitting real-time data between the drone and the GCS or other drones in a swarm.

   The telemetry radio sends and receives data such as flight status, sensor readings, and control commands. This enables operators to monitor the drone's performance, adjust flight parameters, and ensure the drone follows its planned mission. Common telemetry radio systems operate on frequencies like 915 MHz or 2.4 GHz, providing reliable and long-range communication.

   In swarm operations, the telemetry radio also facilitates inter-drone communication, allowing drones to coordinate their actions and share information about their environment. This is critical for tasks that require collective behavior, such as formation flying or collaborative mapping.

4. **Ground Control Station**: As mentioned earlier, the GCS serves as an interface to view telemetry information about the drone and send commands. The GCS is installed on a ground-based device such as a laptop, tablet, or phone and is directly accessed by the user.

   The GCS software provides a user-friendly interface for monitoring the drone's status, viewing real-time data, planning missions, and sending control commands. This allows operators to manage the drone's flight path, adjust settings, and respond to any issues during the mission.

5. **Companion Computer**: The companion computer is an optional component of drones, providing extra processing power for additional utilities such as extra

storage or deploying AI models for tasks like image recognition. This component runs on a Linux-based operating system, making it easy to develop and run custom code. The companion computer can handle complex algorithms and data processing tasks that the primary flight controller might not be able to manage. This includes running machine learning models, processing high-resolution images and videos, and executing advanced navigation algorithms.

6. **Additional Sensors**: The companion computer may require additional sensors for specific missions, such as cameras, 2D LiDARs, and IR (infrared) sensors. These sensors provide the companion computer with the necessary data to perform specialized tasks.

7. **Flight Actuators**: Given the input from the sensors, companion computer, and GCS commands, the flight controller issues commands to the actuators to perform specific actions. Actuators include components such as motors and servos, which execute the flight controller's instructions.

   Motors are responsible for controlling the drone's propellers, enabling it to take off, maneuver, and maintain stable flight. The flight controller adjusts the speed of each motor to achieve the desired movement and orientation.

   Servos are used to control movable parts of the drone, such as camera gimbals or control surfaces on fixed-wing drones. By adjusting the position of these servos, the flight controller can stabilize cameras, change the drone's direction, or manipulate other mechanical components.

5.2 Choosing the Right Flight Stack

The drone's "brain" is known as the autopilot. It typically includes flight stack software running on a real-time operating system (RTOS) on flight controller (FC) hardware. The flight stack provides essential stabilization and safety features, along with pilot assistance for manual flight and automation of tasks like takeoff, landing, and executing predefined missions.

Some autopilots also incorporate a general-purpose computing system for higher-level command and control, supporting advanced networking, computer vision, and other features. This might be implemented as a separate companion computer, but it is increasingly likely to become a fully integrated component in the future.[16]

Choosing the right flight stack is paramount for our project, as it could determine the scope of our work and the flexibility of our development process. as such, two notable flight stacks stand out; **PX4** and **ArduPilot**. Both of these flight stacks are popular options among hobbyists and professionals alike. Table 5.1 provides a comparison between these two flight stacks.

PX4 is usually used in research when drone developers require fine-grained control of their drone system's flight controls and stabilizers. This means that the default configurations for PX4 are relatively naive and may only work under lab conditions.

Meanwhile, ArduPilot has extremely sophisticated control systems. This prevents fine-grained control of our flight controls but results in high-performance flight. ArduPilot has also existed for longer than PX4 and, as such, is the more mature flight stack with a richer and more robust feature set, extensive documentation, and a larger community.

Therefore, the decision was made to use ArduPilot as it aligns better with our project requirements and abstracts away the nuanced fine-grained control that goes well beyond the scope of our project.

Feature	**PX4**	**ArduPilot**
License	BSD License; changes do not need to be shared with the community	GPL License; changes must be shared with the community
Modularity	Highly modular; allows customization and integration of custom modules and sensors	Versatile but less modular than PX4
Vehicle Support	Primarily focused on aerial vehicles but supports other types as well	Supports a wide range of vehicles including planes, VTOLs, rovers, submarines, and boats
Community and Documentation	Active community with continuous innovation; documentation is not as extensive as ArduPilot's	Large, active community with extensive documentation and support resources
Advanced Features	Advanced autopilot capabilities including obstacle avoidance and GPS-denied navigation	Comprehensive feature set including support for various mission-critical applications
Industry Adoption	Widely adopted in commercial drone industry; used for industrial inspections, delivery drones, and aerial photography	Popular among hobbyists and researchers; strong presence in both amateur and professional domains
SDK and APIs	Uses MAVSDK for high-level programming and control	Uses DroneKit and MAVROS for high-level programming and control
Ease of Use	More suited for users with specific needs for precision and advanced control	More user-friendly, especially for beginners due to its extensive documentation and community support

Table 5.1: Comparison of PX4 and ArduPilot

5.2.1 ArduPilot Introduction & Installation

Ardupilot is an advanced open-source autopilot software, designed to control autonomous vehicles across multiple domains including aerial drones, ground vehicles, boats, and even submarines. Initially developed in 2007 by members of the DIY Drones community, Ardupilot has evolved into a robust platform utilized worldwide in numerous applications ranging from hobbyist projects to commercial and research operations. This software provides a rich set of features and supports a wide array of hardware, making it a versatile choice for building and operating unmanned vehicles[3]. Some of the core features of ardupilot include:

1. **Multi-Vehicle Support**: Ardupilot is uniquely versatile, capable of controlling various types of vehicles such as multirotors, fixed-wing aircraft, helicopters, rovers, boats, and submarines. Each vehicle type benefits from specialized control algorithms developed to maximize performance and stability in a wide range of operating conditions.

2. **Comprehensive Flight Modes**: The software offers an extensive range of flight modes, from basic manual and stabilized flight to fully autonomous missions. These include Auto, Guided, RTL (Return to Launch), and Loiter, allowing operators to match the flight mode to the mission requirements precisely.

3. **Highly Configurable Mission Planning and Execution**: Users can plan missions with multiple waypoints, set actions at each waypoint, adjust flight paths dynamically, and manage payload operations during the flight. This is facilitated through integration with GCSs, which provide user-friendly interfaces for detailed mission scripting and real-time vehicle monitoring.

4. **Sensor and Peripheral Compatibility**: Ardupilot supports a broad spectrum of sensors and peripherals, enhancing the vehicle's ability to navigate and perform complex tasks. This includes GPS units, IMUs, optical flow cameras, rangefinders (like lidars and sonars), and various communication peripherals using MAVLink, CAN, and DroneCAN protocols for data exchange.

5. **Advanced Safety and Redundancy Features**: Safety is a critical aspect of Ardupilot's design, incorporating features like geo-fencing, battery failsafes, and emergency landing procedures. It also supports hardware redundancy to enhance reliability, particularly in commercial and high-risk operations.

6. **Open Development and Community Support**: Being open-source, Ardupilot benefits from contributions by a global community of developers who continually enhance its capabilities. This collaborative environment helps keep the platform at the technological forefront and fosters a supportive ecosystem for users and developers alike.

Ardupilot is compatible with a range of flight controller hardware, the most prominent being the Pixhawk family, including Pixhawk, Pixhawk 2, Pixhawk 4, and Pixhawk 5. It is also compatible with other controllers like the Cube Autopilot series and

boards from Matek. These flight controllers can be integrated with a variety of peripherals through interfaces such as I2C, UART, SPI, and CAN, allowing for a modular and scalable vehicle design.

ArduPilot can be installed by cloning the GitHub repository provided on their main website, where they offer OS-specific dependency installation scripts that must be executed. While this step is fairly straightforward, dependency management has proven quite problematic on Arch Linux due to an outdated installation script that was incompatible with Python 3.12. As a result, we had to move our work to the virtual machine provided by Dr. Majdi Maabreh, which was running on Ubuntu 22.04.4 LTS (Jammy Jellyfish).

5.2.2 MAVLink Protocol

MAVLink, or Micro Air Vehicle Link, plays a crucial role in ArduPilot as a communication protocol that facilitates data exchange between the flight controller and external devices, GCSs, on-board peripheral devices, and even other autonomous vehicles. It is a lightweight, header-only message marshaling library designed for efficient, high-latency, and noisy links typically found in drone applications[13]. Here are some of MAVLink's key features:

- Highly efficient. MAVLink 1 has only an 8-byte overhead per packet, including start sign and packet drop detection, while MAVLink 2 has a 14-byte overhead but offers greater security and extensibility. Its lack of additional framing makes it ideal for applications with limited communication bandwidth.

- Extremely reliable. MAVLink has been used since 2009 for communication between various vehicles, ground stations, and other nodes over diverse and challenging communication channels with high latency and noise. It includes methods for detecting packet drops and corruption, as well as for packet authentication.

- Supports numerous programming languages and runs on various microcontrollers and operating systems, including ARM7, ATMega, dsPic, STM32, Windows, Linux, MacOS, Android, and iOS.

- Allows up to 255 systems to operate concurrently on the network, including vehicles and ground stations.

- Facilitates both offboard and onboard communications, such as between a ground control station and a drone, and between a drone autopilot and a MAVLink-enabled drone camera.

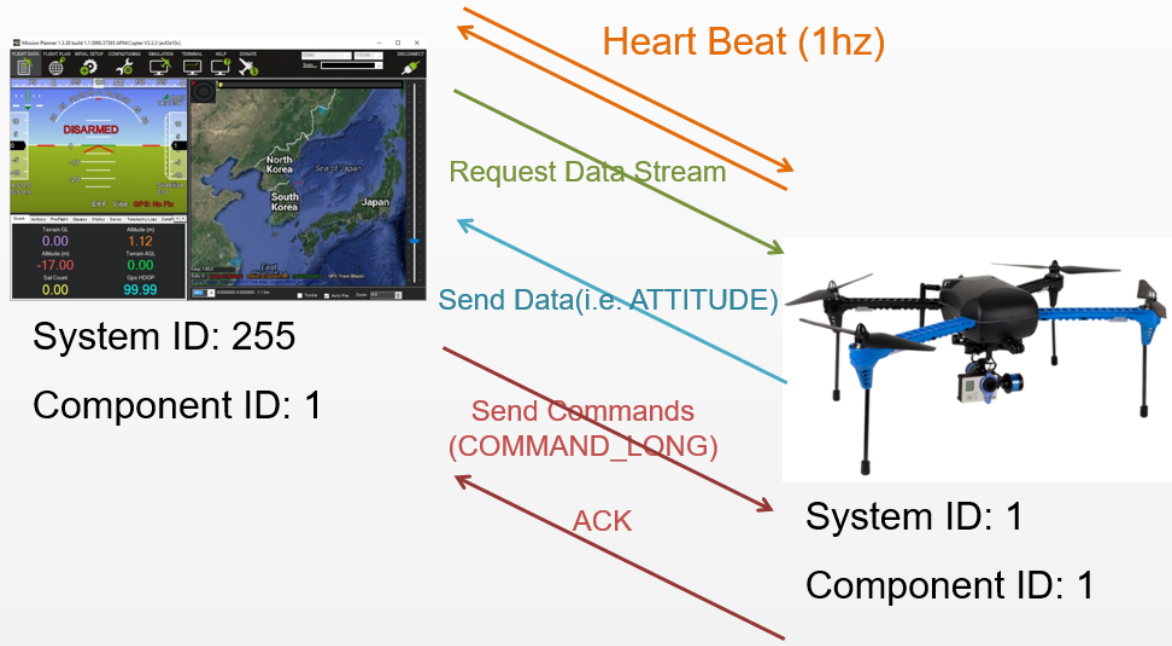


Figure 5.2: MAVLink high level message flow[3]

MAVLink enables the transmission of telemetry data from the vehicle to the ground control station and vice versa. This data can include position, velocity, attitude, battery status, and other vital flight information. It also allows operators to control the vehicle remotely, send commands, and modify flight parameters in real-time, enhancing the flexibility and responsiveness of operations.

5.2.3 Software in the Loop (SITL) Simulation

In drone systems, the general control cycle begins with the flight sensors measuring the state of the drone. This state is then fed into the flight controller, which determines the next action to take based on the control laws. These actions are then sent to the actuators, which perform the actions. The sensors then measure the state again, and the cycle repeats. SITL allows us to simulate this loop without any special hardware. It takes advantage of the fact that ArduPilot is a portable autopilot that can run on a wide variety of platforms, including a PC. When running in SITL, the sensor data comes from a flight dynamics model in a flight simulator. ArduPilot has a wide range of built-in vehicle simulators and can interface with several external simulators, which will be discussed in greater detail in the following sections[3].

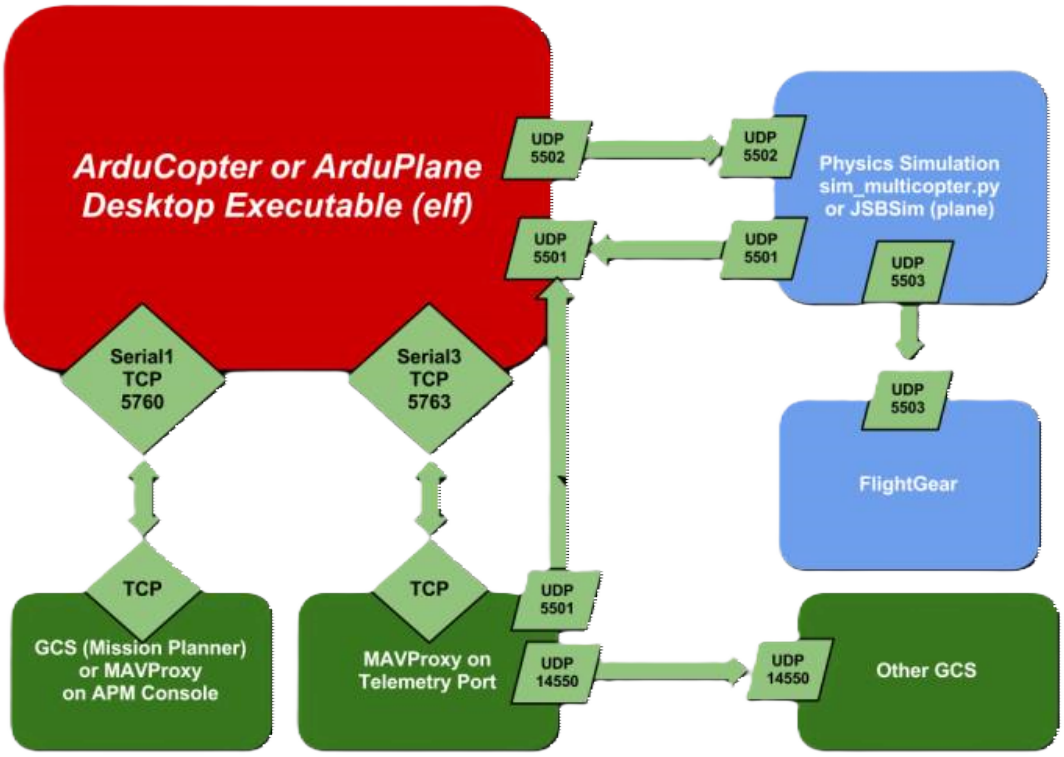


Figure 5.3: SITL Architecture[3]

In real drone systems, the communication between different internal components of the drone, such as sensors and the flight controller, predominantly uses electrical signals over communication protocols like I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface). These protocols are chosen for their suitability for high-speed, low-latency data transfer required by sensors like IMUs, GPS, barometers, and magnetometers. Additionally, Pulse Width Modulation (PWM) signals are often used to control motors and electronic speed controllers (ESCs).

For external communication, including interactions between the flight controller, companion computer, and GCS, the MAVLink protocol is used. This communication is typically facilitated via UART (Universal Asynchronous Receiver/Transmitter) connections and radio telemetry modules. The companion computer and flight controller communicate using MAVLink over interfaces such as UART, USB, CAN bus, or Ethernet, depending on the specific hardware and requirements.

In SITL simulations, however, all communication between components, including simulated sensors and the flight controller, occurs through MAVLink. This ensures that the simulated environment mimics the communication protocols used in real-world scenarios, with MAVLink messages being transmitted over network protocols like UDP (User Datagram Protocol) or TCP (Transmission Control Protocol). This use of MAVLink in SITL provides a realistic testing environment for drone software.

5.2.4 MAVProxy

MAVProxy is a powerful command-line based GCS that allows users to configure and manage their drones through the MAVLink protocol. This initial setup through a CLI ensures that basic configurations and initial troubleshooting can be done efficiently before moving on to more complex GCS software with graphical interfaces[3].

It also acts as a MAVLink router, enabling communication between the drone and various GCSs. This means it can forward MAVLink messages to multiple GCSs simultaneously, allowing for a more flexible and robust setup where multiple devices or applications can monitor and control the drone at the same time[3].

MAVProxy is written in Python and is highly extensible through Python modules. This makes it ideal for developers who need to customize their GCS setup or integrate additional functionalities. Starting with MAVProxy allows for the development and testing of custom modules before integrating them into other GCS software[5].

Installing MAVProxy first sets a solid foundation for drone configuration, communication, and testing. It provides essential functionalities and a robust environment for managing ArduPilot-based systems, which complements the more user-friendly graphical GCS installed later.

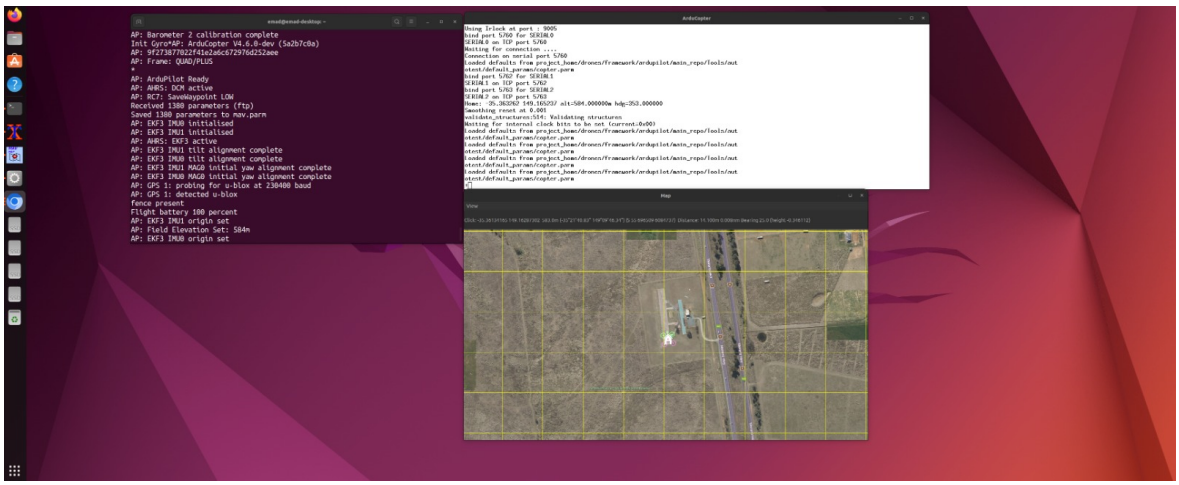


Figure 5.4: MAVProxy running on ubuntu.

5.3 Ground Control Station

5.3.1 Comparing different GCSs

When it comes to drone GCSs, two prominent options are QGroundControl and Mission Planner. Both are compatible with ArduPilot, but they have distinct features and differences that may influence your choice based on specific needs and operating environments.

Table 5.2: Comparison of QGroundControl and Mission Planner

Feature	QGroundControl	Mission Planner
Platform Compatibility	Cross-platform	Windows, macOS, Linux
User Interface	Clean, modern	Feature-rich
Firmware Support	PX4, ArduPilot	ArduPilot
Advanced Features	Geofencing, offline mapping	Dataflash log analysis
Community Support	Active community	Robust ArduPilot support
Mobile Device Support	Yes	Limited
Ease of Use	High	Moderate
Best For	General users	Detailed configuration

1. **QGroundControl**: QGroundControl is a versatile and widely used GCS that supports multiple platforms, including Windows, macOS, Linux, Android, and iOS. This broad compatibility makes it highly accessible for a diverse user base. QGroundControl is particularly noted for its clean, modern, and user-friendly interface, which simplifies navigation and operation, making it suitable for both beginners and experienced users. It supports both PX4 and ArduPilot firmware, offering flexibility for users with different types of autopilots. QGroundControl includes advanced features such as geofencing, dynamic geofencing, and offline mapping capabilities, which enhance its functionality in various scenarios. The active user community provides extensive support and resources, facilitating troubleshooting and continuous learning. Its full support for mobile devices adds to its convenience, especially for field operations using smartphones or tablets[3].

2. **Mission Planner**: Mission Planner is a highly feature-rich GCS primarily designed for Windows, although it can be run on macOS and Linux using Mono, albeit with some performance and compatibility issues. This GCS is well-known for its detailed and customizable interface, providing comprehensive tools for mission planning, real-time monitoring, and in-depth data analysis. Features like dataflash log analysis and real-time data graph plots are particularly useful for advanced mission planning and troubleshooting. Mission Planner is widely supported within the ArduPilot community, ensuring robust support and frequent updates. However, its interface can be less intuitive compared to QGroundControl, which might pose a challenge for new users. The limited support for mobile devices makes it more suited for desktop use. Mission Planner is ideal for users who require detailed configuration and analysis tools, especially those operating on Windows platforms[3].

Given our current requirements and operating system preferences, QGroundControl is a practical choice, with the flexibility to switch to Mission Planner if your needs evolve.

5.3.2 QGroundControl Installation & Testing

To install QGroundControl, these 3 steps had to be done:

1. Adding the user to the dialout group and removing ModemManager to ensure proper access to serial ports.
2. Installing GStreamer plugins and other required libraries for video streaming and application functionality.
3. downloading the latest QGroundControl AppImage, making it executable, and then running the application.

After successfully installing QGroundControl, it was time to take the simulated drone on a test flight. We'll be using QGroundControl to assign a simple mission to the drone. The mission will consist of the drone taking off, traveling to two distinct waypoints, and then returning and descending.

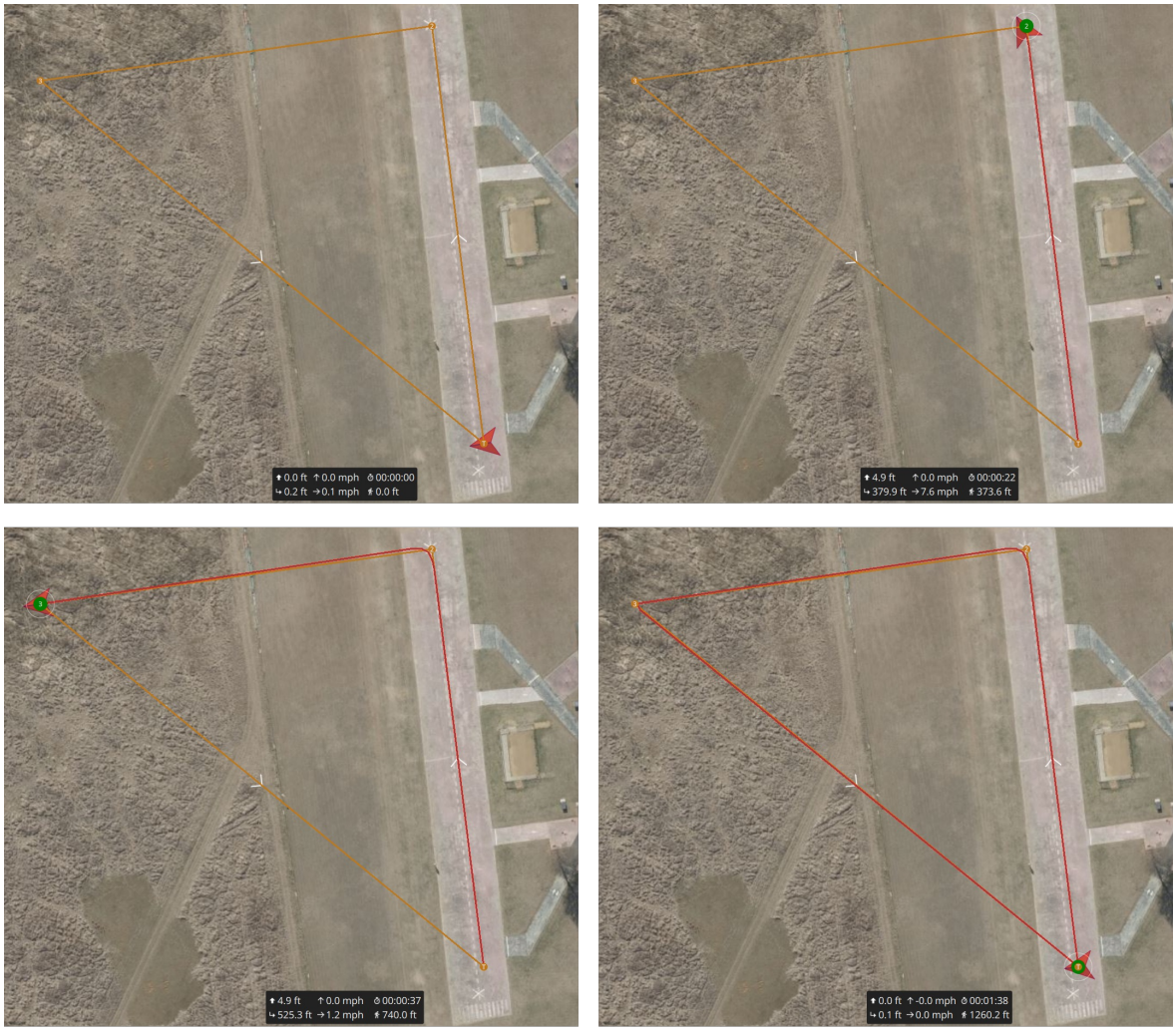


Figure 5.5: QGroundControl test flight

5.4 Gazebo

5.4.1 Gazebo Introduction

The next step in our environment preparation was to install a 3D physics simulation environment. This would allow us to test our drone's collision avoidance capabilities, image recognition, and search efficiency. One popular choice is Gazebo. Gazebo is a highly respected simulation environment in the robotics community. It has been used in numerous robotics simulation challenges for ground, marine, and space-based robots, including the DARPA Robotics Challenge, the DARPA Subterranean Challenge, and the Virtual RobotX Competition. Gazebo provides robust physics simulation, high-quality graphics, and the ability to simulate a variety of sensors and objects, making it an ideal tool for developing and testing complex robotic systems. Additionally, Gazebo integrates well with ROS (Robot Operating System), enabling seamless testing and development of robotic algorithms in a simulated environment before deployment in the real world.

5.4.2 Gazebo Installation & Testing

The Gazebo installation process comprised of these general steps:

1. Setting up our system by installing necessary tools and libraries to support the installation process.

2. Adding the Gazebo package repository to our system and installing Gazebo.

3. Cloning the ArduPilot Gazebo plugin repository, building the plugin, and configuring our system to recognize the new plugin. This step integrates ArduPilot with Gazebo, enabling detailed drone simulations.

4. Adjusting firewall settings to allow necessary network communications, ensuring seamless operation of the simulation environment.

During the installation of Gazebo on Dr. Majdi's virtual machine, we encountered significant performance issues, with the simulation running extremely slow. The root cause of this problem was the lack of nested virtualization support. This technology allows a virtual machine to take advantage of hardware virtualization extensions, such as Intel VT-x or AMD-V, which are used to improve the performance of virtual environments. Without nested virtualization, the virtual machine's performance can be severely degraded because it cannot fully utilize the host machine's hardware capabilities.

After encountering difficulties with enabling the nested virtualization setting on the virtual machine, we decided to move our work to a local installation of Ubuntu 22.04.4 LTS (Jammy Jellyfish).

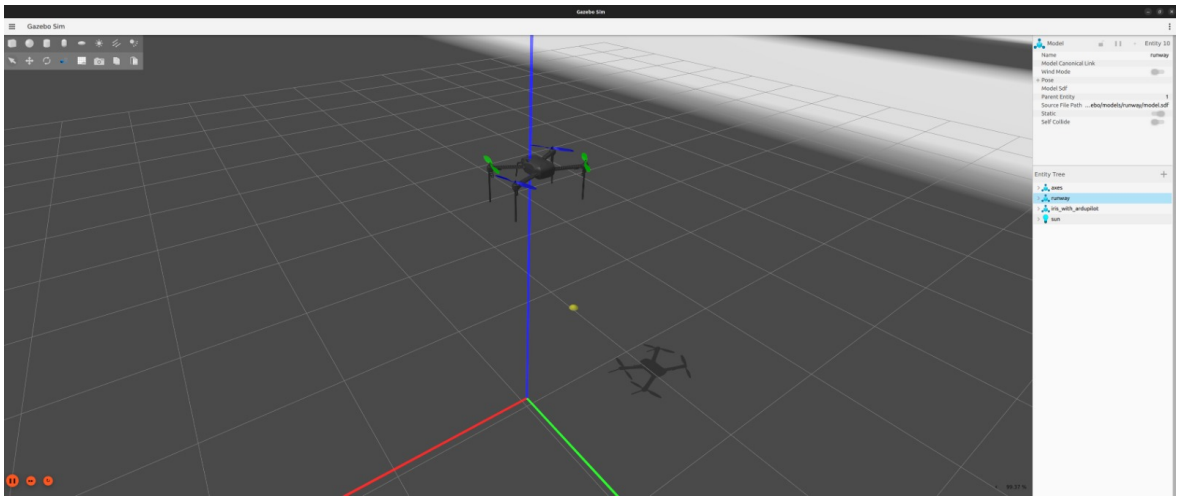


Figure 5.6: Screenshot of Gazebo.

5.5 Testing Intelligent Drone Navigation

As a final test in the current environment, we wanted to initialize two SITL instances, and have it so that whenever a waypoint is set only the closest drone would move towards it with the other drone remaining stationary. This task could by achieved by utilizing the Python **PyMAVLink** library. PyMAVLink is a Python library that provides a convenient interface for interacting with MAVLink protocol-based autopilots. It allows for the creation, sending, and receiving of MAVLink messages, facilitating communication between GCSs, drones, and other MAVLink-compatible devices. PyMAVLink is widely used for telemetry data retrieval, mission planning, and drone control scripting.

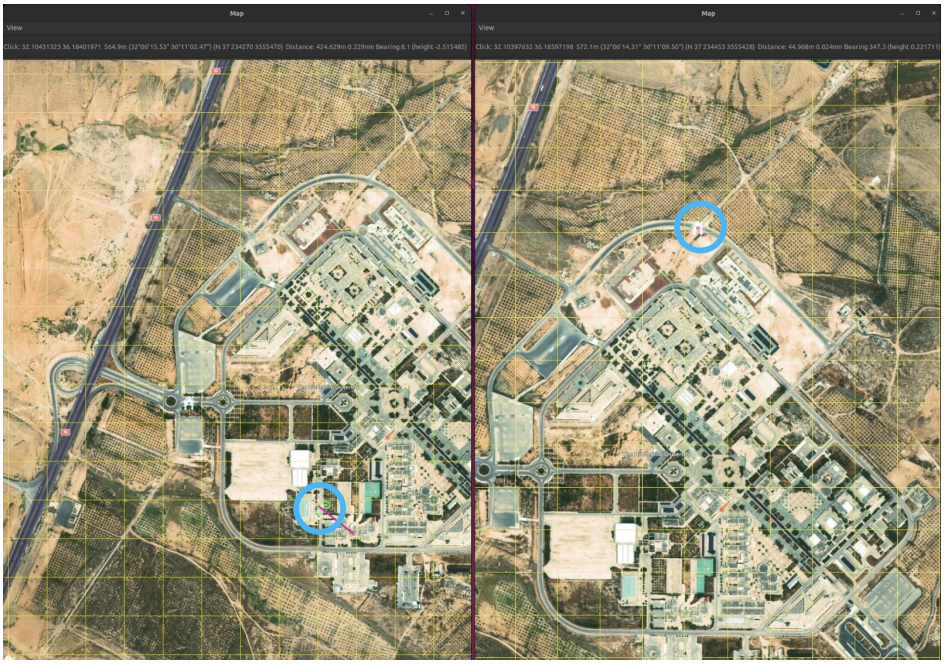


Figure 5.7: The closer drone starts moving with the other remaining stationary.

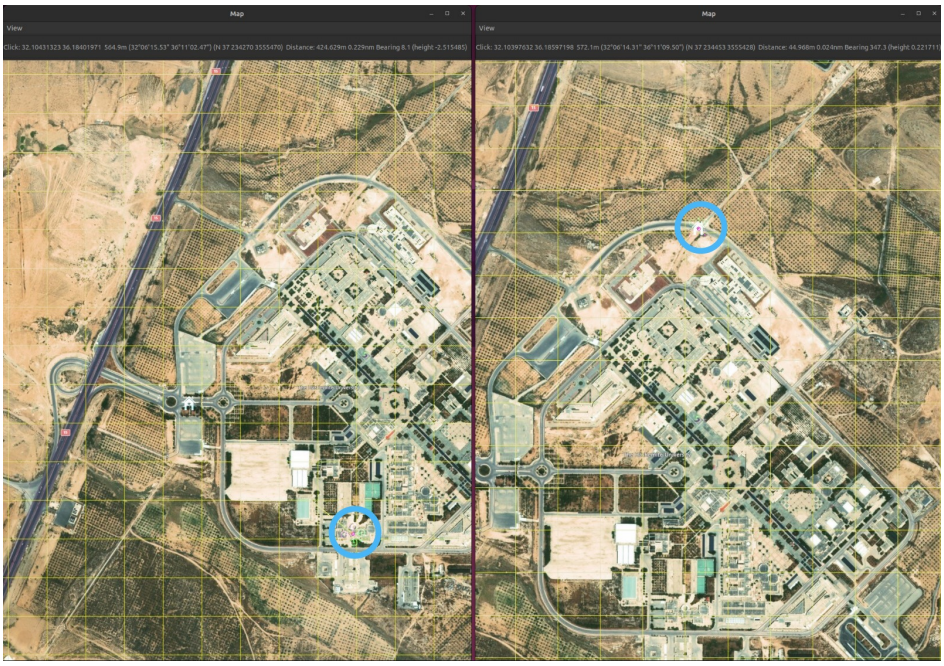


Figure 5.8: The closer drone arrives.

5.6 ROS2

5.6.1 ROS2 Introduction

The Robot Operating System (ROS2) is an advanced, open-source middleware framework design to facilitate the development of robotic systems. It serves as a robust platform for building modular, scalable, and distributed robotic applications. By enabling seamless communication between various software components, ROS2 allows us to focus on functionality without being constrained by inter-process communication.

The ROS graph is a network of ROS2 elements processing data together at the same time. It encompasses all executables and the connections between them. These executables are called **nodes** and they represent the basic unit of computation in ROS2. Nodes are typically responsible for one, modular purpose such as controlling drones or processing camera input.

5.6.2 Communication Between Nodes

Nodes can communicate with other nodes within the same process, a different process, or a different machine by exchanging data using the followsing primay communication paradigms:

1. **Topics (Publish/Subscribe)**: Nodes can send or receive messages via topics. A publisher node sends message to a topic, while a subscriber node listens to that topic for updates. for example, a drone node could publish its GPS data on a GPS topic, while the swarm coordination node subscribers to that topic to track that drone's location in real time.

2. **Services (Request/Response)**: For two-way communication, ROS2 provides services, where one node sends a request, and another node processes that request and sends back a response. This is useful for actions requiring immediate feedback, such as sending arm commands to drones and awaiting acknowledgment.

3. **Actions (Goal/Feedback/Result)**: Actions enable long running tasks, allowing nodes to set a goal, receive periodic feedback, and get the final result. For example, a mission control node might send a drone the goal of exploring a specific area and receive status updates as the task progresses.

5.6.3 The DDS Middleware

At the core of ROS2's communication system lies the Data Distribution Service (DDS), a middleware that manages data exchange between nodes. DDS enables ROS2 to support real-time, decentralized, and reliable communication.

The DDS protocol allows nodes to dynamically discover each other on the network without requiring a centralized master node. This decentralized discovery process enhances fault tolerance and simplifies deployment.

DDS allows for fine-grain control over the communication's reliability, latency, and bandwidth through QoS policies. For example, critical drone data , such as velocity commands, can be configured with high reliability and low latency to ensure timely and accurate execution.

One of DDS's strengths is its flexibility in supporting multiple implementations. These implementations differ in terms of features, performance, and licensing. We will expand more on our choice of implementation in the following section.

5.6.4 ROS2 Installation

For our project, we chose ROS2 Humble Hawksbill as the primary version of ROS2 to work with. ROS2 Humble is a Long-Term Support (LTS) release, ensuring support until 2027. ROS2 Humble was selected because it offers the latest advancements in the ROS2 ecosystem while maintaining compatibility with our chosen technologies.

The installation of ROS2 Humble was carried out on devices running Ubuntu 22.04, the recommended operating system for this version. Following the official ROS2 documentation, the steps included:

1. **System Preparation**: Adding the ROS2 repository, updating system dependencies, and setting up the required keys.

2. **Installing the Base Package**: Installing the ros-humble-desktop package to provide access to the core tools, visualization utilities (like Rviz2), and simulation capabilities (like Gazebo).

3. **Environment Configuration**: Setting up the ROS2 environment by sourcing the setup.bash file, which allows easy execution of ROS2 commands and integration with our development tools.

5.7 Micro XRCE-DDS

In our project, certain components, such as individual drones, operate on resource-constrained hardware with limited processing power, memory, and network bandwidth. To address these constraints, we're using eProsima's Micro XRCE-DDS (eXtremely Resource-Constrained Environment DDS)[10], a lightweight implementation of the Data Distribution Service (DDS) standard, specifically designed for embedded systems with minimal resources.

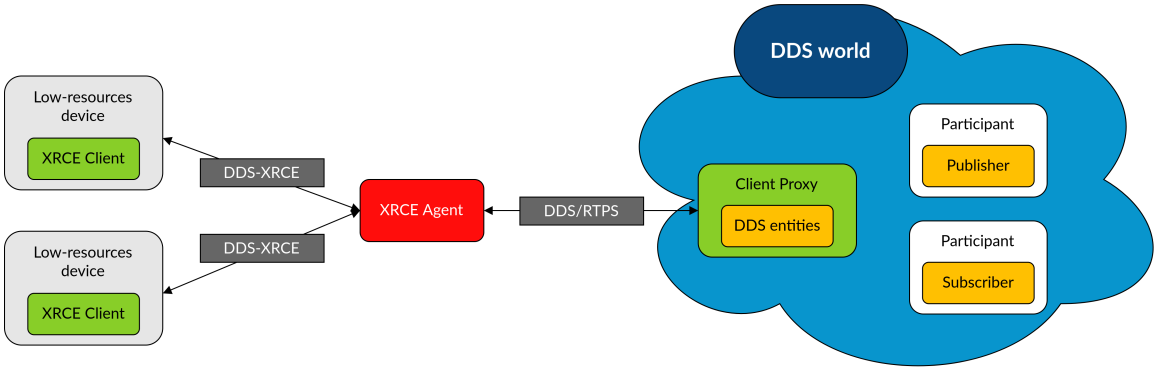


Figure 5.9: Micro XRCE-DDS.[10]

Micro XRCE-DDS adopts a client/server model and comprises two key components: the Micro XRCE-DDS Client and the Micro XRCE-DDS Agent. The clients are lightweight libraries designed to operate in highly resource-constrained environments, while the agent acts as a bridge, connecting the clients to the broader DDS network.

Clients interact with the agent to perform operations such as publishing and subscribing to topics within the DDS global dataspace. They also support remote procedure calls (RPC) as specified by the DDS-RPC standard, enabling communication through a request/reply mechanism. The agent processes these requests, returns the status of the operations, and, for subscription or reply-based actions, provides the requested data.

5.7.1 How Micro XRCE-DDS Works

The Micro XRCE-DDS architecture enables seamless communication between resource-constrained devices (clients) and the larger DDS network via the agent. The diagram 5.10 illustrates this architecture and its key components.

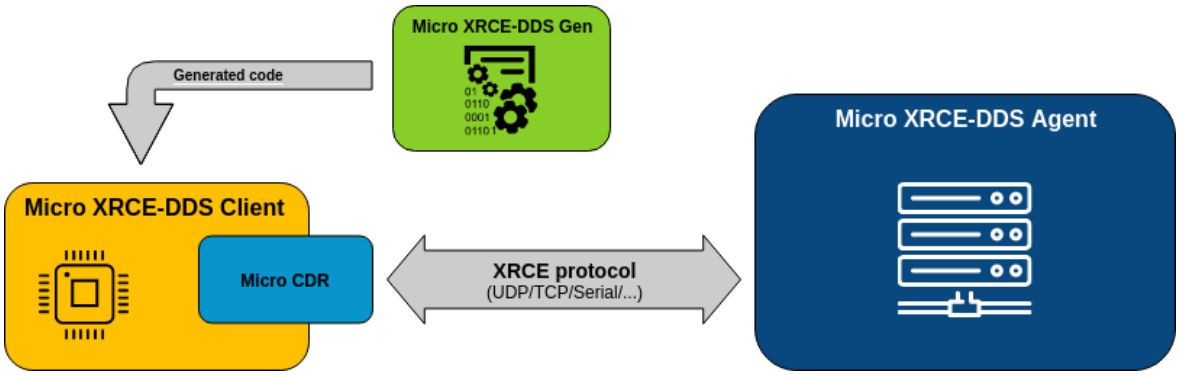


Figure 5.10: General Micro XRCE-DDS Architecutre.[22]

1. **Micro XRCE-DDS Gen**: The Micro XRCE-DDS Generator (Gen) is a tool that automates the creation of client-side code. It takes user-defined data types and communication structures as input and generates optimized code that can run on constrained devices. This simplifies the development process, allowing us focus on functionality without worrying about low-level implementation details.

2. **Micro XRCE-DDS Client**: The client is a lightweight software library that runs on resource-constrained devices, such as drones. It uses the generated code from Micro XRCE-DDS Gen to interact with the DDS network. The client handles basic operations like sending data (publishing) or requesting data (subscribing) while minimizing resource usage.

3. **Micro XRCE-DDS Agent**: The agent acts as a bridge between the client and the larger DDS network. It translates the client's lightweight XRCE protocol messages into standard DDS communication, allowing the client to participate in the DDS global dataspace. The agent processes requests from the client, and relays the responses back to the client.

4. **Communication Protocol**: The client and agent communicate using the XRCE protocol, which supports multiple transport layers such as UDP, TCP, or serial communication.

5.7.2 Micro XRCE-DDS Installation & Configuration

The following steps showcase the general steps we took for installing and configuring Micro XRCE-DDS for our ROS2 workspace:

1. Installing the Java Runtime Environment, which is required for building the Micro XRCE-DDS generator.

2. Cloning the Micro XRCE-DDS Gen repository.

3. Building the generator. This step compiles the required tools to create the client-side code for Micro XRCE-DDS.

5.8 MAVROS

MAVROS is a middleware package that acts as a bridge between the ROS2 ecosystem and the ArduPilot flight stack. It allows for seamless communication between robotic systems running ROS2 and drones controlled by ArduPilot by utilizing the MAVLink protocol. This integration enables ROS2 nodes to send commands to and receive telemetry data from drones.

MAVROS provides interfaces for issuing high-level commands to drones, such as setting waypoints, adjusting flight modes, or commanding takeoff and landing. Additionally, ROS2 nodes can subscribe to telemetry data from drones, including GPS location, battery levels, velocity, and orientation.

5.8.1 MAVROS Installation

To install MAVROS, we followed the following general steps:

1. Installing the main MAVROS package along with its supplementary package, MAVROS Extras, which includes additional plugins and functionalities.

2. Installing GeographicLib datasets. MAVROS relies on GeographicLib datasets to handle geographic and geospatial calculations, such as transforming GPS coordinates.

5.9 Hadoop

Drone swarms can generate very large amounts of data, which may need to be store for further analysis or later use. The storage solution must be able to handle large throughput while efficiently storing and distributing the data. That's where Hadoop comes in.

Hadoop is an open-source framework that facilitates the distributed storage and processing of massive datasets across clusters of computers. It consists of two core components: the Hadoop Distributed File System (HDFS), which handles the storage by breaking data into blocks and distributing them across nodes, and the MapReduce programming model, which processes the data in parallel across the cluster. Hadoop is designed to scale horizontally by adding more nodes to the cluster and ensures fault tolerance through data replication, making it ideal for managing the large, complex datasets generated by drone swarms.

5.9.1 Hadoop Installation

To install and configure Hadoop, we followed the steps provided by the official documentation. For demonstration purposes, we installed Hadoop in pseudo-distributed mode to simulate the cluster environment while remaining on a single machine. The installation process consists of the following general steps:

1. Installing the Java Development Kit alongside Java Runtime Environment, which was already installed for Micro XRCE-DDS.

2. Configuring SSH.

3. Configuring environment variables and Hadoop settings.

4. Formatting HDFS and starting Hadoop services

5.10 Integration of Hadoop with ROS2

This section demonstrates the integration of Hadoop's HDFS with ROS2 to enable real-time transfer and storage of image data. The system is designed to offer seamless communication between multiple drones and a centralized data storage cluster, ensuring scalability and robustness for applications requiring large-scale data handling. This setup not only highlights the potential of combining distributed storage systems with robotics but also provides a foundation for real-time image analysis workflows.

The integration begins with the ROS2 Publisher Node, deployed as the ImagePublisher on each drone. This node reads image data from a local directory using OpenCV and publishes it as sensor_msgs/Image messages via ROS2 topics. Each drone is assigned a unique topic, such as /camera/Image1, /camera/Image2, and /camera/Image3, ensuring organized data streams. The modular design of the publisher node makes it adaptable to additional sensors or data sources, enhancing the flexibility of the system.

The ROS2 Listener Node, referred to as the HDFSListener, subscribes to the published topics and manages the transfer of image data to Hadoop's HDFS. Upon receiving the images, the listener node converts them into HDFS-compatible formats and utilizes Python's HDFS client library to store them in the cluster. The diagram 5.11 illustrating how data flows from drones to the HDFS cluster through the ROS2 communication framework.

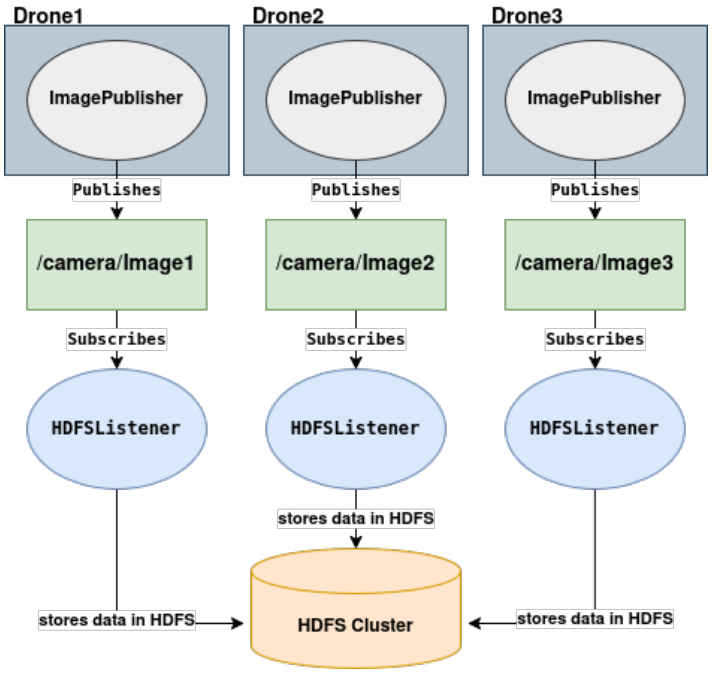


Figure 5.11: Hadoop integration flow.

The system enables centralized storage of drone-captured images, paving the way for future enhancements such as real-time preprocessing before storage or expanding the pipeline to support additional data modalities.

5.11 YOLOv11

YOLO (You Only Look Once) is a family of real-time object detection models known for their speed and accuracy. YOLOv11, the latest iteration, builds upon its predecessors by introducing improved architecture ensuring higher detection accuracy with reduced computational overhead. This makes it particularly suitable for applications requiring real-time inference on edge devices, such as drones in a swarm.

For this project, we chose the YOLOv11-S (Small) variant due to its balance between computational efficiency and detection performance. Drones operate under strict resource constraints. The small-size model allows the drones to run object detection tasks efficiently while maintaining a sufficient level of accuracy for object detection.

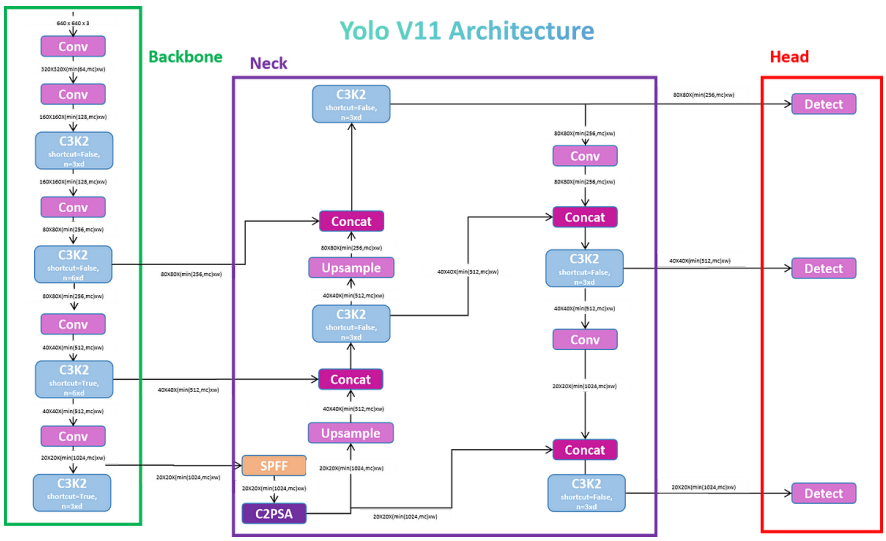


Figure 5.12: YOLOv11 model architecture[19].

5.11.1 Implementation in the Drone Swarm

Each drone in the swarm is equipped with its own YOLOv11 node, implemented as a ROS2 component for modularity and integration with other system components. The YOLO node is responsible for processing the drone's onboard camera feed and publishing real-time object detection results.

The YOLO node is designed to process the drone's camera feed in real time. It subscribes to the camera feed, which is published as a ROS2 topic, and uses the YOLOv11-S model to analyze incoming frames. This analysis generates detection results, including object classes, bounding box coordinates, and confidence scores. The processed results are then published to a dedicated ROS2 topic, making them accessible to other components within the system.

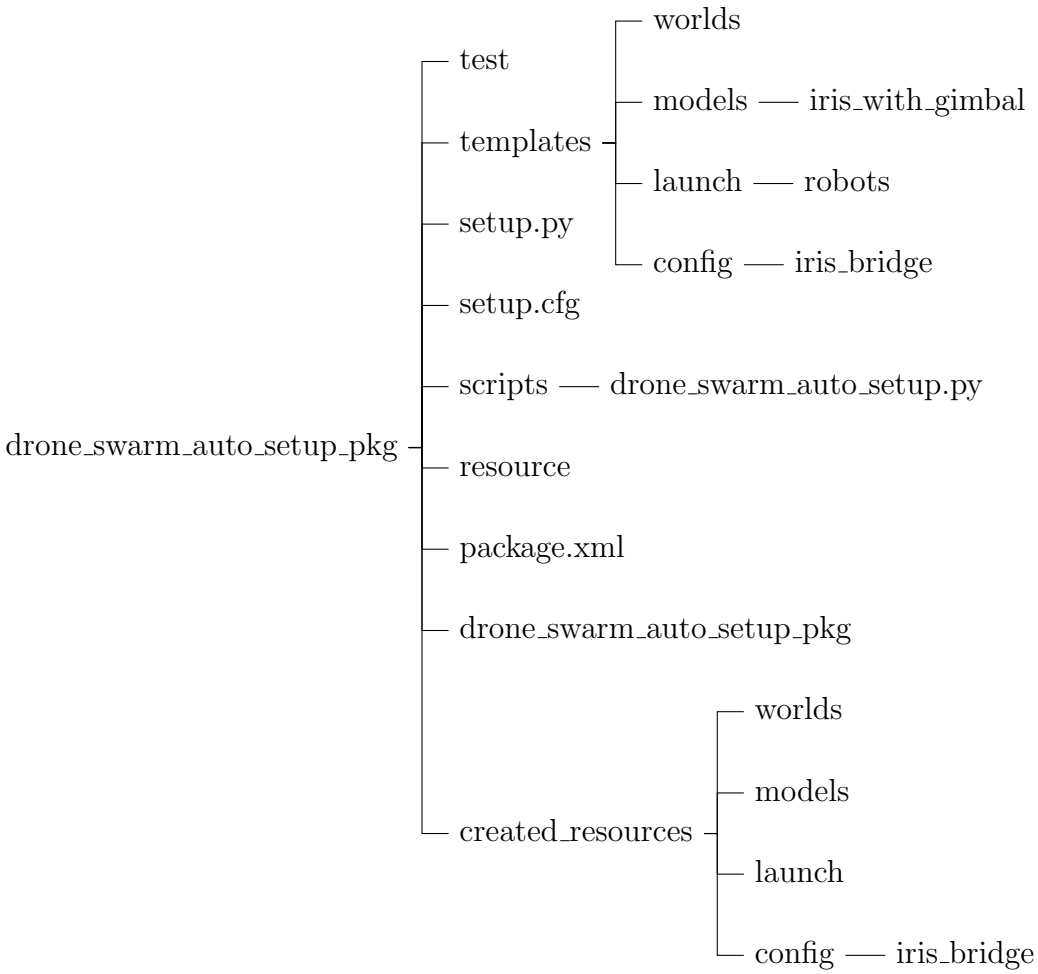
The YOLO node's predictions are streamed as a ROS2 topic in a structured format. Each message a list of detected objects with class labels, bounding boxes, and confidence scores. The detection results can be displayed in RViz, a powerful visualization tool in ROS2, to monitor the drone's real-time object detection performance. And for

long-term analysis, the detection stream is stored in Hadoop Distributed File System (HDFS). A dedicated ROS2 node subscribes to the YOLO prediction topic, converts the data into a suitable format, and uploads it to the HDFS cluster.

5.12 Drone Swarm Auto Setup Package

The `drone_swarm_auto_setup_package` is a robust and efficient solution for automating the configuration of drone swarms. This package generates the necessary resources and configurations to streamline the setup process, enabling the deployment of multiple drones with minimal manual intervention. It supports seamless integration with ROS2, ArduPilot SITL, and Gazebo simulation environments.

Additionally, the `drone_swarm_environment_setup.sh` is a comprehensive tool designed to streamline the setup of the development environment for the drone swarm framework. This shell script automates the installation and configuration of essential components, including ROS2 Humble, ArduPilot SITL, and the Gazebo simulation environment. It also installs all necessary dependencies for MAVROS and the ArduPilot-Gazebo plugin. Also, the script creates the required directory structure, sets up environment variables, and configures project-specific templates and integrations.



1. **Templates**
   Templates provide pre-configured files for various resources required by the drone swarm:

   - **Config**: YAML configurations for ROS-Gazebo bridges (e.g., `iris_bridge.yaml`).
   - **Launch**: Launch files for individual drones and the swarm (e.g., `drone_swarm_launch.py`).
   - **Models**: Gazebo-compatible drone models (e.g., `iris_with_gimbal`).
   - **Worlds**: Gazebo simulation environments (e.g., `iris_runway.sdf`).

2. **Created Resources**
   The `created_resources` directory is dynamically populated during the setup process. It includes:

   - **Config**: Generated YAML files for each drone's bridge.
   - **Launch**: Generated launch files for the swarm.
   - **Models**: Customized drone models with unique configurations.
   - **Worlds**: Simulation worlds updated with the specified drone swarm.

3. **Scripts**
   `drone_swarm_auto_setup.py`
   This script automates the setup of the drone swarm. It creates unique configurations for each drone in the swarm, Generates Gazebo models with unique parameters (e.g., `fdm_port_in`). Then it updates the Gazebo world to include all drones. And finally, it creates unique ROS-Gazebo bridge configurations.

6 — Reinforcement Learning Agent Development & Integration

This chapter explains the role of the reinforcement learning (RL) framework employed in the project, alongside the specific RL algorithm used. The focus is on how the RL agents were designed, trained, and integrated into the drone swarm system to achieve optimal pathfinding capabilities.

In this project, the RL agents play a critical role in pathfinding for the swarm. Starting from a centralized spawn point, each drone in the swarm is assigned a unique target located within houses. The RL agent on each drone is responsible for determining the optimal path to reach its target, navigating through the complex environment of houses.

The primary objective of this chapter is to showcase how the RL agents were developed to autonomously find optimal paths, demonstrating their capability to navigate challenging environments effectively. This functionality directly supports the broader project goal of achieving autonomous drone swarm control, where drones operate with minimal human intervention.

6.1 Reinforcement Learning Basics

Reinforcement Learning (RL) is a type of machine learning where agents learn to make decisions by interacting with their environment. The agent receives feedback in the form of rewards or penalties, allowing it to learn strategies that maximize cumulative rewards over time. In this project, RL is the key mechanism enabling drones to navigate autonomously through complex environments.

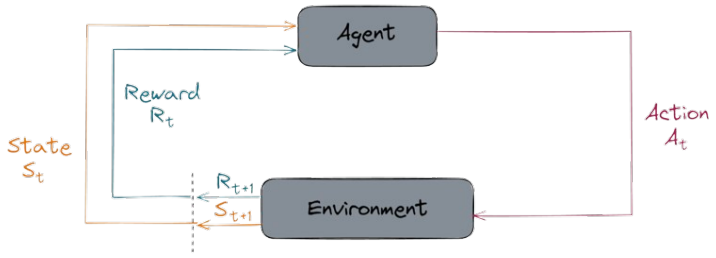


Figure 6.1: The Markov Decision Process

A tailored approach, termed Centralized Planning with Decentralized Execution (CPDE), is utilized to coordinate the swarm. In this framework, task allocation is handled centrally by a simple greedy algorithm that assigns targets to drones based on simple heuristics (e.g. smallest Manhattan distance). This centralized planner uses the global knowledge of the environment. Once targets are assigned, each drone operates independently, using its RL agent to determine the optimal path to its designated target.

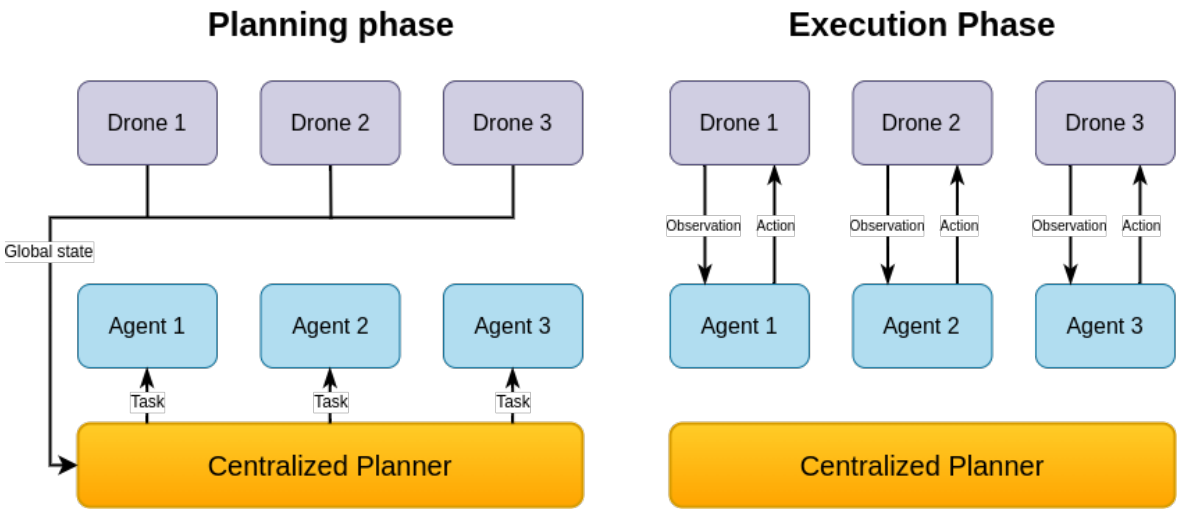


Figure 6.2: CPDE basic workflow

The RL algorithm chosen for this project is Advantage Actor-Critic (A2C). A2C is a policy-gradient algorithm that uses two networks: an actor network, which decides the actions the agent should take, and a critic network, which evaluates the quality of those actions. This approach was selected for its balance between computational efficiency and robust performance in discrete action spaces.

To implement and train the RL agents, we used OpenAI Gymnasium to define the simulation environment and Stable Baselines3 for RL algorithm implementations. These tools provided a structured framework for developing and testing the agents.

Several critical RL concepts were applied during development:

1. **Learning Curriculums**: The RL agents were trained in progressively complex environments, starting with simple obstacle-free scenarios and gradually introducing challenges like narrow corridors. This approach enabled the agents to build foundational skills before tackling more difficult tasks.

2. **Exploration vs. Exploitation**: Balancing exploration (trying new actions to discover better strategies) and exploitation (using known strategies to maximize rewards) was crucial for effective learning.

3. **Reward Shaping**: The reward function was carefully designed to encourage

efficient pathfinding while penalizing collisions or excessive energy use. Reward signals were refined iteratively to ensure that agents learned desired behaviors.

6.2 Design of the RL Agent

The design of the reinforcement learning (RL) agent in this project focuses on enabling efficient pathfinding for drones to locate targets hidden within houses. The primary challenge lies in navigating environments where house walls serve as obstacles, making it essential for the agent to avoid collisions while finding the shortest path to its target.

6.2.1 Problem Definition

The RL agent's task is to determine an optimal path to its assigned target, navigating around obstacles and minimizing unnecessary movements. The problem is made unique by the need to avoid house walls and by the hidden nature of the target, which requires intelligent decision-making at each step, accounting for long term rewards.

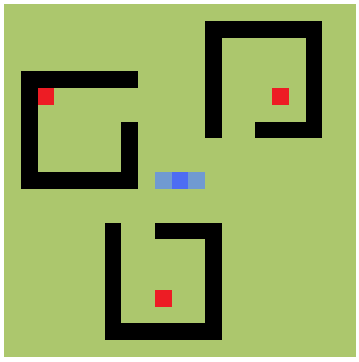


Figure 6.3: grid environment for the agents. The 3 blue points in the center are the drones at their launch position, and the red dots are the targets

6.2.2 State Space

The state space (or observation space) defines the information available to the agent at any given moment, allowing it to perceive its environment and make decisions. For our reinforcement learning setup, the state space for each agent consists of the following components:

The state space for each agent comprises:

1. **Agent's position**: The current (x, y) coordinates of the drone.
2. **Target's position**: The (x, y) coordinates of the assigned target.

Additionally, a complete representation of the environment, including walls and open spaces, was provided to the agents during training. Ideally, we would have preferred for each drone to simultaneously find its optimal policy while also mapping the environment. But that goes well beyond the scope of our project.

These inputs were chosen to strike a balance between providing the agent with sufficient information for effective decision-making and maintaining computational efficiency. By minimizing unnecessary or redundant information, we ensure that the agent can process its state space efficiently while still being capable of navigating complex environments and completing assigned tasks.

6.2.3 Action Space

The action space defines the set of all possible actions that the agent can take at any given state. For our reinforcement learning agent, a discrete action space was chosen, meaning the agent selects from a predefined set of distinct actions. This discrete approach simplifies both the implementation and the training process while still enabling versatile behavior.

The available actions in the action space allow the agent to take a step into any cell in its neighboring window (up, down, right, left, up-right, up-left, down-right, down-left).

This discrete action space was selected for its simplicity, reducing computational complexity while still offering sufficient maneuverability for effective navigation within the environment. Unlike continuous action spaces, which allow an infinite range of actions and require complex function approximations, discrete action spaces are computationally efficient and easier to integrate with standard reinforcement learning algorithms.

6.2.4 Reward Function

The reward function was designed to guide the agent's learning process by incentivizing desirable behaviors and penalizing undesired ones. This helps the agent learn efficient pathfinding strategies while avoiding redundant or harmful actions. The table 6.1 summarizes the reward structure:

Table 6.1: Reward function structure for the RL agent.

Trigger	**Reward**	**Explanation**
Each step taken	-1	Encourages the agent to find optimal paths.
Revisiting the same cell	-2	Discourages redundant paths to improve search efficiency and avoid wasting steps.
Attempting to move into a wall	-3	Penalizes collision attempts to reinforce obstacle avoidance.
Moving closer to the target	$+0.5 \times (\text{old\_distance} - \text{new\_distance})$	Rewards the agent for reducing its distance from the target and penalizes moving further away.
Successfully reaching the target	$+100$	Strongly incentivizes completing the task to ensure the agent prioritizes reaching the target.

This reward function was iteratively refined to balance penalties and rewards effectively. The system motivates the agent to adopt strategies that minimize unnecessary actions, avoid collisions, and efficiently navigate toward the target, ultimately improving overall performance and decision-making.

6.3 RL Agent Development

The development of the reinforcement learning (RL) agent was centered on using widely recognized frameworks and designing a neural network to ensure effective learning and consistent performance. The project utilized the following tools and frameworks:

1. **OpenAI Gymnasium**
   Served as the simulation environment, providing a grid-based world where the agent could interact and learn. Gymnasium was chosen for its flexibility, ease of customization, and wide adoption in the RL research community.

2. **Stable Baselines3 (SB3)**
   Used to implement the RL algorithms. SB3 is a popular library that supports multiple RL algorithms, offering well-tested implementations and extensive documentation. Its integration with Gymnasium enabled a seamless development experience and expedited troubleshooting.

6.3.1 Model Architecture

The RL agent's neural network employs a fully connected architecture, shared by both the policy (actor) and value (critic) functions. This design ensures computational efficiency and effective learning. The architecture includes:

1. **Input Layer**
   Processes the agent's state representation, which includes the agent's position, the target's position.

2. **Hidden Layers**
   Two fully connected layers, each with 64 units, use ReLU (Rectified Linear Unit) activation functions.

3. **Output Layer**
   For the actor, the output layer generates action probabilities for the discrete action space. For the critic, the output layer provides an estimate of the value function, which evaluates the expected future rewards from a given state.

This architecture was chosen for its simplicity and effectiveness. We were able to maintain computational efficiency with this architecture which is crucial for drone swarm environments.

6.3.2 Training Process

The RL agent was trained in a simulated, grid-based environment implemented using Gymnasium. The training process followed an episodic design, where each episode began with the agent at a predefined starting position. The agent's objective was to navigate to the target within a fixed number of steps. Key hyperparameters used during training are shown in table 6.2

Table 6.2: Hyperparameters used during RL agent training.

Hyperparameter	**Value**	**Explanation**
Learning rate	0.0007	Controls the step size during gradient descent.
Discount factor (γ)	0.8	Balances immediate vs. future rewards.
Entropy coefficient (ent_coef)	0.001	Encourages exploration by adding randomness to the policy.
Value function coefficient (vf_coef)	0.5	Balances the weight of the value function loss in the total loss function.
Max gradient norm	0.5	Limits the magnitude of gradients to prevent instability during training.
Number of steps per update (n_steps)	20	Specifies the number of steps collected before updating the policy.

These hyperparameters were tuned iteratively to ensure stability during training and to encourage an optimal balance between exploration and exploitation.

6.3.3 Challenges and Solutions

During development, the agent encountered significant difficulty navigating through doors into houses, as these tasks required precise spatial understanding.

To address this, a curriculum learning approach was adopted. The agent was first trained to complete a simpler task of reaching the door. This allowed it to develop basic navigation skills and an understanding of the environment layout. Once proficient at reaching doors, the training task was extended to include navigating inside houses to locate the target. This incremental learning strategy enabled the agent to gradually adapt to more complex scenarios.

By breaking down the task into manageable stages, the agent was able to build foundational skills before tackling the full problem, resulting in improved performance and reliability.

6.4 Integration with the Drone Swarm System

Integrating the reinforcement learning (RL) agent into the drone swarm system involved establishing seamless communication, deploying the trained RL model, and ensuring effective operation in both simulation and real-world environments.

6.4.1 Communication Setup

All communication within the system is managed using ROS2, with MAVROS serving as the bridge between the ROS2 nodes and the drones. MAVROS facilitates the exchange of positional data and command execution. This setup allows the RL agents to receive real-time position updates and send movement commands to the drones.

6.4.2 Deployment Pipeline

The deployment of the RL agent follows a centralized-to-decentralized process:

1. A centralized Drone Controller Node in ROS2 identifies all active drones in the system.
2. The controller instantiates a unique RL agent instance for each drone and assigns a target based on the task allocation algorithm.
3. The RL agent communicates with the drone through MAVROS:
    - **Inputs**: The drone's current state (drone position and target position), obtained via MAVROS.
    - **Outputs**: The agent's action predictions are translated into MAVROS commands, directing the drone to take a step in a specific direction.

The system also includes a command override mechanism to handle critical situations:

- If an obstacle is detected, the drone ignores RL agent commands to avoid collisions.
- When the battery level drops below 10%, the drone automatically returns to its launch point.
- Upon detecting a person (target) using the YOLO, the drone immediately lands to signify task completion.

6.4.3 Simulation-to-Real Transfer

The RL agent was trained in a simplified 2D grid simulation, while the real-world implementation operates in a 3D environment simulated in Gazebo. Despite the 3D complexity, the project treats the environment as effectively 2D for decision-making purposes, reducing the gap between the training and deployment domains.

6.5 Testing and Evaluation

The testing and evaluation phase focused on observing the performance of the reinforcement learning (RL) agents in both training and simulation environments. The training process and results provided key insights into the agents' capabilities and limitations.

6.5.1 Evaluation During Training

Two primary performance indicators were tracked during training:

Mean Episode Length: The average number of steps taken per episode. This metric decreased exponentially as training progressed, eventually plateauing, indicating that the agents were learning to minimize unnecessary steps and find more efficient paths.

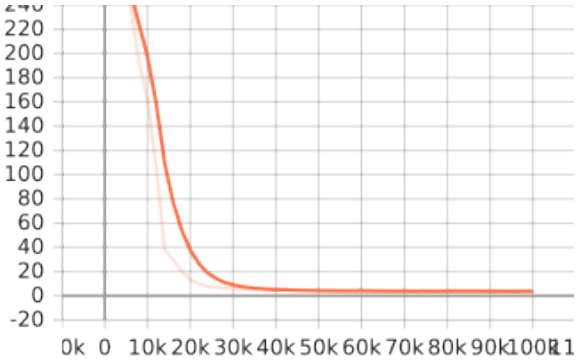


Figure 6.4: Average number of steps per episode over all timesteps plot.

Mean Rewards per Episode: The average reward accumulated per episode. This metric increased exponentially and plateaued as the agents learned to maximize rewards by reaching their targets efficiently while avoiding penalties.

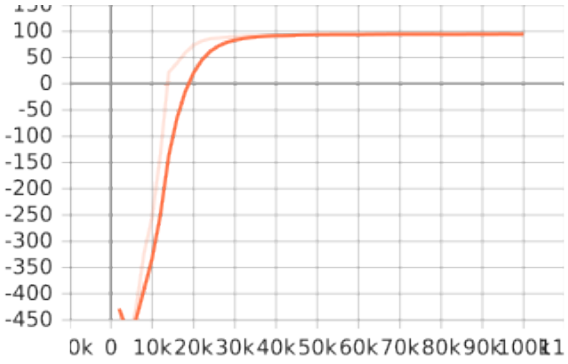


Figure 6.5: Average rewards per episode over all timesteps plot.

6.5.2 Simulation Results

The RL agents were tested in the 2D grid simulation environment, where they consistently exhibited excellent performance. Agents reliably selected the optimal paths to their targets, minimizing travel distance and avoiding obstacles. The behavior patterns observed in the simulation confirmed the agents' ability to effectively balance exploration and exploitation during navigation tasks. The figure 6.6 showcases a single agent moving towards a single target with our trained policy.

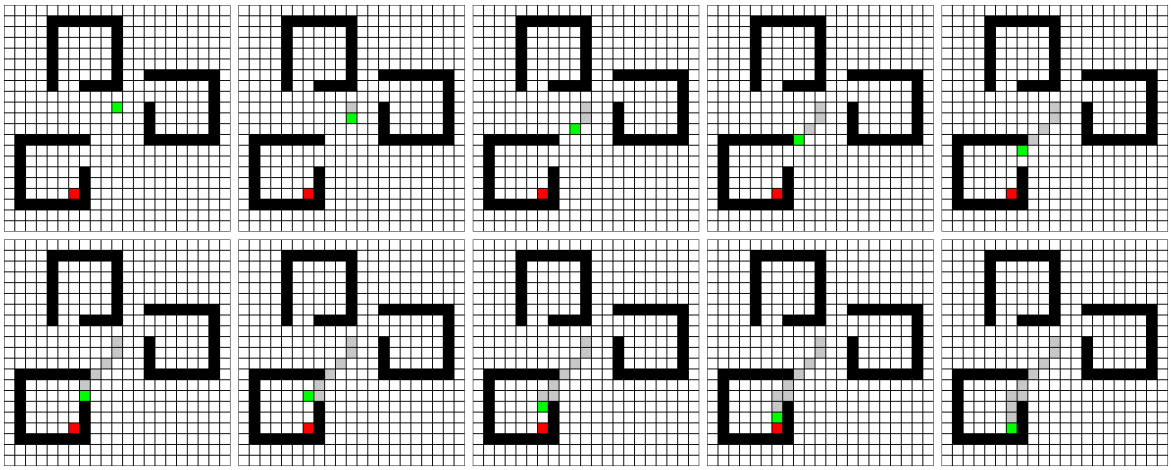


Figure 6.6: Single agent going towards a target.

6.5.3 Limitations

Despite their strong performance, the RL agents have some limitations. The current design does not account for obstacles that appear dynamically during execution. Agents are limited to navigating within a static environment, and incorporating dynamic obstacle handling would improve adaptability. Additionally, While the 2D grid environment simplifies training and deployment, extending the agent's decision-making capabilities to account for real-world 3D complexities could enhance robustness.

7 — Conclusion

7.1 Project Summary

This project explored the development and coordination of a drone swarm system. By utilizing a centralized planning and decentralized execution approach, we aimed to enhance the efficiency and reliability of multi-drone systems for real-world applications.

We began by establishing foundational knowledge about drones, their applications, and the motivations for developing swarm coordination systems. We reviewed existing technologies and identified their limitations, highlighting gaps this project sought to address. The preparation of the development environment included the integration of various tools, such as ArduPilot, MAVProxy, Gazebo, ROS2, and Hadoop, ensuring a robust simulation and testing framework.

The reinforcement learning (RL) agent was designed to enable autonomous coordination of the drone swarm. The agent's state space aggregated critical environmental and drone-specific data, while its action space provided a set of commands for effective navigation and task execution. We defined a reward function tailored to the objectives of path finding.

During development, we implemented and tested the RL agent's model architecture and training pipeline, addressing several challenges such as optimizing training stability and ensuring simulation-to-real transferability. The integration of the RL agent with the drone swarm system was a significant milestone, involving communication setup and deployment pipelines. Testing and evaluation demonstrated promising simulation results, despite acknowledged limitations related to dynamic environments and marking revisited targets.

7.2 Future Work

The completion of this project opens multiple avenues for further research and development:

1. While this project focused on static environments, incorporating dynamic elements such as moving obstacles or evolving disaster scenarios would enhance the system's adaptability.

2. Implementing mechanisms to mark and remember previously visited targets would improve search efficiency and reduce redundant exploration.

3. Including additional parameters like wind speed, temperature, and terrain elevation could enable the RL agent to make more informed decisions.

4. Transitioning from simulation to real-world implementation is critical for validating the system's effectiveness. This includes hardware testing, field trials, and integrating feedback from disaster response teams.

5. Investigating scalability to larger drone swarms with more diverse capabilities could expand the system's applicability to a broader range of missions.

6. Improving energy efficiency by optimizing flight paths and task assignments would prolong mission durations and reduce operational costs.

7. Exploring decentralized learning approaches, where drones share experiences in real time, could further enhance the swarm's collective intelligence.

This project represents a significant step toward advancing drone swarm technology for disaster response. All source code, documentation, and resources will be made available on our project website here: https://drone-swarm-project-hu.web.app/

Bibliography

[1] Commercial drone market. https://www.fortunebusinessinsights.com/commercial-drone-market-102171.

[2] Unmanned aerial vehicles (uav) market. https://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html.

[3] ArduPilot Development Team. *ArduPilot Documentation*, 2024. URL https://ardupilot.org/. Accessed: 2024-05-15.

[4] Godwin Asaamoning, Paulo Mendes, Denis Rosário, and Eduardo Cerqueira. Drone swarms as networked control systems by integration of networking and computing. *Sensors*, 21(8), 2021. ISSN 1424-8220. doi: 10.3390/s21082642. URL https://www.mdpi.com/1424-8220/21/8/2642.

[5] Inphoenix Aviation. Comparative analysis of uav ground control stations: Mission planner, qgroundcontrol, ugcs, betaflight, and inav. *Inphoenix Aviation*, 2024. URL https://inphoenixaviation.com.

[6] Tomas Baca, Matej Petrlik, Matous Vrba, Vojtech Spurny, Robert Penicka, Daniel Hert, and Martin Saska. The mrs uav system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles. *Journal of Intelligent amp; Robotic Systems*, 102 (1), April 2021. ISSN 1573-0409. doi: 10.1007/s10846-021-01383-5. URL http://dx.doi.org/10.1007/s10846-021-01383-5.

[7] Jovan Boskovic, Nathan Knoebel, Nima Moshtagh, Jayesh Amin, and Gregory Larson. Collaborative mission planning autonomous control technology (compact) system employing swarms of uavs. 08 2009. ISBN 978-1-60086-978-5. doi: 10.2514/6.2009-5653.

[8] Civil Aviation Regulatory Commission. Jordanian civil aviation regulation: Unmanned aircraft systems and operations. https://carc.gov.jo/pdf/CARC_UAV_Operations2.docx, October 2019.

[9] Ebrahimi and John Page. Uav swarm search strategy applied to chaotic ship wakes. 02 2013.

[10] eProsima. Micro xrce-dds. https://github.com/eProsima/Micro-XRCE-DDS, 2024. URL https://github.com/eProsima/Micro-XRCE-DDS. Accessed: 2024-12-27.

[11] Aleksandar Erceg, Biljana Činčurak Erceg, and Aleksandra Vasilj. Unmanned aircraft systems in logistics – legal regulation and worldwide examples toward use in croatia. 10 2017.

[12] Miguel Fernandez-Cortizas, Martin Molina, Pedro Arias-Perez, Rafael Perez-Segui, David Perez-Saura, and Pascual Campoy. Aerostack2: A software framework for developing multi-robot aerial systems, 2023.

[13] MAVLink Development Team. *MAVLink Micro Air Vehicle Communication Protocol*, 2024. URL https://mavlink.io/en/. Accessed: 2024-05-15.

[14] Ivana Palunko, Rafael Fierro, and Patricio Cruz. Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach. *2012 IEEE International Conference on Robotics and Automation*, pages 2691–2697, 2012. URL https://api.semanticscholar.org/CorpusID:18475577.

[15] Lenka Pitonakova. Foraging strategies in nature and their application to swarm robotics. 2013. URL https://api.semanticscholar.org/CorpusID:163161196.

[16] PX4 Development Team. *PX4 Autopilot User Guide*, 2024. URL https://docs.px4.io/main/en/. Accessed: 2024-05-15.

[17] Intelligent Quads. Intelligent quads: Autonomous drones and swarm robotics. https://intelligentquads.com. Accessed: 2024-05-16.

[18] Aravinda S. Rao, Marko Radanovic, Yuguang Liu, Songbo Hu, Yihai Fang, Kourosh Khoshelham, Marimuthu Palaniswami, and Tuan Ngo. Real-time monitoring of construction sites: Sensors, methods, and applications. *Automation in Construction*, 136:104099, 2022. ISSN 0926-5805. doi: https://doi.org/10.1016/j.autcon.2021.104099. URL https://www.sciencedirect.com/science/article/pii/S0926580521005501.

[19] S. Nikhileswara Rao. Yolov11 architecture explained: Next-level object detection with enhanced speed and accuracy, October 22 2024. URL https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2d Accessed: 2025-01-04.

[20] Shanza Shakoor, Zeeshan Kaleem, Muhammad Iram Baig, Omer Chughtai, Trung Q. Duong, and Long D. Nguyen. Role of uavs in public safety communications: Energy efficiency perspective. *IEEE Access*, 7:140665–140679, 2019. doi: 10.1109/ACCESS.2019.2942206.

[21] Daniel H. Stolfi, Matthias R. Brust, Grégoire Danoy, and Pascal Bouvry. A cooperative coevolutionary approach to maximise surveillance coverage of uav swarms. In *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, 2020. doi: 10.1109/CCNC46108.2020.9045643.

[22] ArduPilot Development Team. Micro xrce-dds generator (micro-xrce-dds-gen). `https://github.com/ardupilot/Micro-XRCE-DDS-Gen`, 2024. URL `https://github.com/ardupilot/Micro-XRCE-DDS-Gen`. Accessed: 2024-12-27.

[23] Yubing Wang, Peng Bai, Xiaolong Liang, Weijia Wang, Jiaqiang Zhang, and Qixi Fu. Reconnaissance mission conducted by uav swarms based on distributed pso path planning algorithms. *IEEE Access*, 7:105086–105099, 2019. doi: 10.1109/ACCESS.2019.2932008.

[24] Kunlun Wei, Tao Zhang, and Chuanfu Zhang. Research on resilience model of uav swarm based on complex network dynamics. *Eksploatacja i Niezawodność – Maintenance and Reliability*, 25(4), 2023. ISSN 1507-2711. doi: 10.17531/ein/173125. URL `https://doi.org/10.17531/ein/173125`.

[25] A. N. Wilson, Abhinav Kumar, Ajit Jha, and Linga Reddy Cenkeramaddi. Embedded sensors, communication technologies, computing platforms and machine learning for uavs: A review. *IEEE Sensors Journal*, 22(3):1807–1826, 2022. doi: 10.1109/JSEN.2021.3139124.

[26] Yongkun Zhou, Bin Rao, and Wei Wang. Uav swarm intelligence: Recent advances and future trends. *IEEE Access*, 8:183856–183878, 2020. doi: 10.1109/ACCESS.2020.3028865.