Development of a Distributed Learning and Scalable Testbed for UAVs Using Machine Learning

This thesis is submitted

To



For the award

Doctor of Philosophy

By

Jawad Mahmood

Prepared under the supervision of

Dr. Muhammad Adil Raja

Dr. John Loane

Dr. Fergal Mc Caffery

School of Informatics and Creative Arts

Dundalk Institute of Technology

September 2026

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, and that I have exercised reasonable care to ensure that the work is original and have conformed to the regulations on the use of and declaration of Generative AI, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Jawad Mahmood (Candidate) ID No. D00262041 Date: 10/09/2026

Appendix A – Reporting of Gen AI Tool Use in Research Theses

This appendix provides a report of the use of Generative Artificial Intelligence (Gen AI) tools during the preparation of this thesis.

Category	Details
Tools	**ChatGPT:** Used to improve English sentences, remove grammatical errors, and assist with coding and fixing LaTeX code errors. **Grammarly AI:** Used to improve English sentences and remove grammatical errors. The exact model/version and dates of individual uses were not recorded in the thesis.
Outputs	ChatGPT outputs were used for English-language improvement, grammatical correction, and assistance with coding and LaTeX errors. Grammarly AI outputs were used for English-language and grammatical improvement. The tools were used as supporting tools during thesis preparation.
Prompts	The thesis does not contain a preserved record of the exact prompts used. The documented purposes included improving or rephrasing English sentences, correcting grammatical errors, and identifying or fixing LaTeX coding errors. Example prompts include to improve sentence wording, correct grammar, and identify or fix LaTeX errors. These examples describe the purposes of the prompts and are not claimed to be verbatim historical prompts.
Iterations	The thesis does not record the exact number of Gen AI iterations or provide a complete prompt/output history. The documented use involved iterative assistance with English-language improvement, grammatical correction, and LaTeX coding and error fixing. Outputs were reviewed by the author during preparation of the final thesis.
Critical Reflection	The use of Gen AI tools was limited to language improvement, grammatical correction, and assistance with LaTeX coding and error fixing, as acknowledged in the thesis. The research problem, objectives, methodology, testbed design and implementation, experimental validation, results, contributions, limitations, and conclusions constitute the researcher's work. Gen AI tools were used as supporting tools during thesis preparation rather than as a substitute for the research process.

Appendix A: Reporting of Gen AI Tool Use in Research Theses

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors for their guidance, encouragement, and support throughout my research journey. Their insightful feedback, patience, and unwavering belief in my work have been instrumental in shaping this thesis. I am thankful to my father **Chaudhry Khalid Mehmood**, wife **Uzma Azam**, and cousin brother **Abdul Rehman** for supporting me during my PhD and praying for me. A special thanks to HEA for providing the financial support that enabled me to conduct my research and pursue this academic endeavour. Without this support, this work would not have been possible. I am deeply appreciative of IEEE, Springer, and Elsevier Publishers for publishing my Research papers related to my thesis.

I would like to share that Grammarly AI and ChatGPT helped me improve my English sentences and remove grammatical errors. I have written the thesis in LaTeX. ChatGPT helped me in coding and fixing the LaTeX code errors.

Research Papers' Publications Related to this Research Project (PhD Thesis)

Journals

1. **Jawad Mahmood, Muhammad Adil Raja, John Loane, Fergal McCaffery,** A Configurable Intrinsic Curiosity Module of Testbed for developing Intelligent Swarm UAVs. Machine Learning With Application Journal, 2025, **Elsevier Publisher**. https://doi.org/10.1016/j.mlwa.2025.100714

2. **Jawad Mahmood, Muhammad Adil Raja, John Loane, Fergal McCaffery,** A Multi-Objective Scheme for Collision Avoidance, Swarm Cohesion, and Target Tracking for Smart UAVs, Journal of Information and Intelligence, 2025, **Elsevier Publisher** https://doi.org/10.1016/j.jiixd.2025.10.004

Conferences

1. **Jawad Mahmood, Muhammad Adil Raja, John Loane, Fergal McCaffery,** A Multi-UAV Testbed for Real-Time Target Tracking Using Curiosity-Driven Reinforcement Learning, International Conference on Information and Communication Technology for Intelligent Systems, 2025, USA, **Springer Publisher**.
   https://link.springer.com/chapter/10.1007/978-981-95-1357-4_28

2. **Jawad Mahmood, John Loane, Muhammad Adil Raja, Fergal McCaffery,** A Distributed learning and Scalable Testbed for Developing Intelligent UAV Swarm Controllers Based on Curiosity-Driven Machine Learning, Irish Signals and Systems Conference, 2025, Ireland, **IEEE Publisher**.
   https://ieeexplore.ieee.org/document/11291260

3. **Jawad Mahmood, John Loane, Muhammad Adil Raja, Fergal McCaffery,** Comparison of Reinforcement Learning Models of a Testbed for UAV Target Tracking, International Conference on Computer, Software and Modelling, 2025, Italy, **IEEE Publisher**.
   https://ieeexplore.ieee.org/document/11449951

Contents

1 Introduction **17**

1.1 Research Problem Defined 18

1.2 Research Questions and Objectives 19

1.3 Research Motivation 20

1.4 Research Approach 22

1.5 Research Contributions 22

1.6 Structure of Thesis 23

1.7 Summary 24

2 Literature Review **25**

2.1 Adopted Approach 25

2.2 Data Sources 26

2.3 Search Keywords 27

2.4 Search for Existing Literature and Screening 28

2.5 UAV Testbeds 30

2.6 Elements of a Unmanned Aerial Vehicle (UAV) Testbed 42

2.6.1 Fixed-wing UAVs 43

2.6.2 FlightGear 47

2.6.3 Swarm UAVs 52

2.6.4 Reinforcement Learning Algorithms 56

2.7 Research Questions Revisited 73

2.8 Summary 75

3 Research Setting **76**

3.1 Research Methodology 76

3.1.1 Research Philosophies 77

3.1.2 Research Approaches 78

3.1.3 Research Strategies . . . 80
3.1.4 Choices . . . 91
3.1.5 Time Horizon . . . 91
3.2 Summary . . . 92

4 Design and Implementation of the Testbed 93
4.1 Designing the Architecture of the Testbed . . . 93
4.1.1 Interfacing Flight Dynamics Model (FDM) with Reinforcement Learning (RL) Algorithms . . . 94
4.2 Implementation of the Advantage Actor Critic Algorithm (A2C) Algorithm 96
4.2.1 A2C Hyperparameter Tuning . . . 99
4.3 Implementation of the Asynchronous Advantage Actor-Critic Algorithm (A3C) Algorithm . . . 100
4.3.1 Step-by-Step Workflow of Asynchronous Updates in A3C . . . 101
4.3.2 A3C Hyperparameter Tuning . . . 103
4.3.3 Implementation of Intrinsic Curiosity Module . . . 105
4.4 Implementation of Proximal Policy Optimization (PPO) Algorithm . . . 110
4.4.1 PPO Hyperparameter Tuning . . . 111
4.5 Implementation of Gaussian Reward Function Centered on Ellipsoidal Boundary Around UAV . . . 112
4.6 Implementation of Distributed Learning Computing Based Tracking of A2C Aircraft by a Swarm of A3C and PPO Aircraft in a Complex Path . . . 119
4.7 Summary . . . 124

5 Verification and Validation Using Experimental Results 126
5.1 Experimental Validation Roadmap . . . 128
5.2 Experimental Results for the Implementation of A2C Model to Control Target Aircraft . . . 130
5.3 Experimental Results for the Implementation of A3C Model to Control Two Tracking Aircraft . . . 132
5.4 Experimental Results for the Implementation of A3C Model to Control Seven Tracking Aircraft . . . 135
5.5 Experimental Results for the Implementation of PPO Model to Control the Tracking Aircraft . . . 139

5.6 Comparison of Intrinsic Curiosity Module (ICM) Based A3C and PPO Models with Non-ICM Based RL Models . . . 142
5.7 Behaviour of an Untrained Agent . . . 145
5.8 Behaviour of Trained Agent . . . 147
5.9 Tracking of A2C Aircraft by A3C Aircraft in a Straight Path . . . 148
5.10 Tracking of A2C Aircraft by the Swarm of Two A3C Aircraft in a Complex path . . . 149
5.11 Tracking of A2C Aircraft by the Swarm of PPO and A3C Aircraft in a Complex Path in the Distributed Learning Setup . . . 152
5.12 Tracking of A2C Aircraft by a Swarm of Seven A3C Aircraft in a Complex Path . . . 155
5.13 Experimental Simulation Videos of This PhD Project . . . 163
5.13.1 Experimental Simulation Video of the Untrained Agent . . . 163
5.13.2 Experimental Simulation Video of the Trained Agent . . . 163
5.13.3 Experimental Simulation Video of Target Tracking in a Straight Path 164
5.13.4 Experimental Simulation Video of Target Tracking in a Complex Path by the Swarm of Tracking UAVs . . . 164
5.13.5 Experimental Simulation Video of Target Tracking of the Distributed Learning Testbed . . . 164
5.13.6 Experimental Simulation Video of Target Tracking of Seven A3C Workers Controlled Aircraft . . . 165
5.14 Summary . . . 165

6 Discussion and Conclusion 166

6.1 Research Problem and Research Questions Revisited . . . 166
6.2 Research Contribution and Significance . . . 178
6.3 Limitations . . . 181
6.4 Implications and Synthesis of Key Findings . . . 184
6.5 Future Work . . . 186
6.6 Concluding Insights . . . 187
6.7 Summary . . . 187

List of Figures

2.1 Steps taken in conducting the literature review 26

2.2 Figure showing different motions of aircraft. a) Longitudinal motion. b) Vertical motion. c) Pitch motion. d) Lateral motion. e) Roll motion. f) Yaw motion [1]. 32

3.1 Research onion [2]. 77

3.2 Action research [3]. 80

3.3 Action design research [4] 82

3.4 Design science research methodology [5]. 85

3.5 Flowchart showing design stage and evolution stage 87

4.1 Architecture of the Testbed [6] 94

4.2 Flowchart illustrating the working mechanism of JSBSim. 95

4.3 Maturity Model of Testbed 95

4.4 Implementation of A2C algorithm 98

4.5 Implementation of A3C algorithm. 102

4.6 Implementation of ICM. 106

4.7 Framework to overcome the problem of delayed reward using A3C-ICM model. 109

4.8 Reward and penalty based on the distance between UAVs 114

4.9 Illustration of the calculation of reward by placing the Gaussian reward function on the boundary of an ellipsoid. The UAV inside the ellipsoid corresponds to a tracking UAV under consideration that has to compute its reward. The other UAV is either a target or some other tracker. 116

4.10 Flowchart shows Ray handles the communication between the separate Nodes 119

4.11 Distributed learning computing framework of Testbed for swarm of UAVs. 122

5.1 Graphical results to show the implementation of A2C algorithm a) Actor loss b) Critic loss c) Entropy d) Entropy loss e) Cumulative reward f) Episode length mean 131
5.2 Graphical results for the implementation of A3C algorithm a) Actor loss, b) Episode length mean, c) Frames Per Second (FPS), d) Cumulative reward. 133
5.3 Graphical results for the implementation of A3C algorithm e) Learning rate f) entropy loss g) critic loss h) entropy. 134
5.4 Graphical illustration of A3C performance for seven workers a) Cumulative reward b) Episode length mean c) Step rate d) Learning rate e) Actor loss f) Critic loss. 136
5.5 Graphical illustration of A3C performance g) Entropy h) Entropy loss. . . 137
5.6 Graphical results of PPO algorithm a) Clip fraction b) Clip range c) Variance d) Entropy loss e) Policy gradient loss f) Learning rate. 139
5.7 Graphical results of PPO algorithm g) Cumulative reward h) Episode length mean i) Training loss. 140
5.8 Graphical results to show the comparison of performance between A3C with ICM and A3C without ICM a) Cumulative reward b) Episode length mean c) Actor loss d) Critic loss e) Entropy loss f) Entropy. 143
5.9 Graphical results to show the comparison of performance between PPO with ICM and PPO without ICM a) Cumulative reward b) Policy gradient loss c) Training loss d) Variance e) Clip fraction f) Entropy loss. 144
5.10 Behaviour of an untrained agent performing random motions. a) Aircraft performing pitch motion in the downward direction. b) Aircraft performing a combination of pitch and yaw motions and inverted. c) Aircraft performing a combination of roll and yaw motions. d) Aircraft performing pitch motion in the upward direction. e) Aircraft performing a combination of roll and yaw motions. f) Aircraft is inverted. 146
5.11 Behaviour of trained agent performing stable motions. a) The aircraft is turning right. b) The aircraft is turning left. 147
5.12 Target tracking. a) Two windows showing the target and tracking aircraft. b) Console showing the list of pilots. c) Console showing the chatbox. . . . 148
5.13 Flightgear built-in map demonstration. 149
5.14 Swarm UAVs to UAV target tracking demonstration on a) Python editor and b) FlightGear map. 151

5.15 UAV to UAV target tracking a) Target and tracker moving straight line b) Target and tracker turning left c) Target turning right and tracker is turning from left to right d) Target is turning right and tracker is turning from straight to right e) Target and tracker are turning right f) Target is turning straight from right and tracker is turning right. 152
5.16 Demonstration of swarm of UAVs Real-time in FlightGear 153
5.17 The Swarm of Tracking UAVs following the Target in FlightGear Map . . . 154
5.18 FlightGear Built-in map demonstrating the Swarm of 7 Tracking UAVs following the target 156
5.19 FlightGear Pilot List 157
5.20 Various flight trajectories.(The units are in meters and are in coordinates.) 159
5.21 Min. max. distance (meter) plot of various flight trajectories. 160

List of Tables

2.1 Inclusion and exclusion criteria for the literature review 28
2.2 Summary of Literature Search Strategy 30
2.3 State Space Representation of UAV (6-DOF Model) 32
2.4 Comparison of UAV testbeds: focus, features, and limitations. 42
2.5 Summary of fixed-wing UAV research: methods, features, and limitations. 46
2.6 Summary of FlightGear research: focus, features, and limitations. 52
2.7 Summary of swarm UAV research: focus, features, and limitations. 55
2.8 Summary of research studies using the A2C algorithm: focus, features, and limitations. 61
2.9 Summary of research studies using the A3C algorithm: focus, features, and limitations. 65
2.10 Comparison of PPO-based UAV research studies: models, features, and limitations. 69
2.11 Comparison of ICM-related research studies: models, features, and limitations. 73

3.1 Comparison of research approaches [7]. 84
3.2 Mapping of Design Science Research (DSR) Stages to Thesis Activities . . 89
3.3 The selected elements of research methodology of the project. 92

4.1 Final tuned hyperparameters for A2C 99
4.2 Hyperparameters for A3C Model 103
4.3 Comparison of Grid Search and SRCW Tuning Strategies 105
4.4 Self-Reflective Curiosity-Weighted Hyperparameter (SRCW) Tuning [8, 9, 10, 11] 107
4.5 PPO Model's Hyperparameters 112

5.1 Comparison of the Testbed Developed in This Research Project with Previous Studies for Validation 127

6.1 Comparison of the testbed of this research project with previous studies . . 179

List of Abbreviations

Abbrev.	Definition
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Action Research
A2C	Advantage Actor Critic Algorithm
A3C	Asynchronous Advantage Actor-Critic Algorithm
CAD	Computer Aided Design
CAR	canonical action research
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance
CDL	Curiosity Driven Learning
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DL	Deep Learning
DP	Dynamic Programming
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
DQN	Deep Q-Learning Network
EC	Evolutionary Computing
FDM	Flight Dynamics Model
FOV	Field of View
FPS	Frames Per Second
FSM	Finite State Machine
GA	Genetic Algorithm
GAE	Generalized Advantage Estimation
GaTAC	Georgia testbed for autonomous control
GNC	Guidance, Navigation, and Control
GPU	Graphics Processing Unit
GPS	Global Positioning System

Continued on next page

Abbrev.	Definition
HILS	Hardware-In-the-Loop Simulation
ICM	Intrinsic Curiosity Module
IMU	Inertial Measurement Unit
JGF	Joint Geospatial Facility
JSBSim	Flight Dynamics Model
LAN	Local Area Network
LiDAR	Light Detection and Ranging
LSTM	Long Short-Term Memory Networks
MAPPO	Multi-Agent Proximal Policy Optimization
MC	Monte Carlo
MEC	Mobile Edge Computing
MEMS	Micro-electromechanical systems
ML	Machine Learning
MSE	Mean Squared Error
MSFS	Microsoft Flight Simulator
NASA	National Aeronautics and Space Administration
NN	Neural Network
NSGA	Non-Dominated Sorting Genetic Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm II
NUAV	Namal Unmanned Aerial Vehicle
PAROS	Programming Swarm
PID	Proportional-Integral-Derivative
PPO	Proximal Policy Optimization
R2N2	Residual Recurrent Neural Networks
RL	Reinforcement Learning
ROS	Robot Operating System
RNN	Recurrent Neural Network
ROS	Robot Operating System
RUAV	Rotorcraft Unmanned Aerial Vehicle
SARSA	State-Action-Reward-State-Action algorithm
SGD	Stochastic Gradient Descent

Continued on next page

Abbrev.	Definition
SL	Supervised Learning
SQP	sequential quadratic programming
SRCW	Self-Reflective Curiosity-Weighted Hyperparameter
TCP	Transmission Control Protocol
TD	Temporal Difference
UAS	Unmanned Aerial Systems
UCAV	Unmanned Combat Aircraft Vehicles
UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicles
UAV	Unmanned Aerial Vehicle
VRU	Vulnerable Road User
VRX	Virtual RobotX
VTOL	Vertical Takeoff and Landing
WAN	Wide Area Network
XDR	(eXternal Data Representation)

Abstract

This project aims to develop a testbed for designing and training control algorithms for cooperative and self-organizing swarms of UAVs. The primary purpose of developing a distributed learning and scalable testbed based on Machine Learning (ML) algorithms is to enable UAVs to make decisions using real-time data and perform tasks autonomously. In this project, a novel testbed is developed that allows the integration of different ML algorithms with a flight simulator. This testbed supports multiple UAVs that learn to fly and coordinate in a simulated environment to accomplish the objective of target tracking. It employs novel techniques that enable faster learning and higher performance compared to conventional ML methods. FlightGear is the flight simulator used in this project. Once trained, the aircraft can fly with the help of an ML model and show stable behaviour compared to an untrained model. This testbed can be used to train control models for a wide variety of use cases. As proof of concept, a problem is formulated regarding target tracking of UAVs. The tracking aircraft follows the path of the target aircraft. Both tracking and target aircraft are controlled by different ML models and fly on a common flight simulator. This testbed can also scale up the number of tracking aircraft.

Keywords: Testbed, FlightGear, Swarm UAV, ML

Chapter 1

Introduction

A testbed is a controlled and realistic environment designed for testing, experimenting, and evaluating new UAV systems, technologies, or algorithms before they are deployed in the real world. It allows researchers to experiment with flight dynamics, control strategies, communication, autonomy, and multi-UAV coordination without relying on real aircraft. The main benefit of a UAV testbed is that it provides a safe, low-cost, and repeatable environment for testing UAV behaviours. It avoids the risks of real-world flight testing, reduces hardware damage, and allows experiments to be repeated under identical conditions. This makes development faster, safer, and far more efficient.

In this project, I implemented a testbed that is scalable and based on distributed learning. The testbed can perform the task of target tracking of UAVs. I used three different RL algorithms to control the target and tracking aircraft. The first algorithm is A2C, the second is the A3C algorithm, and the third one is the PPO algorithm. Only one target aircraft is controlled by the A2C algorithm, and more than one tracking aircraft are controlled by the A3C and PPO algorithms. I used JSBSim FDM and FlightGear flight simulator for experimental simulations. Previous testbeds, except Georgia testbed for autonomous control (GaTAC), were not scalable and distributed.

The first chapter serves as an introduction, outlining the research problem, objectives, contribution, and the research approach employed in this project. The architectures of the A2C, A3C, and PPO algorithms are detailed. The six degrees of freedom of an aircraft's motion are explained. The second chapter is a literature review. The topics reviewed include swarm UAVs, fixed-wing UAVs, FlightGear, UAV testbeds, and RL algorithms. The research papers were critically reviewed, which helped in finding the research gap for

this project. The third chapter of this thesis is the research methodology. The fourth chapter details the implementation of the testbed. Three major RL algorithms (A2C, A3C and PPO) were implemented in Python. Flowcharts detailing the implementation of the algorithms are included in this chapter. Tables of hyperparameters for each algorithm are also given in this chapter. The fifth chapter is based on experimental validation and verification. The performance of the implemented models is analyzed using graphical results. The behaviour of the trained and untrained RL agents, as well as a detailed description of the target tracking task, is included. The sixth chapter includes a discussion and conclusion of the research questions that are answered in this thesis, as well as the research contribution and future work of this project.

The next section defines the research problem of this thesis.

1.1 Research Problem Defined

The development of autonomous and cooperative UAVs requires extensive experimentation to validate control strategies, ML algorithms, and coordination mechanisms. Conducting such experiments directly on physical aircraft is risky, costly, and often impractical due to safety concerns, hardware damage, and operational constraints [12]. A simulation-based testbed provides a safe, flexible, and cost-effective alternative, allowing researchers to test and evaluate ML models under controlled yet realistic conditions before deployment on real aircraft. Such testbeds enable the acceleration of innovation, exploration of edge cases, and refinement of algorithms without exposing human life or expensive hardware to unnecessary risk [13] .

Despite the existence of several UAV testbeds, prior testbeds suffer from critical limitations that reduce their applicability to large-scale swarm scenarios. Early testbeds were typically restricted to single-agent validation, lacking scalability (except GaTAC) and distribution. GaTAC primarily operates within a centralised computational framework, where control and learning processes are executed on a single machine. While it supports multiple UAVs, it does not explicitly implement distributed machine learning across multiple physical machines or support scalable deployment across networked nodes. It only distributed the FlightGear instances. It only distributes the FlightGear simulation instances across multiple machines, while the control and learning processes remain centralized and are executed on a single system. In this research, GaTAC is used as a reference point for comparison because it demonstrates multi-UAV

coordination within a simulation environment. However, the testbed developed in this work extends beyond GaTAC by enabling distributed machine learning across multiple computers, supporting asynchronous learning through A3C, and integrating multiple Machine Learning algorithms within a single framework. By introducing GaTAC before comparison, the discussion provides a clearer context for evaluating the strengths and limitations of existing testbeds and highlights the specific contributions of this research.

Even more advanced systems, such as the Namal Unmanned Aerial Vehicle (NUAV) testbed, provided valuable foundations but remained centralized, unable to distribute computation across multiple machines. These platforms also struggled with delayed reward issues. Delayed reward means that the agent receives positive feedback only after many actions in the future instead of getting it immediately. This makes it harder for the learning algorithm to connect actions with their actual outcomes. Delayed rewards can destabilize training, hinder convergence, and cause policy degeneration, leading agents to settle for suboptimal behaviours. In policy degeneration, an agent's learning gets stuck by repeating the same poor actions instead of exploring better ones, leading to poor performance of the algorithm [14] .

The testbed should integrate distributed learning and scalable architectures, support multiple aircraft operating concurrently, and incorporate mechanisms to address delayed reward and policy degeneration. By overcoming these limitations, the testbed creates an environment for developing intelligent, cooperative, and real-time swarm behaviours.

1.2 Research Questions and Objectives

The overall research question is: How can a scientifically grounded and methodologically validated testbed be developed to support scalable and distributed learning coordination of intelligent swarm UAVs?
The Research Questions (RQs) and their corresponding Research Objectives (ROs) are linked as follows:

1. **RQ-1:** What are the limitations in the current state of the art in testbeds for self-coordinating and cooperative swarms of UAVs?

    (a) **RO-1:** To conduct a critical analysis of existing UAV testbeds to identify their limitations in their UAV testbeds, forming the foundation for the design and development of an improved testbed.

2. **RQ-2:** Which flight simulators have been used in previous testbeds for self-coordinating and cooperative swarms of UAVs, and which is the most suitable for this research?

   (a) **RO-2:** To select and integrate an appropriate flight simulator and FDM into the testbed to ensure realism, flexibility, and accurate aircraft dynamics.

3. **RQ-3:** What are the requirements for a testbed for self-coordinating and cooperative swarms of UAVs?

   (a) **RO-3:** To identify, analyse, and define the technical and functional requirements necessary for designing a scalable and distributed learning swarm UAV testbed.

4. **RQ-4:** Which ML algorithms are most suitable for enhancing the performance and robustness of UAV swarm testbeds?

   (a) **RO-4:** To evaluate and compare multiple ML algorithms to determine which are most suitable for achieving stable, robust, and cooperative swarm behaviour within a UAV testbed.

With the research questions and objectives established, the next section explains the research motivation.

1.3 Research Motivation

The development of autonomous and cooperative UAVs requires robust testing environments that can support complex multi-agent interactions. It is costly and risky to conduct experiments using real aircraft; therefore, a testbed is needed. Existing UAV testbeds identified in the literature are often limited by centralized architectures, lack of scalability, and insufficient support for distributed learning. These limitations restrict their ability to simulate realistic swarm scenarios where multiple UAVs must operate simultaneously and coordinate in real time.

This research is motivated by the need to address these limitations through the development of a distributed and scalable testbed that enables multi-UAV coordination using Machine Learning techniques. The literature review shows that simulation tools such as FlightGear and JSBSim are widely used because they provide realistic flight

behaviour. However, most existing studies use these tools for single UAV control or small-scale experiments. They are rarely combined with distributed learning across multiple machines or used to train multiple UAVs working together in a coordinated manner. This gap highlights the need for a testbed that integrates realistic simulation with distributed multi-agent learning.

A2C is used to control the target UAV because it provides a stable and consistent learning process through synchronous updates. Since the target UAV is required to follow a controlled or predictable behaviour (such as random but smooth motion), A2C is suitable because it avoids the variability introduced by asynchronous updates and produces steady policy behaviour.

A3C is selected for tracking UAVs because it supports asynchronous learning through multiple workers. This allows different agents to explore the environment in parallel, improving learning speed and making it suitable for multi-agent scenarios. A3C also reduces correlation between training samples, which helps in faster convergence when multiple UAVs are learning simultaneously in a shared environment. This makes it well suited for distributed execution across multiple machines.

PPO is included to address stability limitations observed in A3C. While A3C enables fast and scalable learning, its asynchronous updates can sometimes lead to unstable or inconsistent policy updates. PPO overcomes this by using a clipped objective function, which restricts large policy changes and ensures more stable and reliable learning. This makes PPO useful as a benchmark and as a complementary approach to improve robustness.

Together, these algorithms allow the testbed to balance stability, scalability, and robustness. A2C provides controlled behaviour for the target UAV, A3C enables efficient multi-agent learning, and PPO ensures stable policy optimization. The research also addresses the challenge of delayed reward, which is common in UAV tracking tasks. The integration of curiosity-driven mechanisms enhances exploration and enables agents to learn more effectively in complex environments. From a practical perspective, the ability to coordinate multiple UAVs has significant applications in real-world scenarios such as surveillance, environmental monitoring, disaster response, and search and rescue missions. A scalable and distributed testbed provides a safe and cost-effective platform for developing and evaluating such systems before real-world deployment.

The novelty of this research lies in combining distributed learning, multi-agent coordination, and curiosity-driven exploration within a single testbed. This work

provides new insights into how multiple UAVs can learn to coordinate without explicit communication while maintaining stable performance. These contributions are directly linked to the research questions, which focus on scalability, coordination, and learning efficiency in multi-UAV environments.

1.4 Research Approach

This research adopts the Design Science Research (DSR) methodology as the overarching approach to guide the design and validation of the testbed. DSR is particularly well suited for studies that aim to create and evaluate innovative artefacts, in this case, a scalable and distributed learning testbed for UAV swarm coordination using RL algorithms. The methodology emphasizes both problem-solving and knowledge generation by systematically progressing through the stages of problem identification, objective definition, artefact design and development, demonstration, evaluation, and communication.

In the context of this thesis, the research began by identifying limitations in existing testbeds, such as scalability, centralised architectures, delayed rewards, and policy degeneration. From these challenges, objectives were defined to guide the development of a distributed learning testbed capable of addressing these shortcomings. The artefact "UAV testbed" was then designed and implemented by integrating FlightGear, JSBSim, and RL models such as A2C, A3C, and PPO. Demonstration and evaluation were performed through controlled simulation experiments, where performance was measured in terms of scalability, stability, and the ability to foster cooperative swarm behaviours. Finally, the outcomes and insights are communicated through this thesis and related research paper publications, thereby contributing to both academic knowledge and future research directions in distributed learning swarm UAV systems.

Following the methodological approach, the subsequent section highlights the main contributions of this thesis.

1.5 Research Contributions

The work demonstrates that distributed learning architectures can support stable and coordinated multi-UAV tracking within a simulation environment, even when agents operate without explicit inter-agent communication.

The results show that asynchronous multi-agent learning using A3C improves convergence stability and tracking performance compared to synchronous approaches. The research also provides evidence that curiosity-driven exploration improves learning efficiency and convergence in delayed reward environments.

The integration of the Intrinsic Curiosity Module shows that agents can better adapt to complex target trajectories and maintain stable tracking behavior. In addition, the study shows that a composite reward function based on ellipsoidal Gaussian constraints can simultaneously enforce tracking, cohesion, and collision avoidance within a single optimisation framework, leading to emergent cooperative behaviour.

Finally, the work demonstrates that heterogeneous Machine Learning models can operate within a distributed testbed and maintain stable performance, highlighting the flexibility of combining different learning strategies in a multi-agent setting.

1.6 Structure of Thesis

This thesis is organized into six chapters. Chapter 1 introduces the research motivation, problem statement, objectives, approach, and contributions, along with an overview of the UAV testbed integrating FlightGear, JSBSim, and RL algorithms such as A2C, A3C, and PPO. Chapter 2 provides a structured literature review, addressing the research questions, evaluating existing UAV testbeds, simulators, and swarm coordination studies, and identifying the research gaps. Chapter 3 outlines the research methodology, structured using the DSR framework, and explains the philosophical stance, research strategies, and evaluation choices. Chapter 4 details the design and implementation of the distributed learning testbed, including the system architecture, integration of FDM with RL algorithms, implementation of A2C, A3C, and PPO, and the development of reward functions for swarm tracking tasks. Chapter 5 presents verification and validation through experimental results, comparing trained and untrained models, ICM and non-ICM variants, and analysing UAV behaviours in both straight and complex target-tracking scenarios. Chapter 6 discusses the research contributions in light of the objectives and questions, highlighting the implications and impact of the findings. Chapter 6 also discusses the key outcomes, acknowledges limitations, suggests directions for future research, and concludes the thesis.

1.7 Summary

This chapter establishes that the main limitation of existing UAV testbeds is not a single issue, but the combined effect of limited scalability, lack of distributed learning, and the delayed reward problem, which together restrict the development of cooperative swarm behaviour. These challenges must be addressed in an integrated manner rather than through isolated improvements. The chapter analytically justifies the design choices of the testbed by linking them directly to these challenges. FlightGear and JSBSim provide the required simulation realism and flexibility, while A2C, A3C, and PPO are selected as complementary learning models to balance stability, scalability, and robustness. The inclusion of the Intrinsic Curiosity Module introduces an exploration-driven mechanism to mitigate delayed rewards and improve learning efficiency. A key conclusion of this chapter is that the contribution of the research lies in the integration of distributed learning, curiosity-driven exploration, and high-fidelity simulation into a single testbed, enabling coordinated and scalable swarm UAV behaviour. This foundation leads into Chapter 2, where the limitations and design decisions are critically examined through a structured review of existing literature to validate the research gap.

Chapter 2

Literature Review

The purpose of this chapter is to provide an in-depth analysis of research already undertaken in the area of UAV testbeds, with a particular focus on their integration with RL frameworks for autonomous navigation, coordination, and target tracking. The review examines testbed architectures, control strategies, and simulation platforms.

This chapter begins by outlining the process adopted in conducting the literature review, which ensures that relevant studies are identified, analysed, and synthesised in a structured manner. The selected literature provides the foundation for understanding how UAV testbeds have evolved from conventionally operated platforms to advanced, distributed learning, and scalable multi-agent systems capable of running complex learning-based algorithms in real time. By evaluating existing contributions and identifying technological trends and research gaps, this chapter establishes the groundwork for the UAV testbed presented in later chapters.

2.1 Adopted Approach

A structured literature review was conducted in accordance with the methodology outlined by Kitchenham [15]. The approach taken is given in Figure 2.1.

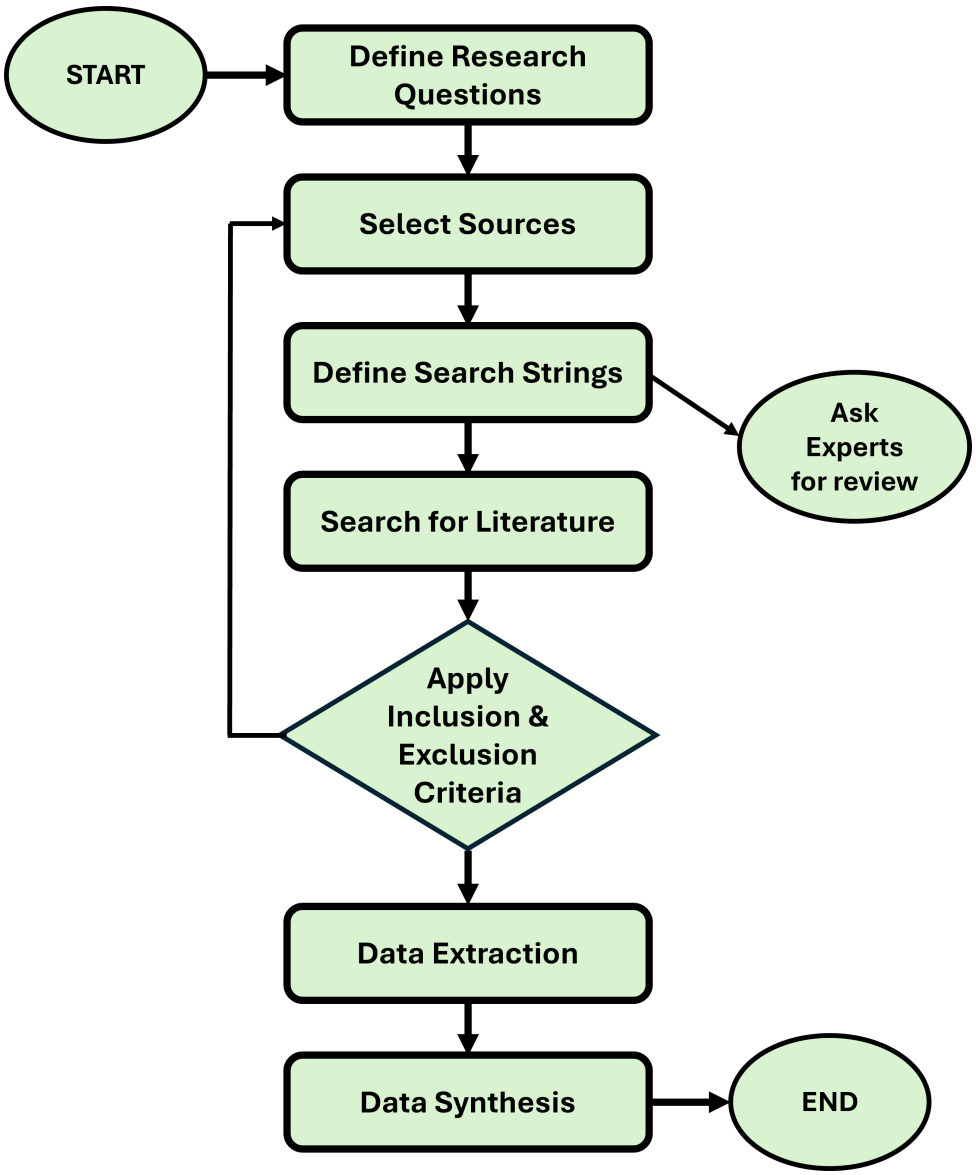


Figure 2.1: Steps taken in conducting the literature review

According to this figure, the literature review begins with defining research questions to focus the review, followed by selecting reliable data sources. Next, search keywords were formulated, and a search was conducted. Finally, inclusion and exclusion criteria were applied to screen, refine, and consolidate relevant studies, ensuring focused review.

2.2 Data Sources

The second step was to choose electronic databases of peer-reviewed papers. The databases were required to contain literature from key journals and conferences concerning UAV swarm coordination, RL-based control, and testbed development. The following electronic databases were used during the search process:

1. Google Scholar.
2. IEEE Xplore.
3. ACM Digital Library.
4. Springer.
5. Elsevier(Science Direct).

These databases were selected because they collectively cover the highest-impact and most authoritative sources in the fields of autonomous systems, UAV testbeds, and ML-based control. Google Scholar provides broad interdisciplinary coverage and helps ensure that no emerging or cross-domain studies are overlooked. IEEE Xplore and the ACM Digital Library are essential for accessing peer-reviewed research in robotics, distributed systems, and embedded control architectures, which underpin the development of the swarm UAV testbed. Springer and Elsevier's Science Direct were chosen for their extensive collections of journal articles in aerospace engineering, optimization, RL, and simulation methodologies. Together, these databases provide complementary coverage, ensuring that the literature review captures both foundational work and the latest advancements relevant to UAV testbeds.

2.3 Search Keywords

The third step in the literature review process involved identifying keywords to be used when searching the selected electronic databases. The broad search terms used in this study were ML, Supervised Learning (SL), Unsupervised Learning (UL), RL, swarm UAVs, UAV testbeds, and flight simulators. The literature review derived the narrowed search terms.

The basic search term to search for the algorithms was the control strategies, which led to RL, which includes Deep Q-Learning Network (DQN), PPO, Deep Deterministic Policy Gradient (DDPG), A2C, and A3C. To narrow the search area for research papers, particularly in Google Scholar, combinations of basic search terms were also used. These included: UAV testbeds–RL algorithms–flight simulators, UAV testbeds–A3C algorithm–A2C algorithm, UAV testbeds–PPO algorithm, FlightGear–swarm UAVs–PPO algorithm, UAV testbeds–A3C algorithm–A2C algorithm–FlightGear, PPO algorithm–FlightGear, swarm UAVs–FlightGear–A3C

algorithm, A2C algorithm–FlightGear–UAV testbeds, ICM-A2C-A3C, and ICM-UAV testbeds FlightGear.

2.4 Search for Existing Literature and Screening

The initial search entailed querying each selected digital library with the search keywords outlined in Section 2.3 and applying the inclusion and exclusion criteria presented in Table 2.1.

Inclusion Criteria	**Exclusion Criteria**
Publication period: 2008–2025	Literature unrelated to ML, UAV testbeds, swarm UAVs, flight simulators, or simulation architectures
Language: English	Position papers, book reviews, or anonymous publications
Full-text available	Duplicate studies
Peer-reviewed journal articles and conference papers	Abstract-only or inaccessible papers, Non-English publications without an official translation

Table 2.1: Inclusion and exclusion criteria for the literature review

Each paper was first reviewed by examining the abstract. If the abstract indicated relevance, the paper was retained for further analysis. A total of 250 papers were reviewed. This research area is very focused and narrow; therefore, there are limited research papers that are relevant to my project. After screening, approximately 70 papers were added to this literature review.

The classification of papers was based on relevance to the core research themes of this study, which include UAV testbeds, multi-agent coordination, distributed learning, and Machine Learning-based control. A structured screening process was followed to evaluate each paper using predefined inclusion and exclusion criteria. The inclusion criteria required that the paper:

1. Focus on UAV systems, testbeds, or simulation environments,
2. Involves Machine Learning or control algorithms relevant to UAV coordination.

3. Provides experimental, simulation, or methodological contributions related to multi-agent or distributed systems. Papers that met these criteria were considered relevant for further analysis.

The exclusion criteria removed papers that:

1. Focused only on hardware design without control or learning components,
2. Unrelated to machine learning-based UAV applications.
3. Lacked sufficient technical detail or validation.

The classification process was performed by a single annotator, where each paper was manually reviewed based on its title, abstract, and, where necessary, full text. A qualitative relevance assessment was used rather than a numerical scoring metric. Papers were categorised as highly relevant, moderately relevant, or not relevant, based on how closely they aligned with the research objectives. The rationale for this approach was to ensure a focused and domain-specific literature review, while maintaining flexibility to include studies that provide valuable insights even if they do not directly match all criteria. This process resulted in a curated set of papers that informed the research gaps and guided the development of the testbed.

Table 2.2 presents a summary of the search strategy, including the databases used, search queries, the number of documents retrieved, and the number of papers selected after screening.

Database	Search Query	Retrieved	Selected	Year Range	Notes
IEEE Xplore	"UAV testbed" AND "Machine Learning"	85	25	2015–2024	Focused engineering studies
ScienceDirect	"UAV swarm" AND "reinforcement learning"	70	20	2016–2024	Included simulation studies
SpringerLink	"Distributed UAV" AND "multi-agent learning"	55	15	2016–2024	Focused distributed systems
Google Scholar	"UAV coordination" AND "FlightGear"	120	10	2015–2024	Broader search, filtered heavily

Table 2.2: Summary of Literature Search Strategy

The next section examines UAV testbeds. It examines different testbed architectures, features, and limitations, to highlight how they have shaped UAV research and where gaps remain.

2.5 UAV Testbeds

A UAV, commonly known as a drone, is an aircraft that operates without a human pilot onboard. It can be remotely controlled or operated autonomously through preprogrammed flight plans and onboard sensors. UAVs are widely used in military, commercial, civilian, and research applications, offering flexibility, efficiency, and cost-effectiveness.UAVs vary in size, design, and functionality. Small UAVs are commonly used for aerial photography, surveying, and agriculture, while larger, more advanced UAVs are utilized in military operations, environmental monitoring, and logistics. Their structure typically consists of an airframe, propulsion system, navigation and communication systems, and payload components, which can include cameras, sensors, or cargo. UAVs can operate more economically than manned aircraft and can carry out vital missions at a low cost without endangering human life[16].

The different motions of a UAV are given below:

Roll motion: Roll motion is when the longitudinal axis is rotated. The aircraft's longitudinal axis runs from its nose to its tail, as shown in Figure 2.2 a). If the aircraft had to spin along this axis, the wings would roll from side to side. The ailerons, controlled by the control wheel, are the control surfaces that permit this movement. The left and right ailerons move up and down using the control wheel to begin a turn. For example, the aircraft rolls along its longitudinal axis to the left since the right aileron produces higher lift on that wing [17].

Pitch motion: In this type of motion, the aircraft revolves around the lateral axis. The aircraft's nose would move up or down as shown in Figure 2.2 c) and the pitch attitude rotates along the lateral axis. The elevator will be the control surface that moves the pitch attitude, and the yoke or control wheel is used to move that surface.

Yaw motion: Visualize a line passing through the aircraft's centre of gravity. The aircraft would yaw along its vertical axis when it rotated around this line, as shown in Figure 2.2 f). [18].

Longitudinal motion: In this type of motion, the aircraft moves in forward and backwards directions only as shown in Figure 2.2 a).

Vertical motion: The aircraft moves up and down only in vertical motion as shown in Figure 2.2 b).

Lateral motion: In this type of motion, the aircraft moves side to side only as shown in Figure 2.2 d).

Throttle: It refers to the thrust of the aircraft. The aircraft is propelled in the direction of motion by thrust.

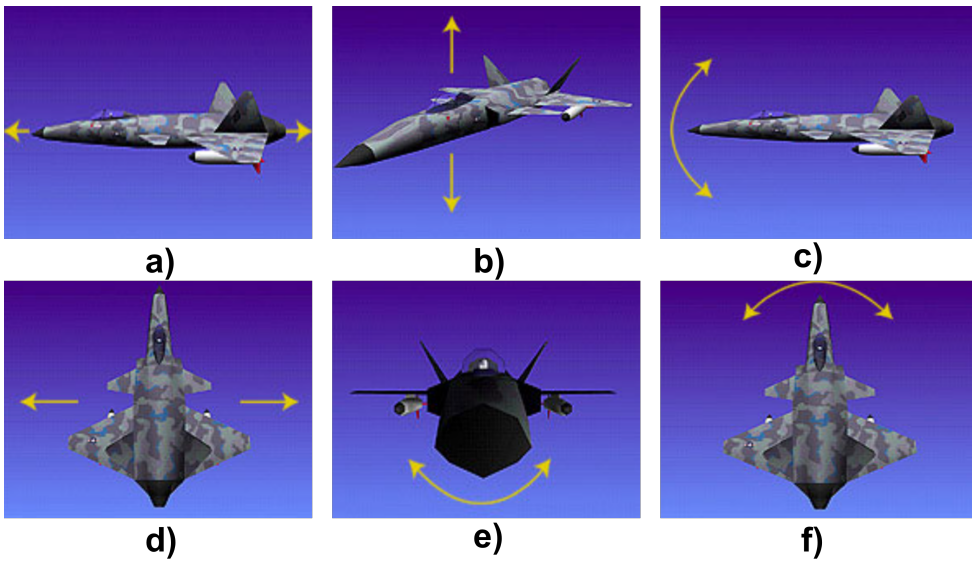


Figure 2.2: Figure showing different motions of aircraft. a) Longitudinal motion. b) Vertical motion. c) Pitch motion. d) Lateral motion. e) Roll motion. f) Yaw motion [1].

The motion of a fixed-wing UAV can be described using a six degrees of freedom (6-DOF) model, which represents both translational and rotational dynamics in a three-dimensional coordinate system. Instead of describing motion in simple directional terms, the UAV state is defined using vectors that capture its position, velocity, orientation, and angular velocity. The translational motion of the UAV is represented by its position vector in three-dimensional space, defined as (x, y, z), along with its linear velocity components (u, v, w) along the body axes. These components describe the forward, lateral, and vertical motion of the aircraft. The rotational motion is defined by the orientation of the UAV, typically represented using Euler angles. The corresponding angular velocity components (p, q, r) describe the rates of rotation about the body axes. Together, these variables define the full state of the UAV and are used by the Machine Learning models to learn control policies and make decisions within the testbed. Table 2.3 presents the state space variables associated with the 6-DOF model, providing a structured representation of the UAV dynamics used in this research.

Category	**Variable**	**Description**
Position	(x, y, z)	UAV position in 3D space
Linear Velocity	(u, v, w)	Velocity along body axes
Orientation	(ϕ, θ, ψ)	Roll, pitch, and yaw angles
Angular Velocity	(p, q, r)	Rotation rates about body axes

Table 2.3: State Space Representation of UAV (6-DOF Model)

Fixed-wing UAVs are a type of UAV that use wings, similar to traditional aircraft,

to generate lift and use aerodynamic principles to remain in flight by maintaining forward motion. These UAVs are highly efficient for long-distance missions, extended flight durations, and operations that require high-altitude surveillance and monitoring. Due to their ability to cover vast areas efficiently, fixed-wing UAVs are commonly used in military, environmental, agricultural, and commercial applications. Their reliance on wings allows them to glide and consume less energy, thus leading to significantly longer flight times [12].

Fixed-wing UAVs are chosen in this project because their extended flight times, higher speeds, and wide operational range align directly with the requirements of target tracking. These characteristics allow the trackers to pursue a moving target over extended distances and time periods without frequent interruptions for recharging or refuelling, ensuring reliable and continuous tracking performance within the testbed. Fixed-wing UAVs were used in NUAV testbed, which is the main inspiration of this project.

A group of artificially intelligent UAVs is known as a swarm of UAVs[19]. The UAVs in a swarm cooperate to perform tasks that would be extremely difficult for a single UAV to accomplish. Compared to single UAV operations, a swarm of UAVs offers more flexibility, safety, and coverage [19]. In a swarm, several UAVs synchronise to accomplish a task by interfacing with computer algorithms. Swarm UAVs are now widely available and employed in various applications due to their extreme adaptability [20].
Swarm UAVs are used in this project because they enable cooperative tracking of a single target by distributing the task across multiple aircraft. Within the testbed, swarm coordination is effectively simulated and managed across multiple machines, ensuring real-time performance and demonstrating the advantages of distributed learning for cooperative target tracking. We wanted to develop a testbed that can train multiple UAVs simultaneously for performing user-specified tasks. This is the major reason why we used swarm UAVs.

Having a testbed for UAV development can speed up the process of creating autonomous and self-coordinating UAVs [21]. Testbeds are important because they provide a controlled, repeatable, and safe environment for designing, testing, and validating algorithms before deploying them in real-world UAVs. Unlike physical flight experiments using real aircraft, which are costly, time-consuming, and risky, testbeds allow researchers to replicate scenarios, fine-tune algorithms, and explore failure cases without endangering hardware or human safety. They bridge the gap between theoretical models and practical implementation, enabling the evaluation of swarm

behaviours and control strategies under realistic but simulated conditions. By using testbeds, researchers can accelerate the development of autonomous and cooperative UAVs, ensuring that algorithms are robust enough to handle uncertainties in real-world environments [22]. If the testbed is scalable, it means we can increase the number of aircraft and distribute the testbed across many computers.

Integrating ML algorithms with a flight simulator is one of the testbed's many desirable attributes [23]. Data exchange between the ML algorithm and the flight simulator is possible due to the integration of ML algorithms, a FDM, and the flight simulator [6]. The desirable attributes of a testbed include flexibility. Flexibility is crucial to allow integration with different ML algorithms, control architectures, and sensor configurations. Scalability ensures that the testbed can handle not just single vehicles but also swarms of UAVs, reflecting the demands of distributed learning coordination. Realism is achieved through accurate flight dynamics models and high-fidelity simulators such as FlightGear, which provide realistic aerodynamic, environmental, and mission conditions. In addition, testbeds should support distribution across multiple machines to emulate large-scale swarm operations and incorporate mechanisms such as intrinsic rewards and reward shaping to address challenges like delayed feedback and policy destabilisation in RL [24] . Cost-effectiveness, open-source accessibility, and compatibility with hardware-in-the-loop configurations are also desirable so that results can smoothly transfer from simulation to real-world deployment.

The history of UAV testbeds begins with platforms that were designed mainly for the training and testing of a single aircraft rather than swarms. For instance, Yasar's Rotorcraft Unmanned Aerial Vehicle (RUAV) testbed [23] demonstrated how aerial and ground vehicles could be coordinated within a mission. Its strength lay in showing the feasibility of multi-vehicle interaction, but like many early works, it was tied to specific configurations and could not extend beyond a small number of vehicles.

Around the same time, Zhang's autopilot-based testbed [25] made use of FlightGear to provide an accessible simulation platform. While more flexible in terms of visualisation and user interaction, it still suffered from the same constraint: it was not designed to handle scalability or distribution. Both of these testbeds highlight the value of simulation for coordination but show that without distributed learning architectures, applications remain narrow and do not generalize to swarm contexts.

A different line of work focused on integrating simulators with ML frameworks to

enable adaptive, learning-based control of UAVs. Shah et al.'s AirSim testbed [26] provided a high-fidelity, visually realistic environment for training deep reinforcement learning agents in tasks such as navigation and obstacle avoidance, while Kaufmann et al.'s DeepDrone [27] extended this idea to vision-based agile flight, using domain randomization to bridge the simulation-to-reality gap. Similarly, Vira et al.'s SwarmRL testbed [28] integrated a flight simulator with reinforcement learning to simulate the swarm behaviors. All four testbeds shared the common feature of coupling modern simulators with ML frameworks to improve autonomy, adaptability, and task performance. However, they were also similar in their limitations: none supported distributed learning across multiple physical machines, all relied heavily on centralized models, and each was constrained by computational bottlenecks that limited scalability and real-world deployment. Together, they illustrate the potential of ML-enhanced simulation for UAV autonomy but also expose the persistent shortcomings of scalable, distributed learning, and robust swarm UAVs coordination. Ahmed et al. [29] have developed a high-fidelity virtual test-bed using Gazebo and Python to simulate fault-tolerant control specifically for octo-rotor UAVs. Their framework enables the dynamic introduction of motor faults and environmental disturbances to test adaptive reinforcement learning approaches for maintaining stable flight. This open-source simulator facilitates faster design iterations and safer testing of complex autonomous behaviors before physical hardware-in-the-loop deployment. Zhang et al. [30] have designed a hardware-in-the-loop simulation (HILS) platform for UAV flight control systems based on the dSPACE real-time environment. The system utilizes Matlab/Simulink for nonlinear 6-DOF modeling and automatically generates C code to streamline the development of flight control laws. Their results demonstrate that this approach significantly increases automation while reducing the development time and costs associated with traditional manual programming methods.

Moving beyond single-aircraft experiments, Michael's micro-UAV testbed [21] introduced a platform that supported coordination among quadrotors for tasks such as surveillance and reconnaissance. Compared to other MATLAB co-simulation testbeds, Michael's work shifted attention toward cooperative behaviors and swarm-like dynamics. However, its reliance on quadrotors alone meant it was still a specialised platform, and it did not extend to heterogeneous or large-scale swarms. This marks an important turning point: while the other testbeds validated control principles, Michael's testbed demonstrated that swarms could be studied in simulation, but it also showed that

without scalability and distribution, such efforts remained limited in scope.

The research studies presented testbeds for multi-UAV systems, focusing on cooperative decision-making in complex and partially observable environments. Tai et al. [31] introduced the PyFlyt testbed as a standardized and flexible simulation testbed for training UAV control algorithms with realistic physics and performing planning tasks. Chen et al. [32] developed the Open testbed to address multi-UAV pursuit–evasion through online planning, adaptive environments, and realistic UAV dynamics, demonstrating improved generalization and real-world deployment. Khalil et al. [33] proposed the Replanner testbed, which uses an economic and auction-based learning strategy to efficiently coordinate UAV trajectories and task allocation within a swarm. Common similarities across these works include focusing on multi-UAV coordination, reliance on simulation-centric testbeds, reward-driven cooperation, and an emphasis on robustness. However, all three studies share limitations such as heavy dependence on simulator fidelity, limited large-scale real-world validation, and challenges in generalizing to larger swarms and highly complex real-world environments.

The GaTAC testbed [34] represented the next major advance. Unlike other platforms, GaTAC was explicitly designed as an open-source, flexible testbed for multi-agent RL. It demonstrated that policies could be evaluated in swarm-like environments, offering a level of scalability not found in any other testbed. Although GaTAC can run multiple FlightGear instances, it does not distribute the machine learning models or the autonomous decision-making processes across multiple systems; all control runs on a single machine. In contrast, my testbed is fully distributed learning, as each UAV is controlled by its own machine learning model running on a separate computer. GaTAC cannot handle large-scale, multi-agent swarm experiments where each aircraft is controlled by its own model running on a separate computer. Its reliance on environment-driven feedback left agents vulnerable to sparse or delayed rewards. In comparison to Michael's micro-UAV testbed, GaTAC was broader and more algorithmically flexible, but both highlighted the risks of unstable policy training when scaled up to swarm contexts.

Building on GaTAC's use of open-source tools and infrastructure, the NUAV testbed [6] took advantage of FlightGear and JSBSim to provide more realistic modelling and communication capabilities. Unlike GaTAC, NUAV incorporated Genetic Algorithms (GAs) and Artificial Neural Networks (ANNs) for multi-objective optimization, making it more versatile for algorithmic exploration. However, like its predecessor, it stopped

short of enabling true distribution across machines. Both GaTAC and NUAV were similar in their attempts to support swarm experimentation and in their reliance on centralized architectures. The common shortcomings, lack of distributed learning, and vulnerability to delayed reward problems, underline why new approaches are necessary.

Previous testbeds such as RUAV, autopilot-based FlightGear integrations, and similar platforms were not inherently incapable of being adapted for distributed computing. However, their architectures were not explicitly designed to support distributed learning across multiple machines or scalable multi-agent coordination in a networked environment. In most cases, these testbeds operated within a single-machine setup, where control algorithms and simulation processes were executed centrally. For example, early UAV testbeds primarily focused on validating control strategies for one or a small number of UAVs, with limited emphasis on scaling to larger swarms. While some platforms allowed multiple UAVs to be simulated, they did not provide mechanisms for distributing learning processes across separate computational nodes or for managing communication between agents in a distributed setting.

The Georgia Testbed for Autonomous Control (GaTAC) represents a step towards multi-UAV coordination within a shared simulation environment. However, even in this case, the focus is on coordinated control by distributing the FlightGear instances rather than distributed Machine Learning across multiple machines. This research addresses this gap by developing a testbed that explicitly supports multi-machine deployment, distributed learning, and scalable multi-agent coordination. In GaTAC, multiple instances of FlightGear are distributed across different machines, allowing several UAVs to be simulated simultaneously. However, this distribution primarily applies to the simulation layer, where each machine runs a visual and physics instance of a UAV. The control logic and decision-making processes remain largely centralised, meaning that the intelligence driving the UAVs is not distributed across machines.

In contrast, distributed Machine Learning refers to distributing the learning agents themselves across multiple machines. In this research, each UAV is controlled by an independent learning agent that runs on a separate computational node. These agents learn locally from their own interactions with the environment and communicate over a network when required. This means that both the simulation and the learning processes are distributed, rather than only the visual simulation.

The key difference is that distributing FlightGear increases simulation capacity, while distributing Machine Learning enables scalable and parallel learning across

multiple agents. By distributing the learning process, the testbed supports real-time multi-agent training, improves scalability, and allows each UAV to operate as an independent intelligent agent. This research addresses the gap by combining both aspects, where simulation instances and learning agents are distributed across multiple machines to enable scalable and coordinated multi-UAV behaviour.

Distributed learning in the context of GaTAC primarily refers to hardware distribution, where multiple UAV agents are executed across separate physical machines connected through a local area network. Each machine operates as an independent node and runs one or more learning agents, while communication between nodes is achieved through a network-based communication framework. This enables the testbed to scale across multiple machines and supports real-time multi-agent coordination. In addition to hardware distribution, the testbed also utilises data parallelism through the A3C algorithm. In this approach, multiple workers interact with separate instances of the environment and update a shared model asynchronously. These workers can run either on a single machine using multiple CPU cores or be distributed across multiple machines. This allows efficient exploration of the state space and improves learning speed. However, the testbed does not implement model parallelism, where a single model is split across multiple devices. Instead, each agent maintains its own model or interacts with a shared global model depending on the algorithm used. Therefore, the distributed nature of this work is primarily based on hardware distribution combined with data parallelism. This distinction clarifies that the testbed extends beyond single-machine execution and supports deployment across networked computational nodes. In the case of GaTAC, the limitation is not that it is incapable of distributed computing, but that its architecture is primarily centralised, with control and learning processes executed on a single machine. It does not explicitly implement hardware-level distribution of machine learning agents across multiple networked nodes, which is a key feature of the testbed developed in this research.

Viewed together, these testbeds demonstrate a clear progression. Early systems worked on the controllers and flight dynamics; mid-stage platforms coupled simulators with external tools for realism and education; later platforms like Michael's micro-UAV testbed moved toward advanced testbeds such as GaTAC and NUAV attempted to address scalability but fell short of reward stability. The similarities across them lie in their reliance on FlightGear and their contribution to shaping UAV simulation, while their differences highlight the gradual shift from single-aircraft validation to multi-agent

autonomy. What unites them all, however, is a persistent limitation: they did not distribute the ML models across machines, and they could not scale effectively to large swarms.

These limitations are important because real-world swarm UAVs must be able to scale across large numbers of agents, operate in distributed learning, and learn stably in dynamic environments. Without solving these issues, testbeds cannot replicate the challenges of real swarm missions. My testbed builds on this history by leveraging the open-source flexibility of FlightGear and JSBSim, while extending beyond GaTAC and NUAV to introduce true distributed learning across machines, scalability to larger swarms, and intrinsic curiosity modules to address delayed rewards. In doing so, it transforms the lessons of earlier testbeds into a robust and generalizable platform for cooperative swarm training. I have developed my testbed to overcome the limitations of the previous testbeds.

The NUAV testbed served as an initial reference point for selecting a simulation environment and baseline architecture; however, the testbed developed in this research introduces several independent architectural decisions and extensions that go beyond a direct continuation of NUAV. The elements adopted from NUAV include the use of FlightGear and JSBSim for realistic flight dynamics, as well as the use of fixed-wing UAV models. These components were selected after evaluating alternative simulators and platforms, as discussed in Chapter 2, where FlightGear was identified as suitable due to its open-source nature, flexibility, and support for real-time communication.

However, several aspects of the NUAV design were critically evaluated and not adopted:

1. NUAV follows a largely centralised control structure, which limits scalability. This approach was not adopted, and instead, a distributed learning architecture was implemented where multiple agents operate independently across different computational nodes.

2. NUAV does not integrate modern Machine Learning algorithms for adaptive control. This limitation led to the introduction of A2C, A3C, and PPO models, enabling learning-based decision-making instead of predefined control strategies.

3. NUAV lacks mechanisms to address delayed reward and exploration challenges. This resulted in the integration of the Intrinsic Curiosity Module to improve exploration and convergence behaviour.

4. NUAV does not support heterogeneous learning agents. In this work, different UAVs can be controlled by different algorithms such as A3C and PPO, allowing flexible and comparative evaluation within the same testbed.

5. NUAV does not provide multi-machine distributed execution. This work introduces a distributed learning framework using parallel workers and multi-node communication, enabling real-time coordination across separate systems.

6. In addition to rejecting these limitations, several independent architectural decisions were made:

7. A distributed learning architecture was designed where each UAV is controlled by an independent agent, enabling scalability and parallel execution.

8. A hybrid multi-algorithm framework was introduced, allowing different Machine Learning models to operate simultaneously within the same environment.

9. A curiosity-driven learning mechanism was integrated to address sparse and delayed reward problems.

10. A comprehensive validation approach was developed, including learning curves, trajectory plots, inter-UAV distance analysis, and simulation-based behavioural validation.

11. A real-time communication framework using UDP and Ray-based distributed execution was implemented to support synchronised multi-agent learning across multiple machines.

These contributions demonstrate that the testbed is not a direct extension of NUAV, but rather a redesigned and expanded architecture that addresses key limitations of earlier testbeds, particularly in terms of distributed learning, multi-agent coordination, and adaptive control.

Table 2.4 compares existing UAV testbeds by examining their models, core features, and major limitations. The table highlights that many earlier platforms were designed for single-aircraft testing and lacked both scalability and distributed learning. It also shows that most testbeds did not support advanced ML algorithms or multi-agent cooperation. By evaluating these gaps, the table demonstrates the need for a distributed learning testbed that can support multiple UAVs simultaneously. This comparison

motivates developing a more flexible and scalable testbed. It also establishes why integrating ML algorithms with high-fidelity simulation is essential for swarm coordination research.

Authors	**Focus / Methods**	**Features**	**Limitations**
Yasar et al. [23]	High-fidelity RUAV flight dynamics, hardware-in-the-loop emulation of Unmanned Ground Vehicless (UGVs), networked cooperative control	Early demonstration of heterogeneous UAV–UGV coordination with shared mission goals	Fixed to helicopter–UGV configuration, lacked scalability to swarms, not distributed learning across nodes
Zhang et al. [25]	FlightGear FDM integrated with custom autopilot, MATLAB-based controllers	Open-source flexible setup for prototyping and visualization of autopilot systems	Single UAV focus, no scalability, no distributed learning multi-agent training
Michael et al. [21]	Finite state machines, closed-loop quadrotor dynamics, cooperative surveillance/reconnaissance	Demonstrated quadrotor coordination in swarm-like behaviours, built on FlightGear + JSBSim	Limited to quadrotors, restricted scalability, lacked distributed learning architecture
Tai et al. [31]	Standardized RL/control testbed, realistic physics, customizable UAV models, planning tasks	Flexible modular design, supports control algorithm training and multi-UAV coordination	Simulator-dependent, limited real-world validation, scalability constraints
Chen et al. [32]	Multi-UAV pursuit–evasion via online planning, adaptive environments, realistic UAV dynamics	Improved generalization, supports real-world deployment, adaptive opponent modeling	Heavy simulator dependence, limited large-scale validation, difficult real-world transfer
Khalil et al. [33]	Auction-based learning, trajectory coordination, task allocation in swarms	Efficient resource allocation, robust to dynamic changes, supports heterogeneous UAVs	Computationally intensive, assumes perfect communication, scalability challenges
Sonu et al. [34]	Multi-agent RL testbed in C++ with agent communication and autonomous control policies	One of first scalable UAV RL testbeds, enabled multi-agent policy evaluation	Not distributed learning (single machine), sparse/delayed rewards, policy instability

Continued on next page

Table 2.4 (continued)

Authors	Focus / Methods	Features	Limitations
Habib et al. [6]	FlightGear + JSBSim, genetic algorithms, ANN-based adaptive control, communication modules	Realistic UAV modelling and algorithmic innovation, demonstrated swarm coordination	Centralised setup only, no true distribution, no intrinsic rewards, limited scalability
Shah et al. [26]	High-fidelity visual/physical simulation, Deep RL (DDPG, PPO, SAC), multi-modal sensors	Photo-realistic environments, supports vision-based navigation and obstacle avoidance	Computationally intensive, sim-to-real gap, limited swarm scalability, no distributed learning ML
Kaufmann et al. [27]	Deep RL and Imitation Learning, domain randomization, vision-only control	Strong sim-to-real transfer for agile flight, focused on high-speed drone racing	Narrow task scope, no swarm coordination, visual-input dependence
Ahmed et al. [29]	High-fidelity octo-rotor simulation for RL-based fault-tolerant control	Gazebo/Python interface, models motor faults and wind disturbances	Computational overhead, lacks multi-agent or swarm scalability
Zhang et al. [30]	dSPACE-based HILS platform for real-time UAV flight control	Matlab/Simulink 6-DOF modeling, automatic C-code generation	Proprietary hardware dependence

Table 2.4: Comparison of UAV testbeds: focus, features, and limitations.

2.6 Elements of a UAV Testbed

A UAV testbed must incorporate several essential elements identified across the literature, including an appropriate aircraft platform, a realistic simulation environment, and control algorithms capable of handling multi-agent, continuous, and long-horizon decision-making. Prior studies use a mix of rotary-wing, fixed-wing, and swarm configurations. A second essential element is the simulation environment. Numerous works rely on engines such as Gazebo, X-Plane, Microsoft Flight Simulator, and FlightGear, each providing different trade-offs in realism, open-source, cost, and extensibility. Research on optimisation, classical control, and various ML approaches demonstrates that swarm coordination requires algorithms capable of coping with delayed rewards, continuous control, and distributed learning multi-agent interactions. Together, these findings guide the selection of aircraft types, simulators, and ML algorithms in later parts of the thesis, ensuring that the testbed's design follows directly from the gaps and requirements identified in the literature rather than predetermined

choices.

The next section is based on the literature review of fixed-wing UAVs. It discusses how fixed-wing platforms have been employed in control, navigation, and mission tasks.

2.6.1 Fixed-wing UAVs

Fixed-wing UAVs are aircraft with rigid wings that generate lift through their forward motion across the air, propelled by engines or propellers. Unlike rotary-wing UAVs that can hover, fixed-wing UAVs require a runway or launcher for take-off and landing, offering greater speed and efficiency over long distances. Yan et al. [35] explored continuous actor–critic with experience replay for flocking control of fixed-wing UAVs, showing how RL controllers could regulate roll, pitch, and yaw motions. Their methodology lacks a collision avoidance mechanism, a similarity also seen in Jiang et al. [36], who used fixed-wing platforms with multispectral cameras to diagnose nitrogen levels in crops under static flight profiles. Both studies demonstrated how ML could enhance UAV mission capability, yet both highlighted a recurring weakness as they did not incorporate as collision avoidance mechanism for UAVs .

Other researchers focused on navigation and control robustness. Giral et al. [37] applied RL for waypoint tracking and target interception, while Bøhn et al. [38] used the soft actor–critic algorithm for attitude control, both demonstrating improved adaptability over traditional controllers similar to Yan et al. [35]. However, their models struggled to generalise when faced with real-world uncertainties such as wind disturbances and sensor noise. Borup et al. [39] approached the problem differently by using ML to estimate air data parameters from low-cost sensors, highlighting the importance of accurate state observation for reliable control. However, this method was constrained by dependence on specific sensor placements, limiting its transferability to different UAV designs. These works collectively illustrate the promise of RL-based controllers, but also the risk of overfitting to narrow conditions, underscoring the need for scalable solutions that can adapt across diverse scenarios.

Collision avoidance and safety were central themes in Zhao et al. [40], who designed a multi-agent RL testbed for dense airspace management, and Yan et al. [35], who later proposed local situation maps for flocking. Both efforts emphasised scalable multi-agent behaviour, but their approaches were computationally expensive and relied on simplified simulations that fell short of real-world unpredictability. Li et al. [41] worked on critical

manoeuvres and safety for automated landing with Proportional-Integral-Derivative (PID)-DQN hybrid framework, Cui et al. [42] worked on Long Short-Term Memory Networks (LSTM)-based fault diagnosis, and Bronz et al. [43] researched on Support Vector Machine (SVM)-based real-time fault detection, addressed important reliability aspects, but suffered from high training costs or limited datasets. Taken together, these studies reveal recurring barriers, computational inefficiency, lack of scalability, and limited generalisation, which matter because they prevent fixed-wing swarms from achieving safe, adaptive, and reliable deployment in uncertain environments. My work synthesises these insights by providing a distributed learning FlightGear-based testbed that incorporates curiosity-driven exploration and Gaussian reward shaping to address UAV collisions and delayed rewards, ensuring robust swarm coordination that earlier studies could not achieve.

Fixed-wing UAVs were selected in this project because they align closely with the requirements of long-range target tracking. Their higher speeds, longer endurance, and energy efficiency allow them to sustain tracking over extended distances without frequent interruptions for refuelling or recharging. Compared to rotary-wing platforms, fixed-wing UAVs are more suitable for demonstrating realistic swarm behaviour in endurance-based missions. Prior testbeds such as NUAV also used fixed-wing aircraft, showing their effectiveness in scalable flight dynamics simulations.

Table 2.5 summarises previous research focused on fixed-wing UAVs, listing the methods used, key features, and limitations. It shows how different studies addressed flight dynamics, stability modelling, and control strategies for fixed-wing aircraft. The table reveals that many works relied on simplified assumptions, limited autonomy, or predefined flight paths. It also highlights the absence of real-time learning and multi-agent target-tracking behaviour in earlier studies. By providing this comparison, the table clarifies why fixed-wing aircraft were selected for this thesis and why they require a more intelligent control setup. This directly supports the need for a scalable testbed capable of integrating learning algorithms.

Authors	Focus / Methods	Features	Limitations
Yan et al. [35]	Continuous Actor–Critic with Experience Replay for flocking control of fixed-wing UAVs	Demonstrated stable control of roll, pitch, yaw, and velocity in continuous spaces; outperformed DDPG and double DQN in multi-agent flocking	Assumed fixed but different altitudes for collision avoidance, limiting realism for 3D manoeuvres and terrain-following scenarios
Jiang et al. [36]	Fixed-wing UAVs with multispectral cameras and ML models for nitrogen diagnostics	Showed integration of aerial sensing with agronomic and meteorological data; validated RL policies for agricultural monitoring missions	Operated at static altitudes, reduced adaptability to dynamic terrains, poor generalisation to real-world variability
Giral et al. [37]	RL for waypoint tracking and dynamic interception of fixed-wing UAVs	Improved robustness and adaptability of Guidance, Navigation, and Control (GNC) systems, enabling pursuit and interception tasks	Policies struggled to generalise under real-world uncertainties such as wind gusts and sensor noise
Borup et al. [39]	ML (Neural Networkss (NNs), regression) for air data estimation from low-cost Micro-electromechanical systems (MEMS) sensors	Cost-effective estimation of angle of attack, sideslip, and airspeed, supporting low-cost UAV design	Strong dependence on sensor placement and calibration; results lacked transferability across platforms
Zhao et al. [40]	Multi-agent RL with Markov games for fixed-wing collision avoidance	Improved safety in dense airspaces with adaptive decision-making and cooperative planning	Computationally expensive, scalability issues in larger swarms, and incomplete conflict avoidance guarantees
Bøhn et al. [38]	Soft Actor–Critic for low-level attitude control of fixed-wing UAVs	Data-efficient learning with improved adaptability and robustness under varying flight conditions	Risk of overfitting to narrow training scenarios; controller performance depended heavily on simulator fidelity
Giral et al. [44]	Actor–Critic approaches for waypoint tracking and interception	Simplified complex GNC tasks using adaptive reinforcement learning policies	Weak generalisation to real-world disturbances and environmental variability

Continued on next page

Table 2.5 (continued)

Authors	Focus / Methods	Features	Limitations
Bøhn et al. [45]	PPO for attitude regulation of fixed-wing aircraft	Outperformed traditional PID controllers under wind and turbulence, demonstrating RL advantages	Low sample efficiency; significant sim-to-real transfer gap limited practical applicability
Yan et al. [35]	Deep RL with local situation maps for scalable flocking and collision-free navigation	Enabled scalable multi-agent navigation and cooperative flocking strategies	Relied on simplified assumptions; lacked adaptability to unpredictable and dynamic environments
Li et al. [41]	Hybrid PID + DQN for automated landing of fixed-wing UAVs	Combined classical control with RL, improving adaptability in complex landing phases	Computationally intensive; required large volumes of training data for convergence
Cui et al. [42]	LSTM-based deep learning for fault diagnosis and health management	Provided predictive fault detection and proactive health management for UAV operations	Over-reliance on simulated datasets; rare failure cases were underrepresented in training
Bronz et al. [43]	SVM for real-time fixed-wing fault detection	Achieved up to 95% accuracy in fault detection under noisy flight conditions	Small datasets increased risk of overfitting; performance sensitive to environmental variability

Table 2.5: Summary of fixed-wing UAV research: methods, features, and limitations.

A UAV testbed requires a suitable flight simulation environment, and the literature highlights several broad options, such as Gazebo, X-Plane, Microsoft Flight Simulator, and SimplePlanes, each offering different levels of realism, extensibility, and multi-agent capability [46]. However, these simulators also share common limitations: many lack accurate fixed-wing modelling, provide limited support for large numbers of simultaneous aircraft, or cannot be deployed in distributed learning configurations needed for modern swarm research [47]. Others impose closed-source restrictions or offer weak interfaces for integrating advanced RL algorithms, which restrict experimentation with adaptive and cooperative behaviours.

Gazebo is mainly designed for robotics research and is widely used for UAV control and autonomy studies. It supports multiple vehicles, physics-based simulation, and easy integration with control software and learning algorithms. However, its visual quality is limited, and creating large, realistic outdoor environments is difficult, which reduces

realism for flight-based experiments. X-Plane is a commercial flight simulator with very accurate aircraft dynamics. It models aerodynamic forces in detail and is useful for studying aircraft behaviour. However, it is not well-suited for distributed or multi-agent learning. It also requires licensing and offers limited flexibility for integrating custom machine learning algorithms across multiple aircraft. Microsoft Flight Simulator focuses heavily on visual realism, real-world terrain, and weather effects. While it provides impressive graphics, it is not designed for research-oriented control or learning tasks. Its architecture is closed, computationally heavy, and not suitable for real-time distributed learning or multi-UAV coordination. Overall, although these simulators are useful in specific contexts, they are not ideal for building a scalable and distributed learning-based UAV testbed, which requires flexibility, real-time communication, and tight integration with learning algorithms.

These shortcomings make it essential to carefully justify the choice of simulator when designing a scalable UAV testbed. This broader assessment motivates the selection of FlightGear, which offers open-source accessibility, high-fidelity JSBSim dynamics, and multi-instance flexibility suitable for distributed learning swarm experimentation [46]. The next section, therefore, examines FlightGear in detail as a core element of the testbed.

The next section is based on the literature review of the FlightGear simulator. It explores its role as an open-source flight simulator in UAV research, emphasizing its strengths, adoption in prior research papers, and the limitations that need to be addressed.

2.6.2 FlightGear

FlightGear is an open-source flight simulator that provides realistic modelling of aircraft dynamics and environments. It supports a wide variety of aircraft models, is highly customizable, and integrates smoothly with FDMs. Because it is free, flexible, and widely used in both academia and research projects, FlightGear is an ideal platform for building scalable and distributed learning UAV testbeds. Metodiev et al. [48] demonstrated how FlightGear can be used to test aircraft models designed in Computer Aided Design (CAD) by converting them into simulation-ready assets. Their work was important because it established a pipeline from geometry to flyable vehicles, which is essential for training and evaluation in UAV testbeds. However, the absence of technical detail on CAD conversion limited reproducibility and hindered scaling to multiple aircraft, as other researchers cannot replicate or adapt the workflow. This lack of

transparency also hinders scalability, because without a documented and repeatable conversion pipeline, it becomes difficult to create, validate, and deploy multiple aircraft models for swarm-based experiments. In contrast, Wood et al. [49] focused on GymFG, an interface linking FlightGear with RL environments. Their contribution was to highlight FlightGear's potential as an agent simulation platform. Without multi-agent or distributed learning demonstrations, it also fell short of proving scalability. Together, these works underline two necessary but incomplete aspects of UAV testbeds: the ability to bring realistic aircraft into the simulator and the ability to train them using ML frameworks.

Pahlevi et al. [50] extended the use of FlightGear to hypersonic control, proving that its dynamics engine coupled with JSBSim could support highly demanding non-linear control experiments. Unlike Wood et al. [49], whose focus was learning-based navigation, Pahlevi et al. [50] concentrated on model-based inverse dynamics, confirming FlightGear's realism but only for single-vehicle settings. Macleod et al. [51] moved toward scalability by proposing a parallel architecture for FlightGear to better support multi-user sessions, which is highly relevant to distributed learning swarms. Yet their proposal lacked integration with ML agents, limiting practical impact. These contributions, while distinct in emphasis, precision control versus performance scalability, highlight how FlightGear can host both advanced controllers and multi-instance environments, but also reveal gaps when applied to distributed learning swarm UAVs.

Other researchers highlighted mission realism and fidelity. Cervin et al. [52] built a 3D mission planning system combining FlightGear with geographic information system-based data, reinforcing the need for topographical realism in simulation, though the technical gaps in terrain data handling limited adoption. Coopmans et al. [53] confirmed FlightGear's fidelity through software- and hardware-in-the-loop comparisons with real flights, validating its suitability for training agents intended for real deployment. Both studies reinforced FlightGear's strength in realism but, like earlier work, remained confined to single-vehicle or non-learning applications. Cao et al. [54] added another perspective by detailing the open interfaces of the simulator's open FDM, which enable external program integration. However, the lack of consistent documentation and coding conventions slowed the development of reusable learning frameworks. These works collectively illustrate that realism and integration are FlightGear's strong points, but they also show the fragmentation and single-agent

limitations that must be addressed for swarm applications.

Sharma et al. [55] demonstrated how FlightGear could be used to evaluate GA-tuned PID controllers for autopilot stability, establishing it as a cost-effective environment for baseline testing. Unlike Wood et al. [49] or Pahlevi et al.[50], Sharma et al. [55] focused on classical control rather than ML, providing a useful framework but not advancing distributed learning or learning-based scalability. Across all of these studies, a pattern emerges: FlightGear has been successfully employed for project development, high-fidelity control, mission realism, hardware integration, and even as a learning sandbox, but each effort stops short of creating a distributed learning, scalable swarm training platform. The limitations, ranging from single-agent focus to low-quality documentation, lack of distributed learning, and absence of intrinsic reward mechanisms, are critical because they prevent these systems from addressing delayed rewards, policy destabilisation, and real-time coordination across swarms.

My work synthesises these contributions by leveraging FlightGear's realism and open interfaces but extending them into a distributed learning testbed architecture. By launching multiple FlightGear processes across nodes, synchronising them via multiplayer and Ray, and embedding A2C, A3C, and PPO controllers with intrinsic curiosity and Gaussian reward shaping, the testbed resolves the recurring weaknesses identified in prior work. This transforms FlightGear from a valuable single-agent simulator into a reproducible, distributed learning, and scalable platform capable of training robust UAV swarms for target tracking and coordination tasks.

FlightGear was selected because it is open-source, cross-platform, and already widely adopted in research and testbeds such as GaTAC and NUAV. Its seamless integration with JSBSim, extensive library of aircraft models, and active developer community make it a flexible and reliable choice for building a distributed learning and scalable testbed for swarm UAV target tracking. Its flight dynamics are powered by JSBSim, providing realistic and accurate models of aircraft behaviour. FlightGear is scalable, lightweight compared to commercial simulators, and supports multi-agent and distributed learning testbed environments.

JSBSim is an open-source Flight Dynamics Model (FDM) that simulates the physical behaviour of aircraft based on aerodynamic, propulsion, and environmental forces. It models the six degrees of freedom motion of an aircraft by computing forces and moments acting on the vehicle in real time, using a set of nonlinear differential equations. These include aerodynamic lift and drag, thrust generated by the propulsion

system, gravitational effects, and control surface inputs such as ailerons, elevators, and rudders. The simulator operates by continuously updating the aircraft state, including position, velocity, orientation, and angular rates, based on these forces and moments. This provides a physically accurate representation of aircraft motion, allowing a realistic simulation of flight dynamics under different conditions. In this research, JSBSim is integrated with FlightGear to provide both accurate physics and visual simulation. While FlightGear is responsible for rendering the environment and visualising the UAVs, JSBSim ensures that the motion of each UAV follows realistic aerodynamic principles. This combination enables the testbed to simulate real-world flight behaviour without requiring physical aircraft. The use of JSBSim strengthens the realism of the testbed by ensuring that the Machine Learning agents interact with a physics-based environment rather than a simplified or abstract model. This is particularly important for UAV coordination tasks, where accurate modelling of motion and control dynamics directly affects learning behaviour and performance evaluation. Therefore, JSBSim plays a critical role in bridging the gap between simulation and real-world flight dynamics, supporting the validity of the experimental results presented in this research.

Table 2.6 presents a summary of studies that used the FlightGear simulator for control, navigation, or system validation. It highlights the main focus of each study, the FlightGear features used, and the limitations that constrained their applicability. The table shows that although FlightGear provides realistic modelling, most studies used it only for single-UAV validation or basic control testing. The table also illustrates the consistent reliability of FlightGear across past research. It reinforces the reasoning behind its selection as the main simulation environment in this thesis.

Authors	**Focus / Methods**	**Features**	**Limitations**
Metodiev et al. [48]	Conversion of CAD-designed aircraft into FlightGear-compatible models, followed by simulated flight tests	Demonstrates an end-to-end pipeline from geometry modelling to flyable assets inside FlightGear, validating design in simulation	Provides no technical detail on the CAD-to-FlightGear conversion process, limiting reproducibility and large-scale asset development for swarms

Continued on next page

Table 2.6 (continued)

Authors	Focus / Methods	Features	Limitations
Wood et al. [49]	Development of GymFG linking FlightGear with RL frameworks	Enables training of self-navigating aerial agents in a high-fidelity simulator; practical interface for control and observation exchange	Lacks comparative evaluation against other simulators; scalability and multi-agent distribution not discussed, restricting relevance for swarm-level research
Pahlevi et al. [50]	Inverse dynamics-based control for hypersonic flight using FlightGear and Flight Dynamics Simulator (JSBSim)	Showcases FlightGear's realism in handling nonlinear dynamics and demanding speed/altitude regulation	Focused only on single-vehicle regulation with model-based control, no exploration of multi-agent training or learning-based adaptability
Macleod et al. [51]	Proposed parallel processing architecture for FlightGear to enhance simulation throughput and multiuser performance	Potentially supports heavier workloads and larger-scale simulation sessions by leveraging multi-core parallelism	Does not include schematics, or integration examples with ML or RL controllers; practical scalability remains untested
Cervin et al. [52]	Real-time 3D mission planning and remote aircraft operation with FlightGear, integrating Geographic Information Systems (GIS) topography and Joint Geospatial Facility (JGF) imagery	Introduces terrain-aware mission planning interface with real-time 3D visualisation for operator decision support	Provides little technical detail on real-time topography processing (latency, tiling, coordinate transforms), leaving scalability questions unanswered
Coopmans et al. [53]	Software- and hardware-in-the-loop validation of a fixed-wing UAV using FlightGear for visualisation and dynamics	Validates fidelity of FlightGear/JSBSim dynamics by comparing with real-world flight tests; enables fault injection and wind testing	Limited to single-vehicle analysis; lacks distributed learning or multi-agent RL integration for swarms
Cao et al. [54]	Study of FlightGear's FDM interface to external programs for algorithm integration	Highlights open-source extensibility; allows researchers to focus on control logic instead of rendering or visualisation	Inconsistent coding styles and sparse documentation in FlightGear's open-source codebase make extension and onboarding challenging

Continued on next page

Table 2.6 (continued)

Authors	Focus / Methods	Features	Limitations
Sharma et al. [55]	GA-optimised PID controllers evaluated in FlightGear	Demonstrates cost-effective and risk-free tuning of autopilot gains in a high-fidelity environment	GA tuning highly sensitive to population size and mutation rates; lacks scalability and no transition towards adaptive or RL-based control

Table 2.6: Summary of FlightGear research: focus, features, and limitations.

The next section is based on the review of the literature on swarm UAVs.

2.6.3 Swarm UAVs

Swarm UAVs are groups of UAVs that work together to achieve a common goal through cooperation and coordination. By sharing information and distributing tasks, they can cover larger areas and track targets more effectively. This collective behaviour makes swarm UAVs well suited for complex missions such as surveillance, search and rescue, and target tracking. Tosato et al. [56] highlighted the industrial potential of swarm UAVs by deploying miniaturized sensors for large-scale gas monitoring, demonstrating why a single platform is often insufficient for complex missions. Their work revealed the importance of cooperative sensing but also showed how turbulence from propellers compromised reliability. In contrast, Majd et al. [57] pursued collision avoidance and safety through genetic algorithms, emphasising real-time trajectory planning within swarms. While both studies demonstrated how collective behaviour extends system capability, they also shared a limitation: neither provided stable, scalable mechanisms for generalising behaviour to new conditions. This is precisely why scalability and policy stability, which are critical for real-world swarms, remain open challenges.

Other authors extended the discussion toward scalability in trajectory prediction and swarm chasing. Raj et al. [58] proposed a new activation function to improve prediction accuracy under noisy conditions, offering value for trajectory management but producing convoluted paths in practice. Ban et al. [59] instead employed asynchronous deep RL for multidimensional swarm chasing, aligning closely with my use of the A3C algorithm for efficient learning. Yet their approach suffered from computational overhead due to sophisticated channel modelling, which restricted practical scalability. Both works addressed the challenge of managing large swarms but revealed different

weaknesses: one generated inefficient paths, while the other introduced computational bottlenecks. These limitations highlight why distributed learning is necessary to support large-scale UAV coordination without excessive computational demands.

Applications of swarm UAVs in critical environments were further demonstrated by Diaz-Vilor [60], who optimised wildfire tracking with trajectory adjustments using deep RL. Gao et al. [61] extended this theme with cooperative wireless coverage in search and rescue, applying machine learning to overcome stochastic communication failures. Both studies showcased the adaptability of swarm UAVs under uncertainty but also revealed the cost of high computational demand and communication overhead, issues that constrain deployment at scale. Dhuheir et al. [62] similarly focused on machine learning inference for trajectory planning in resource-constrained conditions, showing the promise of deep RL but again raising concerns about real-time efficiency. Collectively, these works show that while distributed learning methods are essential for swarms, existing approaches often remain computationally prohibitive and lack efficient mechanisms to ensure distributed learning under real-world constraints.

Further progress was made by Ding et al. [63], who explored multi-agent learning inference for swarm UAVs in wireless networks, and by Venturini et al. [64], who applied RL to cooperative area surveillance. Both demonstrated the potential of RL for swarm cooperation, but both also faced significant challenges of computational cost and scalability. Zhao et al. [65] added another perspective by introducing complementary collaboration for heterogeneous swarms, emphasising adaptability across UAVs with different roles. This connects directly to my testbed, where heterogeneous roles of tracker UAVs are modelled in a distributed learning setting.

Taken together, these studies demonstrate that swarm UAVs have clear advantages in scalability, resilience, and mission flexibility, but they consistently suffer from limitations such as inefficient trajectories, computational bottlenecks, and a lack of robust scalability and distributed learning. These shortcomings matter because without scalable and stable coordination, swarm UAVs cannot reliably operate in complex, dynamic environments. My work builds upon these insights by creating a distributed learning FlightGear-based testbed where A2C,A3C, and PPO algorithms operate in parallel, strengthened by intrinsic curiosity to handle delayed rewards and Gaussian reward shaping to stabilise learning. This synthesis transforms swarm UAV research from conceptually promising but computationally heavy testbed into a practical, scalable, and adaptive platform for real-time coordination.

Swarm UAVs were chosen because they provide robustness, adaptability, and cooperative capability beyond what a single UAV can achieve. Distributing the target tracking task across multiple aircraft increases fault tolerance: if one UAV fails, others can continue the mission. Within a distributed learning and scalable testbed, swarm UAVs demonstrate the advantages of parallel learning, asynchronous coordination, and real-time collaboration.

Table 2.7 summarises existing research on swarm UAVs, focusing on cooperation, communication, formation control, and decision-making. It highlights how different algorithms were used for coordination and what specific tasks each swarm system performed. The table also lists common limitations such as a lack of realistic physics, an absence of high-fidelity simulation, or restricted swarm size. It becomes clear that most studies were constrained by computational limits or non-scalable architectures. This comparison illustrates that although swarm concepts are widely studied, very few testbeds combine realism with scalability. The table also supports the motivation for building a distributed learning testbed that manages several learners in real-time. Overall, it identifies clear gaps that my testbed directly addresses.

Authors	**Focus / Methods**	**Features**	**Limitations**
Tosato et al. [56]	Swarm UAVs with miniaturized onboard sensors for industrial gas sensing	Demonstrates cooperative sensing capability, enabling larger-scale monitoring than a single UAV	Sensor placement sensitive to turbulence from propellers, reducing measurement reliability
Majd et al. [57]	Parallel evolutionary computing for real-time swarm mission planning and collision avoidance	Enhances motion safety and trajectory efficiency using evolutionary optimisation	No details on hyperparameter tuning, reproducibility uncertain, scalability not addressed
Raj et al. [58]	Novel NN activation function (Adapto-Swelli-Gauss) for trajectory prediction and avoidance in large swarms	Increases prediction accuracy for noisy nonlinear data and improves swarm safety	Generated trajectories were convoluted and inefficient, raising real-time applicability concerns
Ban et al. [59]	Deep RL with asynchronous multi-agent learning for multidimensional swarm chasing	Improves adaptability, coordination, and response time across swarm agents	High computational load due to advanced channel modelling, scalability limited for large deployments

Continued on next page

Table 2.7 (continued)

Authors	Focus / Methods	Features	Limitations
Diaz-Vilor [60]	RL-based trajectory optimisation for wildfire tracking using twin-delayed deterministic policy gradients	Maintains network connectivity, resilience, and coverage under dynamic wildfire conditions	Limited by UAV energy constraints and need for frequent recharging, reducing long-term continuity
Gao et al. [61]	distributed learning algorithm for cooperative wireless coverage in swarm UAVs	Enhances downlink coverage in search-and-rescue missions, adapts through local decision-making	Requires high computational resources, limiting real-time scalability and rapid swarm deployment
Dhuheir et al. [62]	Deep RL for distributed learning inference and adaptive trajectory planning in surveillance swarms	Reduces latency, dynamically optimises paths, improves task coordination	Computational overhead significant, restricting efficiency and scalability for large swarms
Ding et al. [63]	ML testbed for swarm trajectory planning, power control, and sensing	Integrates multi-agent RL, distributed learning inference, and emerging communication technologies	Very high computational and communication overhead, weak real-time performance
Venturini et al. [64]	RL-based cooperative area surveillance and monitoring with swarm UAVs	Promotes cooperative exploration, robustness against communication disruptions	Computational burden heavy, limiting scalability and rapid field deployment
Zhao et al. [65]	Complementary collaboration for heterogeneous swarms using adaptive behaviour differentiation	Improves adaptability and decision-making in diverse UAV roles under complex conditions	Real-time behaviour adjustment increases complexity, scalability remains an issue for large heterogeneous swarms

Table 2.7: Summary of swarm UAV research: focus, features, and limitations.

The computational challenges in RL are various, but they all revolve around one common factor: RL is essentially sequential since the training data is created during the learning process [24]. The reason for selecting RL over supervised or unsupervised learning is that we did not have the data in hand in the initial stages. In order to work with supervised and unsupervised learning, we need to have the dataset initially. Then, we apply supervised/unsupervised learning to the data for classification and regression problems. Our research problem was not based on classification or regression. We need an

agent to explore the environment. The agent adjusts its behaviour over time by learning from its environment and receiving feedback, usually in the form of rewards and penalties. The next section is based on the literature review of RL algorithms.

2.6.4 Reinforcement Learning Algorithms

RL is a multidisciplinary branch of ML that studies how intelligent agents should behave in an unpredictable setting to obtain the greatest cumulative reward. The RL agent perceives and interprets its surroundings and learns by mistakes. The agent adjusts its behaviour over time by learning from its environment and receiving feedback, usually in the form of rewards and penalties. RL algorithms are divided into value-based, policy-based, and actor-critic algorithms. Value-based algorithms are designed to learn the value of each environmental state. A state's value is determined by the expected rewards associated with completing a task from the current condition onwards. Policy-based algorithms directly update the policy to maximise the reward. The actor–critic framework is a class of reinforcement learning algorithms that combines two complementary components: an actor, which selects actions according to a learned policy, and a critic, which evaluates those actions by estimating the value function. The actor updates the policy parameters in the direction suggested by the critic, while the critic reduces prediction error through value estimation.

A2C is a type of RL algorithm. Actor and critic neural networks collaborate to find a solution to a given problem in the actor-critic algorithm. The actor NN performs actions that result in a reward or penalty and a change in the state of an agent. The critic generates the value to evaluate the actions of the actor.

A3C is a powerful RL algorithm that enables agents to learn optimal actions in complex environments. It has been successfully applied to various tasks, such as video games, robot control, and traffic optimisation. It works by updating the agent's policy and value functions asynchronously, allowing for faster learning and better performance. The policy function tells the agent what action to take in each situation. The value function tells the agent how good a situation or action is for getting future rewards.

The global network controls every agent. Every agent asynchronously updates the parameters of the global network. Curiosity-driven learning encourages an agent to explore unfamiliar parts of the environment, and the ICM is a popular module that implements this idea by producing an intrinsic reward for novel or unpredictable

experiences.

PPO architecture consists of an actor-critic framework, policy optimisation through clipping, and an advantage estimator, all working together to achieve a stable and efficient RL framework. One of the core innovations in PPO is the clipping mechanism used to stabilise policy updates. Traditional policy gradient methods can suffer from instability because updating the policy too aggressively can lead to poor performance. PPO solves this by limiting how much the policy can change in a single update step, ensuring that the updates remain small but effective. This prevents drastic policy shifts and makes training more stable and reliable.

The A2C algorithm was selected to control the target aircraft because it was the building block of A3C algorithm and provides a synchronous actor–critic framework, which serves as the building block of the A3C algorithm, making it an essential choice for this project, as it establishes the foundational architecture upon which asynchronous extensions are developed. I have only one target aircraft, so I used the A2C algorithm to control it, whereas I have more than one tracking aircraft; thus, I used the asynchronous version, the A3C algorithm.

The A3C algorithm was chosen for the swarm of tracking aircraft because it introduces asynchronous parallel workers that enable diverse exploration of the environment, reduce gradient variance, and accelerate convergence. These characteristics are crucial for swarm coordination, where multiple UAVs must cooperate in real time within a distributed learning testbed. Since A3C builds upon the actor-critic principles of A2C but extends them to an asynchronous setting, it is particularly well suited for managing the scalability and distributed learning nature of training for multi-agent swarm tracking tasks.
The PPO algorithm was also employed for tracker aircraft to overcome a key limitation of A3C, namely policy degeneration, where the agent gets stuck in a habit of repeating poor actions that give low rewards, instead of exploring or improving toward better actions. Policy degeneration does not occur every time, but there is a risk that it can happen rarely. PPO addresses this issue through its clipped surrogate objective, which stabilises policy updates and ensures incremental improvements. Furthermore, PPO improves sample efficiency by allowing multiple training epochs on the same data, leading to faster and more robust learning. These advantages make PPO a strong complement to A3C for reliable swarm tracking.

As the literature on UAV control, swarm coordination, and autonomous decision-making demonstrates, modern testbeds increasingly rely on ML methods

capable of handling continuous dynamics, uncertainty, delayed rewards, and multi-agent interactions [66]. Traditional optimisation or rule-based controllers struggle with these demands, particularly in real-time aerial environments. Across numerous studies, reinforcement learning—especially actor–critic architectures—consistently appears as a promising solution because it combines policy flexibility with value-based stability and can be deployed synchronously or asynchronously depending on the task. Likewise, curiosity-driven exploration methods such as ICM frequently arise in settings where external rewards are sparse or delayed, a common challenge in UAV pursuit and swarm behaviour. Taken together, this evidence from prior research motivates a focused review of algorithms such as A2C, A3C, PPO, and ICM. Academic and technical literature repeatedly identifies them as well-suited for scalable, distributed learning, and adaptive UAV testbeds [25].

The next subsection is based on the literature review of the A2C algorithm. It examines how synchronous actor–critic methods have been applied in UAV control and what limitations arise in dynamic environments.

2.6.4.1 A2C algorithm

The A2C algorithm has been widely explored in different domains, from scheduling to aerial navigation, and its synchronous actor–critic structure makes it particularly suited for stable policy learning. Gerpott et al. [67] demonstrated how A2C can coordinate multiple agents in production scheduling, reducing delays in a machine learning system. Han et al. [68] extended this to obstacle avoidance for UAVs, accelerating training through shared experiences across agents. Both studies show that synchronous actor–critic learning is powerful in multi-agent settings, but they stop short of addressing the long-horizon delayed rewards that arise in continuous flight or the scalability needed for swarms. Yan et al. [69] added to this by employing A2C to achieve collision-aware planning, but this reliance on simplified dynamics limited realism. In my testbed, the target aircraft instead employs pure A2C inside a physics-accurate FlightGear–JSBSim loop, producing trajectories that are more representative of real-world flight conditions.

Other domains also confirm the adaptability of A2C. Zhang et al. [70] used a heuristic duelling-network variant for cognitive jamming, and Lu et al. [71] applied it for dynamic task scheduling in edge–cloud systems. Both underline the value of reward

shaping and synchronous actor–critic for stability in uncertain environments. However, their environments were small-scale or synthetic, lacking the continuous, high-dimensional dynamics of aircraft flight. My contribution extends their insights into a real-time flight simulator, where the target's stability under multi-agent interactions is critical.

Surveys such as Jiménez et al. [14] highlight A2C as a candidate for UAV autonomy, while Choppara et al. [72] applied it to trust-aware scheduling in fog computing, both reinforcing its reputation for robustness under uncertainty. Yet, the lack of implementation in high-fidelity simulators remains a limitation. Liu et al. [73] went further, applying A2C to multi-target allocation with hundreds of agents, showing that the method can remain stable at a non-trivial scale. Still, their reward design was task-specific and limited in generality. By contrast, my testbed embeds A2C in FlightGear/JSBSim as the target-aircraft controller.

In summary, these prior studies collectively confirm that A2C is a strong candidate for stable, multi-agent learning but also reveal persistent gaps: limited realism, insufficient handling of delayed rewards, and restricted scalability. In my distributed learning testbed, A2C governs the target aircraft, generating reliable yet non-trivial trajectories that trackers must pursue. This role requires reproducibility rather than rapid adaptation, which differentiates it from the use of A3C or PPO in tracking agents. Prior studies show both the strengths and the limitations of A2C, which collectively highlight the need for stabilisation mechanisms such as intrinsic curiosity and Gaussian reward shaping to manage delayed and sparse feedback.

The A2C algorithm was chosen to control the target UAV because it provides a straightforward synchronous actor–critic framework. Since only one target aircraft is used in this project, the efficiency of A2C makes it a suitable choice without the added complexity of asynchronous methods. It establishes a reliable baseline policy that enables the tracking aircraft to follow flight trajectories, which is essential for evaluating the performance of multiple tracking UAVs. Furthermore, A2C serves as the foundation of more advanced algorithms like A3C, making it an appropriate starting point for building and validating the distributed learning and scalable testbed.

Table 2.8 summarises studies where the A2C algorithm was used, outlining the task domains, main features, and identified limitations. It shows that A2C performs well in single-agent environments due to its synchronous structure and stable actor–critic updates. However, the table also reveals challenges such as slow exploration, sensitivity

to reward delays, and difficulties when extended to multi-agent setups. Overall, the comparison aligns with the decision to pair A2C with more advanced algorithms in the testbed.

Authors	**Focus / Methods**	**Features**	**Limitations**
Gerpott et al. [67]	A2C applied to production scheduling in hybrid multi-agent flow shops	Demonstrates that synchronous actor–critic reduces aggregate task delays and stabilises coordination	No integration with aerial or continuous-time systems; ignores long-horizon sparse reward problems critical for flight tasks
Han et al. [68]	A2C for UAVs obstacle avoidance with shared multi-agent experience replay	Shows stable obstacle avoidance behaviour and accelerated learning via shared knowledge	Evaluated on small UAV teams; does not address delayed or sparse reward challenges in long pursuit tasks
Yan et al. [69]	Multi-agent A2C integrated with artificial potential fields for collision-aware path planning	Coordinates multiple UAVs, reduces collisions and ensures cooperative motion planning	Simplified to 2D artificial potential fields, unrealistic in 3D aerodynamics; performance drops in dynamic or noisy environments
Zhang et al. [70]	Heuristic duelling-network A2C for cognitive radio jamming and anti-jamming	Achieves faster convergence and improved stability in discrete, adversarial decision-making	Evaluated only in small-scale, discrete state–action spaces; not scalable to continuous UAV flight dynamics
Lu et al. [71]	A2C for dynamic task scheduling in edge–cloud computing with mixed action spaces	Adapts efficiently to fluctuating computational loads and heterogeneous resources	Tested only on synthetic workloads; lacks validation in physical systems or UAV-like real-time environments
Jiménez et al. [14]	Comprehensive survey of RL methods for multi-UAV autonomy, including A2C	Identifies coordination, decentralisation, tracking, and dynamic decision-making as core strengths	Remains conceptual and descriptive; no experimental demonstration of A2C in real or simulated aerial environments
Choppara et al. [72]	Reliability- and trust-aware scheduling in fog/edge computing with A2C	Learns robust policies under uncertain node behaviour, dynamically adapts to unreliable conditions	Restricted to simulation traces; reacts slowly to sudden failures; no link to continuous control domains like aviation

Continued on next page

Table 2.8 (continued)

Authors	Focus / Methods	Features	Limitations
Liu et al. [73]	distributed learning A2C for large-scale multi-target allocation involving hundreds of agents	Demonstrates stable synchronous updates at scale, efficient convergence for resource assignment	Task-specific reward function design limits generalisability; scalability beyond reported settings not fully analysed

Table 2.8: Summary of research studies using the A2C algorithm: focus, features, and limitations.

The next subsection is based on the literature review of the A3C algorithm. It discusses how asynchronous workers improve exploration.

2.6.4.2 A3C algorithm

The A3C algorithm has been explored across diverse domains, from cloud scheduling to aerial manoeuvres, and these studies collectively highlight its strength in asynchronous training tasks. Tuli et al. [74] demonstrated how A3C can coordinate probabilistic edge–cloud scheduling, proving its capacity to manage RL agents under uncertainty. Chen et al. [75] extended this idea to mobile vehicle services, where asynchronous learning enabled dynamic adaptation in fluctuating network conditions. Both works reinforce the ability of A3C to handle decentralised decision-making, but their absence of flight dynamics and focus on computing or vehicular contexts limit their relevance to autonomous aerial systems. My work builds upon these insights by embedding A3C directly into a distributed learning aerial testbed, where tracking aircraft must adapt in real time to the unpredictable motion of a target UAV.

Other researchers have applied A3C to resource-constrained environments. Du et al. [76] used it for blockchain-enabled mobile systems, while Lee et al. [77] applied a hierarchical A3C to reduce large state spaces in multi-agent environments. These approaches share an emphasis on scalability and efficient exploration, yet both suffer from oversimplified setups with high computational costs. My testbed addresses this gap by applying A3C with ICM where intrinsic curiosity is essential to optimize performance.

Parallelisation and exploration efficiency have also been central themes. Babaeizadeh et al. [78] introduced a GPU-Asynchronous Advantage Actor–Critic framework to exploit Graphics Processing Unit (GPU) acceleration for faster training, while Labao et al. [79] modified exploration by diversifying worker policies instead of relying on entropy

regularisation. These innovations underscore the computational advantages and adaptability of A3C, but they either lack transparency in implementation or justification of their gradient updates. In contrast, my testbed improves exploration stability through intrinsic curiosity modules, ensuring diverse behaviours without ad-hoc modifications, and applies Gaussian reward shaping to smooth policy updates, avoiding destabilisation during swarm training.

Applications in aerial domains bring A3C closer to my own research. Fan et al. [80] employed it for manoeuvre decision-making in UAVs, and Yang et al. [81] used it to optimise UAVs placement for 5G/6G networks. Both works validate the potential of asynchronous actor–critic methods in high-stakes aerial environments. Still, each carries key limitations: missing hyperparameter details in Fan et al., and an unrealistic dependence on perfect real-time data in Yang et al. Similarly, Navaneethakrishnan et al. [82] applied A3C to robotic locomotion, showing its cross-domain robustness but also noting its high computational intensity. My testbed addresses these shortcomings by providing systematic hyperparameter tuning, deploying curiosity-driven exploration to reduce reliance on dense external signals, and distributing workloads across multiple nodes with Ray, thereby achieving scalable and realistic training of swarm UAVs.

By embedding A3C directly into a distributed learning FlightGear–JSBSim simulator, integrating ICM for sparse and delayed rewards, and applying Gaussian reward shaping for stability, my testbed overcomes these gaps. This makes A3C the ideal choice for controlling tracker UAVs that must adapt quickly and robustly to dynamic target behaviour in scalable swarm environments.

The A3C algorithm was selected for the tracking swarm because of its asynchronous parallel learning structure. Multiple workers explore the environment simultaneously and update a shared global network, leading to faster convergence and broader exploration of the state–action space. This property is critical in swarm UAV coordination, where several agents must learn and adapt in real time across a distributed learning architecture. A3C also reduces gradient variance, helping agents to learn stable policies while operating in parallel. Its scalability and distributed learning nature make it particularly well-suited for multi-agent UAV target tracking, aligning directly with the objectives of this project.

Actor-critic methods have become a widely adopted approach in Machine Learning for UAV control due to their ability to combine policy-based and value-based learning within a unified framework. Current research trends indicate that these methods are particularly effective in continuous control tasks, where UAVs must operate in dynamic and uncertain

environments.

A key advantage observed across the literature is the ability of actor–critic models to provide stable policy updates by leveraging value estimation to guide the learning process. This makes them suitable for UAV applications such as trajectory tracking, navigation, and control, where maintaining stability is critical. In particular, methods such as A2C and A3C have been used to improve learning efficiency by reducing variance and enabling parallel exploration. However, despite these advantages, several limitations are consistently reported. One common issue is the sensitivity of actor–critic methods to hyperparameter selection, which can significantly affect convergence and stability. In addition, many studies rely on single-agent implementations or small-scale environments, limiting their applicability to multi-UAV coordination scenarios.

Another important observation is that, while asynchronous methods such as A3C introduce parallelism through multiple workers, this parallelism is often confined to a single computational system rather than being extended across multiple machines. As a result, the scalability of these approaches in distributed environments remains underexplored.

Furthermore, challenges such as delayed reward and sparse feedback are frequently highlighted in UAV control tasks. Standard actor–critic methods may struggle in such settings, as the learning signal becomes weak or delayed, leading to slower convergence. These limitations indicate that, although actor-critic methods provide a strong foundation for UAV control, there is a need to extend their application towards distributed multi-agent environments and to incorporate mechanisms that improve exploration and learning efficiency. This research builds on these insights by integrating actor–critic models within a distributed testbed and enhancing their performance using curiosity-driven learning.

Table 2.9 provides a comparison of studies that used the A3C algorithm, highlighting its asynchronous parallel learning mechanism. It shows how A3C improves exploration by running multiple workers that update a shared global network. The table also points out limitations such as instability in complex environments, sensitivity to noise, and the risk of policy degeneration. The comparison demonstrates that A3C is highly suitable for multi-agent scenarios, making it appropriate for controlling multiple tracking aircraft. It also highlights the absence of high-fidelity, real-time swarm experiments using A3C in previous work.

Authors	Focus / Methods	Features	Limitations
Tuli et al. [74]	A3C for probabilistic edge–cloud scheduling in machine learning computing	Demonstrates asynchronous coordination across multiple workers to manage uncertainty in dynamic systems	No detailed pseudocode; reproducibility difficult; not validated on physical agents or aerial domains
Chen et al. [75]	A3C-based resource allocation in mobile vehicle service networks	Shows adaptability to rapidly changing conditions in machine learning vehicular systems, demonstrating robustness of asynchronous learning	Limited to ground vehicles; no validation in continuous control or flight dynamics; lacks scalability analysis for swarms
Du et al. [76]	A3C for pricing and resource allocation in blockchain-enabled mobile systems	Handles stochastic and uncertain environments, machine learning decision-making under variable loads	Justification for choosing A3C is weak; simpler algorithms may achieve similar results; not extended to safety-critical aerial tasks
Lee et al. [77]	Hierarchical multi-agent A3C to reduce state-space complexity in 2D grid tasks	Improves training efficiency by decomposing large state spaces into manageable subspaces	Simplified to discrete grid states; lacks realism for continuous UAV control; does not test robustness in noisy, real-world-like settings
Babaeizadeh et al. [78]	GPU-accelerated GA3C (GPU–A3C) for large-scale tasks such as video games	Parallelisation on GPU significantly reduces training time, enabling more workers and faster policy updates	No pseudocode; applied only to Atari-style benchmarks; does not address real-time physical systems like UAVs
Labao et al. [79]	Modified A3C with diversified worker policies to improve exploration	Reduces reliance on entropy regularisation by encouraging diversity among worker agents	Gradient modification strategy lacks rigorous justification; not tested in continuous high-dimensional flight domains
Fan et al. [80]	Deep A3C for manoeuvre decision-making in Unmanned Combat Aircraft Vehicless (UCAVs)	Demonstrates aerial combat manoeuvre applications; constructs manoeuvre libraries and adaptive control policies	Hyperparameter tuning details missing; stability issues when generalising across varied aerial scenarios

Continued on next page

Table 2.9 (continued)

Authors	Focus / Methods	Features	Limitations
Navaneethakrishnan et al. [82]	A3C applied to robotic locomotion for biped design	Shows robustness of asynchronous actor–critic in balancing stability and adaptability for complex locomotion	Computationally very expensive; high resource requirements limit applicability to real-time aerial swarms
Yang et al. [81]	A3C for optimal UAV placement in 5G/6G communication networks	Enables dynamic, real-time adaptation of UAVs to meet user demand and network conditions	Relies on availability of highly accurate real-time data; assumptions limit applicability to uncertain environments

Table 2.9: Summary of research studies using the A3C algorithm: focus, features, and limitations.

The next subsection is based on the literature review of the PPO algorithm.

2.6.4.3 PPO algorithm

The PPO algorithm has emerged as a popular RL method for UAV research because its clipped updates provide a balance between exploration and stability, making it well-suited for complex, high-risk scenarios. Liang et al. [83] demonstrated this by combining PPO with a Convolutional Neural Network (CNN)–LSTM fusion network for collision avoidance in constrained airspace. Their work shows how PPO enables UAVs to respond adaptively to dynamic environments, but the heavy computational demands of deep recurrent networks limit feasibility for real-time swarm deployment. Bai et al. [84] further explored PPO in military combat simulations, emphasising multi-agent cooperation and strategy. Although their results highlight PPO's value for coordination tasks, their approach struggles to scale to larger swarms, which restricts broader application. Both studies reveal the strengths of PPO in adaptive decision-making but underscore the pressing challenge of scalability and computational efficiency, challenges that my distributed learning testbed explicitly addresses through Ray-based parallelisation, curiosity-driven exploration, and Gaussian reward shaping.

Other works extend PPO into mission-specific domains, further validating its potential. Guan et al. [85] applied PPO to disaster recovery by optimising UAV trajectories for communication restoration, and Chen et al. [86] used it in cellular networks to plan energy-efficient paths. Both highlight how PPO adapts to dynamic conditions, but their reliance on accurate real-time data or susceptibility to

computational delays limits applicability in unpredictable environments. Chikhaoui et al. [87] pursued a similar goal in dense urban navigation, applying PPO to collision avoidance in 3D spaces, yet failed to provide reproducible hyperparameter configurations, raising concerns about stability. These approaches illustrate how PPO is consistently applied to trajectory optimisation and safety-critical navigation but remain hindered by oversimplified assumptions and missing methodological details. My testbed advances this line of work by systematically tuning PPO in physics-based FlightGear–JSBSim environments and embedding curiosity modules to reduce dependence on perfect state information, ensuring that learned policies remain robust under uncertainty.

Further innovations in PPO target specialised scenarios. Zhan et al. [88] employed the Ray framework for the training of PPO in combat simulations, demonstrating its viability for heterogeneous UAV teams while also highlighting computational costs. Li et al. [89] proposed a PPO-based approach for UAV spectrum cartography, optimising trajectories for accurate signal mapping while conserving energy. While both highlight PPO's flexibility, Zhan et al. expose computational overhead as a key bottleneck, and Li et al. reveal the pitfalls of overly complex reward functions that reduce transferability. My research integrates PPO into a distributed learning testbed that balances computational loads across multiple nodes and simplifies reward design with Gaussian functions to maintain stability, while intrinsic curiosity ensures continued exploration in sparse-reward conditions.

Taken together, these studies confirm PPO's promise for adaptive UAV navigation, coordination, and trajectory optimisation across a wide range of scenarios. Yet, they also show recurring weaknesses, computational inefficiency, poor scalability, reliance on idealised data, and fragile reward designs that hinder their broader adoption in swarm contexts. By embedding PPO into my distributed learning FlightGear–JSBSim testbed alongside A2C and A3C, and enhancing it with intrinsic curiosity and Gaussian reward shaping, I provide a stable, scalable, and reproducible testbed for real-time swarm UAV tracking. This makes PPO an essential complement to other actor–critic methods in ensuring both adaptability and robustness within multi-agent aerial systems.

The PPO algorithm was introduced alongside A3C to address a key challenge: policy degeneration. In swarm settings, A3C workers sometimes get stuck repeating poor actions that yield low rewards instead of improving towards better strategies. PPO solves this with its clipped surrogate objective, which stabilizes policy updates by restricting drastic

changes during optimization. This ensures steady, incremental learning and reduces the risk of unstable behaviours. PPO also improves sample efficiency by allowing multiple epochs of training on the same data, which is valuable in complex swarm simulations. Its robustness and stability make PPO an effective complement to A3C for reliable swarm tracking in the distributed learning testbed.

Proximal Policy Optimization has emerged as a robust policy gradient method that addresses several limitations of earlier actor-critic approaches. A key trend in the literature is the adoption of PPO for tasks requiring stable and reliable policy updates, particularly in environments where large policy changes can lead to performance degradation.

The main strength of PPO lies in its clipped objective function, which constrains policy updates and prevents abrupt changes during training. This results in improved stability and more consistent convergence compared to traditional policy gradient methods. For UAV control tasks, this stability is particularly important, as abrupt policy updates can lead to unsafe or unrealistic behaviours. Across existing studies, PPO has been successfully applied to UAV navigation, trajectory optimisation, and control tasks. However, similar to actor–critic methods, most implementations are limited to single-agent or small-scale scenarios. The use of PPO in multi-agent UAV coordination, particularly within distributed environments, remains relatively limited.

Another observation is that PPO improves training stability but does not inherently address challenges such as delayed reward or insufficient exploration. As a result, agents may still struggle to learn effectively in complex environments where feedback is sparse or delayed. In addition, while PPO provides improved robustness, it is often used in isolation rather than in combination with other learning strategies. The integration of PPO with complementary methods, such as asynchronous learning or curiosity-driven exploration, is not widely explored in existing UAV testbeds.

These observations highlight a gap in the literature, where stable policy optimisation methods such as PPO are not fully utilised within distributed and multi-agent learning frameworks. This research addresses this gap by incorporating PPO alongside A3C within a distributed testbed, allowing different UAV agents to utilise different learning strategies while maintaining coordinated behaviour. By combining PPO with curiosity-driven mechanisms and distributed execution, this work extends the application of PPO beyond conventional single-agent settings and demonstrates its effectiveness in scalable multi-UAV coordination scenarios.

Table 2.10 compares research studies using the PPO algorithm for UAV control tasks, focusing on model architecture, capabilities, and performance limitations. It shows that PPO is known for stable policy optimisation due to its clipping mechanism and sample-efficient training. However, the table also identifies limitations such as dependence on hyperparameter tuning and possible fluctuations in complex scenarios. By reviewing these past works, the table demonstrates that PPO has strong potential for multi-agent control but remains under-explored in swarm-based contexts. This supports my decision to evaluate PPO trackers and compare them against A3C. The table also highlights that previous studies rarely used PPO with distributed learning systems or high-fidelity simulators.

Author(s)	**Models/Methods**	**Features**	**Limitations**
Liang et al. [83]	PPO with hybrid CNN–LSTM fusion architecture	Achieves adaptive collision avoidance in constrained airspace, handles temporal dependencies via LSTM and spatial features via CNN	Very high computational cost due to deep recurrent–convolutional model; unsuitable for real-time onboard UAV systems
Bai et al. [84]	Enhanced multi-agent PPO for military combat simulations	Enables strategic coordination and cooperation in adversarial multi-UAV combat environments	Scalability issues when increasing swarm size; communication and synchronisation overhead limit applicability to large formations
Guan et al. [85]	PPO integrated with modified K-means clustering for trajectory planning	Applied to disaster recovery scenarios; optimises flight paths for restoring communication networks under uncertainty	Heavy reliance on highly accurate real-time situational data; performance degrades in noisy or chaotic environments
Zhan et al. [88]	PPO using the Ray framework for training heterogeneous swarms	Supports heterogeneous UAVs coordination in large-scale combat simulations, improves training throughput via distributed learning rollouts	Extremely high computational resource demands; difficult to deploy without access to large-scale compute clusters

Continued on next page

Table 2.10 (continued)

Author(s)	Models/Methods	Features	Limitations
Chen et al. [86]	PPO-based Markov Decision Process (MDP) path planning for UAV-assisted cellular networks	Optimises energy consumption and coverage, framing UAV trajectory control as a decision process	Limited real-time adaptability due to computational delays; scalability challenges for fast-changing network conditions
Chikhaoui et al. [87]	PPO framework for UAV navigation in dense 3D urban environments	Focuses on collision avoidance in cluttered urban airspaces with dynamic obstacles	Hyperparameters not fully disclosed, reducing reproducibility; stability concerns remain under highly dynamic conditions
Li et al. [89]	PPO-based spectrum cartography for UAV path optimisation	Balances energy efficiency and signal measurement accuracy in trajectory planning	Complex handcrafted reward function reduces generalisability; difficult to adapt across diverse UAV swarm scenarios

Table 2.10: Comparison of PPO-based UAV research studies: models, features, and limitations.

The next subsection is based on the literature review of ICM

2.6.4.4 Intrinsic Curiosity Module

The ICM is a mechanism in RL that provides agents with intrinsic rewards based on prediction error. It utilizes a forward model to predict the next state from the current state and action, and an inverse model to infer the action required to transition between two states. When the agent encounters novel or unexpected transitions, the prediction error is high, generating intrinsic motivation to explore further. This allows agents to learn effectively even in sparse or delayed reward environments. The concept of intrinsic curiosity has been explored in diverse domains, and many authors have shown how ICM can overcome the challenges of sparse and delayed rewards. Colas et al. [90] pioneered this idea in the CURIOUS algorithm by combining intrinsic motivation with multi-goal RL. Their work demonstrates that novelty-driven exploration signals allow agents to learn across diverse goals. This is significant because it shows how curiosity can guide agents even when extrinsic rewards are absent, a property directly relevant to UAV swarms, where external feedback is often sparse. Building on this foundation, Burda et

al. [91] conducted one of the most comprehensive validations of curiosity-driven methods, applying ICM-inspired exploration across 54 environments. Their results confirmed that intrinsic signals can sometimes substitute for extrinsic rewards entirely, but also revealed limitations in stochastic settings where prediction errors may no longer align with task objectives. This limitation matters in UAV applications because flight environments often contain stochastic disturbances like wind, where careless curiosity signals could destabilise learning.

Zhelo et al. [92] extended these insights into robotics by applying curiosity to mapless navigation, showing that intrinsic rewards help overcome inefficiency in sparse navigation tasks such as dead ends and long corridors. Their approach shares similarities with UAV swarm navigation, but the scale is restricted to single-agent robots. In contrast, Bougie et al. [93] proposed a skill-based curiosity model that shifts the focus from prediction error to uncertainty in mastering specific goals. While their testbed demonstrates generalisation across skills, it also illustrates that prediction-based curiosity is not the only viable pathway. However, other researchers adapted ICM for stability improvements. Li et al. [94] addressed catastrophic forgetting in ICM using fixed random networks to stabilise intrinsic rewards. This work highlights that without careful design, curiosity can collapse into trivial behaviours, where agents exploit spasticity rather than learning.

Thiede et al. [95], meanwhile, transferred curiosity into molecular RL, proving that exploration principles extend far beyond robotics. Although effective, their work demonstrates that applying ICM in complex domains requires careful calibration of novelty signals to avoid wasted exploration. The versatility of ICM has also been highlighted in data science. Moran et al. [96] incorporated ICM into feature selection, showing curiosity can drive the discovery of informative features missed by other ML algorithms. Similarly, Luo et al. [97] applied curiosity to visual paragraph generation, mitigating mode collapse and tackling long-horizon delayed rewards in natural language tasks. While these domains are different from UAVs, they collectively confirm the universal strength of ICM in environments where external signals are insufficient, delayed, or misleading.

Taken together, these works demonstrate that ICM is a proven mechanism to drive exploration, overcome sparse rewards, and stabilise policy learning across robotics, simulation, and even abstract domains such as data and language. Yet, limitations such as a lack of scalability across multi-agent systems remain unresolved. My testbed

incorporates ICM within A3C and PPO frameworks for swarm UAV tracking, where it provides intrinsic signals that complement sparse extrinsic feedback. Combined with Gaussian reward shaping, ICM ensures policies remain stable even under stochastic flight dynamics, while distributed learning execution guarantees scalability across swarms. This synthesis justifies the inclusion of ICM in my work: it directly addresses the delayed reward problem, ensures efficient exploration, and prevents policy destabilisation, enabling reliable distributed learning UAV swarm coordination.

The ICM was integrated to overcome the delayed reward problem inherent in UAV tracking. In such scenarios, an agent may not receive immediate feedback for its actions, making it difficult to connect decisions with outcomes. ICM addresses this by generating an intrinsic reward based on the agent's ability to predict the consequences of its own actions, encouraging exploration even when external rewards are sparse or delayed. By combining curiosity-driven learning with Gaussian reward shaping, the testbed promotes stable training and prevents policy degeneration. This mechanism improves adaptability, accelerates learning in complex environments, and ensures that swarm UAVs maintain effective tracking behaviours even in challenging scenarios.

The couch-potato problem refers to a situation where an agent exploits the intrinsic reward mechanism by generating high prediction error without performing meaningful task-related behaviour. Since ICM provides a reward based on the agent's ability to predict the consequences of its actions, the agent may learn to repeatedly visit states that maximise prediction error rather than states that contribute to the task objective.

In the context of UAV flight, this behaviour can manifest in several undesirable ways. For example, a UAV may perform unnecessary or erratic manoeuvres, such as continuous oscillations, rapid changes in orientation, or deviations from the target trajectory, simply to generate novel or unpredictable state transitions. Instead of improving tracking performance, the agent focuses on maximising curiosity-driven reward.

This can negatively impact task performance by reducing tracking accuracy, increasing instability in flight paths, and potentially leading to inefficient or unsafe behaviour. In multi-UAV scenarios, such behaviour may also disrupt coordination between agents, as individual UAVs prioritise intrinsic reward over cooperative objectives.

To mitigate this issue, the intrinsic reward must be balanced with the extrinsic task-based reward. In this research, the reward design combines curiosity-driven exploration with task-specific rewards such as tracking accuracy and formation

maintenance. This ensures that exploration is encouraged, but remains aligned with the overall objective of stable and coordinated UAV tracking.

Table 2.11 summarises studies that employed the ICM to improve exploration by generating intrinsic rewards for unseen or unpredictable states. It highlights how ICM improved learning in sparse-reward or delayed-reward environments across different tasks. The table also lists limitations such as computational overhead or instability in environments with noisy dynamics.

Author(s)	**Models/Methods**	**Features**	**Limitations**
Colas et al. [90]	CURIOUS algorithm combining Universal Value Function Approximator, hindsight learning, and ICM	Introduces modular multi-goal RL with intrinsic rewards; curriculum learning based on learning progress; supports cross-goal transfer	Evaluated only in robotic domains; lacks demonstration in aerial or swarm UAV systems; scalability for distributed learning untested
Burda et al. [91]	Large-scale curiosity-driven study using ICM with random vs. learned features	Demonstrates that curiosity-only agents succeed in 54 benchmark environments; validates ICM exploration across diverse domains	Struggles in highly stochastic environments where prediction error is noisy; does not address continuous flight dynamics or delayed UAV rewards
Zhelo et al. [92]	ICM-based intrinsic rewards for mapless navigation in Deep Reinforcement Learning (DRL)	Improves policy learning under sparse navigation rewards; enhances generalisation in long-corridor and dead-end tests	Single-agent focus; no distributed learning scalability; not validated in multi-UAV cooperative or swarm environments
Bougie et al. [93]	Goal-based Curiosity (GoCu), a skill-driven extension of ICM	Rewards uncertainty in skill mastery; exploration generalises across tasks in RL; effective in MiniGrid, MuJoCo, Atari	Not evaluated in robotics or UAV domains; lacks distributed learning architecture for swarm coordination

Continued on next page

Table 2.11 (continued)

Author(s)	Models/Methods	Features	Limitations
Li et al. [94]	Random Curiosity Module (RCM) using fixed random target networks	Stabilises intrinsic signals, prevents catastrophic forgetting, avoids "couch-potato" exploitation; state-of-the-art Atari performance	Focused on gaming benchmarks; no validation in real-time, continuous control tasks such as UAV flight dynamics
Thiede et al. [95]	ICM-inspired algorithms for molecular RL exploration	Encourages agents to explore large chemical search spaces; improves novel molecule discovery and design efficiency	Domain-specific; no link to robotics or UAVs; cannot handle time-critical distributed learning coordination
Moran et al. [96]	Deep Curious Feature Selection (DCFS) applying ICM-style novelty to features	Curiosity-driven feature subset evaluation; improves predictive performance by selecting informative and diverse features	Data science application; unrelated to robotics or UAV control; lacks real-time distributed learning deployment
Luo et al. [97]	Curiosity-driven RL for visual paragraph generation using ICM	Tackles mode collapse and delayed feedback in language models; increases output diversity by 38.4%	Limited to NLP tasks; not transferable to UAV swarm coordination or physical environments

Table 2.11: Comparison of ICM-related research studies: models, features, and limitations.

2.7 Research Questions Revisited

RQ1: What are the limitations in the current state of the art in testbeds for self-coordinating and cooperative swarms of UAVs?

Existing UAV testbeds often suffer from critical shortcomings, including the inability to handle the delayed reward problem, which hinders agent learning in sparse-reward or dynamic environments. They also lack scalability, except for the GaTAC testbed, and do not feature a distributed learning architecture, limiting applicability to multi-agent real-time systems [6]. In addition, they fail to implement mechanisms to prevent policy destabilisation during training, leading to inconsistent performance, and do not incorporate advanced reward shaping techniques such as Gaussian or ellipsoidal reward

functions, which are essential for precise target tracking and stability. The testbed addresses these gaps by integrating A2C and A3C algorithms with ICM-based intrinsic motivation, Gaussian reward functions, and distributed learning execution across multiple systems via FlightGear's multiplayer architecture [34].

RQ2: Which flight simulators have been used in previous testbeds for self-coordinating and cooperative swarms of UAVs, and which is the most suitable for this research?

Prior work has explored simulators such as Gazebo, Microsoft Flight Simulator, X-Plane, and SimplePlanes. While each has unique advantages, most are closed-source, subscription-based, or lack flexibility for customised RL integration [48]. This research selects FlightGear due to its open-source nature, high-fidelity physics via JSBSim, compatibility with multiple operating systems, and demonstrated success in prior scalable testbeds such as NUAV and GaTAC. Its multiplayer networking enables both local and distributed learning UAV coordination, making it highly suitable for the system.

RQ3: What are the requirements for a testbed for self-coordinating and cooperative swarms of UAVs?

The testbed is built on an architecture that integrates ML models with a flight simulator to support real-time control and decision making. It is designed to be scalable, so multiple autonomous aerial agents can operate together within the same simulated environment. The system also follows a distributed approach, allowing different ML agents to run on separate networked machines while sharing information and coordinating their actions. Learning is based on actor–critic style methods, where agents learn in parallel to explore efficiently. Carefully designed reward shaping is applied to improve learning stability, reduce oscillations, and ensure consistent performance during training.

RQ4: Which ML algorithms are most suitable for enhancing the performance and robustness of UAV swarm testbeds?

Enhancing the performance and robustness of UAV swarm testbeds requires learning methods that can handle continuous control, dynamic interactions, and non-stationary multi-agent environments. Value-based approaches are generally unsuitable for such settings due to instability in continuous and multi-agent scenarios, which motivates the use of policy-gradient and actor–critic methods. Actor–critic algorithms such as A2C and A3C, together with the policy-gradient optimisation method PPO, provide stable learning, support continuous action spaces, and scale effectively in distributed settings. Their suitability is further strengthened by integrating the ICM, which addresses

delayed and sparse rewards by generating intrinsic motivation that encourages sustained exploration and prevents policy stagnation. In addition, Gaussian-based reward shaping provides smooth feedback that improves tracking accuracy and flight stability, while the distributed learning architecture, particularly with A3C, reduces gradient variance and prevents policy destabilisation. Together, these characteristics make A2C, A3C, and PPO well-suited for enhancing the performance and robustness of scalable UAV swarm testbeds [87].

2.8 Summary

This chapter critically analyses existing research on UAV testbeds, swarm coordination, flight simulators, and machine learning approaches, revealing that current solutions address these aspects in isolation rather than as an integrated system. The review shows that many testbeds lack scalability and distributed learning capability, while others fail to effectively handle delayed reward challenges in learning-based control. A key insight from the literature is that although FlightGear and JSBSim are widely used for realistic simulation, and algorithms such as A2C, A3C, and PPO have shown strong performance individually, their combined use within a distributed and scalable swarm framework remains limited. Furthermore, curiosity-driven approaches such as ICM are not commonly integrated into UAV testbeds despite their potential to improve exploration and learning stability. The chapter, therefore, identifies a clear research gap: the absence of a unified testbed that combines realistic simulation, distributed learning, and curiosity-driven reinforcement learning for cooperative swarm UAV coordination. This analysis directly informs the design requirements and methodological choices of the research, leading to Chapter 3, where the research methodology is defined to systematically develop and evaluate the proposed testbed.

Chapter 3

Research Setting

A research methodology provides a structured framework that guides how a research study is designed, conducted, and evaluated. It defines the philosophical stance, logical approach, and practical strategies that shape the process of gathering, analysing, and interpreting data. In this chapter, the methodological choices are justified in relation to the specific aims of this project: the design and validation of a distributed learning, scalable swarm UAV testbed supported by RL algorithms. By outlining the underlying philosophies, approaches, strategies, choices, and time horizon, this chapter demonstrates how the methodology ensured both scientific rigour and practical relevance in addressing the challenges of scalability, delayed reward, and swarm coordination.

3.1 Research Methodology

Research methodologies encompass the various tactics, processes, or techniques employed in conducting experiments and data analysis to extend the boundaries of knowledge or enhance understanding of a topic. Saunders et al. [2] have demonstrated the research onion illustrated in Figure 3.1. The research onion is designed for management sciences. However, recently, it has also been used in various engineering and scientific research endeavours. The research onion is composed of five layers. Each layer corresponds to a theme that has to be adopted for the successful completion of a research project.

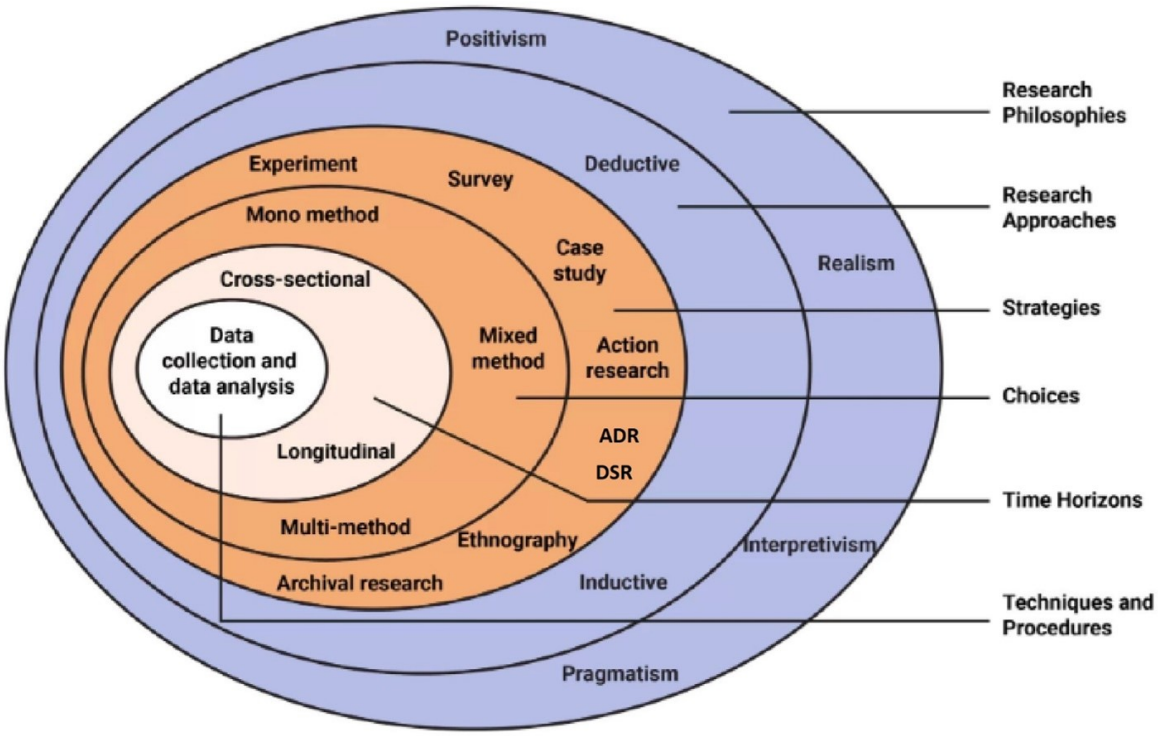


Figure 3.1: Research onion [2].

3.1.1 Research Philosophies

Research philosophies are the outermost layer of the onion. A research philosophy is an idea of how information about phenomena should be obtained, examined, and used. There are three main research philosophies, which are given below.

Positivism: It doesn't involve personal thoughts or viewpoints. It holds that knowledge can only be obtained through empirical research based on observation and measurement.

Interpretivism: Interpretivism strongly focuses on the impact that social and cultural elements may have on a person. This viewpoint is centered on people's ideas and opinions.

Pragmatism: Pragmatism emphasises the significance of employing the best methods to examine phenomena. The primary goal of a pragmatic research philosophy is to start with dynamic information subjected to ongoing interpretation and questioning.

I have adopted positivism for my project because knowledge is obtained through empirical research, observation, and measurement. Additionally, pragmatism also aligns well with my research project. This research adopts a combination of positivism and pragmatism. The choice is rooted in the work undertaken, which involves developing

and experimentally validating a scalable and distributed learning testbed for swarm UAVs controlled by RL algorithms.

Positivism was essential because much of the study required the generation of objective, measurable results through controlled experimentation. The behaviour of UAVs under different RL models (A2C, A3C, PPO, and ICM) was captured using values such as reward functions, training loss, actor-critic losses, and episode lengths. These values were not based on subjective interpretation but on systematic observation, empirical data collection, and repeatable validation. Adopting a positivist stance ensured that the conclusions drawn from the experiments were reliable, generalisable, and grounded in evidence rather than speculation.

Pragmatism was equally important because the research was not purely theoretical, but rather oriented toward solving a real-world problem: enabling the autonomous, scalable, and distributed learning coordination of UAVs. In this project, pragmatism guided the methodological flexibility needed to integrate diverse tools, FlightGear as a simulator, JSBSim as the FDM, Python-coded RL algorithms, and distributed learning computing frameworks such as Ray. Each tool was chosen not for philosophical alignment but because it was the most effective for achieving the objectives of testbed development.

3.1.2 Research Approaches

In the next layer, we can examine the two research approaches: deductive and inductive. It's critical to specify one's research method precisely since it will guide the choices one makes about gathering and interpreting data.

Inductive Approach: The inductive approach includes developing theories via research rather than beginning a project with a theory as a foundation.

Deductive Approach: The deductive approach starts with an idea and works toward developing a research project.

I employed both deductive and inductive approaches because I had a novel research idea, which I developed into a research project. I also gained some new ideas for implementing suitable ML algorithms during the development of the research project. This research employs both deductive and inductive approaches, since the nature of the work combines theory-driven experimentation with data-driven insight.

In this research, the inductive element does not refer to the development of a new

formal theory. Instead, it relates to the generation of empirical insights based on the observed behaviour of Machine Learning agents during experimentation. These observations include patterns in convergence behaviour, the impact of curiosity-driven exploration on learning efficiency, and the effect of reward design on multi-UAV coordination. The insights gained from these observations informed design decisions within the testbed, such as adjustments to reward structures and tuning of learning parameters. However, these outcomes do not constitute theory-building in the traditional sense, but rather empirical findings that contribute to understanding how different learning mechanisms influence UAV coordination. Therefore, the research is primarily deductive and experimental in nature, guided by existing Machine Learning frameworks and evaluated through quantitative analysis. The inductive component is limited to interpreting experimental results and refining the implementation based on observed behaviour. This clarification ensures that the use of the term inductive is consistent with its role in the research and avoids overstating the theoretical contribution.

The deductive element lies in the fact that the project started with a clear theoretical foundation: the development of a distributed learning and scalable testbed for UAVs using RL. The testbed design was guided by existing knowledge in the literature, including RL principles, the actor–critic family of algorithms (A2C, A3C, PPO), and the recognised limitations of previous testbeds, such as lack of scalability, distributed learning architectures, and the delayed reward problem. From these theoretical premises, hypotheses were formed, for example, that introducing the ICM into the testbed would alleviate the delayed reward problem and that distributed learning architectures would improve scalability. The deductive process involved testing these assumptions through carefully structured simulation experiments, verifying whether the expected behaviours emerged when applying the designed framework.

The inductive element emerged during the implementation and experimentation phases. While executing the algorithms in the distributed learning testbed, new observations were made that were not fully predicted by theory. For instance, the analysis of training dynamics results (for example, convergence rates, cumulative reward values, and the behavior of trained versus untrained agents) contributed to understanding how RL models behave in swarm coordination.

3.1.3 Research Strategies

The third layer of the onion reveals a research strategy. This layer describes how research can be done depending on the study's objectives. Examples of research strategies are given below.

Experimental Research: Validating, rejecting, or supporting a research idea is the aim of experimental research. This research approach adheres to the principles of the scientific method. Experimental research is deductive in nature since its goal is to test established research ideas.

Case study research: A case study is a comprehensive and in-depth investigation of a particular topic. For instance, an individual, an organization, an event, a phenomenon, or a problem.

Ethnography: It is the study of individuals in their natural environments to interpret cultural relations. Using ethnography, the goal is to record the participants' subjective experiences.

Archival Research: This research strategy uses pre-existing resources, and then the meaning is determined by analyzing the data.

Action Research: The four unique stages of an Action Research (AR) cycle are plan, act, observe, and reflect. The results of one stage serve as the foundation for the subsequent cycle's stage of ongoing investigation, as demonstrated in Figure 3.2. As a result, every cycle has the potential to produce fresh perspectives [98].

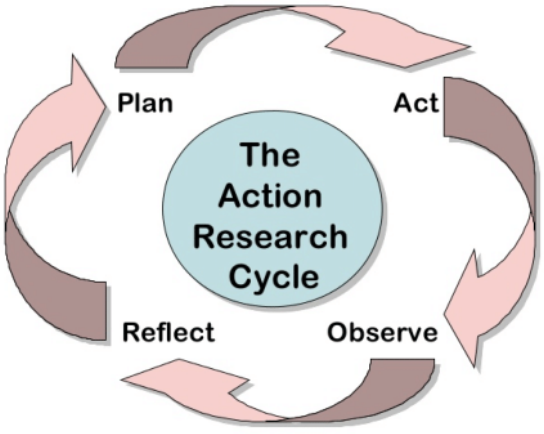


Figure 3.2: Action research [3].

Action Design Research (ADR) means researchers work together with a company or practitioners to build and test a new system. The artefact is developed and improved with the help of real users in a real organisation.

DSR means the researcher creates and tests a new artefact independently, without needing a company or practitioners. The focus is on designing a solution and proving that it works through experiments. The main difference between AR, ADR, and DSR lies in the roles of the researcher and practitioners. In AR, the researcher works as a facilitator alongside practitioners, with success measured by organisational impact. ADR combines this collaboration with the design of artefacts, requiring practitioners to actively shape and evaluate the solution. By contrast, DSR places the researcher as the main driver, with practitioner involvement optional and success measured through rigorous evaluation of the artefact.

AR involves close collaboration between researchers and practitioners to solve immediate organisational problems while simultaneously generating knowledge. In the context of this work, it would require ongoing engagement with industry stakeholders or operational UAV teams to co-develop and refine the testbed. This project does not adopt AR because the primary goal is to design and validate a novel, distributed learning testbed within a controlled research environment rather than to iteratively solve problems in partnership with practitioners. The absence of continuous collaboration with external organisations makes AR unsuitable for this research study.

Action Design Research: ADR is recognized as an important variation of the DSR methodology. The ADR methodology comprises four important stages: problem formulation, intervention and evaluation, reflection and learning, and formulation of learning, as shown in Figure 3.3 [99]. ADR seeks to create and evaluate artefacts in close collaboration with practitioners, addressing real organisational problems while producing research contributions. In the context of this thesis, applying ADR would require developing the swarm UAV testbed iteratively with industry partners, refining it directly according to their practical needs. This approach was not adopted in this project because the thesis focuses on designing and validating the testbed within a controlled academic environment. The goal was to establish a scalable and distributed learning testbed for RL-based swarm coordination rather than to co-develop solutions in partnership with external stakeholders.

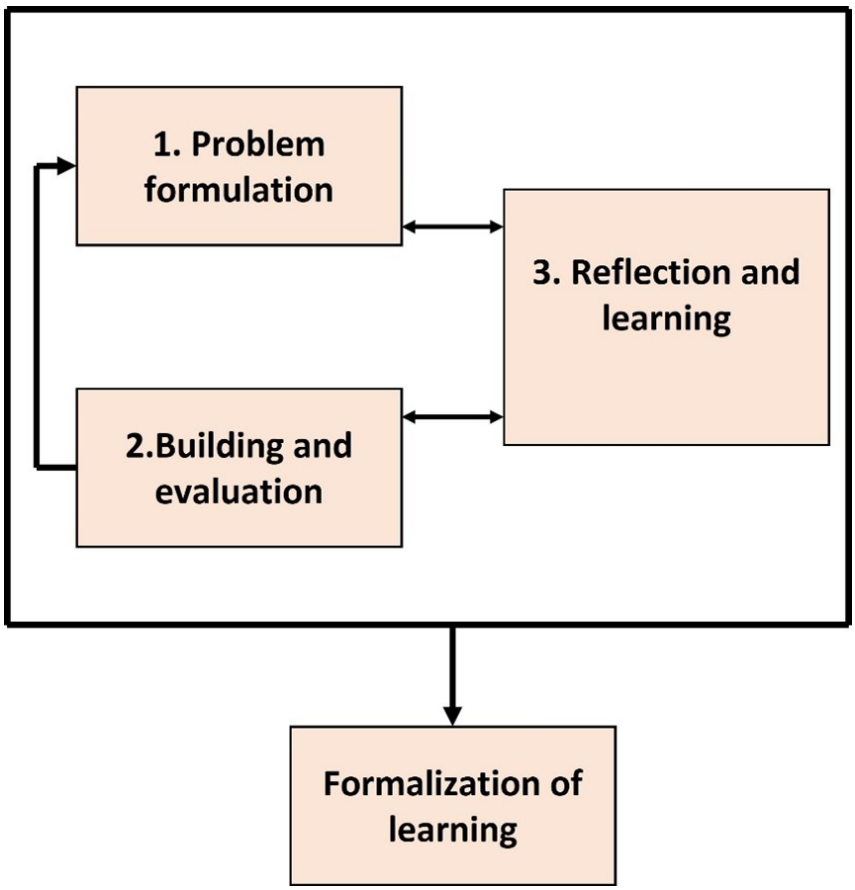


Figure 3.3: Action design research [4]

The main difference between AR and ADR lies in their focus and process. AR focuses on solving real organisational problems in collaboration with practitioners, with the emphasis on reflection and learning during the process of change. The artefact or system itself is not the central outcome; instead, the knowledge gained from the intervention is the key contribution. ADR, by contrast, combines AR with design science. Its aim is not only to address practical problems with practitioners but also to create and evaluate an artefact during the research process.

3.1.3.1 Comparison between AR, ADR and DSR

Iivari and Venable [100] defined DSR as a "Research activity that invents or builds new, innovative artefacts for solving problems or achieving improvements, i.e., DSR creates new means for achieving general goals, as its major research contributions. Such new and innovative artefacts create a new reality, rather than explaining the existing reality or helping to make sense of it."

AR is defined as "An iterative process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action

intervention, and reflective learning" [98]. AR is an approach for investigation that includes identifying an issue in a real-world setting, creating solutions, gathering and evaluating data, and applying the results to guide decisions and promote constructive change. AR is best suited for solving specific existing problems inside an organization [101], but it is not the most appropriate for establishing a testbed for my project. The purpose of this project was to develop an innovative testbed that is both scalable and supports distributed learning, enabling multiple UAVs to be coordinated in real time. The testbed was designed to integrate state-of-the-art RL algorithms, allowing experimentation with single agents as well as swarms. A key aim was to address limitations of existing testbeds by supporting large-scale coordination, reducing reliance on external rewards through ICM, and providing a flexible platform for future research in autonomous swarm behaviours.

There has been no collaboration between researchers and clients from the initial stages of our project, which is a requirement of AR [98]. Thus, DSR is more suitable for our project than AR because the focus was on designing and validating a novel testbed rather than solving an immediate organisational problem. Unlike AR, our work did not involve continuous collaboration with practitioners or clients. Instead, it required a structured methodology to create and evaluate an artefact that advances knowledge in distributed learning swarm UAV coordination.

ADR aims to help research teams develop IT artefacts by using certain conditions and principles, and evaluating them at the end by the company after collaborating with the practitioners during the project [100]. The ADR approach regards organizational intervention as secondary and helps support abstractions and invention. In the initial stage of the project, the researchers are dominant. However, during the development of the IT artefact in the middle stage of the project, the practitioners make a considerable contribution and evaluate it at the end of the project [100].

The differences between the three research approaches are given in Table 3.1. In summary, AR and ADR rely on mandatory practitioner collaboration and measure success through organisational impact, whereas DSR focuses on building and evaluating artefacts. Transferability mentioned in Table 3.1 in this context refers to how the knowledge and testbed design can be applied beyond the original project. For this work, success is measured by convergence of RL models, stability, scalability, and the ability to mitigate delayed rewards, making DSR the most suitable approach.

Property	DSR	AR	ADR
Subject participation in research design	Possible	Mandatory	Mandatory
Organizational impact	Peripheral	Central	Central
Subject feedback	Discrete	Continuous	Continuous
Transferability	Explicit	Implicit	Explicit
Success measure	Quantifiable measures of artifact behaviour	Organizational impact	Organizational learning and artifact generalizability

Table 3.1: Comparison of research approaches [7].

3.1.3.2 Design Science Research

The DSR methodology is an organized approach for investigating and designing an artefact, such as an algorithm that aims to address a technical problem. A methodological paradigm known as DSR recognizes design as a crucial element of scientific investigation. It considers design as a cognitive process with three primary components: defining the problem, conceptualizing the artefact, and evaluating it. To effectively design and implement innovative solutions to pertinent problems, DSR aims to generate knowledge in this area. The knowledge produced must include details about the solution and proof of how effectively the approach solves the issue. Therefore, it should be demonstrated how the solution may be used successfully in the real world to meet the needs of the parties involved in the problem [102]. The DSR methodology is based on six important stages, as shown in Figure 3.4. The first stage is to identify the problem, followed by defining the objectives of the solution. The third and fourth stages are development and demonstration, respectively. The final stage is communication.

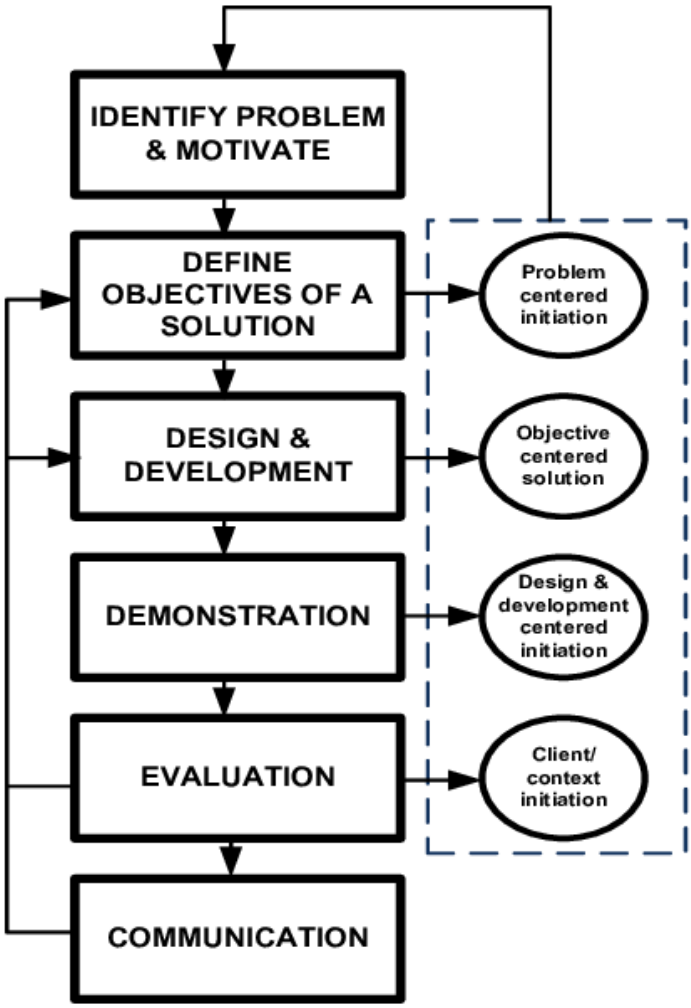


Figure 3.4: Design science research methodology [5].

Identify research problem: DSR is a qualitative research strategy in which the study focuses on the design process, generating information about the method used to develop an artefact, as demonstrated in Figure 3.4.

The starting point for this research was the original project specification, which secured funding to build on the existing NUAV testbed. The specification outlined the need to extend that work by implementing additional RL algorithms, enabling swarms of UAVs, and ensuring that the testbed could scale in a distributed learning manner across multiple machines. To refine this initial scope into a well-defined research problem, a literature review was conducted. This review highlighted several key limitations in the state of the art: most existing testbeds were single-agent, lacked distributed learning architectures, and were unable to support real-time swarm coordination. Furthermore, many studies suffered from the delayed reward problem, where RL agents struggled to learn effectively when feedback was sparse or delayed. By combining the requirements from the original project specification with the insights gained from the literature review, the research

problem was defined as the need to design and validate a novel testbed that supports distributed learning, is scalable, and capable of supporting swarm coordination. The testbed also needed to address fundamental RL challenges, particularly delayed reward, through mechanisms such as the ICM. This structured process of moving from project specification to literature-research gap analysis provided the foundation for the subsequent design and development of the artefact.

Define Objectives of Solution: Following the identification of the research problem, the objectives of this work were shaped both by the original project specification and by insights from the literature review. The funding proposal outlined the need to extend the NUAV testbed by making it scalable and supporting distributed learning, as well as by integrating additional RL algorithms for multi-UAV coordination.

The literature review provided a broader perspective on what an effective testbed should encompass. Previous studies highlighted the need for attributes such as flexibility to integrate multiple algorithms, realism through accurate FDMs, and scalability to support swarms rather than individual UAVs. At the same time, limitations were identified: most existing testbeds were confined to single-agent scenarios, lacked distributed learning architectures, or required expensive hardware-in-the-loop setups that reduced accessibility. A distributed learning testbed means the distribution of ML models over several computers.

In addition, the review of RL research revealed specific challenges. Although algorithms such as A2C, A3C, and PPO had been applied in UAV domains, prior work often overlooked practical issues such as policy degeneration, sparse or delayed rewards, and stability in multi-agent coordination. Mechanisms like the ICM were emerging as promising solutions to these challenges but had not yet been systematically applied in a distributed learning UAV swarm context.

Bringing these elements together, the objectives of the project were defined as follows: to design and implement a distributed learning and scalable testbed capable of supporting swarms of UAVs; to integrate state-of-the-art RL algorithms such as A2C, A3C, and PPO; and to address fundamental learning problems such as delayed reward through the use of curiosity-driven exploration. These objectives not only aligned with the original funding specification but also directly addressed the research gaps identified in the literature.

Design and Development: The development of the testbed followed an iterative process in which functionality was expanded step by step. The initial objective was to establish a working connection between the simulation environment and RL algorithms, and this

was progressively extended to support multiple algorithms and larger swarms of UAVs.

The first stage of development focused on connecting all components to allow a single UAV to be controlled by the A2C algorithm. This required establishing real-time communication between the Python-based RL implementation and FlightGear via the FDM, ensuring that data could be read from the simulator and that control actions could be applied reliably. Once stable control was achieved, the algorithm was tested to verify that training convergence and basic flight behaviours of trained agents were stable.

In the second stage, the testbed was expanded to support the A3C algorithm. This extension introduced multiple asynchronous workers, each maintaining itself in its own copy of the environment and sharing updates with a global network. The asynchronous structure allowed faster and more robust training, and the testbed was updated to manage communication between workers and the simulator while maintaining stability in real-time operations. The flowchart in Figure 3.5 shows the design stage and evolution stage.

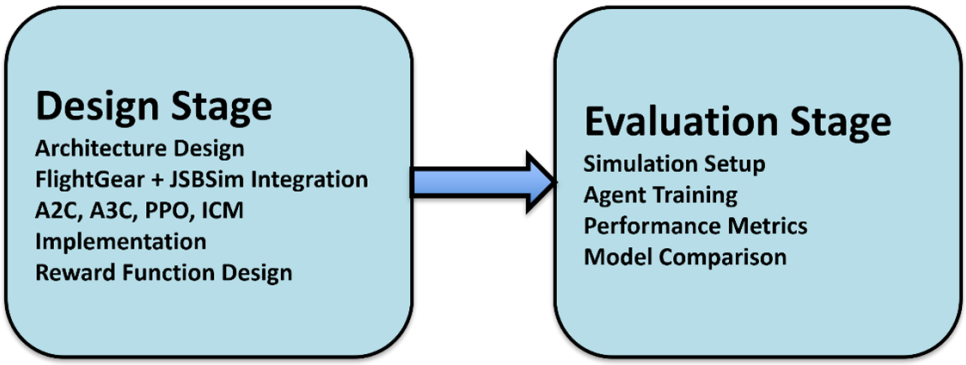


Figure 3.5: Flowchart showing design stage and evolution stage

The third stage extended A3C to control more than one UAV simultaneously. This required enabling multiplayer functionality within FlightGear, managing inter-agent communication, and ensuring that distributed learning processes remained synchronised. At this point, the testbed began to demonstrate swarm behaviours, such as cooperative pursuit of a target, which aligned with the project's objectives of scalability and distributed learning coordination.

The mechanism of distributed learning has been clarified to explain how Machine Learning models are executed across multiple computers. In this research, distribution is achieved through a multi-node architecture where each UAV agent is assigned to an independent worker process. These workers can run on separate machines connected through a local area network. Each worker interacts with its own instance of the

simulation environment and performs learning updates based on its local experience. The communication between workers and the overall learning framework is handled using a distributed execution framework, where model parameters and state information are exchanged across nodes. In the case of A3C, workers update a shared global model asynchronously by sending gradient updates, while in PPO-based setups, agents maintain their own policy updates based on collected trajectories. Data exchange between the simulation environment and the learning agents is implemented using a network-based communication protocol, such as UDP, which allows real-time transmission of state and action information between FlightGear, JSBSim, and the Machine Learning models. This architecture enables the distribution of learning across multiple machines, allowing the testbed to scale beyond the limitations of a single system. The mechanism combines hardware distribution, asynchronous updates, and network-based communication to support real-time multi-agent learning.

In the fourth stage, the testbed was further extended to incorporate the PPO algorithm. This algorithm was selected for its stability during policy updates, which is particularly valuable in high-risk multi-agent environments such as collision avoidance and target tracking. The testbed was modified to allow for clipped policy updates and to evaluate PPO in both single-agent and multi-agent configurations for swarm UAVs.

Across all stages, the development process was iterative: new components were integrated, tested in isolation, and then evaluated in combination with the existing framework. This incremental approach ensured that the system remained stable while growing in complexity. By the end of the development process, the testbed supported multiple algorithms (A2C, A3C, PPO), distributed learning execution across multiple machines, and real-time swarm coordination in FlightGear.

To improve clarity, the Design Science Research (DSR) stages have been explicitly mapped to the corresponding thesis chapters. The problem identification and motivation stages of DSR are addressed in Chapter 1, where the research problem, objectives, and research questions are defined based on gaps identified in existing UAV testbeds. The literature review in Chapter 2 further supports these stages by analysing existing work and identifying limitations related to scalability, distributed learning, and multi-agent coordination. The design and development stage of DSR is presented in Chapter 4, where the architecture of the testbed is described, including the integration of FlightGear, JSBSim, and Machine Learning algorithms such as A2C, A3C, and PPO. This stage focuses on the implementation of the testbed and the development of reward functions and

learning mechanisms. The demonstration and evaluation stages are covered in Chapter 5, where experimental results are presented to validate the performance of the testbed. This includes analysis of learning behaviour, convergence, and multi-UAV coordination under different scenarios. Finally, the communication and reflection stages are addressed in Chapter 6, where the research contributions are discussed, limitations are identified, and conclusions and future research directions are also included. Table 3.2 shows the mapping of Design Science Research (DSR) Stages to Thesis Activities.

DSR Stage	**Research Activities**	**Thesis Chapter**
Problem Identification	Identification of limitations in existing UAV testbeds, definition of research gap	Chapter 1, 2
Objectives Definition	Formulation of research questions and objectives	Chapter 1
Design and Development	Design of distributed testbed architecture, integration of FlightGear, JSBSim, and Machine Learning models	Chapter 4
Demonstration	Implementation of A2C, A3C, and PPO for UAV coordination tasks	Chapter 4, 5
Evaluation	Experimental validation using reward, convergence, and tracking performance metrics	Chapter 5
Communication and Reflection	Discussion of results, contributions, limitations, future work, and conclusion	Chapter 6

Table 3.2: Mapping of Design Science Research (DSR) Stages to Thesis Activities

Demonstration and Evaluation: The testbed was demonstrated and evaluated through a structured process that combined both empirical validation and verification. Demonstration began with controlled experiments to show that the artefact functioned as intended: a single untrained agent was first deployed in FlightGear to illustrate baseline behaviour, followed by trained agents that displayed progressively more stable flight control. As the testbed expanded, demonstrations included multiple tracking UAVs pursuing a target.

Beyond demonstration, rigorous evaluation and validation were carried out to ensure the reliability and generalisability of the testbed. Evaluation and validation steps included the analysis of the experimental results. Validation focused on whether the

testbed met its design objectives: supporting distributed learning, scalability, enabling swarm coordination, and mitigating the delayed reward problem through ICM.

Quantitative evaluation included convergence analysis of cumulative rewards, episode lengths, and training losses and performance comparisons across different algorithms (A2C, A3C, PPO).

Communication: The final stage of the DSR process involves communicating the artefact and the knowledge generated to the wider community. For this project, communication was planned and undertaken at several levels.

At the academic level, research papers were published in peer-reviewed journals and international conferences. These publications focus on different aspects of the work: the design and implementation of the distributed learning and scalable testbed, the integration of A2C, A3C, and PPO algorithms, the solution to the delayed reward problem through the use of ICM, and the introduction of adaptive tuning methods such as the SRCW technique.

At the community level, results are presented at conferences, enabling dialogue with academic peers interested in applying distributed learning swarm UAV systems.

This project was conducted entirely within an academic setting, with funding from the university. From the initial stage through to completion, the development of the testbed has been solely the responsibility of the researcher, rather than a joint effort with a company or client. While evaluation of the testbed has taken place at the end of the project, in line with the requirements of DSR, there has been no practitioner involvement at any stage. For this reason, DSR is more suitable for this research than AR or ADR, both of which require continuous engagement with practitioners.

Evaluation of the testbed was planned through a verification and validation process. Verification involved checking that each component of the testbed functioned correctly. Validation focused on whether the artefact met its research objectives: scalability, distributed learning support, support for multi-agent coordination, and mitigation of the delayed reward problem through ICM. In addition to these technical evaluations, the project has undergone an academic evaluation at the final stage. This includes assessment by internal and external examiners as part of the thesis examination process, as well as peer review through publication in journals and conferences. Both forms of academic scrutiny provide an independent review of the artefact and its contributions, complementing the systematic verification and validation carried out during the development process.

3.1.4 Choices

"Choices" is the next layer of the research onion. There are three different options.

Mono-method refers to the use of only one type of data, either quantitative or qualitative. Multi-method allows the use of more than one methodology. However, these may still focus on a single type of data (for example, using several quantitative techniques without including qualitative data). Mixed-methods explicitly combine both quantitative and qualitative data, ensuring that results are evaluated from multiple perspectives. In this research, the evaluation of the testbed is primarily based on quantitative methods. Performance is assessed using measurable metrics such as cumulative reward, convergence behaviour, episode length, tracking accuracy, and inter-UAV distance. These metrics provide objective evidence of learning performance and system behaviour. In addition to numerical metrics, visual analysis of UAV trajectories and behaviours is used to support the interpretation of the results. However, this form of observation is not treated as a formal qualitative method, but rather as an extension of quantitative evaluation, where graphical outputs and behavioural patterns are analysed to validate learning outcomes. In this project, a mixed-methods approach was adopted. Quantitative data were collected through FlightGear. At the same time, qualitative insights were obtained by observing agent behaviour in simulations.

3.1.5 Time Horizon

The next layer is the time horizon. The time horizon describes the number of points in time at which you intend to collect data. A longitudinal time horizon involves collecting data over a fixed and continuous period, often to study changes or trends over time. A cross-sectional time horizon, by contrast, involves collecting data at different points in time without being tied to a fixed schedule.

In this work, the learning process is inherently longitudinal, as the Machine Learning models are trained over multiple episodes and time steps. During training, agents continuously interact with the environment, update their policies, and improve performance based on accumulated experience. The previous description of the research as cross-sectional was based on the evaluation of results at specific stages of training. However, this does not accurately represent the overall methodology, as the learning process itself evolves over time. The model performance is observed and analysed across multiple training episodes and iterations. Metrics such as cumulative reward,

convergence behaviour, and tracking performance are evaluated over time to assess learning progress. This ensures that the terminology used in the thesis is consistent with the nature of reinforcement learning and aligns with established research methodology practices.

We have now arrived at the centre of the onion, where I need to use research to make decisions on specific methods and approaches. At this stage, I must collect and analyze my project's data. Table 3.3 shows the selected elements of the research methodology of this project.

Table 3.3: The selected elements of research methodology of the project.

Research philosophy	**Research approach**	**Research choice**	**Research strategy**	**Techniques and procedures**
Positivism and pragmatism	Deductive+Inductive	Mixed method	Design science research	Data collection and analysis

3.2 Summary

This chapter defines the methodological foundation of the research by adopting Design Science Research (DSR) to systematically develop and evaluate the UAV testbed. Rather than presenting the methodology as a formality, the chapter justifies DSR as appropriate because the research focuses on creating and validating a practical artefact that addresses identified limitations in existing testbeds. The methodology is tightly aligned with the research problem. The stages of problem identification, objective definition, design, implementation, and evaluation are directly mapped to the development of a scalable and distributed learning testbed. The chapter also clarifies that the evaluation strategy is based on experimental validation using real-time simulation, allowing the performance of A2C, A3C, PPO, and ICM-based models to be assessed in controlled yet realistic conditions. Overall, the chapter concludes that a structured and artefact-driven methodology is essential to bridge the gap between theoretical models and practical swarm UAV coordination. This methodological framework leads directly to Chapter 4, where the testbed is designed and implemented based on these principles.

Chapter 4

Design and Implementation of the Testbed

This chapter presents the design and implementation of the testbed, demonstrating its integration with RL algorithms for real-time UAV coordination. It describes the implementation of A2C, A3C, and PPO models, along with hyperparameter tuning, and their interfacing with the FDM. A Gaussian ellipsoidal reward function and curiosity-driven mechanisms are employed to address the delayed reward problem. The testbed is demonstrated with scenarios ranging from single-agent tracking to multi-agent swarm coordination. Finally, the implementation of the distributed learning architecture of the testbed using the Ray framework for scalability and the real-time performance of the system are presented.

4.1 Designing the Architecture of the Testbed

To design and implement the testbed and enable real-time flight simulations, the RL algorithms were integrated with a FDM and a flight simulator. A major challenge during the development of the testbed was ensuring smooth and reliable communication between FlightGear, the JSBSim FDM, and the RL algorithms. Achieving this integration was critical to maintain synchronized control loops and realistic flight behaviour throughout the simulation process. FlightGear served as the primary simulation platform, offering real-time graphical rendering, while JSBSim functioned as the FDM, responsible for simulating realistic flight dynamics.

The architecture of the testbed was influenced by design principles from the NUAV

testbed. The testbed transmits control commands such as elevator, rudder, and aileron adjustments to FlightGear. Concurrently, FlightGear relays updated flight information, including altitude, latitude, and longitude, back to the system through the same testbed. To ensure efficient and low-latency data transmission, the system utilized the User Datagram Protocol (UDP), facilitating bidirectional communication in real-time. This continuous data exchange enables the UAV to dynamically update its control strategies based on real-time environmental feedback, thereby ensuring accurate and adaptive simulation behaviour. Figure 4.1 shows the overall architecture of the testbed, where the RL algorithms, that are A2C, A3C, and PPO, are integrated with FlightGear and the JSBSim FDM.

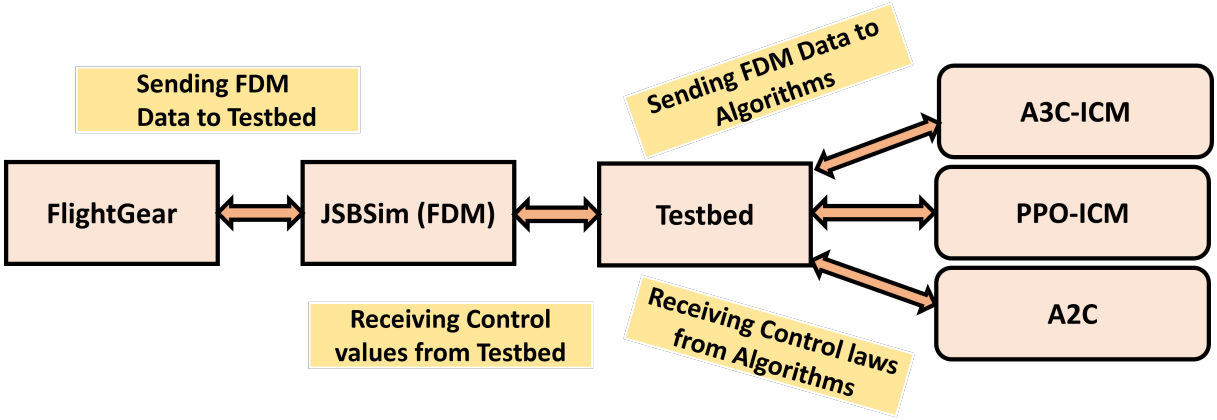


Figure 4.1: Architecture of the Testbed [6]

4.1.1 Interfacing FDM with RL Algorithms

FlightGear was used for the simulation environment and utilized a FDM to replicate the flight behaviour of the aircraft. Specifically, JSBSim delivered a highly detailed physics-based modelling approach capable of accurately reproducing various aspects of flight, including aerodynamics.

Dispatching control instructions: The RL model generated control outputs, which are throttle, rudder, elevator, and aileron commands based on the agent's learned policy and forwarded them to FlightGear through JSBSim.

Receiving real-time flight data: In response, FlightGear returned the current state data of the aircraft, which are its 3D coordinates in x,y, and z-axis, velocity, and altitude, to the testbed.

As illustrated in Figure 4.2, the control commands are relayed from FlightGear to

JSBSim, which then processes and simulates the behaviour of the aircraft. In return, JSBSim provides real-time updates that include spatial location (latitude, longitude, altitude), flight speed and direction, orientation angles (roll, pitch, yaw), and control surface deflections.

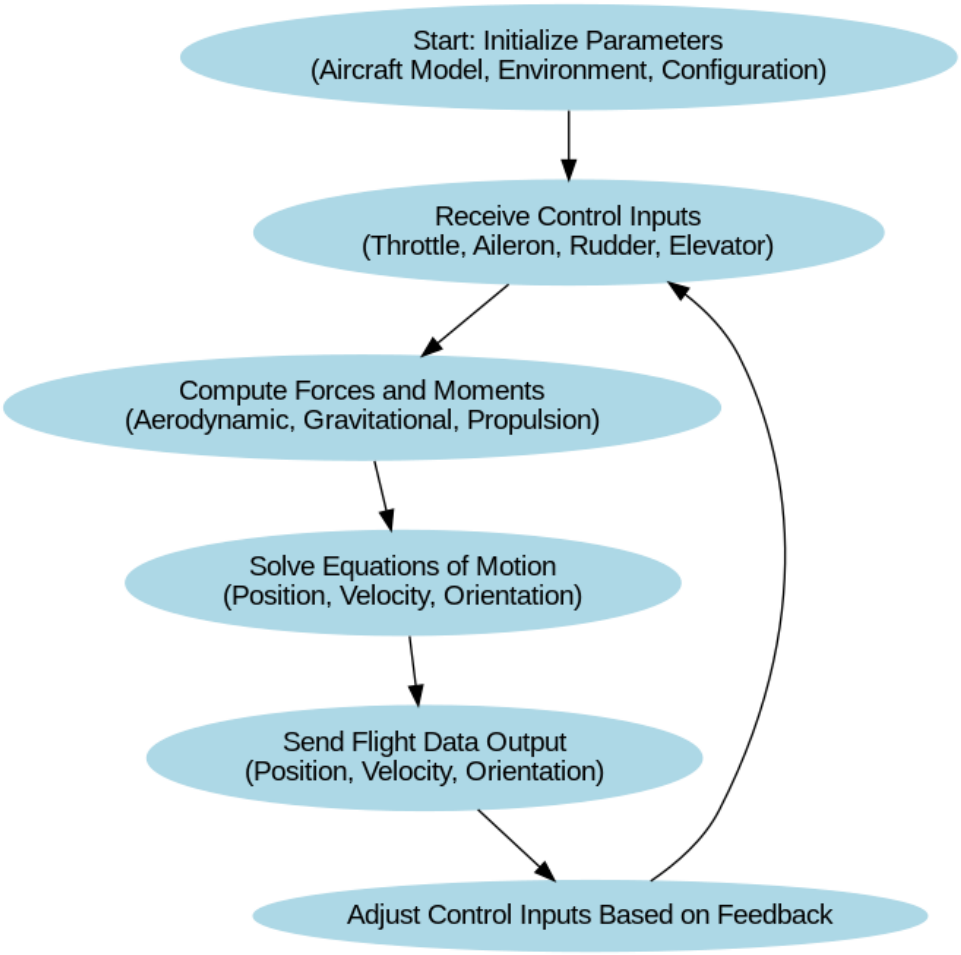


Figure 4.2: Flowchart illustrating the working mechanism of JSBSim.

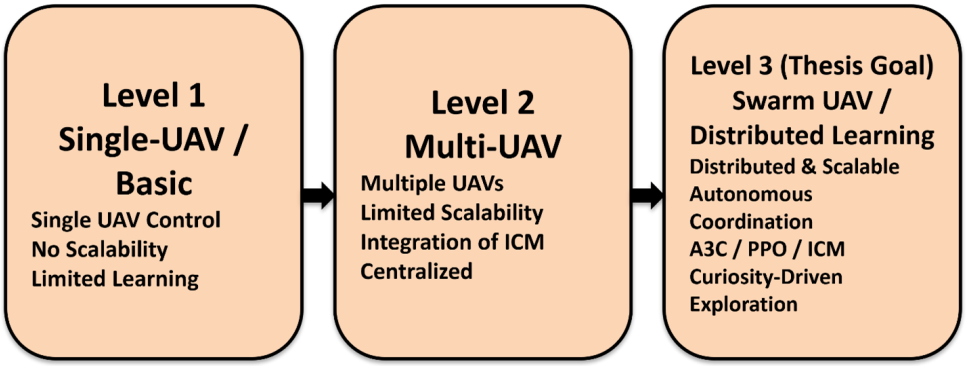


Figure 4.3: Maturity Model of Testbed

Figure 4.3 shows the Maturity model of the testbed. This testbed is considered the next version of the NUAV testbed because the basic architecture and concepts have

been derived from it. Previous UAV testbeds suffered from several shortcomings that limited their applicability for swarm coordination. Many were restricted in scalability except GaTAC, often focusing on single-aircraft setups or small groups. Most testbeds operated on centralized architectures, hindering the distribution of ML models, and relied heavily on extrinsic rewards, which slowed convergence and destabilized policies. To address these gaps, my testbed integrates FlightGear and JSBSim for high-fidelity dynamics, employs distributed learning RL algorithms across a network of computers, and incorporates curiosity-driven exploration with Gaussian reward shaping to ensure scalability, robustness, and stable real-time coordination of swarm UAVs.

The A2C algorithm was selected for controlling the target aircraft because it provides a good baseline for continuous-control tasks. In this research, the target aircraft needs to move randomly, and A2C is well suited for generating such behaviour because it offers a good balance between exploration and stable policy learning. Additionally, A2C serves as the foundational building block of A3C. Since A3C extends the A2C structure into a distributed learning, asynchronous form, it is essential to first implement A2C to establish a consistent control policy for the target before deploying more advanced multi-agent tracking algorithms. Using A2C for the target and A3C for the trackers ensures a clear separation of roles in the testbed and provides a reliable moving reference for the swarm to follow.

4.2 Implementation of the A2C Algorithm

A2C is a synchronous policy-gradient RL algorithm that combines value-based and policy-based approaches. It is implemented to optimize decision-making in environments where agents select actions to maximize long-term rewards. A2C consists of an actor and a critic network. The actor learns a policy $\pi(a|s)$ that maps states to actions, and the critic estimates the value function $V(s)$ to evaluate the quality of a given state. The algorithm uses the advantage function to measure how much better an action is compared to the average, defined as:

$$A(s, a) = Q(s, a) - V(s) \tag{4.1}$$

At each time step, the agent observes its current state within the environment, including its position, speed, altitude, and orientation. The actor network takes this state as input and produces a probability distribution over possible actions. An action is

selected from this distribution and applied to the environment (e.g., change heading or throttle). The environment then returns a reward and the next state based on the action taken. Figure 4.4 shows the implementation of the A2C algorithm, which employs an actor–critic architecture, where the actor network generates a probability distribution over possible control actions (rudder, aileron, elevator, and throttle), and the critic evaluates the value of the current state. The critic computes the value function and helps update the actor through the advantage function, improving the policy over time.

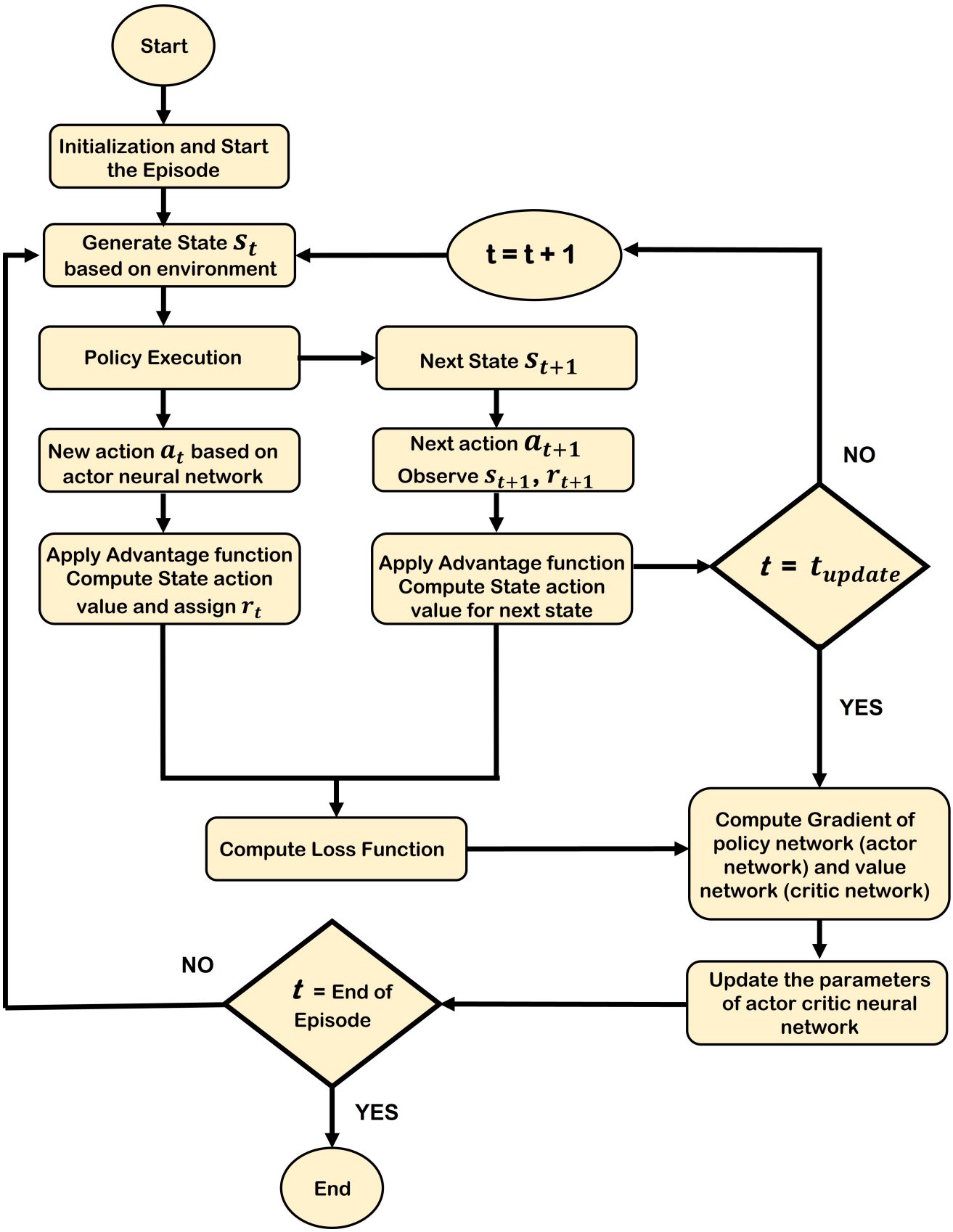


Figure 4.4: Implementation of A2C algorithm

4.2.1 A2C Hyperparameter Tuning

Hyperparameters for A2C were tuned to achieve stable flight for fixed-wing UAVs. Key parameters included the learning rate, discount factor (γ), GAE (λ), rollout length (n_steps), entropy and value-loss coefficients, gradient clipping, and advantage normalization. Using $\lambda = 0.95$ helped balance bias and variance, and advantage normalization ensured consistent training across different reward scales. The A2C approach relies on hyperparameters that are given in Table 4.1, which are necessary for fine-tuning models to attain peak performance.

A coarse search was first carried out by testing a wide range of values to quickly see which settings allowed the model to train without crashing or behaving unpredictably. After identifying the values that worked reliably, a more focused search was performed by testing smaller variations around these good settings. These refined tests used longer training runs and were evaluated based on overall reward, tracking accuracy, and how smoothly the aircraft responded to control actions. This process ensured that the final hyperparameters were stable, effective, and suitable for fixed-wing UAV tracking. Other key hyperparameters included the discount factor, GAE parameter, rollout length, entropy coefficient, value-loss coefficient, and gradient clipping, all of which contributed to reliable convergence and safe behaviour during training.

Hyperparameter	**Value**
Learning rate (`lr`)	3×10^{-4}
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Rollout length (n_steps)	20
Entropy coefficient (c_H)	0.01
Value loss coefficient (c_V)	0.5
Max gradient norm	0.5
Advantage normalization	On
Batch size (per update)	1024

Table 4.1: Final tuned hyperparameters for A2C

Traditional single-agent learning methods are often too slow or unstable for real-time UAV scenarios, especially when multiple aircraft must coordinate in complex paths. The A3C algorithm is well-suited to these challenges because it allows several agents to learn in

parallel asynchronously, each exploring different parts of the environment. This parallel learning approach produces faster, more diverse experiences and leads to quicker and more stable policy improvement. A3C is therefore selected as one of the main tracking algorithms in this testbed because it offers strong performance, efficient exploration, and reliable convergence in multi-agent and distributed learning UAV environments.

4.3 Implementation of the A3C Algorithm

The A3C algorithm integrates the actor-critic structure with an asynchronous training mechanism to enable efficient RL. The A3C algorithm comprises two primary components: the actor and the critic. The actor is responsible for defining the policy by selecting actions based on the current state of the agent. This policy produces a probability distribution over possible actions. One of the defining qualities of A3C lies in its asynchronous training approach, where multiple workers interact with separate instances of the environment simultaneously. Each worker interacts with its respective part of the environment and periodically calculates gradients from its local experience. These gradients are then applied asynchronously to update the global network. In TensorFlow, this is implemented using Gradient$_{T}apetocomputegradients, andoptimizer.apply.gradients$ is used to apply updates to the shared global network. PyTorch is also widely used, but TensorFlow was chosen because it provides better support for parallel execution and distributed learning training, which are essential for A3C. At the time of implementation, TensorFlow's mature ecosystem, built-in optimizers, and stable integration with simulation frameworks offered more reliability for real-time UAV experiments compared to PyTorch's then less-developed multi-threading and distributed learning features.

TensorFlow was selected for this research due to its mature ecosystem, strong support for large-scale machine learning workflows, and integration capabilities with distributed execution frameworks. It provides stable APIs for implementing reinforcement learning models and supports efficient handling of computational graphs, which is beneficial for managing complex training pipelines involving multiple agents. In addition, TensorFlow integrates effectively with distributed computing tools such as Ray, which is used in this research to manage multi-agent execution across multiple nodes. This compatibility supports the implementation of asynchronous and distributed learning required for the testbed.

4.3.1 Step-by-Step Workflow of Asynchronous Updates in A3C

The following steps outline the complete workflow using two workers as an example [103, 104, 105]:

1. **Initialization:** A global neural network is created with shared parameters for the actor (policy) and critic (value function). Each worker creates its own local copy to interact with the environment.

2. **Local roll-outs:** Both workers independently observe the current state, choose actions based on their local policy, receive rewards, and store the experiences (state, action, reward, next state) in a local buffer.

3. **Independent experience collection:** Workers proceed for a fixed number of steps or until a terminal state is reached. This process is done entirely independently by each worker, with no synchronization required during rollout.

4. **Gradient computation:** Each worker computes the policy gradient and value loss using its local experience. The advantage function is estimated as:

$$A(s_t, a_t) = R_t - V(s_t) \tag{4.2}$$

   where R_t is the cumulative reward and $V(s_t)$ is the estimated value.

5. **Global update:** Each worker sends its computed gradients to the global network. The global network immediately applies these updates using an optimizer (e.g., RMSProp or Adam), without waiting for other workers.

6. **Parameter synchronization:** After updating the global network, each worker periodically synchronizes by copying the latest global parameters into its local network to benefit from the improvements learned by others [103].

   Figure 4.5 shows the implementation of the A3C algorithm, where multiple asynchronous workers interact with separate environments in parallel to gather experiences. Each worker computes gradients for the actor and critic networks, then sends updates to a shared global network. The global model aggregates these updates and redistributes parameters back to the workers, enabling faster learning, better exploration, and stable policy improvement.

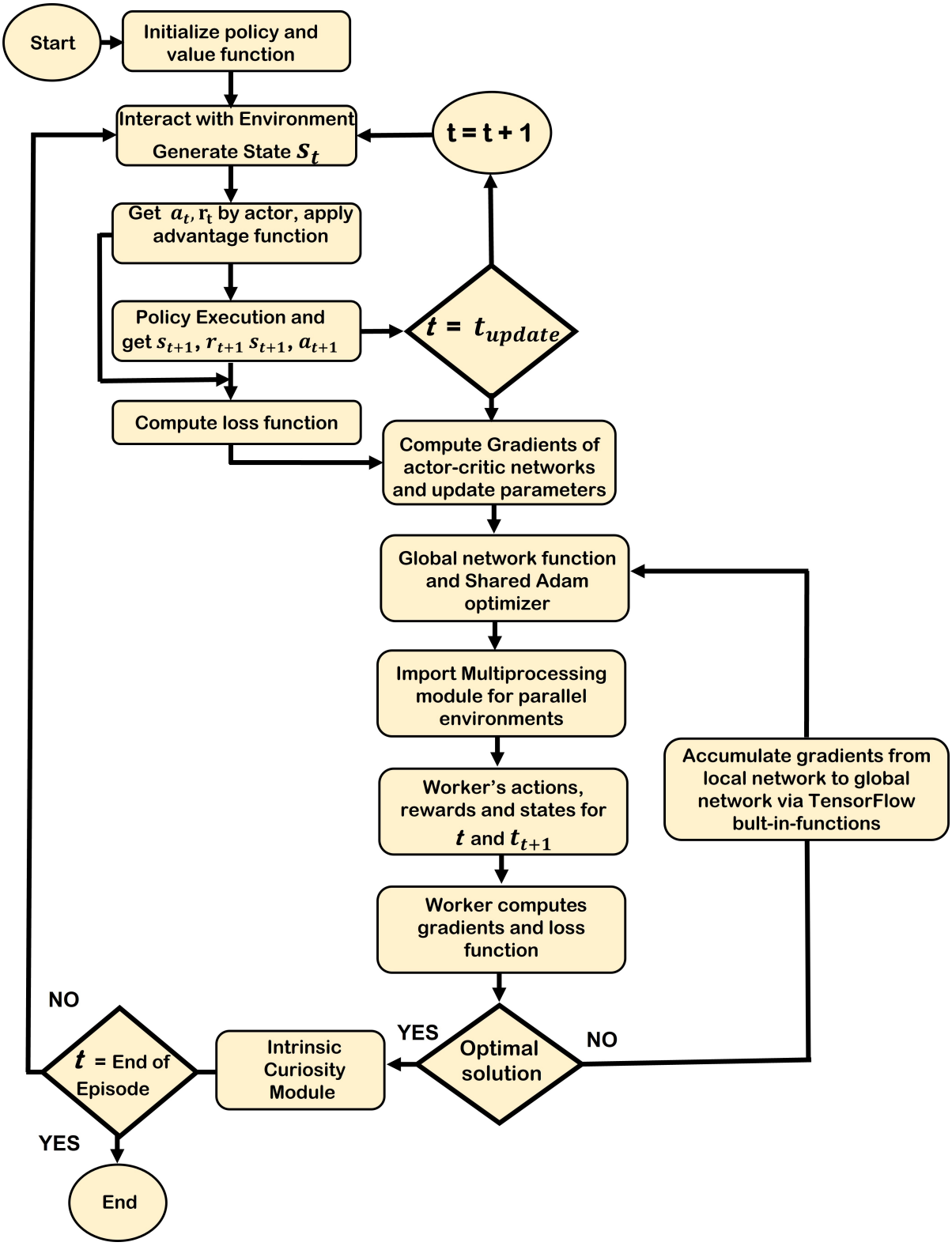


Figure 4.5: Implementation of A3C algorithm.

In UAV applications, A3C allows multiple simulated UAVs to train simultaneously with diverse experiences, accelerating policy development for navigation, tracking, or coordination tasks. It is particularly well-suited for large, continuous environments where traditional single-threaded learning would be too slow or unstable [103].

4.3.2 A3C Hyperparameter Tuning

Model performance can be optimized through hyperparameter adjustments. This is a fundamental step in the ML process, and suitable hyperparameter values are important for the results. Table 4.2 has all the hyperparameters used in training the model. Initially, the discount factor was set as 0.3, but after the completion of training, the performance was not optimal. Thus, the values were modified until improvement. The A3C relies on hyperparameters, which are necessary for fine-tuning models to attain peak performance. The number of training steps is the most critical hyperparameter in this project. Initially, the agent was trained with 1 million time steps. But when the training was completed after one week, there was no difference in the trained and untrained agents' behaviour. The number of time steps increased from 1 million to 3 million, and then there was a significant difference in behaviour. The time steps were further increased to 30 million using a trial-and-error approach, and the results were excellent as the agent showed an improvement in the stability of the aircraft. There were incremental steps in between. The model inputs are velocity, altitude, acceleration, and aircraft position in the x,y, and z planes. The model outputs are values for the rudder, elevator, aileron, and throttle actuators.

Time steps	3 million, 30 million
Gamma (Discount factor)	0.5
Lambda	0.92
Learning rate	1e-3
Max. grad. norm	0.7
Num. mini batch	8
Update epochs	1
Entropy coefficient	0.01
Activation function	ReLU

Table 4.2: Hyperparameters for A3C Model

The ReLU activation function was used because it is lightweight, fast to compute, and generally provides more stable gradient flow compared to older functions such as sigmoid or tanh, which often suffer from vanishing gradients. ReLU helps the network learn faster and handle the continuous-control demands of fixed-wing UAVs more effectively. Other activation functions were considered, but ReLU offered the best balance of stability,

simplicity, and performance for the actor–critic models used in this testbed, making it a suitable choice for real-time learning and control.

In this research, ReLU is used in the hidden layers due to its computational efficiency and ability to mitigate vanishing gradient issues during training. However, it is acknowledged that ReLU can lead to the problem of dead neurons, where neurons become inactive and stop contributing to learning if they consistently receive negative inputs.

Alternative activation functions such as Leaky ReLU and Tanh were considered. Leaky ReLU allows a small gradient for negative inputs, which helps to reduce the risk of dead neurons and can improve learning stability. Tanh, on the other hand, provides bounded outputs in the range of -1 to 1, which is particularly useful for control tasks where actions need to remain within a defined range. In the implementation of this testbed, ReLU is used for hidden layers to maintain computational efficiency, while bounded activation functions such as Tanh are used in the output layer to ensure that control signals remain within valid limits for UAV operation. This combination provides a balance between efficient training and stable control behaviour, while addressing the limitations associated with using ReLU alone.

It is important to clarify that grid search was utilized using the scikit-learn library in this research, particularly during the initial tuning of the A3C model. Specifically, Table 4.2 in the thesis presents the selected hyperparameters for A3C that were obtained through a grid search process. This baseline tuning step ensured that the learning models operated under reasonable and stable configurations before introducing more advanced tuning strategies. However, the primary motivation for introducing the SRCW curiosity tuning strategy was to address the limitations of static hyperparameter optimisation methods such as grid search. Grid search provides fixed parameter values and does not adapt during training, whereas the SRCW approach dynamically adjusts curiosity-related parameters based on the agent's learning behaviour. The key distinction is that:

1. Grid search is a static tuning method that identifies suitable parameter combinations prior to training.

2. SRCW is a dynamic, adaptive tuning mechanism that modifies parameters during training based on feedback from the learning process.

3. The SRCW strategy was designed to improve exploration and convergence in

environments with delayed rewards, where static tuning methods may not be sufficient.

The table compares the traditional Grid Search method with the proposed SRCW tuning strategy. Grid Search performs static hyperparameter optimisation before training begins and keeps the selected parameters fixed throughout the learning process. In contrast, SRCW dynamically adjusts curiosity-related parameters during training based on the agent's learning behaviour, enabling real-time adaptation. This makes SRCW more suitable for reinforcement learning environments with delayed rewards, where fixed parameter settings may not provide optimal exploration and convergence. Furthermore, SRCW achieves this adaptability with moderate computational cost while improving learning efficiency in complex UAV tracking tasks. Table 4.3 shows the comparison of Grid Search and SRCW Tuning Strategies.

Aspect	**Grid Search**	**SRCW**
Type	Static tuning	Dynamic adaptive tuning
When Applied	Before training	During training
Adaptability	Fixed parameters	Adjusts in real time
Handling Delayed Reward	Limited support	Specifically designed to improve learning under delayed reward
Computational Cost	High (exhaustive search)	Moderate
Suitability for RL	Basic baseline	More suitable for complex RL

Table 4.3: Comparison of Grid Search and SRCW Tuning Strategies

4.3.3 Implementation of Intrinsic Curiosity Module

Curiosity-driven learning is an approach that incentivizes agents to explore their environment by providing internal rewards for encountering novel or unexpected states. Unlike conventional RL methods that depend solely on externally provided rewards (extrinsic rewards), this method introduced intrinsic motivation, allowing the agent to seek and learn from unfamiliar experiences. This mechanism is particularly useful in environments where external rewards are sparse or delayed, as it promotes consistent learning progress. Figure 4.6 illustrates the implementation of the ICM, which supports curiosity-driven learning by generating intrinsic motivation based on prediction

discrepancies. The ICM also supports adaptability to dynamic target paths, helping to avoid getting stuck in suboptimal trajectories. This leads to more robust, accurate, and responsive tracking performance in a complex environment.

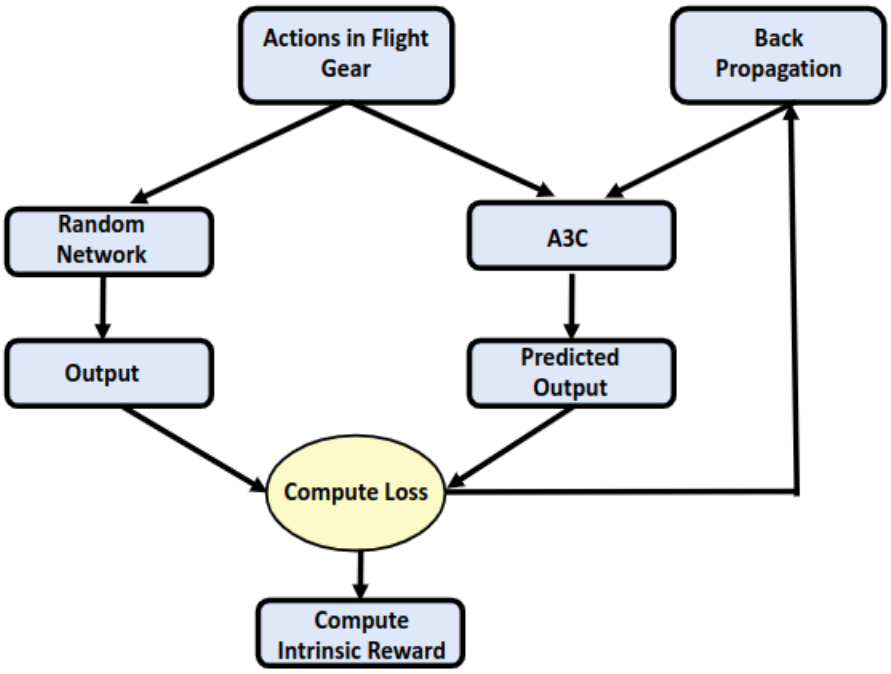


Figure 4.6: Implementation of ICM.

SRCW tuning is an adaptive hyperparameter tuning mechanism specifically for ICM. It dynamically adjusts critical hyperparameters within a specific range, such as ICM-learning rate, entropy coefficient, and reward normalization based on real-time feedback from intrinsic rewards.[8, 9]. Table 4.4 shows the hyperparameters of the model.

Hyperparameter	Range	Description
icm_beta	0.1 to 0.6	Balances intrinsic (curiosity) and extrinsic rewards
icm_learning_rate	1×10^{-4} to 5×10^{-4}	Learning rate for the ICM's internal forward and inverse models
entropy_coefficient	0.005 to 0.02	Controls randomness in policy to promote exploration
gradient_clipping_norm	20 to 40	Upper limit for gradient norms to stabilize training
rollout_length (Tmax)	5 to 20	Number of steps before the worker sends updates to the global model
inverse_loss_weight	0.6 to 0.9	Weight assigned to inverse model loss in the ICM
forward_loss_weight	0.1 to 0.4	Weight assigned to forward model prediction loss in the ICM
feature_dim	64 to 256	Size of the learned feature representation used in the ICM
curiosity_learning_rate	3×10^{-5} to 5×10^{-4}	Learning rate specifically for curiosity-based updates
reward_normalization	True / False	Whether to normalize total rewards for stable training

Table 4.4: SRCW Tuning [8, 9, 10, 11]

4.3.3.1 Solving the problem of delayed reward using A3C-ICM model

In RL, a delayed reward means that the agent, such as a UAV, did not receive immediate feedback after taking an action. Instead, it only gets a reward much later, perhaps after completing a task or reaching a certain point. During this time, the UAV has no clear idea whether the actions it is taking are good or bad. This makes learning slow and confusing because the UAV cannot immediately connect its earlier movements with the final result. For UAVs, this is a serious problem. They operate in a swarm in continuous environments, where the goal, such as staying close to a moving target at a constant distance, is critical. Without quick feedback, the UAV may continue flying by following poor strategies, wasting time and fuel, and could cause collisions, and fail to

improve its behaviour. To solve this problem, we utilized the A3C-ICM model, which incorporates a special type of reward known as the intrinsic reward. The UAV doesn't have to wait for a delayed external reward to learn something useful. The curiosity reward fills in the gaps, keeping the UAV motivated and helping it learn good tracking behaviours much earlier and learn to track moving targets more efficiently, even in complex or changing environments. Figure 4.7 presents the framework where the A3C model is combined with the ICM to address delayed rewards. The ICM generates intrinsic rewards from prediction errors, providing immediate feedback to the agent. These intrinsic signals, added to the external rewards, enable faster learning and more effective tracking in sparse-reward environments.

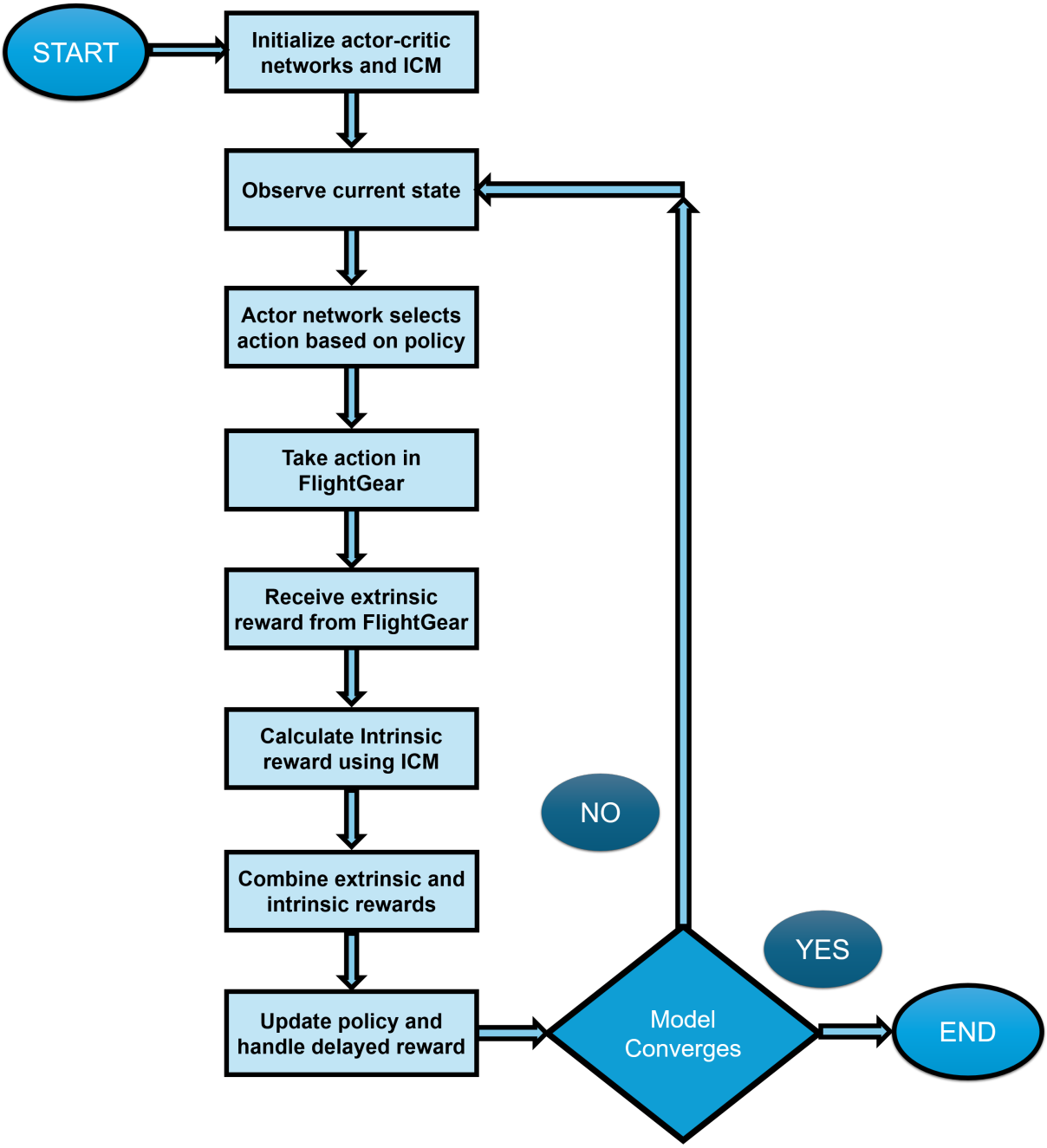


Figure 4.7: Framework to overcome the problem of delayed reward using A3C-ICM model.

The ICM provides the agent with an additional reward signal, called the intrinsic reward, which reflects how novel or unpredictable a state transition is. It does this by encoding the current and next states into feature vectors and using a forward model to predict the next state features from the current state and action. The difference between the predicted and actual next state features determines the intrinsic reward: if the prediction is poor, the reward is higher, and if the prediction is accurate, the reward is lower. This intrinsic reward is then combined with the normal task reward from the environment, so the agent is motivated both to achieve the task and to explore new or uncertain states. In UAV tracking tasks, this helps overcome sparse or delayed external

rewards by ensuring the agent continues to learn and explore effectively.

PPO was incorporated because it addresses a key limitation of many policy-gradient methods, including instability caused by overly large policy updates. PPO uses a clipped surrogate objective that explicitly constrains policy changes between successive updates, which helps prevent sudden performance collapse and policy oscillation. This characteristic is especially beneficial in UAV control tasks, where aggressive or unstable policy updates can lead to loss of tracking.

4.4 Implementation of PPO Algorithm

PPO is categorized as a policy gradient technique for RL agents. In essence, PPO uses a very small policy update to train the appropriate policy network, allowing the agent to consistently arrive at the best solution. If a policy update step size is too small, it will reduce the overall efficiency of the model. However, a large step size might lead policy in the wrong direction with little chance of recovery. As a result, PPO applies a clip function that prevents an agent's policy update from being too big or small (policy degeneration). [106, 107].

Policy Network: The policy network is a NN that generates a probability distribution across potential courses of action based on the current state of the RL agent in the simulated environment. The probability distribution represents the agent's policy, which outlines the likelihood of each action by considering the current situation [108].

Value Network: The value network calculates the predicted cumulative reward for being in a particular state. This estimation reduces the variation of policy gradient estimations. The value network's design is similar to that of the policy network. However, it generally produces a single value rather than a probability distribution [109].

Workings of PPO Algorithm: The policy gradient method, which directly optimizes the policy function that associates states with actions, is the foundation of the PPO algorithm. It seeks to maximize the predicted cumulative reward from the interaction of an agent with the environment. This is usually accomplished by optimizing a surrogate objective function that approximates the policy improvement. The clipped surrogate objective is a crucial component of PPO since it prevents abnormal policy modifications that might result in negative outcomes [110]. PPO ensures that the policy update step size stays within safe boundaries by constraining the ratio of action likelihoods between the new and previous policies through clipping. This stabilizes training and maximizes

sample efficiency [111].

PPO directly optimizes the policy, which maps states to actions, using gradient ascent to maximize cumulative rewards. The PPO objective function improves the policy while ensuring updates are small to maintain training stability. The advantage function measures how much better an action is compared to the baseline. The clipping mechanism limits the policy change by clipping the probability ratio to a specific range. The workings of PPO also include collecting trajectories, computing the advantage function, and optimizing the policy. The agent interacts with the environment using the current policy to collect trajectories of states, actions, and rewards. It uses the trajectories to estimate the advantage function. The agent uses a stochastic gradient to maximize the PPO objective function, ensuring updates remain within the clipped range [112].

In the next section, PPO hyperparameter tuning is discussed. The selected hyperparameters are given in Table 4.5.

4.4.1 PPO Hyperparameter Tuning

The PPO approach relies heavily on hyperparameters, which are necessary for fine-tuning models to attain peak performance. The performance of the policy may collapse suddenly and never recover if it is updated in too large a step size. To maintain the transition from the old policy to the new policy within a safe range, PPO employs a surrogate loss function. The duration of the training is an obvious one. This can be a predetermined number of timesteps until a certain end condition is satisfied. Another is the optimizer's learning rate. The Adam optimizer is used by the PPO algorithm [113]. The hyperparameters used for PPO algorithm are given in Table 4.5.

Gamma	0.99
Learning rate	0.001
Clip ratio	0.2
Epochs	10
Batch size	16
Timesteps per batch	4096
Lambda	0.97
VF_coeff	0.5
Entropy_coeff	0.01
Optimizer	Adam

Table 4.5: PPO Model's Hyperparameters

4.5 Implementation of Gaussian Reward Function Centered on Ellipsoidal Boundary Around UAV

In UAV target tracking, the goal is for the tracking UAV to follow the target UAV at an optimal distance. Aircraft should not come too close to each other to avoid collisions. They should not go too far from each other so that they can maintain visibility and communication amongst themselves. To model this behavior, we utilized a combination of a 3D ellipsoidal spatial model and a Gaussian reward function. A virtual ellipsoid is drawn around each aircraft to find the distance between itself and its neighbouring aircraft. This is shown in Figure 4.8. According to this scheme, if the distance between the target and tracking aircraft increases or decreases too much, then the reward model will give a penalty. However, the model will provide rewards if one of them remains close to the ellipsoidal boundary of the other, as shown in Figure 4.8.

The trackers are forced to maintain a safe distance from each other following this logic. To ensure safety and spatial awareness in UAV target tracking, we draw a 3D virtual ellipsoid around each tracking UAV. This ellipsoid defines its safety zone. It is centered at the UAV's centre position (x_0, y_0, z_0), and its general equation is given as [114, 115]:

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} + \frac{(z-z_0)^2}{c^2} = 1 \tag{4.3}$$

This is the standard equation of a unit 3D ellipsoid centered at (x_0, y_0, z_0). The parameters a, b, and c are the semi-axes. These govern the spread of the ellipsoid in each of the

orthogonal dimensions. They also define the shape and volume of the ellipsoid. A point (x, y, z) lies on the ellipsoid surface if it satisfies this relation. Let the position of the target UAV be: $\mathbf{p}_t = (x_t, y_t, z_t)$ and the position of the tracking UAV be: $\mathbf{p}_r = (x_r, y_r, z_r)$. The relative position vector is calculated according to equation 4.4:

$$\Delta\mathbf{p} = \mathbf{p}_r - \mathbf{p}_t = (x_r - x_t, y_r - y_t, z_r - z_t) \tag{4.4}$$

Using this relative vector, we define the normalized ellipsoidal distance D_e as in equation 4.5 [116, 115]:

$$D_e = \frac{(x_r - x_t)^2}{a^2} + \frac{(y_r - y_t)^2}{b^2} + \frac{(z_r - z_t)^2}{c^2} \tag{4.5}$$

D_e represents the normalized ellipsoidal distance from the tracking UAV to the target UAV. Although (x_r, y_r, z_r) and (x_t, y_t, z_t) are the coordinates of the tracking and target UAVs respectively, it should be noted that this relation can be used to compute the distance between any two aircraft in 3D space. In this context, $D_e = 1$ indicates the tracker is precisely at the ellipsoid boundary of the target, which is an ideal tracking distance. If $D_e < 1$, it implies the tracker is within the ellipsoid, representing a possible collision risk. If $D_e > 1$, the tracker is outside the ellipsoid, and it may be too far from the target aircraft to maintain reliable tracking. We use a Gaussian function to reward or penalize the tracker. The Gaussian is centered on the target's ellipsoidal boundary (i.e., $D_e = 1$). This reward function is defined as [117] and is given in equation 4.6:

$$R(D_e) = \exp\left(-\frac{(D_e - 1)^2}{2\sigma^2}\right) \tag{4.6}$$

where $R(D_e)$ is a Gaussian-based reward function. σ controls the spread of the curve. The reward is maximized when $D_e = 1$ (on the boundary of the ellipsoid).

Here, σ is similar to the standard deviation as in a Gaussian distribution, controlling the spread of the reward function. A small value of σ yields a sharp peak around $D_e = 1$, whereas a larger value of σ produces a shallow and wide reward curve.

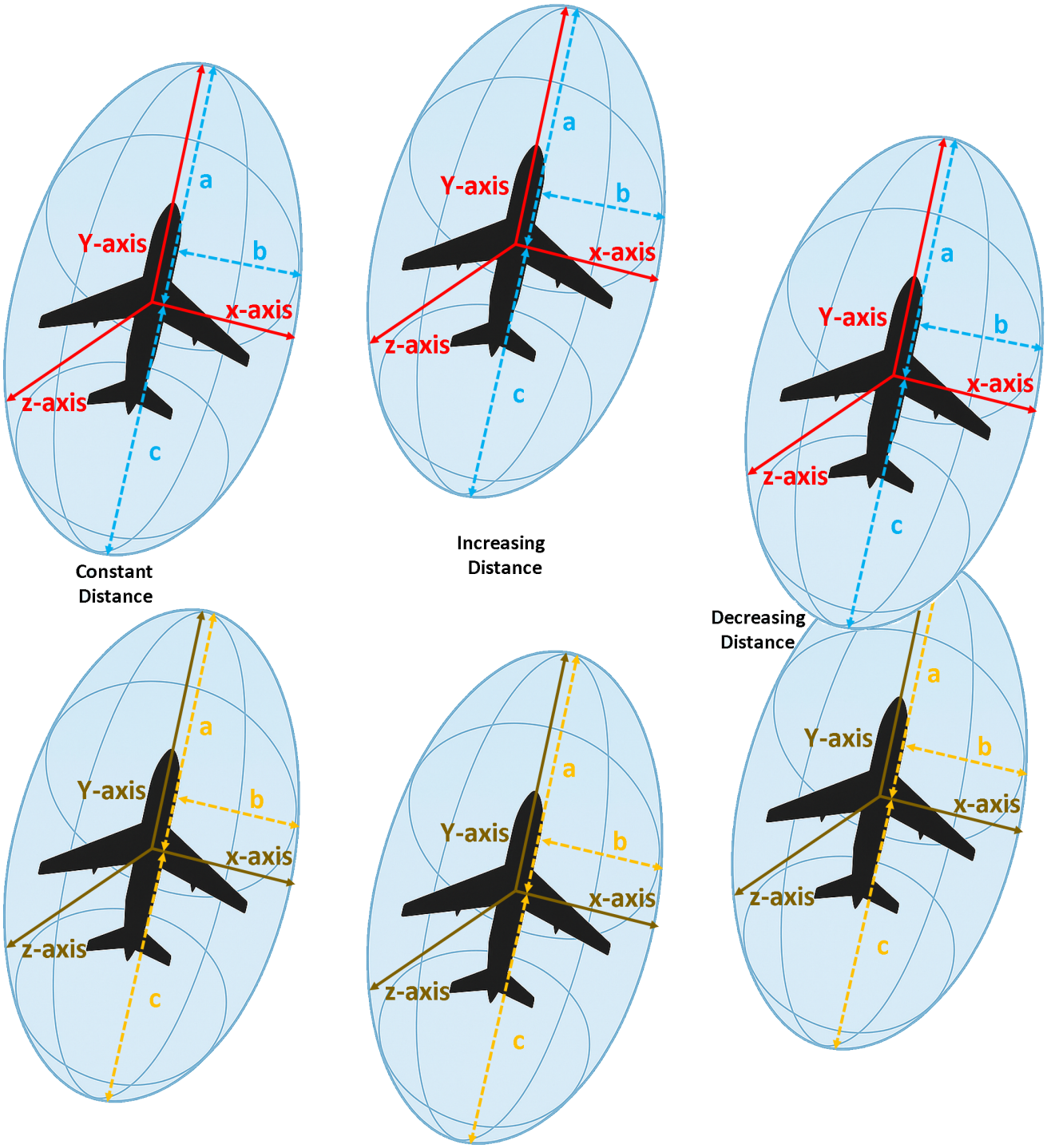


Figure 4.8: Reward and penalty based on the distance between UAVs

After having established this, we face the problem of dealing with the rapidly changing orientations of UAVs during flight. The orientation of a UAV is defined by three basic rotations: roll, pitch, and yaw. The 3D rotation matrix is based on Euler angles: roll ϕ, pitch θ, and yaw ψ. It transforms vectors between the coordinate frames of a UAV.

These rotations define the attitude of a UAV and are captured in the Euler rotation

matrix given in equation 4.7 [118]:

$$R = \begin{bmatrix} \cos\phi\cos\psi & \sin\theta\sin\phi\cos\psi - \cos\theta\sin\psi & \cos\theta\sin\phi\cos\psi + \sin\theta\sin\psi \\ \cos\phi\sin\psi & \sin\theta\sin\phi\sin\psi + \cos\theta\cos\psi & \cos\theta\sin\phi\sin\psi - \sin\theta\cos\psi \\ -\sin\phi & \sin\theta\cos\phi & \cos\theta\cos\phi \end{bmatrix} \tag{4.7}$$

Let the position vectors of the target UAV and the tracking UAV be given by equation 4.8 [119]:

$$\mathbf{p}_t = (x_t, y_t, z_t), \quad \mathbf{p}_r = (x_r, y_r, z_r) \tag{4.8}$$

The relative position vector is defined according to equation 4.9 [119]:

$$\Delta\mathbf{p} = \mathbf{p}_r - \mathbf{p}_t = (x_r - x_t, y_r - y_t, z_r - z_t) \tag{4.9}$$

In real-world UAV flight scenarios, the ellipsoidal safety boundary may not always be aligned with the global coordinate axes. To account for such changes in orientation, a robust and general approach is to transform the global point of interest into the local coordinate system of the ellipsoid, in which it appears axis-aligned. Once transformed, the standard equation of the ellipsoid can be applied to determine spatial relationships such as collision risk or tracking optimality. This process involves two key steps: translation and rotation. First, the global point (e.g., the position of a tracking UAV) is translated so that the centre of the ellipsoid becomes the origin. Next, the translated point is rotated using the inverse of the rotation matrix of the ellipsoid. Since rotation matrices are orthogonal, the inverse is simply the transpose. This maps the global point into the local, unrotated coordinate frame of the ellipsoid.

$$p_{\text{local}} = \mathbf{R}^\top(p_{\text{global}}) \tag{4.10}$$

The matrix $\mathbf{R}$ is a rotation matrix constructed from the Euler angles of the target UAV's orientation. This transformation first translates the global position relative to the ellipsoid's centre and then rotates it into the ellipsoid's local coordinate system aligned with its principal axes, where $R^\top$ is the transpose of the rotation matrix of the ellipsoid. This transformation aligns the analysis with the principal axes of the ellipsoid,

enabling the use of the standard ellipsoid inequality, as in equation 4.11:

$$D_e = \frac{x'^2}{a^2} + \frac{y'^2}{b^2} + \frac{z'^2}{c^2} \tag{4.11}$$

where $(x', y', z') = p_{\text{local}}$ are the local coordinates and a, b, and c are the semi-axes lengths. A value of $D_e < 1$ indicates the point is inside the ellipsoid (potential collision), $D_e = 1$ lies exactly on the surface, and $D_e > 1$ lies outside the safety boundary. This transformation-based approach is both mathematically sound and computationally efficient for tracking, reward shaping, and collision avoidance in rotated 3D ellipsoidal models. Grab the values of roll, pitch, and yaw, and plug them into the rotation matrix in 4.7. Figure 4.9 shows the calculation of the distance from a point of a tracker aircraft to another ellipsoid around a target aircraft.

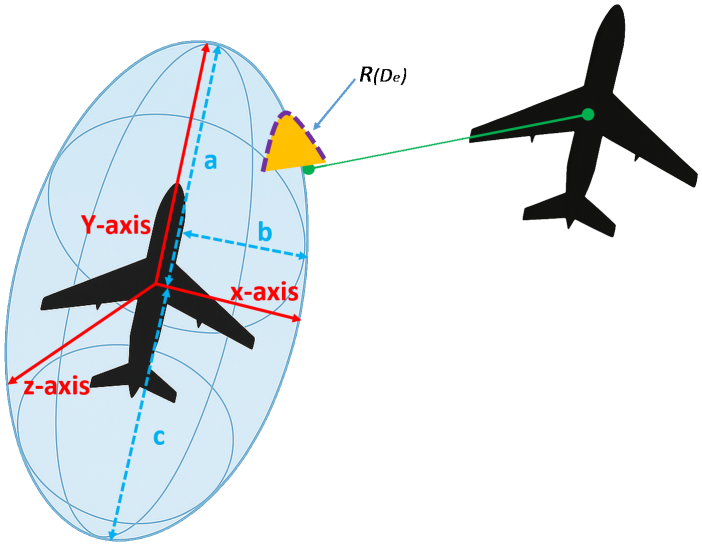


Figure 4.9: Illustration of the calculation of reward by placing the Gaussian reward function on the boundary of an ellipsoid. The UAV inside the ellipsoid corresponds to a tracking UAV under consideration that has to compute its reward. The other UAV is either a target or some other tracker.

The updated reward equation defines the total reward R_i for each tracker UAV i based entirely on ellipsoidal distances and Gaussian reward shaping. The first term computes the sum of Gaussian rewards for all neighbouring trackers UAVs $j \in \mathcal{T}$ that lie within the ellipsoidal safety region of tracker i, specifically those satisfying $D_{e_{ij}} \leq 1$. If no such neighbours exist, this term is replaced by a single Gaussian reward corresponding to the tracker with the minimum $D_{e_{ij}}$. The second term represents a global reward signal derived from the ellipsoidal distance between the tracker i that is closest to the target and the target UAV. The target-focused reward is based on the tracker that is closest to

the target, and this reward is uniformly shared with all trackers.

$$R_i = \sum_{\substack{j \in \mathcal{T} \\ D_{e_{ij}} \leq 1}} \exp\left(-\frac{(D_{e_{ij}}^{(1)} - 1)^2}{2\sigma^2}\right) + \exp\left(-\frac{(D_{e_{it^*}}^{(2)} - 1)^2}{2\sigma^2}\right) \tag{4.12}$$

The reward equation defines the total reward R_i for each tracker UAV i using two separate ellipsoidal Gaussian components. The first term captures the local reward contribution from neighbouring trackers j whose ellipsoidal distance is $D_{e_{ij}}^{(1)} \leq 1$. A Gaussian reward is calculated for each such neighbour and summed. This term tries to avoid collisions when multiple UAVs are within the virtual ellipsoid of a given UAV. When no UAV is within the ellipsoidal region of this UAV, it bases the reward (or penalty) on its distance from its closest other tracker UAV in the swarm, which enforces cohesion within the swarm.

The second term introduces a global reward based on the ellipsoidal distance $D_{e_{it^*}}^{(2)}$ between the tracker i and the target UAV, determined by the tracker with the minimum ellipsoidal distance to the target. Unlike the local term, this global reward is assigned as such to all tracking partners to maintain collective alignment toward the target's location.

It should be noted that it is the nature of the Gaussian reward function that it automatically rewards or penalizes a UAV depending on its distance from the ellipsoidal boundary of the aircraft. At any point in time during calculations, the Gaussian reward function is centered at the ellipsoidal boundary of the aircraft under consideration at a point that is perpendicular to another aircraft. This is illustrated in Figure 4.9. If that aircraft lies on the ellipsoidal boundary of another aircraft, it gets the maximum reward. In contrast, the further an aircraft is from the ellipsoidal boundary of the other aircraft (under consideration), the reward diminishes in a Gaussian sense and begins to convert into a penalty as the distance increases. The Gaussian reward function penalizes an aircraft that is too close or too far from another aircraft that is under consideration.

The multi-objective reward function combining ellipsoidal and Gaussian components was designed to simultaneously enforce tracking accuracy, formation cohesion, and collision avoidance within a unified optimisation framework. In the testbed, each UAV optimises a single composite reward function that integrates these objectives, rather than optimising them independently. The ellipsoidal Gaussian reward defines a spatial constraint around the target aircraft, encouraging UAVs to maintain a desired formation region while avoiding excessive deviation. This creates a shared behavioural structure across all agents. The experimental results in Chapter 5 provide evidence of coordinated

behaviour emerging from this reward design. For example, trajectory plots and minimum–maximum inter-UAV distance graphs show that the UAVs maintain stable relative positioning and avoid collisions while tracking the target aircraft. These results indicate that the agents do not act independently but instead exhibit consistent spatial organisation and cooperative tracking behaviour. However, it is acknowledged that the evaluation presented in the thesis primarily demonstrates coordination through behavioural observation and aggregate metrics, rather than through explicit quantitative measures of coordination. As a result, while the observed behaviour strongly suggests coordinated multi-agent performance, it does not formally distinguish between emergent coordination and parallel optimisation of similar reward structures. To address this, the thesis has been revised to clarify that the coordination observed in the testbed is emergent and driven by a shared reward formulation, rather than enforced through explicit coordination mechanisms. The contribution of the reward function is therefore framed as enabling coordinated behaviour through implicit coupling of objectives within a single optimisation framework.

The Gaussian reward function is used in this research because it provides a non-linear relationship between reward and distance from the desired tracking region. Unlike a simple linear distance-to-target reward, where the reward changes at a constant rate with distance, the Gaussian reward changes according to an exponential function. Consequently, the reward decreases gradually as the UAV moves away from the desired region and increases as the UAV approaches that region. This characteristic is particularly useful for UAV target tracking because it places greater emphasis on maintaining the desired tracking position once the UAV is close to the target. Small deviations near the desired region result in noticeable reward changes, encouraging more precise tracking behaviour. In contrast, a linear reward treats all distance changes equally, regardless of the UAV's position relative to the target. In addition, the reward architecture incorporates an ellipsoidal boundary that defines a desired tracking region around the target aircraft. This allows the UAV to be rewarded for remaining within an acceptable formation region rather than attempting to reach a single fixed point. Such behaviour is more consistent with practical UAV tracking scenarios, where maintaining an appropriate relative position is often more important than minimizing distance to an exact location. As a result, the Gaussian reward function improves learning stability, provides smoother convergence, and enables more consistent tracking behaviour compared to a simple linear distance-based reward.

4.6 Implementation of Distributed Learning Computing Based Tracking of A2C Aircraft by a Swarm of A3C and PPO Aircraft in a Complex Path

A swarm is a group of UAVs that work together to accomplish a specified task. In this research, we aim to make a swarm of tracking aircraft that follows a target plane. The swarm of UAVs is distributed over four computers. Two tracking aircraft are controlled by two workers of the A3C model, and one tracking aircraft is operated by using the PPO model. The ICM is integrated into both PPO and A3C models. In this research, distributed learning was implemented. Networking was achieved using UDP socket programming, allowing four interconnected systems to communicate in real time while controlling a swarm of UAVs. Each system was assigned a specific UAV, with one system controlling the target UAV and the other three systems handling tracking UAVs.

The Ray framework played a crucial role in managing the distributed learning execution of RL agents. It converted RL workers into Ray actors, allowing parallel computation and synchronization of UAV actions across multiple machines. Ray ensured fault tolerance, automatic workload distribution, and state recovery, preventing disruptions if a node failed. There is only one server, the A3C global network, which acts as the central coordinator, handling communication with multiple clients, including A3C worker-1, A3C worker-2, and PPO. Figure 4.10 shows the architecture of the Ray framework.

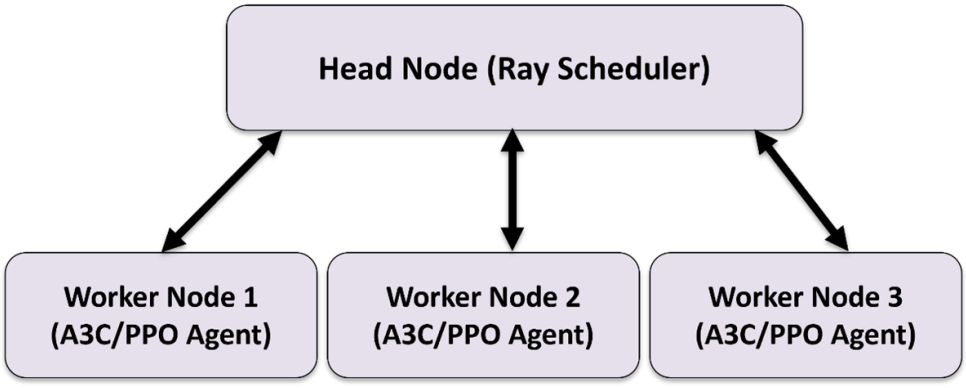


Figure 4.10: Flowchart shows Ray handles the communication between the separate Nodes

The server described in this work should be understood as a central coordination server rather than the traditional global network of the A3C algorithm. While the Actor Critic

customized function is implemented, the role of the server in this testbed is broader and focuses on managing distributed communication, synchronization, and parameter exchange among multiple agents operating across different machines.

In contrast to a standard A3C global network, which aggregates asynchronous gradient updates from homogeneous A3C workers using a unified loss function, the central coordination server in this testbed does not directly combine heterogeneous gradients from different algorithms. Specifically, the PPO agent operates using its own clipped surrogate objective and maintains its own optimization process. Therefore, PPO updates are not mathematically merged with A3C updates in a single global model.

The server in this architecture functions as a central coordination server rather than a traditional A3C global network performing unified gradient updates. The A3C workers follow the standard asynchronous update mechanism, where their gradients are applied to a shared A3C global model. In contrast, the PPO agent maintains a separate policy network and performs updates using its own clipped surrogate objective. Therefore, there is no direct combination of clipped (PPO) and unclipped (A3C) gradients within a single optimization step. The role of the central server is limited to coordination, communication, and synchronization across distributed agents, rather than merging heterogeneous gradient updates.

There are also several limitations of the A3C algorithm. Extensive updates to the policy in A3C can destabilize training and lead to poor performance. Over-aggressive updates might push the policy far from the current solution, making recovery difficult. The limitations of the A3C algorithm have been overcome by using the PPO algorithm in the distributed learning RL model.

The combination of A3C workers and a PPO agent as trackers was chosen to exploit the complementary strengths of both algorithms in a swarm setting. A3C is well-suited for distributed learning because its asynchronous workers explore different parts of the state–action space in parallel and update a shared global network, leading to faster convergence and diverse policy exploration. By deploying two A3C workers, the testbed ensured broader coverage of the search space and reduced the risk of all trackers converging to similar suboptimal behaviours. On the other hand, PPO was included because of its stability through policy clipping, which prevents overly large policy updates and improves robustness in high-risk or complex flight paths. Figure 4.11 shows the distributed learning computing framework.

In this research, distributed learning is defined as the execution of multiple learning

agents across separate computational nodes connected through a local area network, rather than only parallel execution on a single machine. It is important to distinguish between parallel learning and distributed learning. In this research, distributed learning is viewed from the perspective of transparency, where the user interacts with the testbed as a single unified environment without needing to know whether learning agents are executed on one machine or across multiple networked machines. The underlying computational resources are abstracted from the user, while the testbed manages communication and coordination transparently. The distributed architecture, therefore, provides scalability and resource sharing without exposing implementation details to the end user. The machine learning practitioner interacts with a unified learning environment without needing to be aware of whether agents are executed on a single machine or across multiple networked nodes. Furthermore, the A3C algorithm introduces concurrent asynchronous learning through multiple workers operating independently and updating shared parameters without synchronization barriers. The thesis has been revised to clarify the distinction between the distributed computational architecture and the concurrent learning behaviour inherent in A3C. This definition is reflected in the experimental validation, where multiple UAVs are deployed across separate machines to demonstrate real-time coordination and learning. The testbed developed in this research combines a distributed architecture, where learning agents may execute across multiple networked machines, with concurrent asynchronous learning through the A3C algorithm. The thesis has been revised to avoid equating parallel and distributed computing and to more clearly distinguish between the distributed deployment of resources and the concurrent learning behaviour of the reinforcement learning agents.

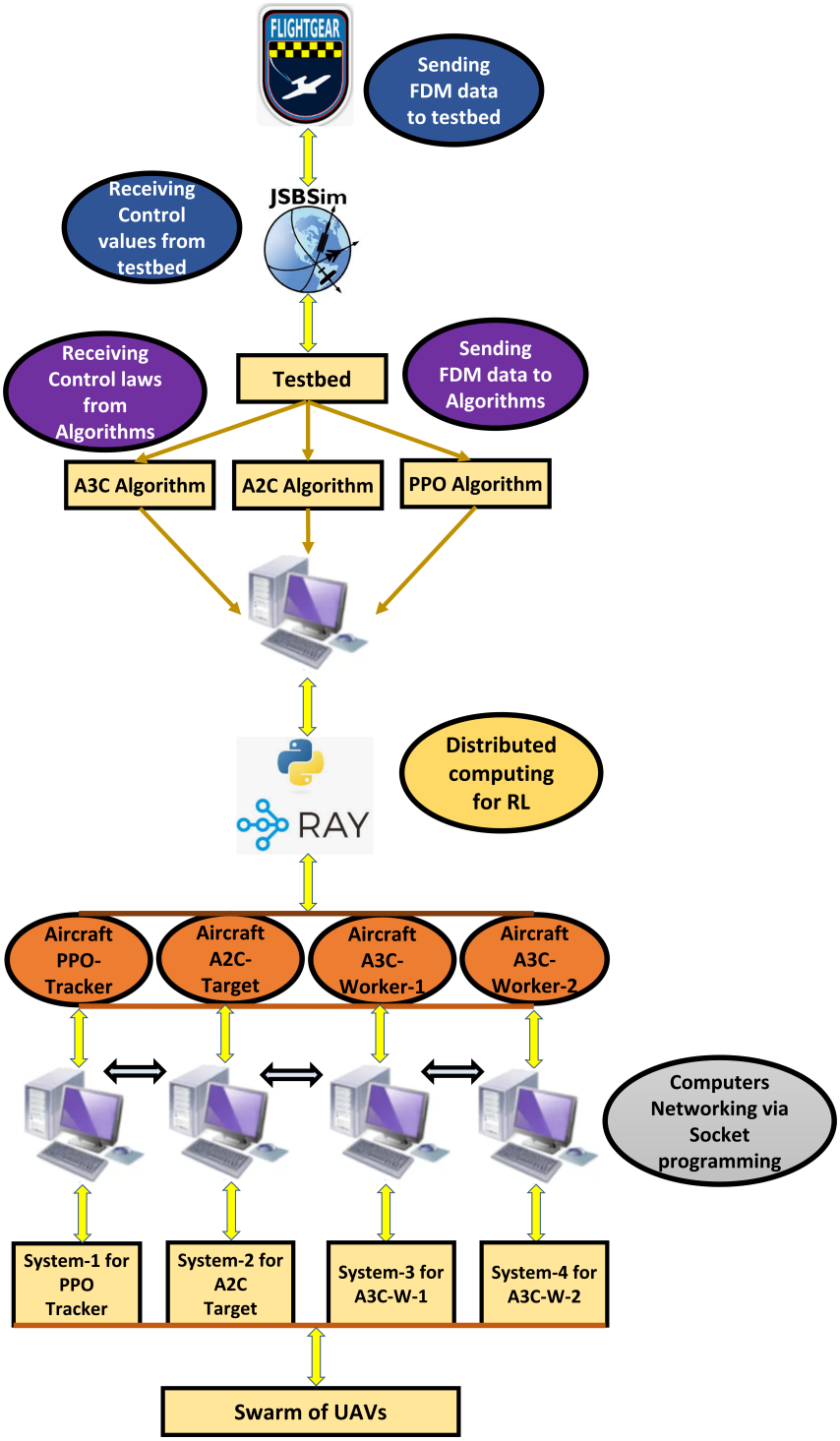


Figure 4.11: Distributed learning computing framework of Testbed for swarm of UAVs.

In this research, it is important to clearly distinguish between distributed learning and asynchronous learning, as these terms refer to different aspects of the testbed. Distributed learning in this research refers to a learning environment in which computational resources and learning agents may be distributed across multiple networked nodes, while the underlying architecture remains transparent to the user. From the perspective of the machine learning practitioner, the testbed operates as a

unified system regardless of whether computation is performed on a single machine or across multiple machines. The distribution of resources, communication between nodes, and coordination of learning processes are managed by the testbed, thereby abstracting the underlying infrastructure from the user. Asynchronous learning, on the other hand, refers to the algorithmic structure used within the A3C model. In this approach, multiple workers operate in parallel and update a shared model independently without requiring synchronisation at every step. This allows faster learning by enabling agents to explore different parts of the state space simultaneously. Distributed learning refers to the architectural organisation of computational resources, where the underlying execution may be distributed across multiple networked nodes while remaining transparent to the user. In contrast, asynchronous learning refers to the learning mechanism of A3C, where multiple workers operate concurrently and update shared parameters independently. Therefore, distributed learning and asynchronous learning are complementary concepts that describe different aspects of the testbed rather than direct counterparts. Therefore, distributed learning in this work is associated with multi-machine execution, while asynchronous learning is associated with the internal operation of the A3C algorithm. The testbed combines both concepts, where asynchronous workers can be deployed across multiple machines, resulting in a distributed and scalable learning framework. This distinction is reflected in the experimental results, where some experiments use parallel workers on a single machine, while others deploy UAV agents across multiple machines to demonstrate true distributed learning. In the context of A3C, the workers are more accurately described as concurrent asynchronous workers rather than parallel workers, as they operate independently and update shared parameters asynchronously.

A2C is used to control the target UAV due to its synchronous learning behaviour, which ensures random motion of the target aircraft. A2C is used to control the target UAV because it provides a stable and predictable learning process through synchronous updates. In this testbed, the role of the target UAV is not to optimise complex cooperative behaviour, but to generate a consistent and dynamically changing trajectory that the tracking UAVs must follow. For this purpose, it is important that the target exhibits smooth, stable, and reproducible motion rather than highly variable or unstable behaviour. The synchronous nature of A2C ensures that updates are consistent across training steps, reducing variance in the learned policy. This leads to more controlled and stable motion of the target UAV, which is important for creating a reliable learning

environment for the tracking agents. If a method such as A3C were used for the target aircraft. The main advantage is that A3C is asynchronous. A3C is used for tracking UAVs because it supports asynchronous learning through multiple workers, enabling efficient exploration and faster convergence. This makes it suitable for multi-agent scenarios where multiple UAVs must learn simultaneously within a shared environment. However, a known limitation of A3C is the potential for policy instability or degeneration, particularly in complex environments where asynchronous updates from multiple workers can lead to inconsistent gradient updates. This may result in oscillatory behaviour or degradation of learned policies over time.

To address this limitation, PPO is incorporated as a complementary algorithm that provides more stable policy updates. PPO uses a clipped objective function to constrain policy changes, preventing large updates that could destabilise learning. This makes PPO more robust to policy collapse and ensures smoother convergence. In this research, PPO is used as a reference and stabilisation mechanism alongside A3C. By comparing the performance of PPO and A3C within the same testbed, it becomes possible to evaluate the impact of asynchronous updates on learning stability. The inclusion of PPO also ensures that stable policy optimisation is maintained in scenarios where A3C may exhibit instability.

This combination allows the testbed to balance exploration and stability, where A3C provides scalability and parallel learning, while PPO acts as a stabilising approach to mitigate policy degeneration. This design choice directly supports the research objectives related to learning robustness and performance in multi-UAV coordination.

We have used the open source Ray framework for distributed learning computing in ML. Distributed learning solves a ML problem using multiple machines. We can break heavy, computationally intensive algorithms into parts and distribute them among multiple processors. This method enables us to do highly computationally demanding tasks in the least amount of time. Using Ray's remote command, workers are converted into Ray actors, making executing them in parallel across multiple machines or cores easier.

4.7 Summary

This chapter presents the design and implementation of the UAV testbed, translating the methodological framework into a functional and integrated system.

Rather than treating components independently, the chapter demonstrates how simulation (FlightGear and JSBSim), learning algorithms (A2C, A3C, PPO), and reward mechanisms are combined to enable scalable and distributed swarm coordination. The architecture is designed to support parallel learning and real-time interaction, where A3C enables asynchronous updates for multiple UAVs, and PPO provides stability against policy degradation. The integration of the Intrinsic Curiosity Module and Gaussian reward shaping addresses the delayed reward problem by improving exploration and providing continuous learning signals. The chapter shows that the effectiveness of the testbed lies not only in implementing individual algorithms, but in how these elements are coordinated within a distributed framework to support cooperative behaviour. This implementation establishes the technical foundation required for evaluating swarm performance. This leads directly to Chapter 5, where the testbed is validated through experimental results to assess its scalability, learning efficiency, and tracking performance.

Chapter 5

Verification and Validation Using Experimental Results

This chapter presents the verification and validation of the testbed through the experimental results of implementing actor–critic algorithms, including A2C, A3C, and PPO. The experiments cover straight-path and complex-path target tracking scenarios involving single and multiple UAVs, thereby providing a validation of the scalability, robustness, and effectiveness of the curiosity-driven multi-agent coordination achieved in the distributed learning environment. Table 5.1 shows a comparison of the validation of previous testbeds with the testbed of this research project.

Testbed	**Graphical Results of ML Models**	**Flight Trajectory Plots**	**Aircraft Inter-Distance Plots**	**Experimental Simulation Videos**	**Hardware In Loop (HIL)**	**Real Flight Data Comparison**
GaTAC Testbed [34]	✓	✗	✗	✗	✗	✗
NUAV Testbed [6]	✓	✗	✗	✗	✗	✗
Micro-UAV Testbed [21]	✓	✗	✗	✗	✗	✗
RUAV Testbed [23]	✓	✗	✗	✗	✓	✓
AutoPilot Testbed [25]	✓	✗	✗	✗	✓	✓
AirSim Testbed [26]	✓	✗	✗	✗	✗	✗
DeepDrone Testbed [27]	✓	✗	✗	✗	✗	✓
PyFlyt Testbed [31]	✓	✗	✗	✗	✗	✗
OPEN Testbed [32]	✓	✗	✗	✗	✗	✗
REPLANNER Testbed [33]	✓	✗	✗	✗	✗	✗
Octo-rotor RL Testbed [29]	✓	✗	✓	✗	✗	✗
dSPACE Testbed [30]	✓	✗	✗	✗	✓	✗
My Testbed	✓	✓	✓	✓	✗	✗

Table 5.1: Comparison of the Testbed Developed in This Research Project with Previous Studies for Validation

Table 5.1 highlights a fundamental difference in how validation is performed in my testbed compared with earlier UAV testbeds. All prior studies, including GaTAC [34], NUAV [6], Micro-UAV [21], RUAV [23], AutoPilot [25], AirSim [26], DeepDrone Testbed [27], and other testbeds, primarily relied on graphical plots of machine learning models as their main form of validation. Flight trajectory plots, lacking in previous testbeds, are critical for validating UAV testbeds because they directly show how an aircraft moves in three-dimensional space under the control of a learned policy. Minimum–maximum distance plots, lacking in previous testbeds, provide an additional and crucial layer of validation for UAV testbeds.

Experimental simulation videos represent another major advancement in validation. In my journal publications [120],[121], the experimental simulation videos were highly encouraged. Videos provide qualitative evidence that complements quantitative plots by showing real-time system behaviour, synchronisation between agents, and responsiveness to dynamic target motion. In my research, simulation videos demonstrate coordinated swarm manoeuvres, smooth tracking of the target aircraft, and stable control under changing flight conditions. Such visual evidence is particularly important for a complex multi-UAV testbed, where emergent behaviours may not be fully captured by numerical plots. The lack of experimental simulation videos in earlier testbeds reduces

transparency and makes independent assessment and reproduction of results more challenging.

In reviewing prior UAV testbeds, it is clear that certain validation approaches adopted by other research groups were not implemented in my work. Specifically, Hardware-In-The-Loop (HIL) simulation and direct comparison with real-world flight data were not conducted. Testbeds such as the RUAV, DeepDrone, and AutoPilot platforms integrated physical hardware to validate control algorithms under realistic conditions and sensor noise constraints. DeepDrone also performed real-flight data comparison to measure simulation-to-real transfer performance. The absence of HIL integration and flight-log comparison remains a limitation in my testbed.

5.1 Experimental Validation Roadmap

This chapter presents the experimental verification and validation of the distributed and scalable testbed for swarm UAV coordination. The main purpose of this chapter is to demonstrate that the UAV testbed can learn stable flight behaviour, support cooperative target tracking, scale to multiple UAVs, mitigate delayed reward challenges, and operate in a fully distributed learning environment.

The experimental evaluation begins with the behaviour of a single untrained UAV. In Section 5.7, the untrained agent exhibits random and unstable motions due to the absence of a learned control policy. The video of an untrained agent is given in section 5.13.1. This section is important because it establishes a baseline behaviour and clearly illustrates that meaningful and stable flight control cannot emerge without learning.

Following this baseline, Section 5.8 presents the behaviour of a trained UAV. After training, the same UAV demonstrates smooth and stable flight behavior and consistent motion; the video of the trained agent is provided in Section 5.13.2. This section is crucial because it validates the effectiveness of the training process and confirms that the learning framework implemented in the testbed can successfully produce stable control policies.

Once single-agent learning is verified, the chapter advances to the target tracking scenario. Section 5.2 shows the graphical results of the target UAV using the A2C model implementation. In this experiment, a single UAV is trained to act as the target aircraft and follows a random path. This section is important because it establishes a randomly moving target behaviour, which is required for evaluating the performance of tracking UAVs in subsequent experiments. There are many use cases and applications of my UAV

testbed, but the task of target tracking was the one use case of this research work.

After implementing the ML model of the target aircraft, Section 5.3 presents the graphical results of the implementation of the A3C model to control two tracking UAVs. These tracking UAVs learn to cooperatively follow the A2C-controlled target UAV in real time. This section demonstrates the effectiveness of asynchronous learning for multi-agent coordination and validates cooperative target tracking within the testbed. The corresponding video of target tracking of this experiment is given in sections 5.13.3 and 5.13.4. We increased the number of tracking UAVs from one to two to show the scalability of our testbed. We increased the number of UAVs in other experiments depending on our computational resources.

To evaluate the scalability of the testbed, Section 5.4 extends the previous experiment by increasing the number of tracking UAVs from two to seven, all controlled using the multi-worker A3C model implementation, and the video is given in 5.13.6. This section is essential because it shows that the testbed can scale to a larger swarm size to show scalability, thereby validating the scalable design of the testbed architecture. Section 5.12 presents a large-scale distributed learning experiment in which seven tracking UAVs are deployed across seven separate machines, while the target UAV controlled by the A2C model runs on an eighth machine. This section is vital because it provides strong experimental evidence that the proposed testbed supports fully distributed learning at scale, which is one of the key contributions of this thesis. We could not increase the number of UAVs to more than seven because of our limited computational resources. However, theoretically our testbed can support an unlimited number of UAVs.

In addition to scalability, the chapter also evaluates learning robustness and delayed reward mitigation. Section 5.5 compares ICM-based A3C and PPO models with their non-ICM counterparts. This section is important because it demonstrates that the integration of ICM significantly improves exploration, increases cumulative rewards, and mitigates the delayed reward problem, resulting in faster convergence and more stable learning.

To further demonstrate the flexibility of the proposed testbed, Section 5.5 presents the implementation of the PPO model for controlling a tracking UAV. This experiment shows that the testbed is not restricted to a single learning algorithm and can support heterogeneous machine learning models within the same simulation environment. The video of this experiment is shown in Section 5.13.5. The distributed learning capability of the testbed is validated in Section 5.11, where one tracking UAV controlled by the PPO

model is executed on one machine, while two tracking UAVs controlled by A3C workers are executed on two separate machines. This section shows that the testbed can support heterogeneous machine learning models.

In summary, this chapter validates the testbed through a structured sequence of experiments, progressing from single-agent learning to cooperative tracking, scalability analysis, curiosity-driven learning, and fully distributed learning execution. Each section plays a specific role in demonstrating that the testbed is based on distributed learning, scalable, can support heterogeneous machine learning models, and is suitable for real-time swarm UAV target tracking.

5.2 Experimental Results for the Implementation of A2C Model to Control Target Aircraft

The performance of the A2C model is illustrated through multiple graphs that highlight the agent's learning behaviour. A2C is employed to control the target aircraft. Hence, the performance of the A2C model can be used to validate the performance of the target aircraft. The cumulative reward begins with low values, showing the agent's initial struggle, but gradually increases and stabilizes into a plateau, reflecting successful learning and convergence. The mean episode length fluctuates with longer curves initially during training, indicating that the agent survives longer and interacts more effectively with the environment as it improves. The actor loss, which is negative because the policy gradient objective formula is written with a minus sign to turn a maximization problem into a minimization, and decreases exponentially, showing that the policy is moving from exploration to consistent action selection. Similarly, the critic loss is fluctuating and unstable, representing difficulty in value prediction, but it steadily decreases overall as the critic network becomes more accurate. Graphical results for the implementation of the A2C algorithm are given in Figure 5.1.

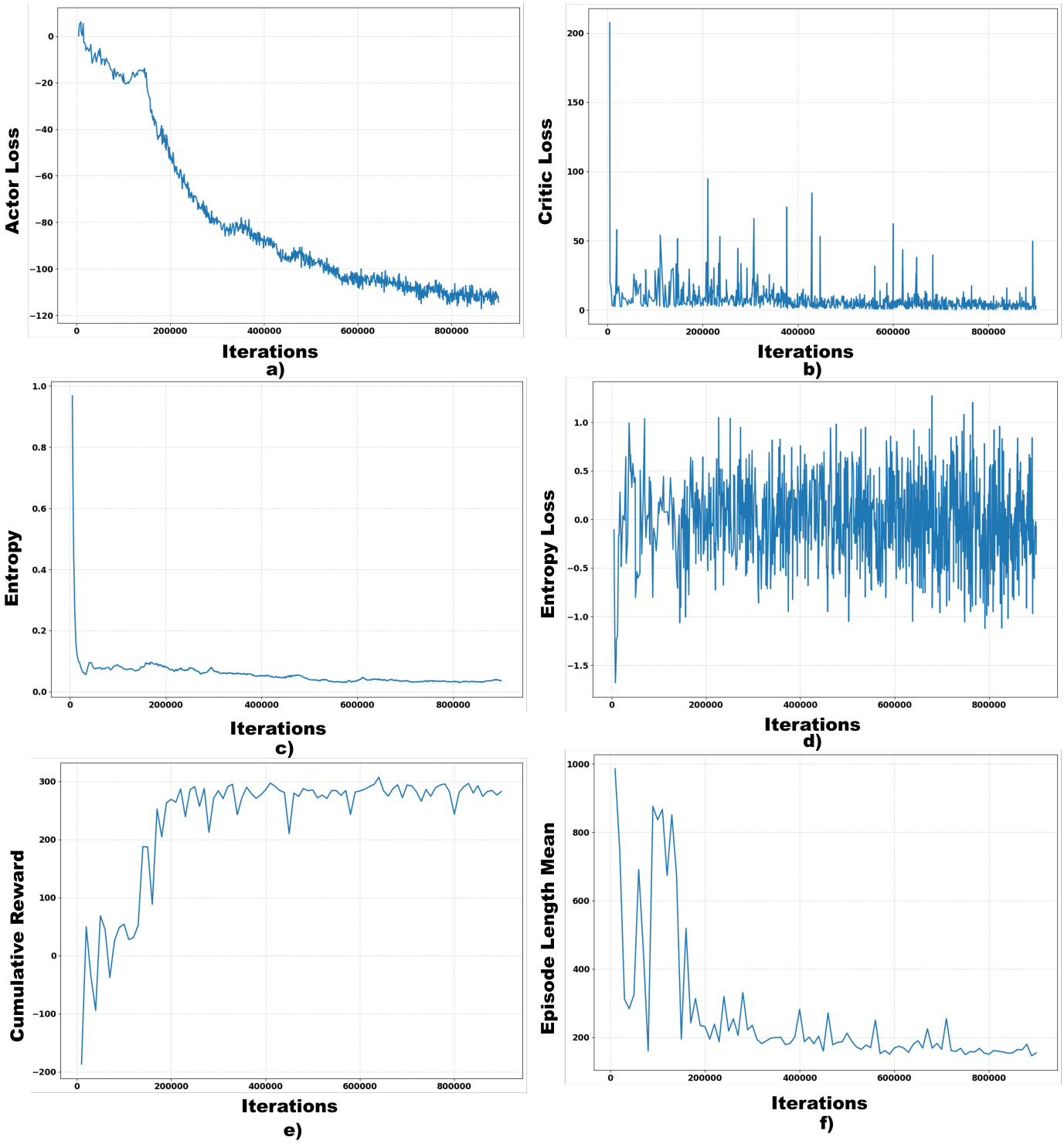


Figure 5.1: Graphical results to show the implementation of A2C algorithm a) Actor loss b) Critic loss c) Entropy d) Entropy loss e) Cumulative reward f) Episode length mean

The entropy loss, which also appears negative because its formula is defined with a minus sign to encourage exploration, shows large variations in the early phase, highlighting random behaviour, while the entropy itself follows a downward trend, confirming the policy's shift from high randomness to a more deterministic and refined strategy.

5.3 Experimental Results for the Implementation of A3C Model to Control Two Tracking Aircraft

The A3C model is implemented to control two tracking aircraft by using two local networks (workers) and one global network. Therefore, the performance of the A3C model can be used to validate the performance of a swarm of two tracking aircraft. The actor loss is a measure of how well the policy network (actor) is learning to choose optimal actions based on the observed state. It is derived from the policy gradient method and encourages the agent to take actions that lead to higher returns (rewards). Episode length mean refers to the average number of steps the agent takes in the environment per episode; higher values generally signify better performance. FPS is the step rate that measures the speed of training and reflects how efficiently the system processes the simulation steps. The cumulative reward is the total accumulated reward across iterations and is a direct indicator of the learning progress and performance of the agent. Entropy loss is used to encourage exploration by penalizing overly deterministic policies; high entropy loss values promote randomness, while low values indicate convergence. Critic loss quantifies the error between the predicted value of a state and the actual observed return. The critic network is responsible for learning the value function, which estimates how good a particular state (or state-action pair) is in terms of expected future rewards. Entropy itself measures the randomness in action selection, with high entropy indicating exploration and low entropy suggesting a more deterministic policy.

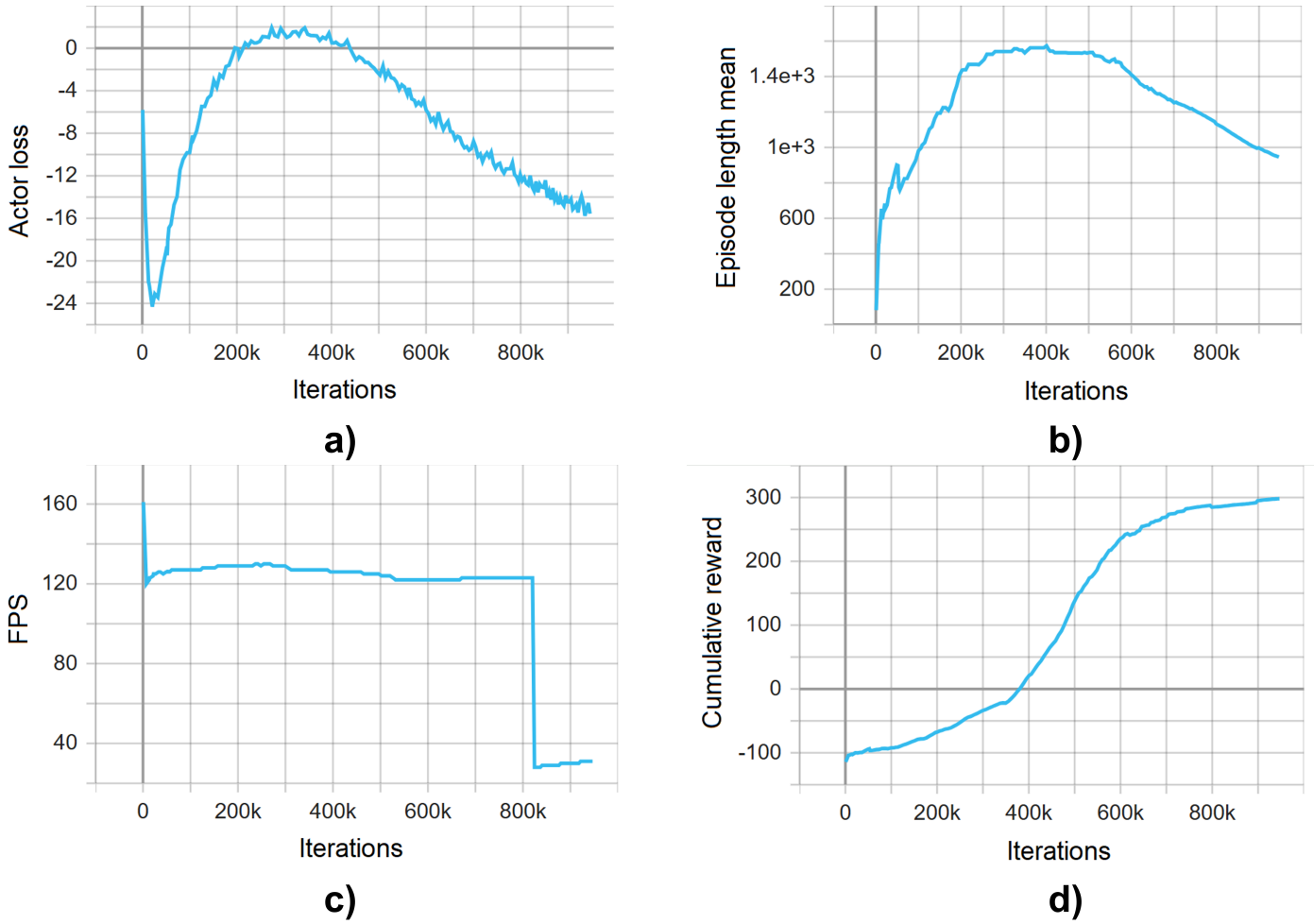


Figure 5.2: Graphical results for the implementation of A3C algorithm a) Actor loss, b) Episode length mean, c) FPS, d) Cumulative reward.

In Figure 5.2 (a), the actor loss gradually rises to a peak before slowly decreasing. This pattern reflects the agent's early exploration and the gradual stabilization of its policy over time. The actor's goal is to maximize rewards, but since ML frameworks work by minimizing a loss, the formula is written with a minus sign, which makes the values negative. The entropy term, which encourages the agent to keep exploring instead of repeating the same actions, is also written with a minus sign for the same reason. This is why both losses show as negative, and it is a normal part of the A3C framework's implemented equations. In Figure 5.2 (b), the episode length mean curve suggests the agent learns to perform well. In Figure 5.2 (c), the FPS curve shows that training speed stabilizes in the mid process but eventually drops considerably in later stages, likely due to computational overhead. In Figure 5.2 (d), the cumulative reward demonstrates a consistent upward trend, indicating successful learning and policy improvement.

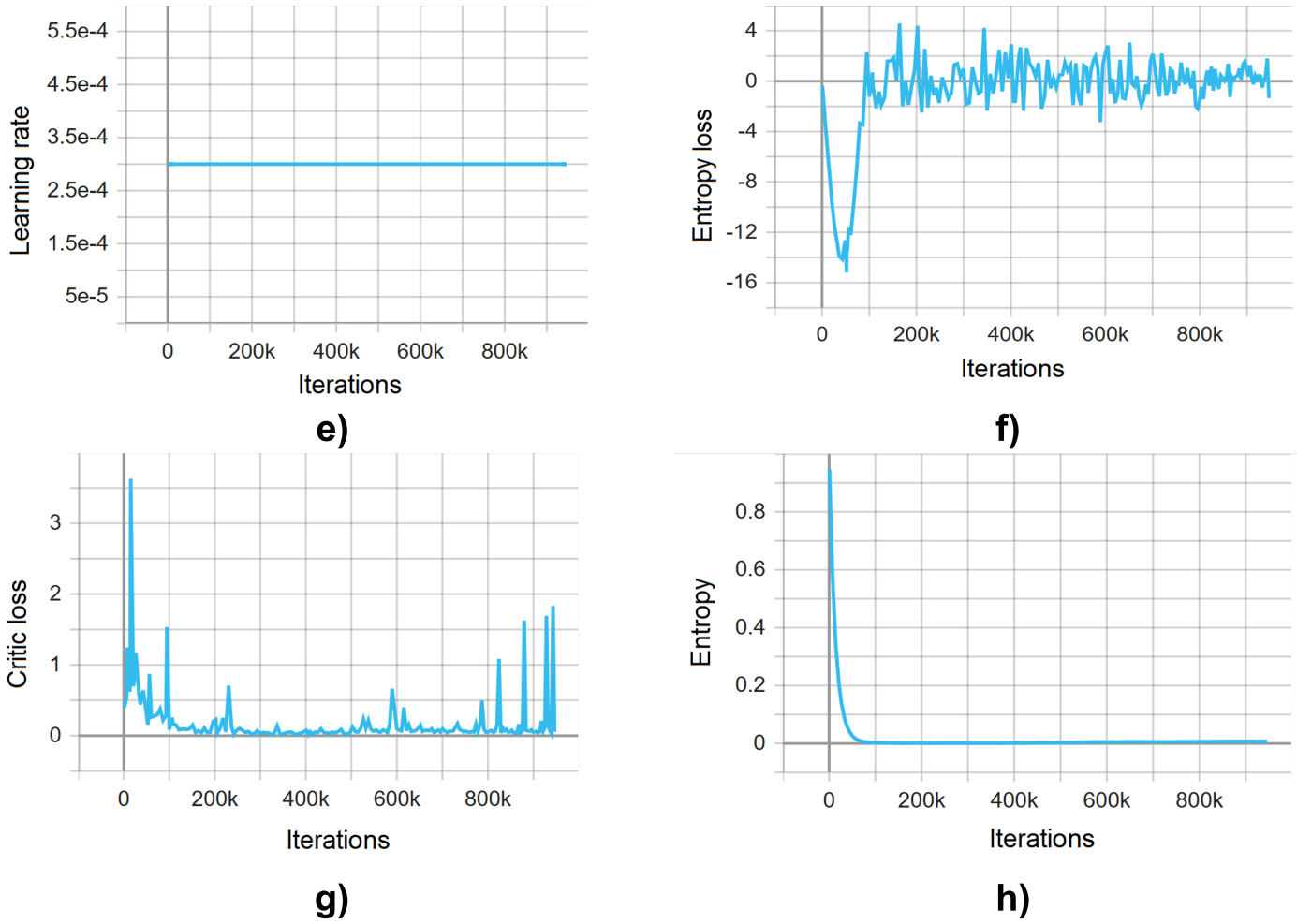


Figure 5.3: Graphical results for the implementation of A3C algorithm e) Learning rate f) entropy loss g) critic loss h) entropy.

In Figure 5.3 (e), the learning rate is an important hyperparameter because it controls how much the model updates its weights after each iteration, directly affecting the speed and stability of learning. If the learning rate is too high, the model may become unstable and fail to converge, while too low a value can make learning very slow. The entropy loss in Figure 5.3 (f) fluctuates, reflecting high randomness and exploration. The critic loss in Figure 5.3 (g) displays high variability at the beginning and end of training, with a long stretch of stability in the middle. In this critic loss plot, the values decrease sharply at the start, indicating that the critic network quickly learns to accurately estimate state values. However, the increase near the end likely indicates temporary instability or fluctuations in value estimation as the policy continues to explore. This happens because the actor (policy) changes slightly during training, and as it explores new states or actions, the critic must relearn their corresponding value estimates, causing short spikes in loss. Additionally, the critic loss can rise when the advantage estimates vary strongly or when rewards become sparse toward the end of training. These spikes are common and not

harmful if the overall trend remains low and stable, which indicates that the critic is still providing reliable value predictions while adapting to the actor's ongoing updates. Finally, the entropy graph in Figure 5.3 (h) shows a sharp decline from high to low values, confirming the transition of the agent's policy from exploratory to deterministic as training.

The interpretation of critic loss spikes has been clarified to address their relationship with UAV flight behaviour. While spikes in critic loss are generally associated with sudden changes in the environment, it is not always possible to directly link each spike to a specific flight event. This is due to the complexity of the UAV dynamics and the continuous interaction between multiple agents within the simulation environment. The flight behaviour involves simultaneous changes in position, velocity, orientation, and inter-agent interactions, which makes it difficult to isolate a single event responsible for a particular spike.

In addition, the asynchronous nature of the learning process means that updates from multiple workers occur at different time steps, further complicating the direct mapping between loss spikes and individual flight actions. However, it can be observed that spikes typically occur during periods of rapid change in the environment, such as when the target UAV follows a complex trajectory or when tracking UAVs adjust their positions to maintain coordination. These spikes reflect the model adapting to new state transitions rather than indicating instability. This clarification highlights that, due to the complexity of multi-agent UAV dynamics, critic loss spikes cannot always be attributed to a single identifiable flight event, but instead represent the combined effect of dynamic interactions within the environment.

5.4 Experimental Results for the Implementation of A3C Model to Control Seven Tracking Aircraft

The A3C model was used to control a swarm of seven tracking UAVs using seven local networks (workers) and one global network. The performance of the A3C model incorporating seven workers can be used to validate the performance of the swarm of tracking UAVs. The cumulative reward of the A3C model refers to the total amount of reward an agent accumulates over an entire episode. It serves as a direct indicator of how effectively the agent is learning and performing its task. A higher cumulative reward

suggests that the policy being followed is successful in achieving the intended goals. The mean episode length is the average number of steps taken in an episode before it ends, either by achieving the objective or hitting a terminal condition. A longer episode length may indicate sustained tracking.

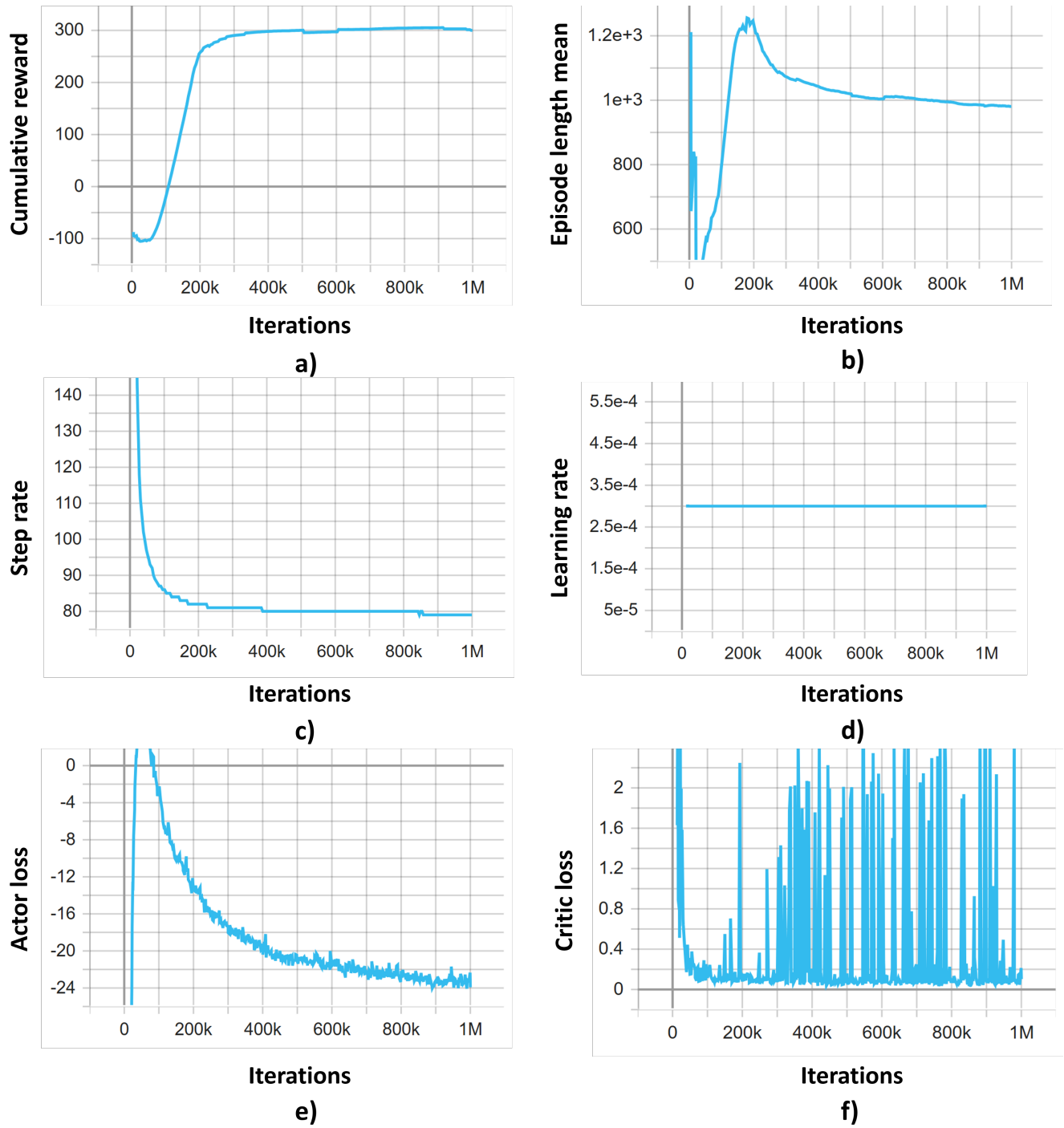


Figure 5.4: Graphical illustration of A3C performance for seven workers a) Cumulative reward b) Episode length mean c) Step rate d) Learning rate e) Actor loss f) Critic loss.

The step rate indicates how fast the training is progressing in terms of the environment steps taken per second. Entropy measures the randomness or uncertainty in the agent's

action selection process. High entropy implies that the agent is still exploring the action space, while low entropy shows that the policy has become more deterministic, favouring the exploitation of learned strategies. Actor loss quantifies the performance of the actor (policy) network. A decreasing negative actor loss indicates that the policy is learning effective actions. Critic loss measures the error between the predicted value and the actual return.

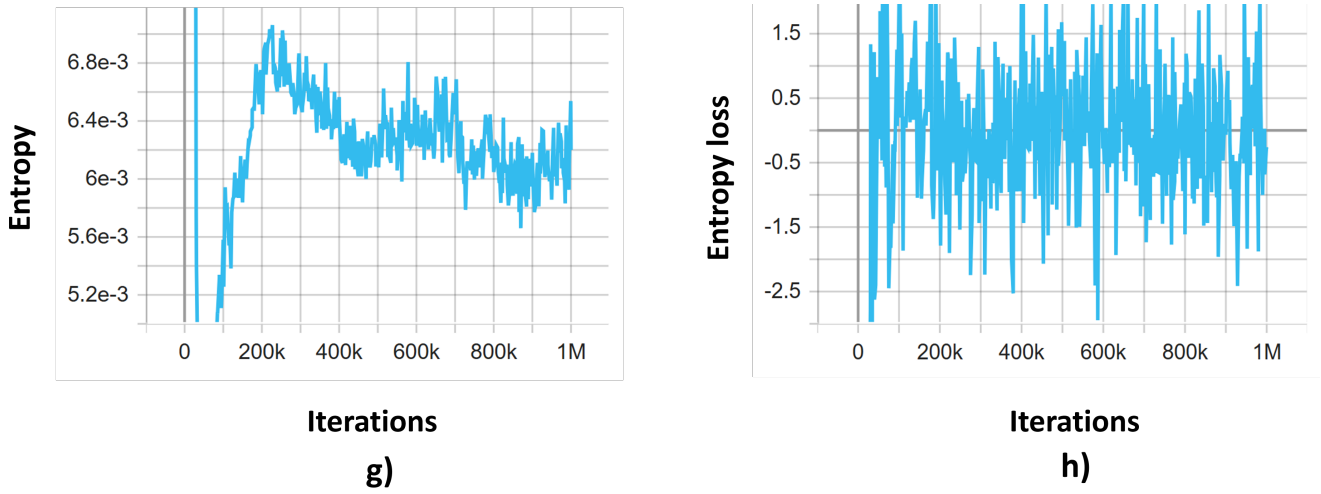


Figure 5.5: Graphical illustration of A3C performance g) Entropy h) Entropy loss.

In Figure 5.4, the curve for cumulative reward initially increases slowly, indicating that the swarm UAVs are beginning to learn effective tracking and coordination behaviours. As training progresses, the curve shows a more consistent upward trend, reflecting improved policy learning and better decision making by the A3C agents. The episode length mean curve also increases and reaches the peak value, then stays constant after decreasing slightly, which implies that the UAVs are maintaining their tracking and coordination for longer durations, avoiding early termination caused by collisions or poor manoeuvring. The step rate remains relatively stable after decreasing from the peak value, suggesting that the simulation is running efficiently and training is progressing at a steady pace. The actor loss curve generally trends downward, indicating that the policy network is learning to select better actions over time. In actor–critic methods like A3C, fluctuations in critic loss are normal. The critic's job is to estimate the value function, the expected future reward for constantly changing policies produced by the actor. Because the actor keeps updating its behaviour during training, the data distribution seen by the critic also keeps changing

In Figure 5.5, the entropy curve starts at a high value, representing a high degree of randomness in the agents' action selection as they explore different strategies. Over

time, this curve decreases and then ultimately fluctuates, indicating that agents are shifting from exploration to exploitation, relying more on the learned policy to make consistent decisions. Correspondingly, the entropy loss curve shows a larger fluctuation pattern. This overall behavior across both figures demonstrates the effectiveness of the A3C-based swarm coordination framework in achieving efficient learning for multi-agent UAV tracking.

The results of the two-tracker and seven-tracker A3C experiments show consistent learning behaviour and confirm the scalability of the system. In both cases, the cumulative reward increases steadily during training, showing that the agents successfully learn effective tracking strategies. The episode length also grows and then stabilizes, indicating improved control and longer, more stable flight paths as training progresses. The actor loss decreases over time, demonstrating better policy convergence, while the critic loss shows natural fluctuations due to continuous policy updates but remains within a stable range, reflecting reliable value estimation. The entropy decreases gradually in both experiments, signifying that the agents transition from high exploration to focused and deterministic behaviour as they gain confidence in their learned policies. Overall, the similar trends across the two-tracker and seven-tracker results confirm that the A3C model maintains stable learning performance and coordination even when scaled up to a larger number of agents in the distributed learning testbed.

5.5 Experimental Results for the Implementation of PPO Model to Control the Tracking Aircraft

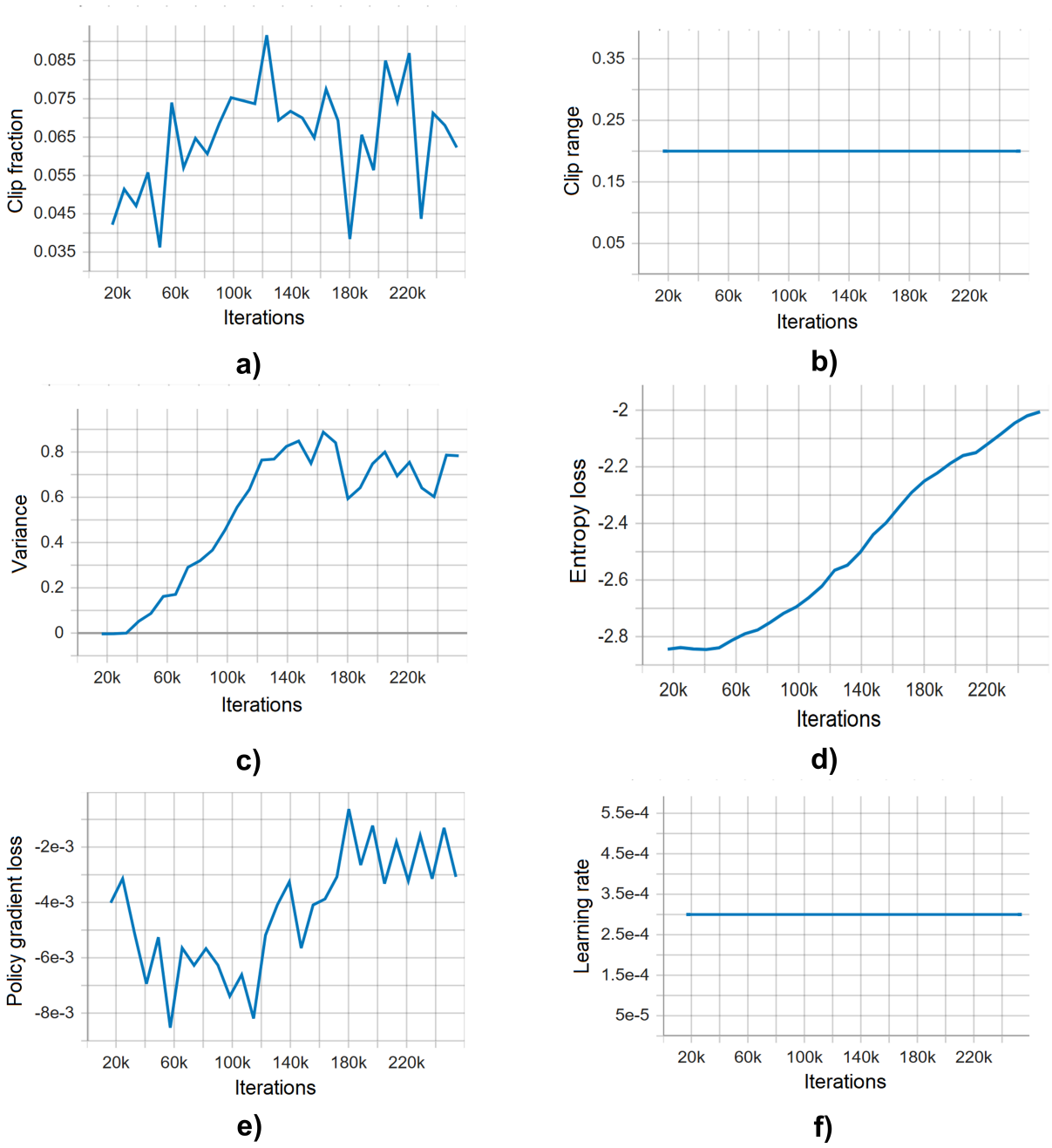


Figure 5.6: Graphical results of PPO algorithm a) Clip fraction b) Clip range c) Variance d) Entropy loss e) Policy gradient loss f) Learning rate.

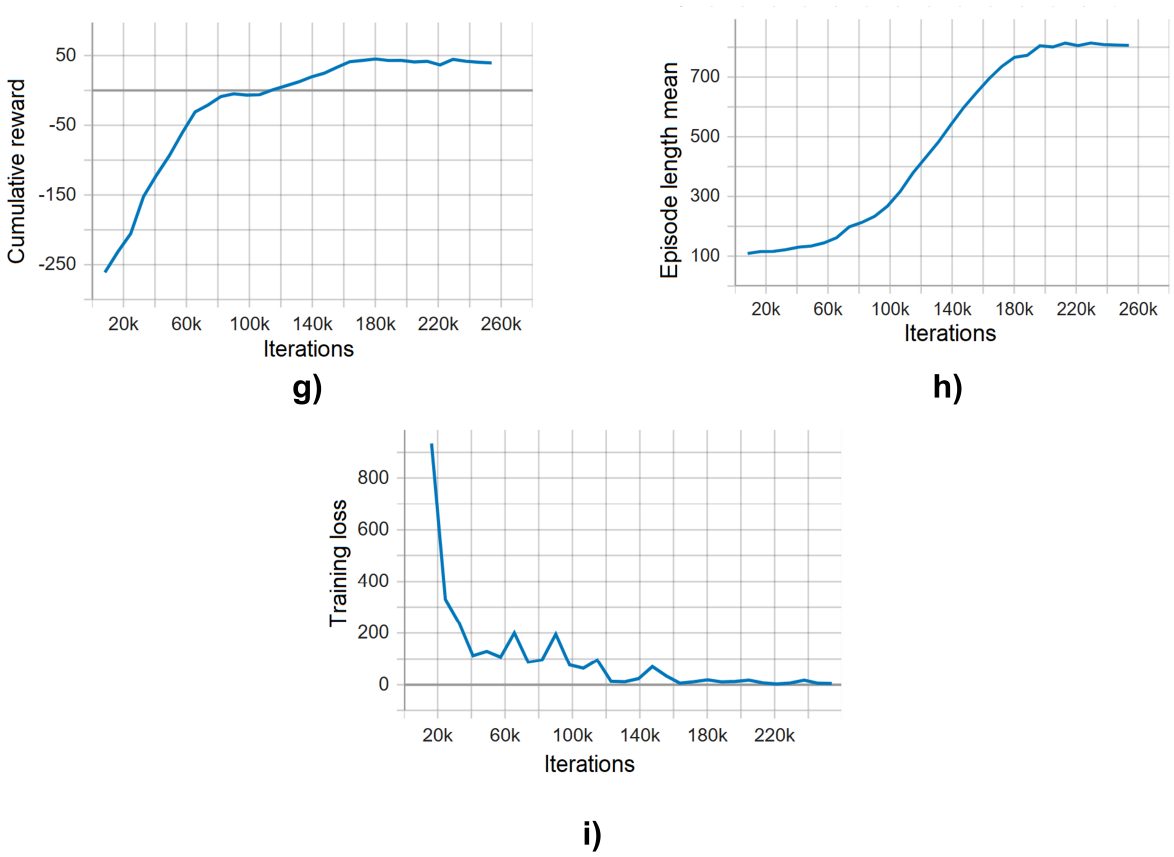


Figure 5.7: Graphical results of PPO algorithm g) Cumulative reward h) Episode length mean i) Training loss.

The PPO model is used to control one tracking aircraft. Therefore, the performance of the model can be used to validate the performance of the tracking aircraft. The term variance in Figure 5.6 refers to the percentage of variability in an agent's predictions. The clip fraction in Figure 5.6 is the fraction of times the clipping mechanism is triggered during PPO training. In PPO, the policy update ratio is clipped within a specified range to prevent excessively large updates. This ensures stable and controlled learning by limiting how much the policy changes in a single update. The clip range in Figure 5.6 is the range within which the policy's update ratio is clipped to stabilize learning. It constrains the policy update, preventing drastic changes while allowing sufficient learning progress. Variance is the measure of how well the value function predicts rewards. It indicates the accuracy of the value function as a predictor of future rewards. Entropy loss is the penalty added to encourage exploration by increasing the randomness of actions. It is calculated based on the entropy of the policy's action distribution. It prevents the policy from converging prematurely to a deterministic strategy, especially in early training. Policy gradient loss is the loss component related to the policy update in PPO. It's derived from the advantage function and governs how the policy is adjusted to improve

performance. The episode length mean is the average length of an episode across all agents during training. The cumulative reward in RL represents the sum of rewards an agent accumulates over time while interacting with an environment. It helps evaluate the agent's performance, with higher cumulative rewards indicating better decision-making strategies.

In PPO, the policy gradient loss optimizes the policy using a clipped surrogate objective to prevent large updates. The training loss in PPO includes policy gradient loss, value function loss (for the critic), and entropy loss (to encourage exploration). This combination ensures stable learning by balancing policy improvement, value estimation, and exploration. Clip range in PPO limits how much the new policy can deviate from the old one, preventing instability. Clip fraction is the measure of the proportion of updates where the policy change exceeds this limit. A high clip fraction means frequent clipping, which may slow learning, while a low clip fraction suggests smoother updates. The clipped surrogate objective function and entropy regularization are core components of PPO that inherently prevent policy degradation. Unlike A2C and A3C, which suffer from large, unstable updates, PPO is designed to ensure controlled policy improvement, preventing drastic shifts that could lead to degeneration [83]. By stabilizing training dynamics and ensuring incremental policy updates, PPO provides a reliable mechanism for maintaining policy consistency and preventing collapse, making it highly effective in RL applications.

The PPO training curves show a clear trend of convergence and stabilisation. The cumulative reward increases steadily and then plateaus, while the episode length mean also rises, indicating that the agent survives longer as learning improves. The training loss drops sharply at the start and then stabilises, confirming convergence as shown in Figure 5.7. The clip fraction fluctuates within a controlled range, reflecting stable clipping behaviour. Variance increases initially and then settles as the policy converges, while entropy loss increases significantly as shown in Figure 5.6, showing reduced exploration and a more deterministic policy over time. The policy gradient loss oscillates around small values, which means the policy is being updated with small and stable corrections rather than large changes.

5.6 Comparison of ICM Based A3C and PPO Models with Non-ICM Based RL Models

This section presents the performance of the A3C and PPO models with and without Intrinsic Curiosity RL. Integrating ICM with PPO and A3C helps overcome the delayed reward problem by generating intrinsic rewards that encourage exploration even when external rewards are sparse. This leads to faster learning, better policy stability, and prevents the agents from getting stuck in repetitive or suboptimal actions. In Figure 5.8 a), the A3C-ICM curve shows a steady upward trend, indicating that the agent is learning effectively and consistently improving. The A3C curve exhibits fluctuating progress and grows irregularly. ICM allows the agent to learn faster and more consistently, even when external (environmental) rewards are sparse or delayed. In Figure 5.8 b), the A3C-ICM curve grows gradually and consistently, showing more efficient behaviour over time. The A3C graph shows the shorter average episodes, indicating less stable strategies. ICM helps the agent stay longer in episodes when needed for exploration. In Figure 5.8 c), the A3C-ICM actor loss decreases smoothly, indicating the agent is making better decisions over time. This reflects the dynamic exploration encouraged by the ICM. Unlike the non-ICM version, which shows a sharp drop indicative of premature convergence, the ICM variant maintains adaptive policy updates, leading to more robust long-term performance. The A3C graph shows a sudden drop in values, representing weak policy learning. With ICM, the actor can adjust its actions more effectively, guided by internal motivation.

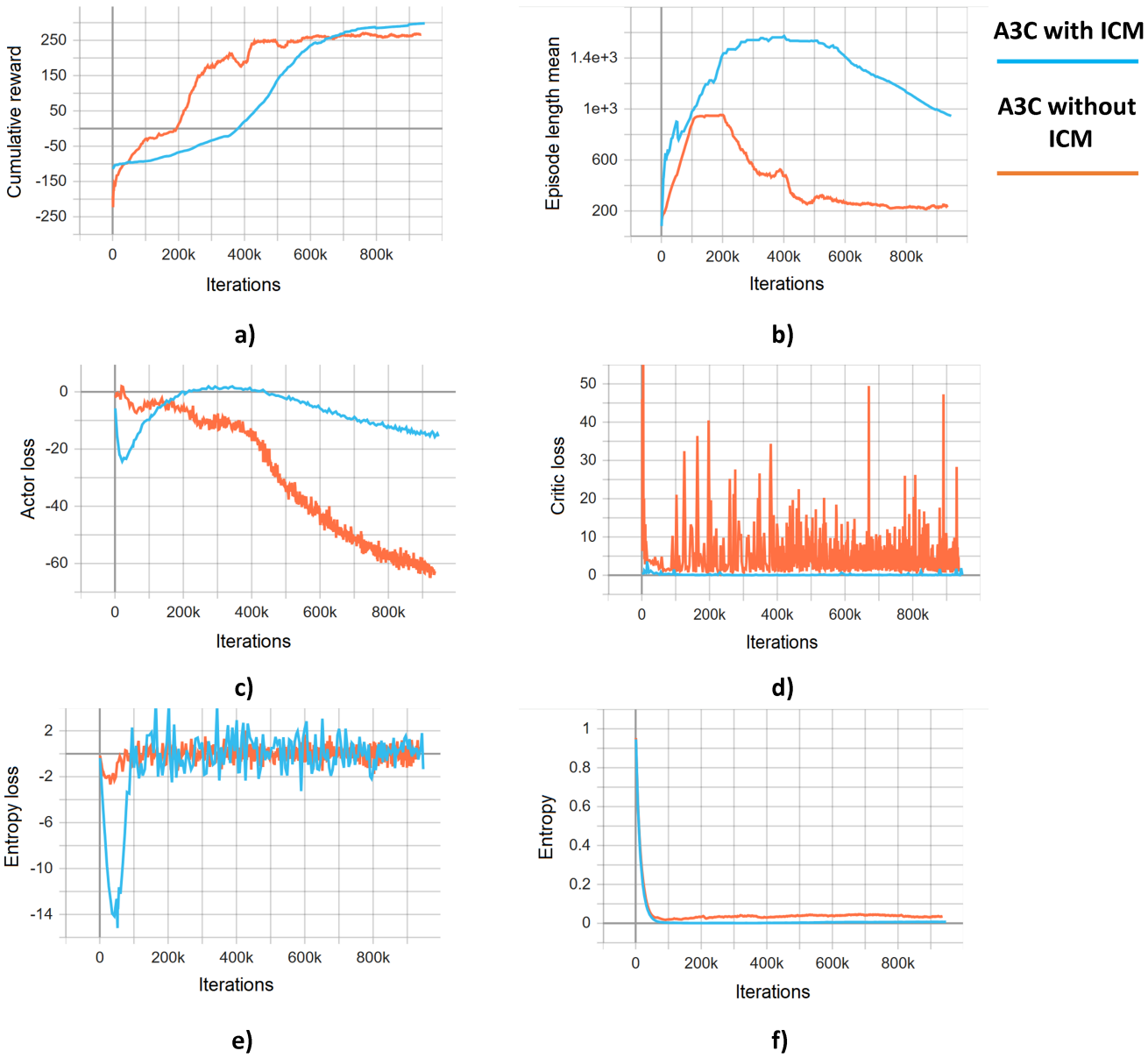


Figure 5.8: Graphical results to show the comparison of performance between A3C with ICM and A3C without ICM a) Cumulative reward b) Episode length mean c) Actor loss d) Critic loss e) Entropy loss f) Entropy.

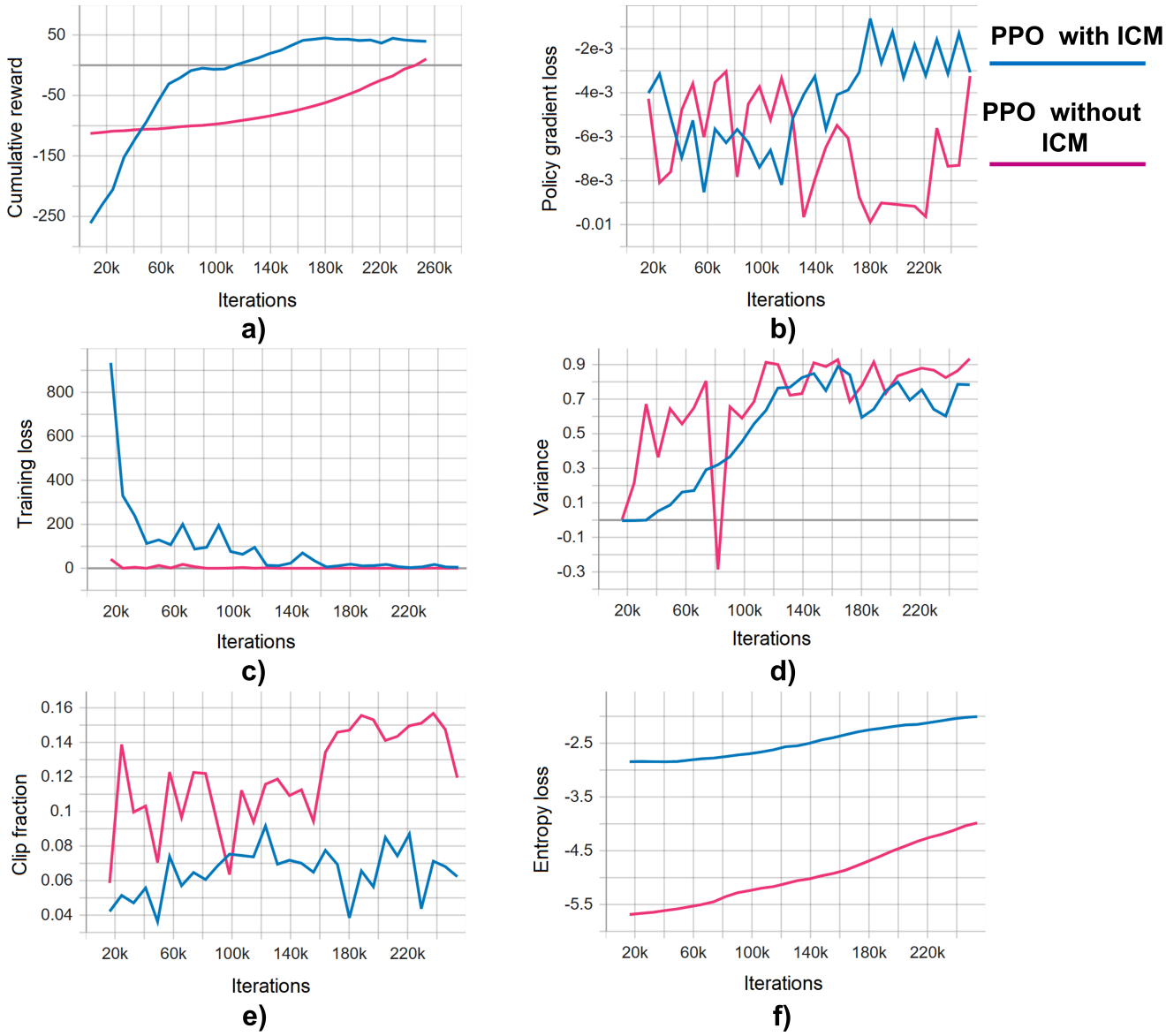


Figure 5.9: Graphical results to show the comparison of performance between PPO with ICM and PPO without ICM a) Cumulative reward b) Policy gradient loss c) Training loss d) Variance e) Clip fraction f) Entropy loss.

In Figure 5.8 d), the A3C-ICM curve shows a stable curve but remains lower than the A3C graph. The A3C curve represents high variation and unstable learning. ICM helps the critic learn more accurate state values, leading to better decision-making by the actor. In Figure 5.8 e), the entropy loss of A3C-ICM shows more randomness for ongoing exploration. The A3C curve shows less exploration in the behaviour of the agent. In Figure 5.8 f), the entropy graphs for both models show similar behaviour.

In Figure 5.9 a), the curve of the PPO-ICM rises sharply and levels off at a higher value, indicating faster and better learning. However, the curve of PPO rises slowly and remains lower throughout. ICM helps the PPO agent achieve higher and faster learning performance in sparse-reward environments. In Figure 5.9 b), the graph of PPO-ICM

shows minor fluctuations and then gradually increases, whereas the PPO curve is more volatile and dips deeper. ICM keeps policy updates more controlled and stable, improving learning reliability.

In Figure 5.9 c), the PPO-ICM graph starts high but drops and then stays consistent, whereas the PPO curve starts low and remains flat. In Figure 5.9 d), the PPO-ICM curve increases gradually, and the PPO graph also starts from the same point but fluctuates more. ICM helps PPO maintain more consistent predictions, which supports stable learning. In Figure 5.9 e), the curves of both PPO and PPO-ICM models look similar. However, the PPO-ICM curve stays low while the PPO graph remains higher, showing excessive clipping. In Figure 5.9 f), the plots of both models show similar behaviour, but the PPO-ICM curve remains high throughout as compared to the PPO curve. This indicates sustained exploratory behavior encouraged by intrinsic curiosity, preventing premature policy convergence and improving long-term learning robustness compared to standard PPO, which exploits earlier and may stagnate.

In the next section, the behaviour of the untrained aircraft is discussed during flight with the help of figures.

5.7 Behaviour of an Untrained Agent

One key aspect of RL is that the agent does not have to be aware of the dynamics of the environment beforehand [122]. For example, the learning agent can use prior knowledge of a flying vehicle in flight control. Even without knowledge about the weight of the RL agent (aircraft), it will still be able to determine how to gather rewards.

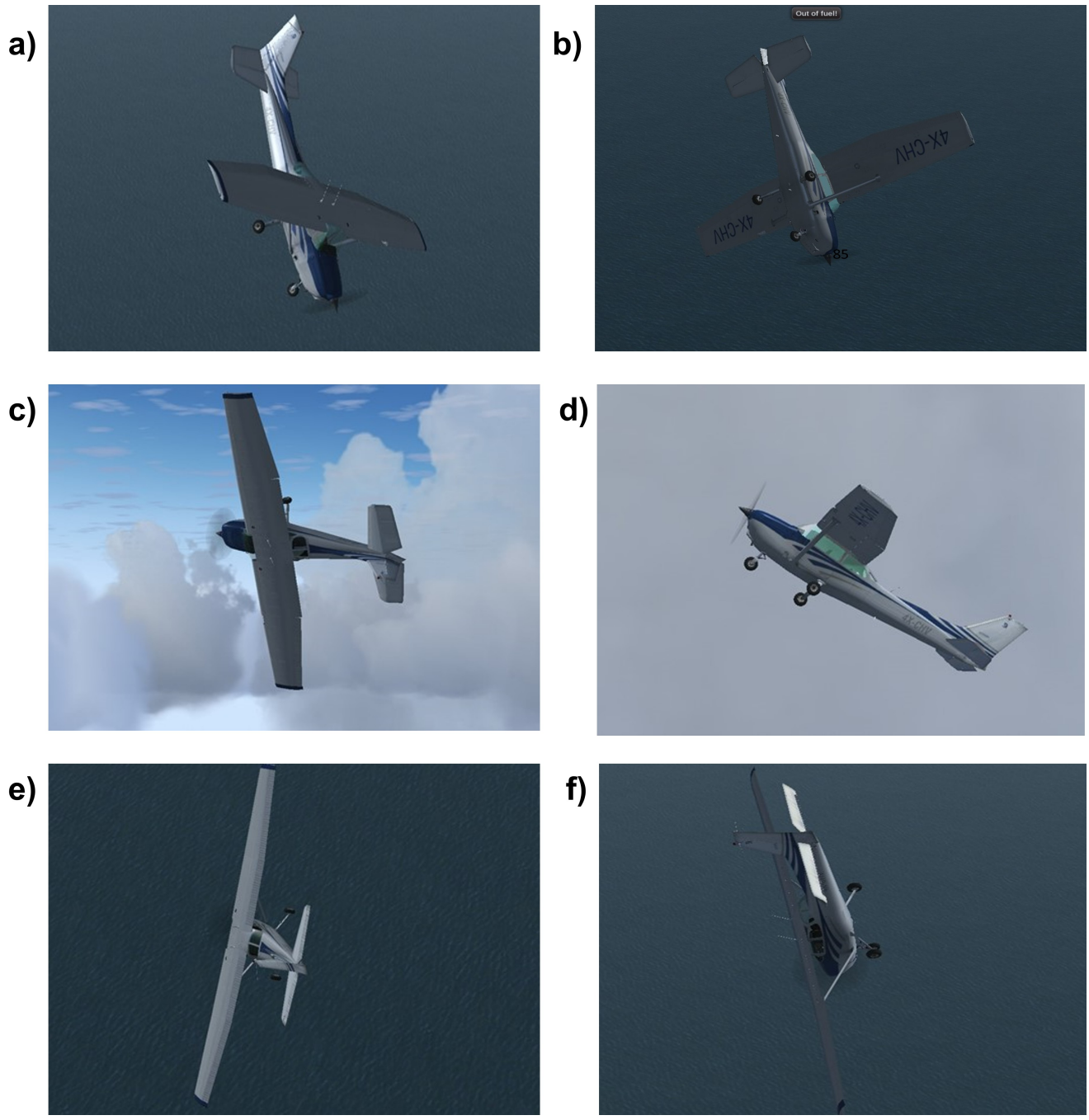


Figure 5.10: Behaviour of an untrained agent performing random motions. a) Aircraft performing pitch motion in the downward direction. b) Aircraft performing a combination of pitch and yaw motions and inverted. c) Aircraft performing a combination of roll and yaw motions. d) Aircraft performing pitch motion in the upward direction. e) Aircraft performing a combination of roll and yaw motions. f) Aircraft is inverted.

The RL agent was specifically trained for the control mission, which is target tracking and involves exploration. When the aircraft is untrained, it will perform random behaviours like random roll, pitch, and yaw motions as shown in Figure 5.10.

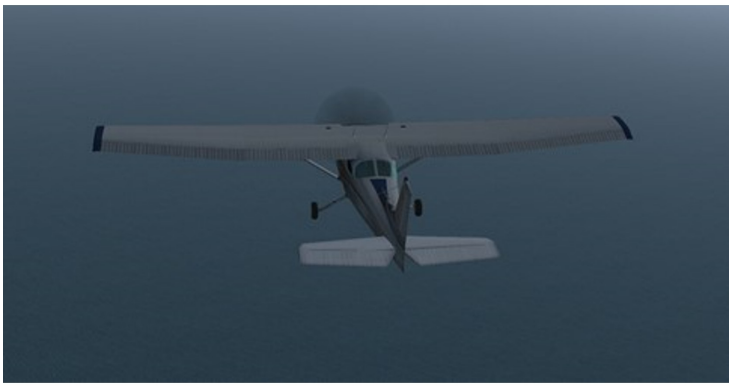
In the following section, the training of RL algorithm was completed, and the behaviour of the trained aircraft during the flight was observed.

5.8 Behaviour of Trained Agent

The trained RL agent exhibits smooth, stable, and coordinated flight motions after learning optimal control policies through RL. Unlike the untrained agent's random and unstable behavior, the trained model performs precise pitch, roll, and yaw motions while maintaining balance, stability, and altitude, as shown in Figure 5.11. This stability confirms that the learning process effectively minimized oscillations and improved control efficiency, resulting in consistent trajectory tracking.



a)



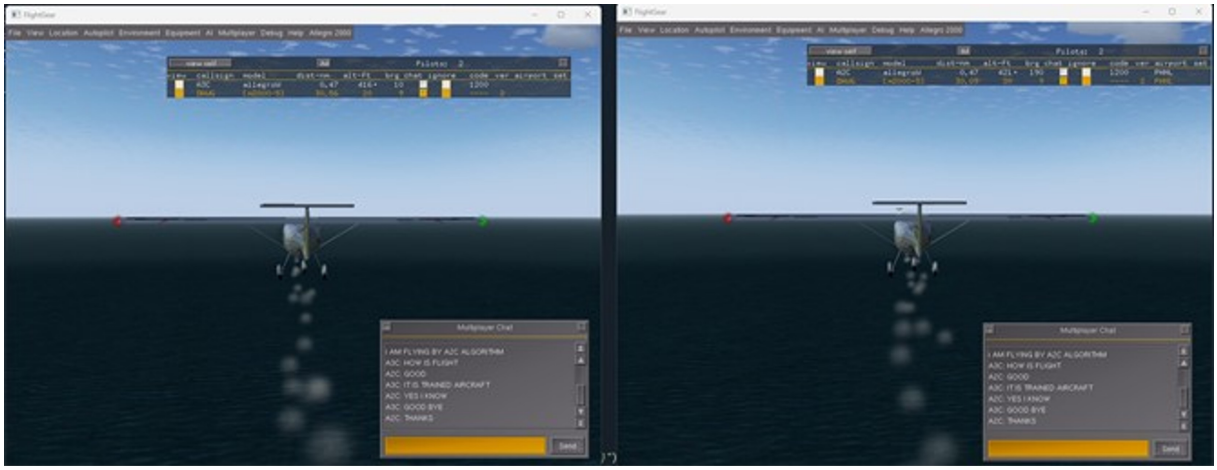
b)

Figure 5.11: Behaviour of trained agent performing stable motions. a) The aircraft is turning right. b) The aircraft is turning left.

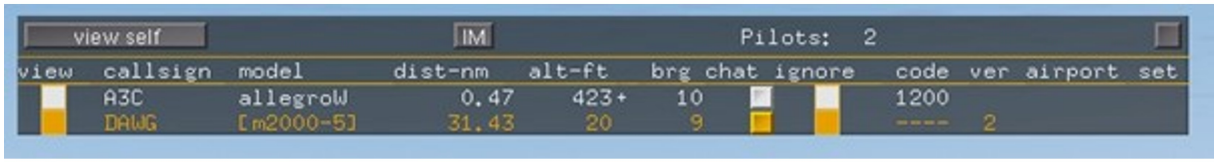
The following section is based on target tracking of UAVs.

5.9 Tracking of A2C Aircraft by A3C Aircraft in a Straight Path

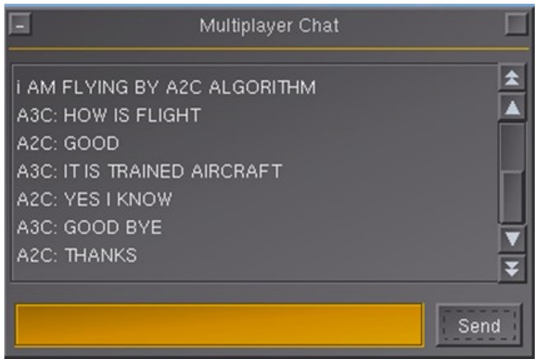
In the straight-path target tracking scenario, the A3C-controlled tracking UAV successfully follows the A2C-controlled target UAV. Both aircraft maintain a steady formation, demonstrating accurate positional alignment. This shows that the A3C agent effectively learned to interpret the target's motion dynamics and respond in real time, ensuring smooth and reliable tracking behavior. Figure 5.12 shows the target tracking in FlightGear, and Figure 5.13 validates the target tracking in FlightGear's built-in map.



a)



b)



c)

Figure 5.12: Target tracking. a) Two windows showing the target and tracking aircraft. b) Console showing the list of pilots. c) Console showing the chatbox.

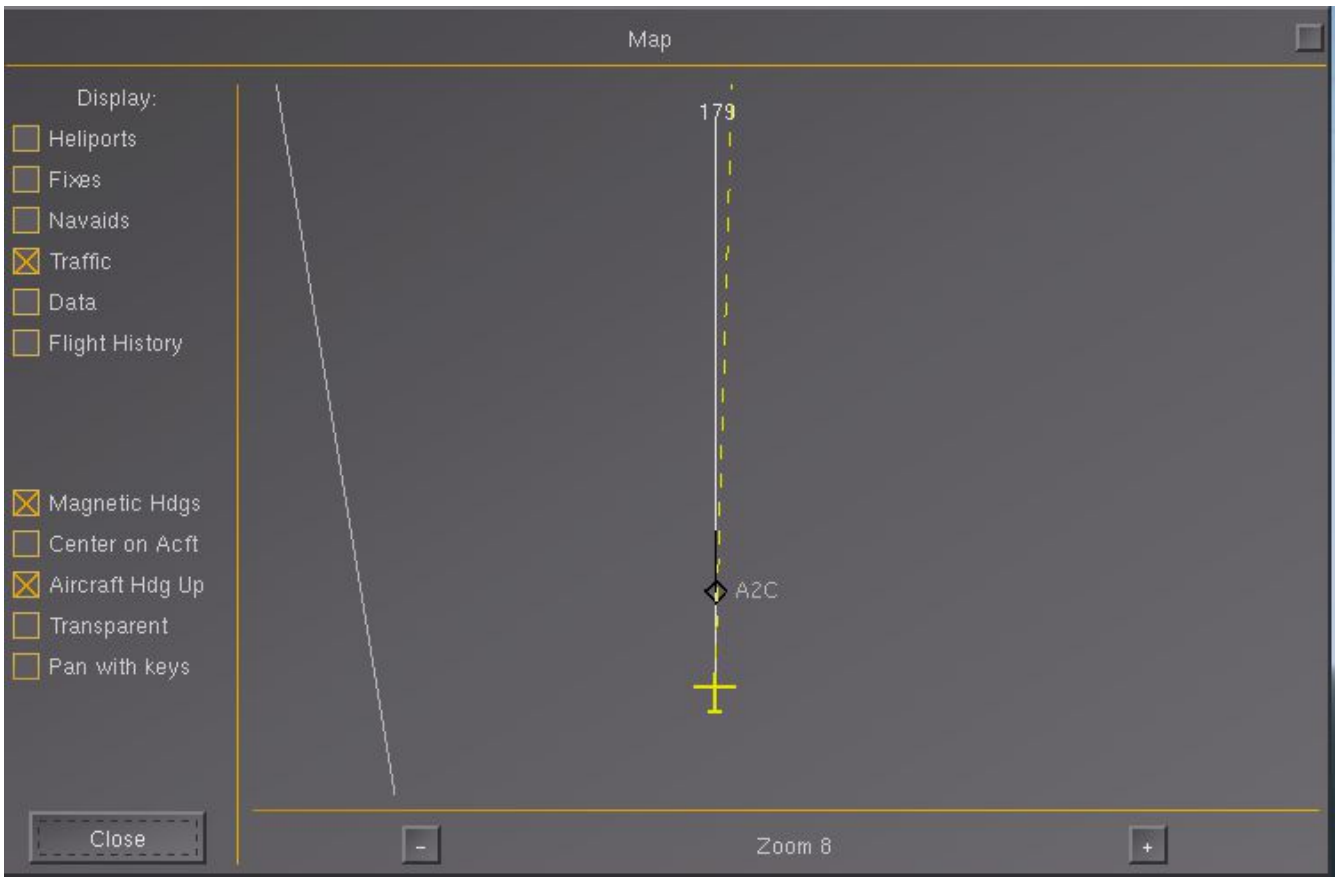


Figure 5.13: Flightgear built-in map demonstration.

5.10 Tracking of A2C Aircraft by the Swarm of Two A3C Aircraft in a Complex path

The asynchronous learning architecture of the A3C algorithm makes it highly effective for handling complex and dynamic tasks, such as UAV target tracking. In this study, we utilized the multiplayer feature of FlightGear, which allows real-time interaction among multiple users. This feature supports collaborative or adversarial flight scenarios by allowing pilots to observe and respond to each other's actions within a shared virtual airspace. The target UAV is controlled using the A2C algorithm, enabling it to follow a random trajectory. This setup creates a challenging target for the trackers, as the movement is not predefined but rather evolves based on learned policies. The two tracking UAVs operate under the control of two workers of an A3C model integrated with the ICM, which encourages the trackers to explore effectively even when external rewards are sparse or delayed. This helps the agents to stay engaged in goal-directed behaviour by generating internal rewards. This is particularly useful in dynamic path scenarios where the feedback is not immediate. This swarm-based UAV target tracking

task involves coordinating multiple tracking agents to follow a single target UAV in real-time. Figure 5.14 illustrates the FlightGear map interface used to visualize the real-time execution of the tracking scenario. Figure 5.15 demonstrates different motion patterns of the tracking aircraft as they respond to the manoeuvres of the target UAV.

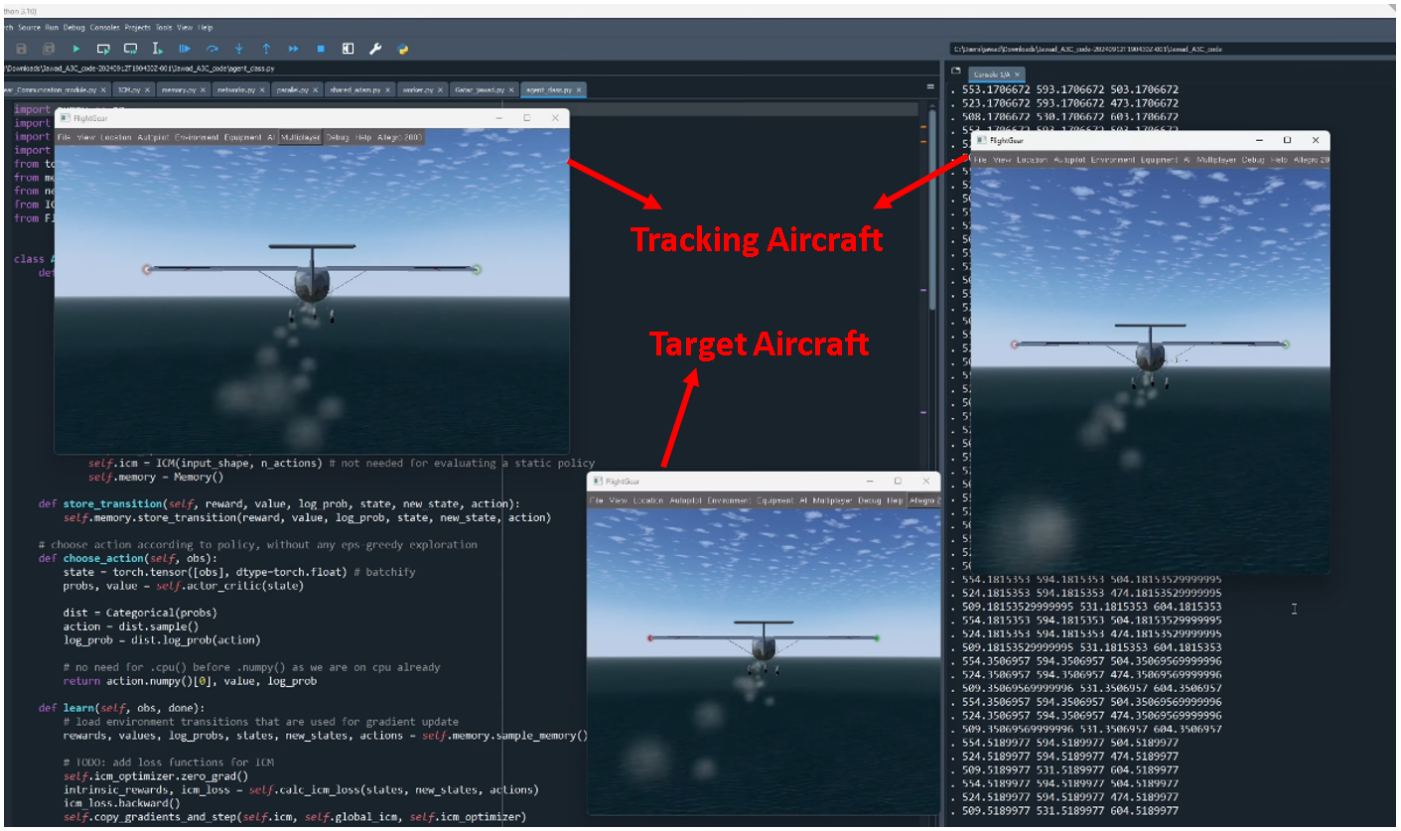


Figure 5.14: Swarm UAVs to UAV target tracking demonstration on a) Python editor and b) FlightGear map.

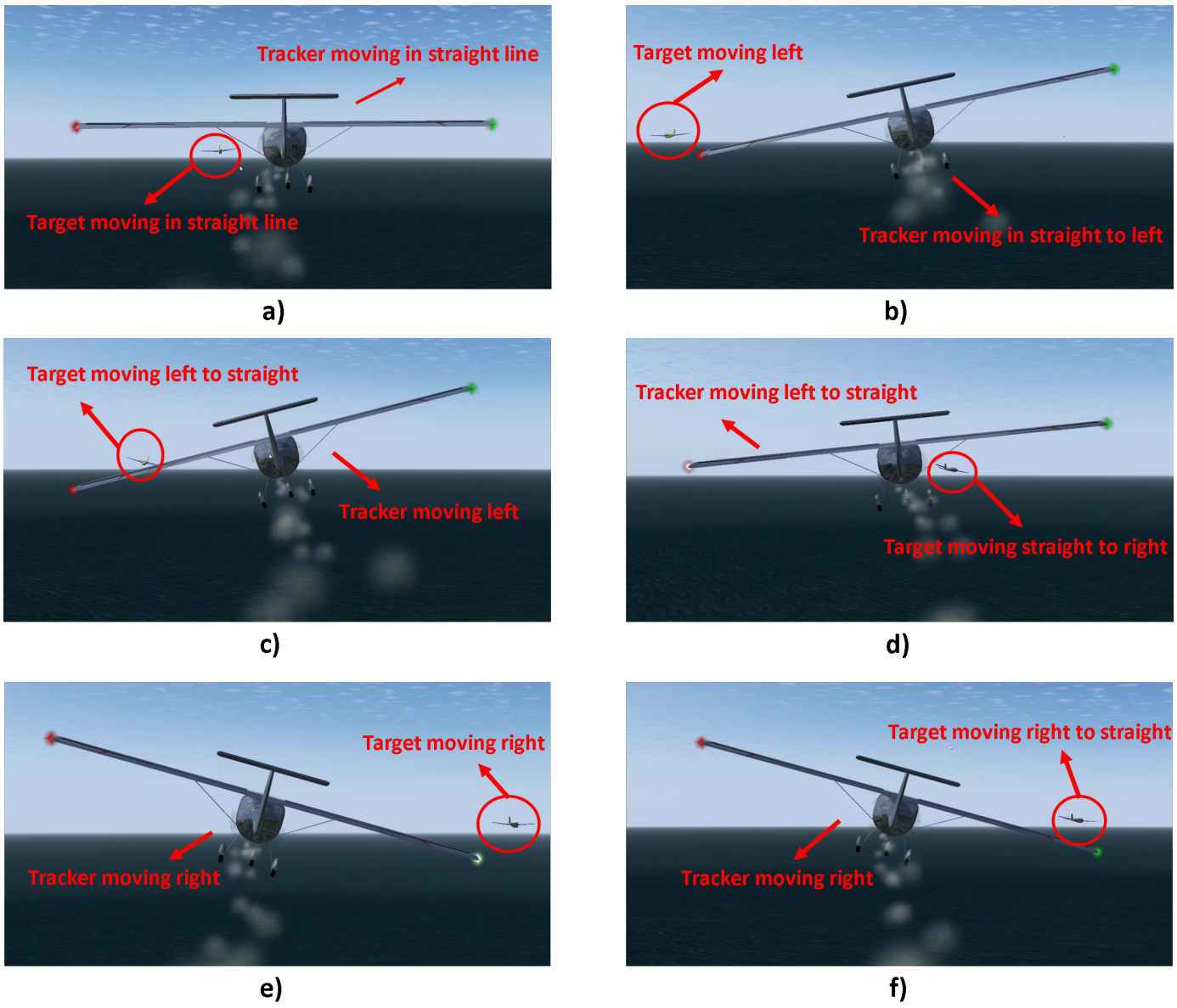


Figure 5.15: UAV to UAV target tracking a) Target and tracker moving straight line b) Target and tracker turning left c) Target turning right and tracker is turning from left to right d) Target is turning right and tracker is turning from straight to right e) Target and tracker are turning right f) Target is turning straight from right and tracker is turning right.

5.11 Tracking of A2C Aircraft by the Swarm of PPO and A3C Aircraft in a Complex Path in the Distributed Learning Setup

This experiment demonstrates swarm-based UAV target tracking using a distributed learning RL architecture that combines the strengths of the PPO and A3C algorithms. The A2C-controlled target UAV follows a complex, continuously adaptive path

generated from its learned policy. The tracking swarm consists of three UAVs: one controlled by the PPO model and two controlled by A3C workers (worker-1 and worker-2), all interconnected through a distributed learning network built using UDP sockets and the Ray framework. This architecture enables real-time synchronization and communication between multiple systems, where each UAV operates on a dedicated machine. The global A3C network acts as the coordinator, receiving gradients from workers, updating global parameters, and redistributing them across clients to maintain policy consistency. Figure 5.16 shows the real-time target tracking and 5.17 shows the map. The first tracking aircraft is controlled by the PPO model. However, the second and third tracking aircraft operate based on worker-1 and worker-2 of the A3C model interconnected via distributed learning networking.



Figure 5.16: Demonstration of swarm of UAVs Real-time in FlightGear

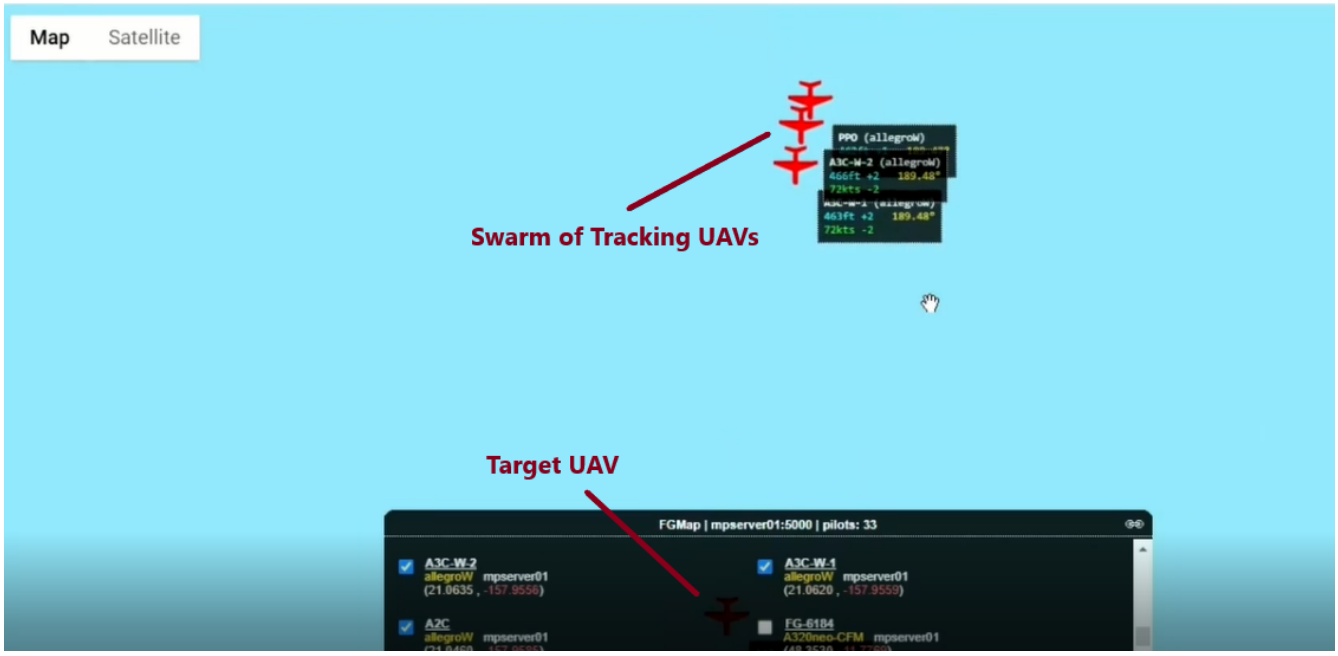


Figure 5.17: The Swarm of Tracking UAVs following the Target in FlightGear Map

Figure 5.17 shows the Swarm of Tracking UAVs following the Target in FlightGear Map.

The interaction between PPO and A3C agents in the distributed testbed is best understood as complementary rather than hierarchical. The PPO-controlled UAV exhibits more stable and conservative behaviour due to its clipped policy updates, which reduce variance and prevent abrupt changes in action selection. In contrast, the A3C-controlled UAVs demonstrate more exploratory behaviour as a result of their asynchronous updates and intrinsic curiosity-driven learning. There is no explicit mechanism for one agent to "lead" another, as each UAV operates with its own policy and learning process. However, an implicit behavioural interaction emerges through the shared environment. The PPO agent tends to follow smoother and more consistent trajectories, which can act as a stabilizing reference within the swarm. Meanwhile, the A3C agents explore a wider range of state–action spaces, allowing them to adapt to dynamic target movements and discover diverse tracking strategies. Over time, this results in a form of cooperative equilibrium, where the PPO agent contributes stability and the A3C agents contribute exploration. This balance improves the overall robustness of the swarm in complex tracking scenarios without requiring direct coordination or shared optimization between the algorithms.

5.12 Tracking of A2C Aircraft by a Swarm of Seven A3C Aircraft in a Complex Path

The seven tracking UAVs are controlled by seven workers of the A3C algorithm, which utilizes asynchronous updates and parallel processing to optimize decision making and ensure real-time coordination. The A3C algorithm allows each UAV to learn from its local observations and experiences while contributing to a global policy, enabling the swarm to operate efficiently and to allow it to adapt to dynamic environments. The target UAV, controlled by the A2C algorithm, exhibits intelligent and dynamic behavior, introducing realistic challenges for tracking UAVs. The RL testbed incorporates a reward system that encourages desirable behavior such as reducing the distance to the target, maintaining swarm formation, and avoiding collisions, while penalizing inefficiencies or deviations from the objective. Figure 5.18 shows the map, and Figure 5.19 shows the aircraft in FlightGear with a pilot list.

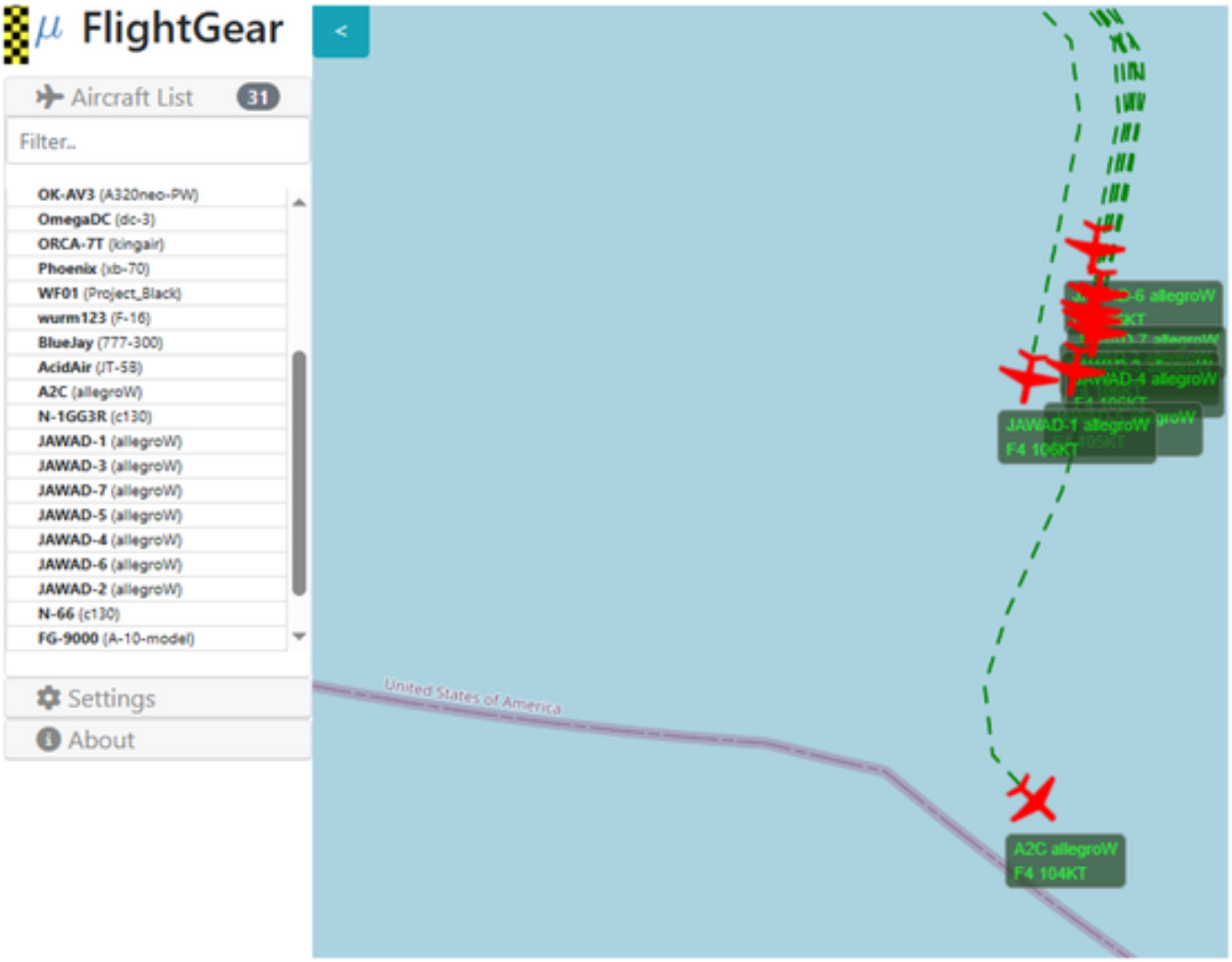


Figure 5.18: FlightGear Built-in map demonstrating the Swarm of 7 Tracking UAVs following the target

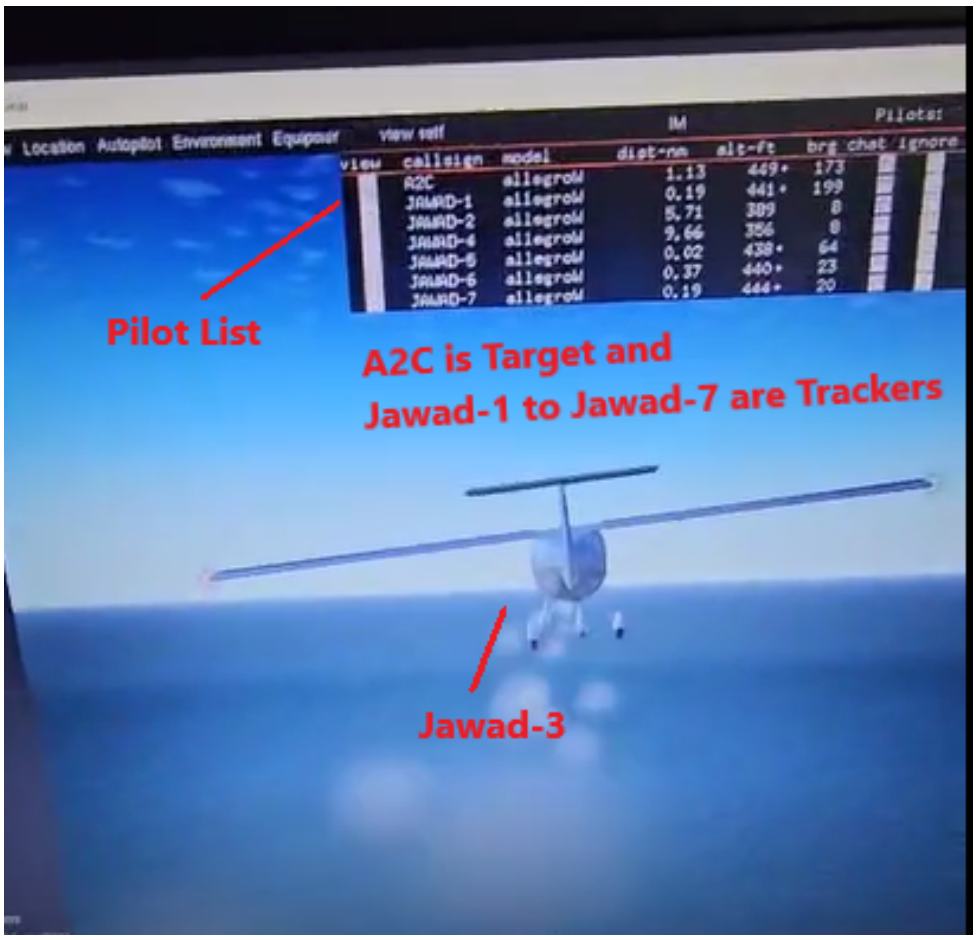


Figure 5.19: FlightGear Pilot List

The 3D trajectory plots of the one target aircraft controlled by the A2C algorithm and seven tracking aircraft controlled by seven workers of the A3C algorithm are shown in Figure 5.20. It illustrates the spatial trajectories of all UAVs, including the dynamically manoeuvring target UAV controlled by A2C, and the seven tracking UAVs governed by the distributed learning A3C-ICM model. The plot demonstrates the capability of the tracking agents to follow the target UAV closely over time while maintaining safe separation and coordinated movement.

The plots showing the distance between UAVs are given in Figure 5.21, which are interesting as they show the minimum and maximum distance between any UAV in the swarm and its nearest neighbour. Figure 5.21 a) is the min. max. plot of the trajectory plot in Figure 5.20 a). Likewise, Figure 5.21 b) corresponds to the trajectory in Figure 5.20 b), and Figure 5.21 c) corresponds to the trajectory in Figure 5.20 c). We use them to quantify the performance of the swarm concerning target tracking. The plots present the minimum and maximum inter-agent distances recorded during the mission, offering a quantitative view of swarm cohesion and safety. The plots also

illustrate that the minimum distances between the UAVs never fall below the collision threshold, demonstrating effective collision avoidance throughout the trajectory. Simultaneously, the maximum distances remain within acceptable bounds, indicating that the UAVs do not drift too far from one another or from the target. These patterns confirm that the Gaussian ellipsoidal reward function not only prevents collisions but also enforces spatial coherence among agents, enabling effective target tracking and group coordination under the multi-objective learning strategy. The minimum distance between two aircraft can be used to assess if there is a possibility of a collision in the swarm. This is computed as the minimum distance between two neighbouring aircraft. On the other hand, the maximum distance between two neighbouring aircraft is a measure of the swarm's cohesiveness, as well as how well the target is being tracked.

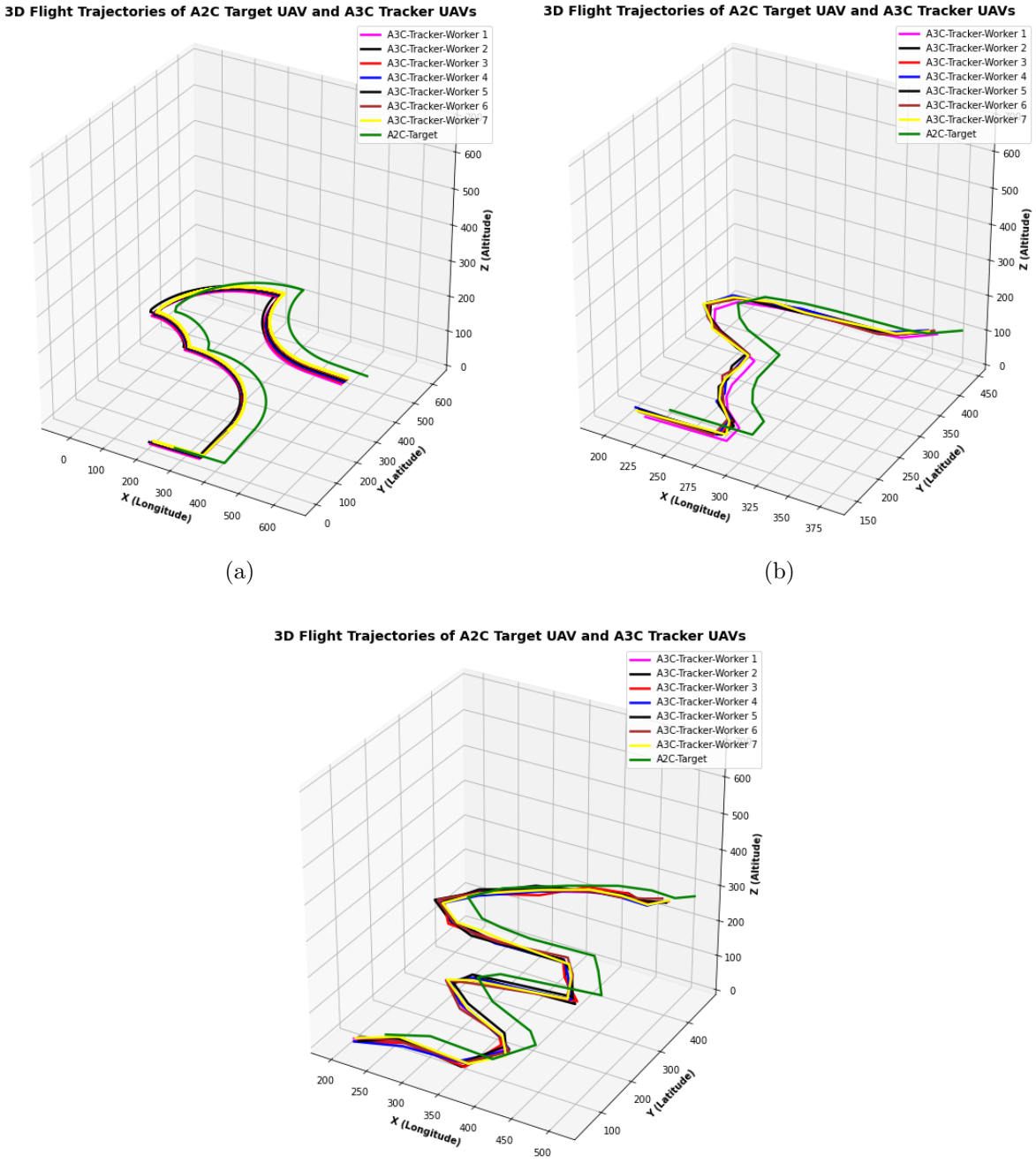
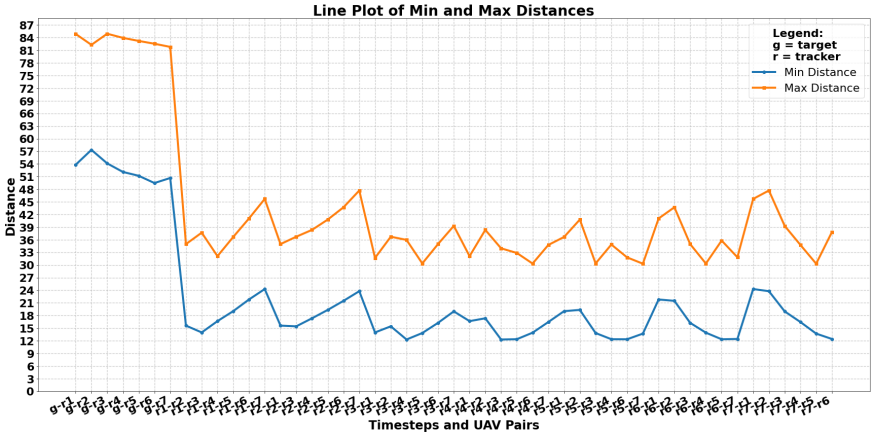
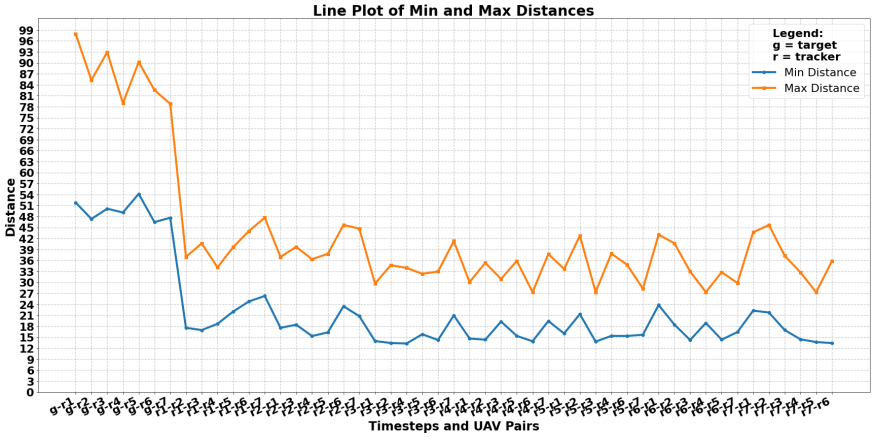


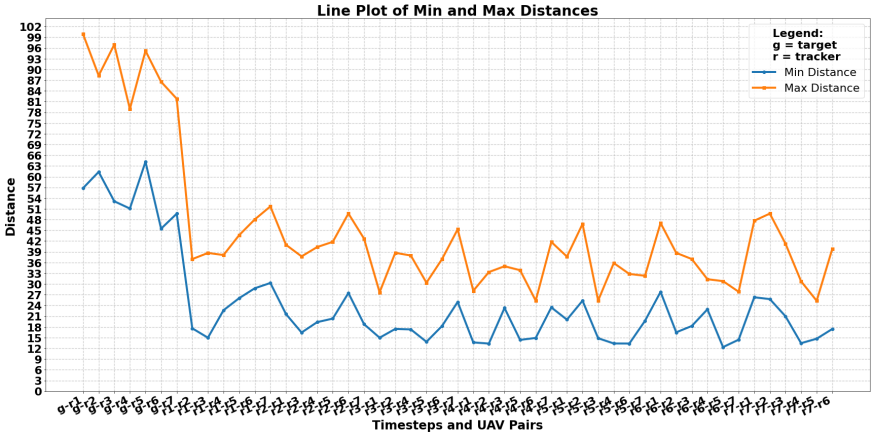
Figure 5.20: Various flight trajectories.(The units are in meters and are in coordinates.)



(a)



(b)



(c)

Figure 5.21: Min. max. distance (meter) plot of various flight trajectories.

The scalability of the proposed testbed is demonstrated in the thesis through a combination of architectural design and experimental validation. In Chapter 5, scalability is demonstrated through experiments involving increasing numbers of UAVs, progressing from a two-UAV configuration to a maximum of seven-UAV configuration. The results show that key learning metrics, such as cumulative reward, episode length,

and actor–critic losses, maintain stable convergence behavior even as the number of agents increases. For example, the cumulative reward continues to increase and stabilize, while entropy decreases, indicating successful learning and policy convergence in multi-agent settings. Additionally, the Frames Per Second (FPS) metric provides indirect evidence of scalability constraints, where a reduction in training speed is observed due to increased computational load as more agents are introduced. This reflects the expected overhead associated with scaling distributed learning systems. However, it is important to clarify that the experiments were not designed as a controlled scalability benchmark. A key reason for this is that the architecture of the experiments evolved as the number of UAVs increased, rather than maintaining a fixed experimental configuration. Specifically:

1. In smaller-scale experiments (e.g., two UAVs), a homogeneous learning setup using A3C workers was employed.

2. In larger-scale distributed scenarios (e.g., three UAVs across multiple machines), heterogeneous algorithms were introduced, where different UAVs were controlled using different learning models (e.g., PPO-ICM and A3C-ICM).

3. In the seven-UAV configuration, multiple A3C workers were used to control each UAV independently within a shared global framework.

This variation in experimental architecture was intentional and reflects the goal of demonstrating the flexibility and adaptability of the distributed testbed, rather than performing a strict like-for-like scalability comparison. As a result, the experiments highlight that the system can support:

1. Multiple agents

2. Multiple learning algorithms

3. Multi-machine distributed execution

But they do not isolate scalability as a single controlled variable. Revised Future work will focus on conducting controlled scalability experiments where:

1. The same algorithm and environment are maintained

2. The number of UAVs is incrementally increased (e.g., 2, 5, 10, 20+)

Such an approach will enable a more rigorous and comprehensive validation of the scalability claims. The scalability of the testbed is supported by its distributed architecture and experimental results, where stable learning behaviour is observed as the number of UAVs increases. The testbed is architecturally designed to support high-density swarms through distributed learning across multiple machines. However, it is acknowledged that the empirical validation is limited to a seven-UAV configuration. Therefore, this work should be considered a proof-of-concept demonstration of the distributed learning framework rather than a full-scale scalability evaluation. Given the availability of more resources, this testbed and the learning mechanism can clearly be scaled up, as it is limited only by the computing power available to the testbed.

The testbed is described as a distributed multi-agent system with emergent coordination, rather than a fully decentralised swarm with explicit inter-agent communication. In the current architecture, each UAV operates as an independent learning agent that optimises its own reward function. The UAVs share 3D coordinates and velocity with each other. Specifically, the reward function is designed to encourage behaviours such as:

1. Maintaining proximity to the target aircraft
2. For tracking trajectories
3. Avoiding divergence from the desired formation

As a result, coordinated behaviour emerges naturally because all agents are optimising similar objectives within the same environment. This is supported by the experimental results in Chapter 5, where multi-agent tracking scenarios demonstrate stable formation and consistent target-following behaviour across multiple UAVs. For example, the seven-UAV experiment shows coordinated tracking with stable inter-agent distances and convergence of learning metrics.

Instead, the contribution of this work lies in demonstrating that coordinated multi-UAV behaviour can emerge in a distributed learning framework using reinforcement learning and shared environmental feedback.

5.13 Experimental Simulation Videos of This PhD Project

The experimental simulation videos of the project are uploaded to the Open Science Framework (OSF). The Digital Object Identifier (DOI) links are given below. Access the files tab and click on the MP4 video link to watch the video.

5.13.1 Experimental Simulation Video of the Untrained Agent

In the initial experiment, the untrained RL agent was deployed to test the environmental response and system integration. As observed in the simulation, the aircraft exhibited unstable behaviour, frequent oscillations, and uncontrolled movements due to the absence of learned policy weights. This untrained phase was essential for validating the communication between the FlightGear simulator, JSBSim flight dynamics, and the RL framework before introducing trained policy models. The video demonstrates the raw, exploratory behaviour of the agent without reward feedback or stable control, confirming that the system could capture real-time state transitions and reward feedback signals.

`https://doi.org/10.17605/OSF.IO/VJSRQ`

5.13.2 Experimental Simulation Video of the Trained Agent

Once the RL model was trained through multiple episodes, the agent achieved a stable control policy, resulting in smooth navigation and altitude control. The simulation video illustrates significant improvement over the untrained case, where the aircraft follows a trajectory with steady heading, pitch, yaw, throttle, and roll stability. The trained agent effectively minimises flight oscillations and maintains balance, demonstrating convergence of the actor–critic learning process. This phase validated that the implemented model successfully learned the flight control policy and produced excellent performance.

`https://doi.org/10.17605/OSF.IO/UYQFS`

5.13.3 Experimental Simulation Video of Target Tracking in a Straight Path

In this experiment, the target UAV controlled by the A2C model flew along a straight trajectory, while tracking aircraft controlled by A3C-ICM followed it in real time. The video captures accurate tracking of the target path, with the tracking agent successfully aligning positions while maintaining distance and orientation. The results validate the testbed's ability to achieve coordinated flight. The straight-path tracking case also served as a baseline for later complex path scenarios, showing that all aircraft could maintain synchronisation without collision or divergence.

`https://doi.org/10.17605/OSF.IO/UNTER`

5.13.4 Experimental Simulation Video of Target Tracking in a Complex Path by the Swarm of Tracking UAVs

This simulation demonstrates the adaptability of the testbed in a complex flight scenario. The A2C-controlled target aircraft followed a complex trajectory, while two A3C-ICM tracking aircraft pursued it dynamically. The recorded video shows that both tracking agents efficiently learned to predict and respond to the target's motion, adapting to altitude changes and abrupt turns. The integration of the ICM enhanced exploration and improved responsiveness, enabling the swarm to maintain cohesion and reduce cumulative tracking error. This experiment highlights how curiosity-driven intrinsic rewards accelerate convergence and stabilize policy learning under complex flight conditions.

`https://doi.org/10.17605/OSF.IO/EGH6J`

5.13.5 Experimental Simulation Video of Target Tracking of the Distributed Learning Testbed

The video demonstrates full swarm coordination under a distributed learning multi-machine configuration. In this setup, one A2C-controlled target aircraft is tracked by three agents, one using PPO-ICM and two using A3C-ICM, each running on a separate interconnected system within the distributed learning testbed. The Ray framework enables parallel execution, synchronised learning, and fault-tolerant coordination. The video shows the swarm maintaining formation during complex and

dynamic flight, confirming the scalability and robustness of the distributed learning design. The experiment proves that the system can support real-time multi-agent tracking across multiple computing nodes while preventing policy degeneration and maintaining convergence.

https://doi.org/10.17605/OSF.IO/RXG3U

5.13.6 Experimental Simulation Video of Target Tracking of Seven A3C Workers Controlled Aircraft

The target tracking task is demonstrated in the experimental simulation video. The video shows that the seven aircraft controlled by seven A3C workers are tracking one target aircraft controlled by an A2C algorithm.

https://doi.org/10.17605/OSF.IO/B2CX8

5.14 Summary

This chapter evaluates the implemented testbed through experimental validation, focusing on learning performance, stability, and swarm coordination. The results demonstrate that trained agents achieve stable and effective target tracking, while untrained agents exhibit random and inefficient behaviour, confirming the effectiveness of the learning framework. The integration of the Intrinsic Curiosity Module improves exploration and helps overcome the delayed reward problem, leading to faster convergence and more robust policies compared to non-ICM models. The comparison between A3C and PPO further shows that while PPO provides stable updates, A3C performs more effectively in the distributed and asynchronous setting of the testbed. The experiments also confirm that the testbed can scale from single-agent to multi-agent scenarios, including swarm configurations, while maintaining coordination and tracking performance. This demonstrates that scalability and distributed learning are successfully achieved in practice. Overall, the chapter concludes that the effectiveness of the testbed arises from the combined impact of distributed learning, curiosity-driven exploration, and reward shaping. These results provide the basis for deeper interpretation, which is discussed in the next chapter.

Chapter 6

Discussion and Conclusion

This chapter discusses the results obtained from the implementation and experimental validation of the testbed. The aim is to connect the findings with the research questions and objectives, highlighting how the outcomes address the limitations identified in earlier studies. The discussion revisits the research questions and objectives, outlines the specific contributions of the testbed, and reflects on the broader significance of the work. The limitations of my testbed are also discussed in detail. This chapter demonstrates that the goals of this thesis have been achieved and establishes a foundation for future developments in distributed learning swarm UAV coordination using ML.

6.1 Research Problem and Research Questions Revisited

This research aimed to address a fundamental gap in the existing UAV testbeds: the absence of a scalable, distributed learning, and realistic testbed capable of evaluating intelligent multi-agent behaviours under dynamic flight conditions. Existing testbeds are often centralised, limited in scope, and do not support meaningful scalability or realistic flight dynamics. This thesis set out to overcome these limitations by designing, implementing, and validating a distributed learning testbed that integrates high-fidelity flight simulation with modern ML techniques to enable cooperative swarm behaviours. The overarching objective was to develop a testbed that not only supports multiple aircraft operating across multiple machines but also allows intelligent agents to learn, coordinate, and interact in complex aerial scenarios.

This section revisits the main research question and each Research Question (RQ) and its corresponding Research Objective (RO), linking them directly with the detailed analysis, design decisions, and experimental results presented throughout this thesis. By connecting the findings from Chapter 2, Chapter 4, and Chapter 5, this section demonstrates that each research task was accomplished, providing a sound foundation for the development of a distributed learning and scalable UAV testbed.

Main Research Question: "How can a distributed and scalable UAV testbed be developed to design intelligent control models for cooperative, coordinating, and self-organizing swarms of UAVs?"

This research demonstrates that such a testbed can be designed and validated by following the principles of the DSR methodology described in Chapter 3. The problem identification stage of DSR highlighted major limitations in earlier testbeds such as NUAV, RUAV, and GaTAC, including their centralised architectures, lack of distribution, difficulty in scaling beyond a few agents, and challenges such as delayed reward and policy degeneration. Guided by these gaps, the design objectives were established to ensure that the new testbed would be scalable across multiple machines, support real-time multi-agent learning, and incorporate mechanisms to stabilise training.

The design and development stages of DSR, as detailed in Chapter 4, demonstrate how the testbed was engineered by integrating the FlightGear flight simulator with the physics-based JSBSim FDM and a suite of actor–critic–based learning algorithms. The implementation of A2C for the target aircraft and A3C and PPO for multiple tracking aircraft showcases how distributed learning is achieved in practice. The asynchronous architecture of A3C enables workers to explore different trajectories concurrently, which directly supports scalability, while PPO provides enhanced training stability by mitigating the risk of policy collapse. The use of the Gaussian-based ellipsoidal reward (Section 4.5) and the ICM (Section 2.6.4) further strengthens the testbed's capability to address delayed reward and improve exploration.

The evaluation stage of DSR is reflected in Chapter 5, where extensive experimental results validate the performance and scalability of the testbed. The results of A2C, A3C, and PPO models (Sections 5.2 – 5.5) reveal that trained agents achieve stable motion, improved cumulative reward trends, and successful target-tracking even in complex paths. The comparison of curiosity-driven models with non-ICM models (Section 5.6) demonstrates better convergence. The multi-agent tracking experiments involving two and seven A3C workers (Sections 5.3 and 5.4) confirm the distributed

learning nature of the testbed, while the distributed learning PPO and A3C experiments (Section 5.11) confirm that the system maintains real-time performance across networked machines. **RQ-1: What are the limitations in the current state of the art in testbeds for self-coordinating and cooperative swarms of UAVs?**

RO-1: To conduct a critical analysis of existing UAV testbeds to identify their limitations, forming the foundation for the design and development of an improved testbed.

The review of existing UAV testbeds in Chapter 2 highlighted several significant limitations in the state of the art. Previous testbeds lacked scalability (except GaTAC) and distributed learning execution capabilities. They were generally limited to single-computer or small-scale setups. Section 2.5 further showed that these testbeds did not address challenges such as delayed reward signals, sparse extrinsic rewards, and policy degeneration, all of which are critical in multi-agent environments. These limitations directly influenced the design of the new testbed. My new testbed introduces multi-machine execution, parallel learning through Ray, and reward shaping using ICM, addressing gaps identified in the literature.

A major limitation identified in the literature was the lack of scalability in most existing testbeds. Except for GaTAC [34] and AirSim [26], other testbeds such as the NUAV [6], Micro-UAV [21], RUAV [23], Open [32], Replanner [33], and DeepDrone [27] testbeds were constrained to single-agent experiments. These restrictions prevented meaningful evaluation of coordinated behaviours within larger swarms. In contrast, my testbed supports scalability through multi-worker A3C learning and the execution of multiple agents across independent systems. Chapter 5 demonstrated this capability with seven parallel A3C workers whose cumulative reward curves converged smoothly (Figure 5.4 a), confirming that the testbed overcomes the scalability limitations observed in previous studies.

Another widely observed limitation was the absence of distributed learning architectures. Testbeds such as NUAV [6], Micro-UAV [21], RUAV [23], Open [32], Replanner [33], and DeepDrone [27] testbeds, all operated on a single machine, preventing real-time exchange of state information between physically separated agents. This limited the ability to simulate realistic distributed learning swarm operations. My testbed directly addresses this by implementing distributed learning RL models using Ray, as described in Chapter 4. Chapter 5 validated that real-time distributed learning

execution was achieved without desynchronisation, demonstrating that the testbed resolves important shortcomings found in prior testbeds.

Earlier testbeds could not support real-time multi-agent coordination. Systems such as AirSim [26], PyFlyt [31], DeepDrone [27], RUAV testbed [23], and Micro-UAV testbed [21] were designed primarily for single-vehicle control with no mechanism for cooperative decision-making. My testbed overcomes this limitation by enabling real-time state exchange among multiple agents. The experimental demonstrations in Figures 5.16 – 5.21 show coordinated movement of numerous UAVs, confirming that real-time adaptive swarm behaviour is supported.

A further limitation found in the literature was the inability of previous testbeds to support challenges, particularly delayed reward problems, sparse reward settings, and policy degeneration. None of the literature reviewed testbeds included mechanisms for intrinsic motivation or advanced reward shaping. As a result, UAV agents trained on these testbeds tended to converge slowly or fall into suboptimal repetitive behaviours. My testbed addresses this through integration of the ICM module and Gaussian-based reward shaping, which significantly improves exploration and stabilises policy learning. The improvement can be quantified by examining the graphical results in Chapter 5. With ICM and Gaussian-based reward shaping, the seven-worker A3C model achieved stable cumulative reward after approximately 150,000 training steps, compared to unstable or highly fluctuating reward when intrinsic motivation was not used. The entropy curve (Figure 5.8) decreased smoothly from around 0.65 to below 0.15, indicating a clear and consistent transition from exploration to exploitation, which did not occur in the non-ICM configuration. Furthermore, actor loss (Figure 5.4 e) showed a steady downward trend rather than oscillatory behaviour, demonstrating that the learning process remained stable throughout training. Chapter 5 shows reduced entropy (Figure 5.5) and smoother actor-loss curves (Figure 5.4 e), demonstrating that the testbed effectively resolves these limitations identified in earlier studies. However, the limitations of my testbed include not supporting HILS and not comparing the data with real flight data.

Previous UAV testbeds suffered from several limitations that strongly restricted their usefulness for swarm research. A major problem was the lack of scalability, as systems such as the NUAV testbed [6] and the Micro-UAV platform [21] supported only one aircraft, preventing the study of realistic multi-agent behaviours or the evaluation of how ML algorithms react as the number of agents increases. Earlier testbeds were also not

based on distributed learning, meaning all agents ran on a single computer, which created computational bottlenecks and prevented realistic modelling of UAV coordination and decision-making; even GaTAC [34] ran all control logic on one machine despite supporting remote FlightGear instances [123].

This made it impossible to study how autonomous agents behave when operating on separate systems, which is essential for real-world swarm deployment. Another harmful limitation was the inability to prevent policy destabilisation, as earlier platforms did not include mechanisms such as clipped policy updates or clipped surrogate objective function; this caused agents to fall into oscillatory or repetitive behaviours, resulting in unstable controllers that could not be relied upon for safe swarm coordination [24]. Finally, previous testbeds did not address the delayed reward problem, relying entirely on sparse extrinsic rewards that made exploration inefficient and caused algorithms to converge slowly or collapse into suboptimal behaviours. These limitations collectively restricted progress in swarm autonomy, showing the need for a scalable and distributed learning that can support intrinsic exploration and robust policy development [14] .

Table 5.1 shows that previous testbeds were validated using only limited forms of evidence, often focusing on graphical results of ML models rather than a complete validation framework. For example, the GaTAC testbed [34], Open [32], Replanner [33] and AirSim [26] reported only graphical outputs of ML models without providing flight trajectory plots, aircraft distance plots, or experimental simulation videos, making it difficult to assess real-time behaviour or multi-agent interaction. The NUAV platform [6], Micro-UAV testbed [21], and RUAV testbed [23] also lacked comprehensive validation, presenting only basic graphical performance indicators but no trajectory-based or distance-based evaluation to demonstrate stability, safety, or accuracy. Hardware-in-the-loop systems such as the AutoPilot testbed [25] similarly did not provide aircraft distance plots or multi-agent simulation videos, limiting their ability to show coordination quality or collision avoidance behaviour. In contrast, the testbed developed in this research includes ML performance graphs, flight trajectory plots (Figures 5.16 – 5.21), aircraft minimum–maximum distance plots (5.21), and experimental simulation videos, allowing multi-dimensional evaluation of learning stability, tracking accuracy, and collaborative swarm behaviour. This broader and more rigorous validation approach addresses the shortcomings of earlier studies and provides stronger evidence of system performance.

My testbed supports distributed learning across multiple computers, enabling larger

experiments and real-time mission coordination. Chapter 5 validated this capability through distributed learning experiments where multiple agents communicated over UDP while performing coordinated target-tracking tasks.

RQ-2: Which flight simulators have been used in previous testbeds for self-coordinating and cooperative swarms of UAVs, and which is the most suitable for this research?

RO-2: To select and integrate an appropriate flight simulator and FDM into the testbed to ensure realism, flexibility, and accurate aircraft dynamics.

Chapter 2 presented a detailed comparison of existing simulators, including Gazebo, X-Plane, and Microsoft Flight Simulator. These platforms were found to be unsuitable for this research because they are either computationally intensive, require costly subscriptions, or lack support for real-time distributed learning swarm coordination [124]. Their limitations (except GaTAC and AirSim) prevented smooth multi-agent integration, restricted extensibility, and made them impractical for a scalable and distributed learning UAV testbed. In contrast, FlightGear stands out due to its lightweight architecture, cross-platform compatibility, and full open-source availability, making it more appropriate for academic research and system-level customisation. These considerations provided the initial basis for choosing FlightGear over other simulation platforms [23].

Another important factor in selecting FlightGear was its proven use in earlier UAV testbeds such as GaTAC [34] and NUAV [6]. These systems demonstrated that FlightGear can reliably support fixed-wing UAVs with realistic dynamics, and consistent performance across various mission scenarios. The success of these testbeds inspired and gave confidence to adopt the same simulator.

The integration of FlightGear with JSBSim also aligned directly with the technical requirements identified in Chapter 4. FlightGear offers real-time UDP-based data exchange, enabling continuous transmission of aircraft state information between multiple computers. This capability of FlightGear allowed the testbed to support distributed learning execution, where separate RL agents operate on independent machines while sharing real-time positional and environmental data. Furthermore, JSBSim provides high-fidelity aerodynamic modelling with configurable aircraft parameters, allowing the system to simulate realistic roll, pitch, yaw, and thrust responses that are essential for RL-based control. The combination of these features

ensured that FlightGear could meet both the computational and architectural needs of the testbed.

Chapter 5 provided strong experimental evidence that FlightGear and JSBSim were suitable for real-time learning and swarm coordination tasks. The simulator showed stable behaviour during all RL algorithm training sessions, maintaining consistent state updates even under multi-agent distributed learning execution. Plots in Figures 5.16 – 5.21 demonstrated accurate dynamic responses at different headings, confirming that aircraft motion within the simulator appears to be like realistic behaviour. The smooth integration between FlightGear, JSBSim, and the RL algorithms further validated the choice of simulator. These experimental results confirm that the selected simulator met all requirements and that the research objective was achieved.

RQ-3: What are the requirements for a testbed for self-coordinating and cooperative swarms of UAVs?

RO-3: To identify, analyse, and define the technical and functional requirements necessary for designing a scalable and distributed learning swarm UAV testbed.

The requirements for the testbed were detailed in Chapter 4. These requirements included scalability, distributed learning operation, realism, flexibility, and support for curiosity-driven exploration. Each requirement was necessary to overcome the limitations identified in earlier work, particularly in systems such as GaTAC [34], and the NUAV testbed [6].

Scalability was required to support larger swarms of UAVs, as earlier systems were limited to only one or two aircraft operating within a single-machine environment. Prior studies, except GaTAC, did not provide scalable ML-based coordination across multiple vehicles, and therefore could not be used to evaluate cooperative behaviour across larger swarms. In this testbed, scalability was achieved through the use of seven parallel A3C workers (increased from 2 and then 3 in the initial experiments) that coordinated gradient updates efficiently across multiple systems. Chapter 5 validated this requirement by demonstrating stable convergence in cumulative reward (Figure 5.4 a), increasing episode length (Figure 5.4 b), and consistent learning behaviour across all workers, confirming that larger-scale multi-agent RL can be supported without instability.

Distributed learning was required to simulate realistic swarm environments where aircraft operate as physically separated units communicating over a network. Most earlier testbeds were constrained to a single computer, preventing any form of true distributed

learning swarm experimentation, for example, the NUAV [6] and RUAV testbeds. The GaTAC [34] supported only local communication or limited networking functionality, and therefore could not emulate multi-machine execution for parallel ML. In contrast, this testbed allows each ML agent to run on an independent system while exchanging real-time state information. Chapter 4 explains this architecture in detail, and Chapter 5 verified that distributed learning operation was achieved with no desynchronisation across multiple systems (Section 5.9 to Section 5.12), demonstrating that the requirement was fully met.

Realism was required to ensure that the learning environment accurately represented fixed-wing aircraft dynamics and could support RL agents without introducing unrealistic behaviours. In this testbed, realism was ensured through the integration of FlightGear with the JSBSim FDM, which provides physics-based aerodynamic responses. The results in Chapter 5 demonstrated stable roll, pitch, and yaw behaviours, and trajectory plots in Figures 5.16 – 5.21 showed accurate aircraft motion at varying speeds and headings.

Flexibility was required for supporting multiple algorithms and enabling experimentation with different control strategies. Earlier testbeds lacked this flexibility, as they were often tied to a single controller or a fixed behavioural model used in testbeds like GaTAC, RUAV [23] and NUAV testbeds. In my testbed, flexibility was achieved by integrating several algorithms—A2C, A3C, and PPO, all of which were successfully trained and tested separately and also worked in parallel.

Support for curiosity-driven exploration was necessary to address challenges such as sparse rewards, delayed reward feedback, and policy degeneration. This testbed incorporated an ICM and Gaussian reward shaping, enabling agents to explore more effectively and reduce reliance on sparse extrinsic rewards. Chapter 5 demonstrates this requirement through the entropy curve (Figure 5.5), which shows a progressive and stable shift from exploration to exploitation, and through stable actor-loss behaviour (Figure 5.4 e). These results confirm that curiosity-driven learning was implemented and that this requirement was achieved. These combined results show that RO-3 was fully met.

RQ-4: Which ML algorithms are most suitable for enhancing the performance and robustness of swarm UAV testbeds?

RO-4: To evaluate and compare multiple ML algorithms to determine which are most suitable for achieving stable, robust, and cooperative swarm behaviour within a UAV

testbed.

Chapters 4 and 5 provided extensive validation of ML algorithms using multiple measures, such as cumulative reward, actor–critic losses, entropy, positional tracking plots, and minimum–maximum inter-UAV distances in the swarm. The seven-worker A3C configuration showed consistent and stable learning behaviour: cumulative reward increased steadily before stabilising (Figure 5.4 a), episode length increased progressively (Figure 5.4 b), and actor loss declined throughout training (Figure 5.4 e), indicating improved policy quality. Entropy reduction in Figure 5.5 demonstrated a smooth transition from exploration to exploitation, and trajectory validation in Figures 5.16 – 5.21 confirmed that trained agents maintained accurate and stable tracking of the target aircraft, in contrast to untrained agents that exhibited oscillatory motion (Sections 5.13.1 – 5.13.4). Swarm-level experiments (5.21) further showed stable inter-agent distances and coordinated follower behaviour, proving the robustness of multi-agent performance.

In comparison to A3C, the A2C implementation primarily served as a baseline and was used to model the target aircraft behaviour, as described in Chapter 4. While A2C worked for a random target UAV trajectory, its learning dynamics were more sensitive to delayed reward and sparse feedback. The reward curves for A2C exhibited greater fluctuations and slower convergence than those observed for the multi-worker A3C configuration. Because A2C relies on synchronous updates, it lacked the diversity of experience and stabilising effect of asynchronous workers that are inherent in A3C. As a result, A2C was less effective for swarm-level tracking and coordination, and its role in this thesis was mainly to provide a random trajectory for a target aircraft.

When compared with PPO, A3C also demonstrated more favourable learning characteristics in the context of this testbed. PPO was introduced to benefit from clipped policy updates to improve stability, but in practice, it required careful tuning of clipping ranges and other hyperparameters to avoid oscillatory reward behaviour. In the experiments, PPO-based agents showed less cumulative reward as compared to A3C, particularly under complex target trajectories and swarm configurations. The seven-worker A3C setup, especially when combined with ICM, achieved smoother reward curves and more reliable convergence. This indicates that A3C was better suited to the distributed learning, asynchronous architecture and computational constraints of the testbed.

The selection of A2C, A3C, and PPO was based on their suitability for

continuous-control environments, their compatibility with distributed learning architectures, and their proven effectiveness in real-time decision-making tasks. These algorithms represent three widely used actor–critic methods capable of handling continuous state and action spaces, which are essential for fixed-wing UAVs. A2C was adopted because it provides a synchronous baseline that is simpler to train and serves as an effective model for generating dynamic target behaviours. Its stability and lower computational footprint made it an appropriate choice for modelling the target aircraft and evaluating tracker performance under controlled conditions [125] .

A3C was selected because it is designed for asynchronous learning to begin with. It is also suitable for distributed learning, which aligns directly with the multi-machine architecture of the testbed [126]. Unlike A2C, which relies on synchronous updates, A3C benefits from parallel workers exploring different parts of the state space simultaneously. This reduces correlation between samples, accelerates convergence, and improves policy stability, all of which were confirmed in the seven-worker experiments presented in Chapter 5. The ability of A3C to scale naturally across multiple systems made it a strong candidate for swarm-level coordination tasks, and its performance results demonstrated more consistent convergence, smoother learning curves, and better robustness than the other algorithms used.

PPO was included due to its clipped surrogate objective, which is designed to stabilise policy updates and prevent large, destabilising gradient steps. This makes PPO particularly appealing for problems with delayed or noisy rewards, as is typical in UAV tracking and swarm coordination. While PPO required careful tuning of hyperparameters, it offered a useful comparison to asynchronous methods and allowed the testbed to evaluate both synchronous and asynchronous algorithm families. Although PPO exhibited greater variability in cumulative reward compared to A3C, its inclusion helped demonstrate how different algorithmic designs respond to delayed reward signals and distributed learning execution [127].

Other algorithms were considered, but did not perform well in this environment. Value-based methods such as DQN failed to handle continuous action spaces without substantial modification, and extended variants such as DDPG failed completely during early testing. These algorithms diverged, which made them unsuitable for inclusion in the final evaluation. Their poor performance highlighted the importance of selecting algorithms that are specifically designed for continuous control and that can operate reliably in distributed learning and asynchronous learning settings [128].

Collectively, the results demonstrate that A3C, particularly in combination with ICM, offered the strongest balance of stability, adaptability, and cooperative swarm coordination among the algorithms implemented. A3C leveraged asynchronous parallel workers to stabilise learning and handled delayed rewards more effectively. The evidence from Chapters 4 and 5, including the convergence trends in Figures 5.4 and 5.5 and the swarm behaviours shown in Figures 5.16 – 5.21, confirms that A3C-ICM is the most suitable algorithm within this testbed.

Entropy in reinforcement learning reflects the level of exploration in the agent's policy. A higher entropy value indicates more exploratory actions, while a lower value suggests that the agent is converging towards a more deterministic policy. In this research, the observed decrease in entropy corresponds to the transition from exploration to exploitation as the agents learn effective tracking strategies. While a rapid decrease in entropy can sometimes indicate premature convergence, this is not observed in this case. The reduction in entropy is accompanied by stable improvements in performance metrics such as cumulative reward, convergence behaviour, and tracking accuracy, as shown in Chapter 5 . This indicates that the agents are not converging to a suboptimal policy, but rather reaching a stable and effective control strategy. Furthermore, the agents continue to respond to changes in the environment, such as variations in the target UAV trajectory, which suggests that some level of adaptive behaviour is retained despite lower entropy values. Therefore, the observed entropy reduction represents a desirable transition towards stable tracking behaviour rather than premature convergence. It reflects that the agents have reached a balance between exploration and exploitation that is suitable for the given task[129].

The use of the Intrinsic Curiosity Module encourages the agent to explore state transitions that are novel or less predictable, rather than relying solely on task-based rewards. This is particularly important in UAV tracking scenarios where the reward signal may be delayed or sparse. In the experiments conducted in this research, agents using curiosity-driven learning demonstrated improved exploration behaviour compared to standard reward-based agents. Specifically, these agents were more effective in adapting to dynamic target trajectories, including sudden changes in direction and complex flight paths. Instead of following a direct but limited tracking strategy, curiosity-driven agents explored a wider range of manoeuvres, allowing them to recover more effectively from tracking errors. For example, when the target UAV performed abrupt turns, curiosity-driven agents were able to adjust their trajectories more quickly

and avoid getting trapped in suboptimal tracking patterns. In contrast, agents trained without curiosity tended to follow more rigid paths and required more time to adapt to changes in the environment. These observations indicate that curiosity-driven learning enables the discovery of more flexible and adaptive tracking behaviours that may not emerge from standard reward-based learning alone. The intrinsic reward encourages the agent to explore alternative strategies, leading to improved robustness and performance in complex scenarios[130].

PPO is widely regarded in the literature as a stable and reliable reinforcement learning algorithm due to its clipped objective function, which constrains policy updates and prevents instability. However, PPO typically relies on batch-based and more synchronised updates, where trajectories are collected and processed before performing optimisation steps. This introduces implicit synchronisation requirements that can be sensitive to delays in distributed environments [1,2]. In contrast, A3C is inherently asynchronous, where multiple workers interact with the environment independently and update the global model without requiring strict synchronisation. This allows each worker to operate at their own pace, reducing the impact of delays caused by network communication or heterogeneous computational resources. Asynchronous reinforcement learning architectures are known to improve scalability and reduce latency issues by decoupling environment interaction from model updates In the distributed testbed used in this research, UAV agents were deployed across multiple machines connected through a network. This setup introduces variability in communication latency and processing speed between nodes. In such conditions, synchronous or batch-based methods like PPO may experience delays due to the need to wait for complete batches of data, leading to reduced efficiency [131]. On the other hand, A3C's asynchronous nature allows continuous updates without waiting for all agents to synchronise. This makes it more tolerant to network latency and better suited for distributed multi-agent environments. Studies have shown that asynchronous methods can achieve faster learning and better responsiveness in dynamic environments due to reduced waiting times and improved resource utilisation [132]. Furthermore, comparative studies indicate that A3C often achieves faster training and adaptation, while PPO prioritises stability at the cost of slower updates Reference. In this research, the ability of A3C to adapt quickly to dynamic UAV trajectories and distributed execution conditions appears to have provided an advantage over PPO. In this research, the advantage of A3C over PPO is observed through faster convergence and improved responsiveness to dynamic target

trajectories. For example, A3C agents reached stable tracking behaviour in fewer training steps and exhibited quicker adaptation when the target UAV changed direction. This is reflected in the experimental results in Chapter 5, where A3C shows a steeper increase in cumulative reward and faster stabilisation of episode length compared to PPO. Therefore, the improved performance of A3C in this work can be attributed to its asynchronous learning structure, which aligns well with the distributed, multi-machine architecture of the testbed. PPO remains a strong baseline for stable policy learning, but A3C demonstrates advantages in environments where latency, parallelism, and real-time adaptation are critical.

6.2 Research Contribution and Significance

The contributions of this research are demonstrated clearly when comparing the testbed with previous testbeds, as shown in Table 6.1.

Testbed	**Scalable**	**Distributed Learning**	**Prevent Policy Destabilization**	**Solved Delayed Reward Problem**
GaTAC Testbed [34]	✓	✗	✗	✗
NUAV Testbed [6]	✗	✗	✗	✗
Micro-UAV Testbed [21]	✗	✗	✗	✗
RUAV Testbed [23]	✗	✗	✗	✗
AutoPilot Testbed [25]	✗	✗	✗	✗
AirSim Testbed [26]	✓	✗	✗	✗
DeepDrone Testbed [27]	✗	✗	✗	✗
PyFlyt Testbed [31]	✗	✗	✗	✗
OPEN Testbed [32]	✗	✗	✗	✗
REPLANNER Testbed [33]	✗	✗	✗	✗
Octo-rotor RL Testbed [29]	✗	✗	✗	✗
dSPACE Testbed [30]	✗	✗	✗	✗
My Testbed	✓	✓	✓	✓

Table 6.1: Comparison of the testbed of this research project with previous studies

While earlier testbeds such as GaTAC, Open, Replanner, AirSim, NUAV, Micro-UAV testbed, and others provided important foundations, they exhibited critical limitations in terms of scalability, distribution, and the ability to handle core ML challenges. The testbed developed in this research project has successfully addressed all of these shortcomings, thereby establishing itself as a significant advancement in the domain of swarm UAV coordination. A key contribution of this research is the development of a scalable and distributed learning testbed capable of supporting multiple UAVs operating across interconnected systems. Earlier testbeds given in Table 6.1, except GaTAC are not scalable. This limitation prevented realistic swarm experiments or multi-agent training. In contrast, the testbed developed in this research scales and distributes ML models across several machines, allowing each UAV to run on an independent machine. Chapter 5 demonstrated this capability by successfully

training seven parallel A3C workers whose cumulative reward curves stabilised after approximately 150,000 steps (Figure 5.4 a). This behaviour shows that the architecture can support larger swarms without compromising learning stability, clearly addressing the scalability gaps in earlier research.

Most earlier testbeds given in Table 6.1 were not based on distributed learning, meaning they could not run control algorithms or ML models across multiple machines. Among these, GaTAC is considered distributed because it can launch multiple FlightGear instances across different computers. However, GaTAC is not a distributed learning testbed in terms of ML models' distribution. All ML models, control logic, and autonomous decision-making processes run on a single machine, while the remote FlightGear instances only act as visual or simulation front-ends distributed on different machines. As a result, it cannot assign a separate ML model to each aircraft. In contrast, the testbed developed in this research is a fully distributed learning.

Each UAV operates with its own ML model running on an independent computer. In the first experiment, four different machines independently executed the A2C target aircraft model, two A3C tracking models, and the PPO tracker. In the second experiment, the system was expanded to eight machines, where one system executed the A2C target model and seven additional systems operated seven separate A3C worker models. Each machine performed its own inference, learning, and communication with FlightGear, demonstrating genuine distributed learning execution. This capability to distribute ML models across multiple systems represents a level of scalability and distribution that earlier testbeds, including GaTAC, do not provide. Even though GaTAC can launch simulation FlightGear instances on remote computers, it cannot distribute control algorithms or ML processes across several systems. All decision-making occurs on a single machine.

Another major contribution is the demonstration that this testbed prevents policy destabilisation, where agents may fall into repetitive, oscillatory, or suboptimal behaviours. Prior studies did not incorporate mechanisms, particularly in multi-agent scenarios where delayed or inconsistent feedback can easily destabilise the policy. This research overcame these issues by integrating PPO alongside A3C, using PPO's clipped surrogate objective to regulate policy updates. Chapter 5 showed that the actor loss steadily declined (Figure 5.4 e), and the entropy curve decreased smoothly (Figure 5.5), confirming that the learning process remained stable across training sessions. These results demonstrate that the testbed successfully avoids the degeneration problems

observed in earlier systems.

A further contribution is the solution to the delayed reward problem, which is one of the most significant barriers for aerial systems. In many real-world UAV missions, meaningful rewards occur only after long flight durations, making traditional methods inefficient and unstable. Previous testbeds relied only on extrinsic reward signals and therefore struggled with sparse or delayed rewards. This research incorporated ICM combined with a Gaussian ellipsoidal reward shaping approach, enabling the agents to explore effectively even when extrinsic rewards were minimal. Chapter 5 showed that trained agents achieved smooth and consistent tracking trajectories (Figures 5.16 – 5.21) and demonstrated reduced exploration entropy (Figure 5.5), proving that the delayed reward problem was effectively addressed.

The significance of this research lies in what it adds to the field of intelligent swarm aviation. This thesis contributes a testbed that unifies high-fidelity flight simulation, distributed learning execution, and curiosity-driven multi-agent learning. The capability to scale the number of agents, distribute ML based computation across multiple systems, and compare different models in a controlled environment represents a meaningful advancement over existing testbeds such as NUAV. Moreover, by demonstrating that curiosity-based learning improves both stability and convergence, the research opens new pathways for developing autonomous swarms that operate reliably in real-world conditions. These contributions, taken together, establish the novelty and value of the work within the broader research landscape.

6.3 Limitations

Although the testbed demonstrated strong performance and achieved the research aims, several limitations should be acknowledged. These limitations arise mainly from the scope of the experiments and the use of a simulation-only environment.

First, the testbed was validated using a single primary use case, namely, target tracking. While this scenario is effective for demonstrating scalability, coordination, and the behaviour of multiple agents, it does not cover the full range of swarm missions. As a result, the generalizability of the testbed across different mission profiles has not yet been verified. Second, all experiments were conducted in a simulation environment based on FlightGear and the JSBSim FDM. Although these tools provide realistic physics and accurate aircraft behaviour, they cannot fully replicate real-world conditions

such as unpredictable wind patterns, Global Positioning System (GPS) noise, and sensor failure. The absence of real hardware validation limits the ability to confirm how well the trained models would transfer to physical UAVs. In summary, the main limitations of this research relate to the focus on a single use case and the absence of real hardware experiments. We did not perform HIL and comparison with real flight data for our testbed. These limitations provide important directions for the future development of the testbed.

Another limitation is that the testbed was evaluated using a maximum of seven UAVs. Although the distributed learning architecture supports additional agents, the experiments focused only on seven aircraft due to computational constraints and the need to stabilise the learning process. Larger swarms may introduce new challenges, such as increased communication load, more complex coordination patterns, and slower training convergence. Future work should explore experiments with greater numbers of agents to confirm whether the testbed maintains stability and performance at larger scales.

Another limitation is that the experiments were conducted exclusively with fixed-wing aircraft models. While fixed-wing platforms represent an important category of aerial vehicles, other configurations, such as rotary-wing and hybrid aircraft, exhibit different aerodynamic behaviours, control surfaces, and mission profiles. The current testbed is technically capable of supporting other aircraft types through JSBSim, but this capability was not explored in the experimental validation. Testing additional vehicle types would help determine the versatility and general applicability of the testbed.

Although the simulation environment is realistic, all experiments were conducted purely in software, and no hardware-in-the-loop validation was performed. Real-world UAV systems introduce additional challenges such as communication delays, weather conditions, noise in sensor readings, and physical hardware constraints that cannot be fully captured in simulation. Furthermore, the algorithms were tested primarily on the task of target tracking; while this serves as a strong proof of concept, it does not encompass the full range of potential swarm applications. The computational cost of training multiple agents concurrently also remains a practical limitation, especially as the number of UAVs in the swarm increases.

The present work has several limitations that offer opportunities for future improvement. The HIL was not performed for our testbed, and our data was not compared with real flight data. The experiments were conducted using only seven UAVs.

Although the architecture is designed to scale, the testbed has not yet been empirically evaluated with larger swarm sizes that would more rigorously test its distributed learning performance. The study focused exclusively on fixed-wing UAV, and the behaviour of rotary-wing or hybrid platforms, whose dynamics differ significantly, was not examined, limiting the generalisability of the findings across diverse aircraft types. All experiments were carried out entirely in simulation using FlightGear and JSBSim, and no hardware-in-the-loop or real-world deployment was performed. This means that the framework has not yet been validated under real environmental uncertainties such as wind disturbances, communication delays, or sensor noise. The distributed learning implementation was tested across a small number of networked systems, and larger or heterogeneous computing environments may introduce latency or synchronisation challenges not captured in the current setup. The evaluation also focused on a limited range of environmental conditions and did not assess the robustness of the models under extreme scenarios such as adverse weather configurations.

The research also implemented only three RL algorithms: A2C, A3C, and PPO. These algorithms were chosen because they align well with the distributed learning and continuous-control nature of the testbed. Although other algorithms, such as DQN and DDPG, were considered or tested, they failed in the fixed-wing swarm environment. They showed complete failure to converge during early tests and were therefore not included in the final evaluation. As a result, the conclusions of this research are limited to the algorithms that demonstrated acceptable performance in this specific context. Future studies should investigate whether alternative algorithms or hybrid approaches can provide further improvements in stability or coordination.

The primary limitation arises from CPU and memory requirements associated with running multiple simulation instances and learning agents simultaneously. Each UAV requires an independent simulation process using FlightGear and JSBSim, along with a dedicated learning agent. As the number of UAVs increases, the computational load grows significantly due to the need to process flight dynamics, state updates, and learning computations in real time. The computational load naturally grows due to the need to process flight dynamics, state updates, and learning computations in real time. This increase in load is expected and does not contradict the scalability of the testbed. Instead, it indicates that additional computational resources are required to support larger swarm sizes. The testbed is designed with a distributed architecture, where each UAV and its learning agent can be deployed on separate machines. This means that

scalability is achieved by adding more computers to the system, allowing the workload to be distributed rather than overloading a single machine. Therefore, while the experiments in this thesis were limited by the available hardware, the architecture itself supports scaling to larger numbers of UAVs by increasing the number of computational nodes. In addition to computational constraints, network overhead also contributes to the limitation. Communication between distributed nodes using UDP introduces latency and bandwidth requirements, particularly when multiple UAVs are exchanging state and action information at high frequency. However, this was a secondary constraint compared to CPU and memory usage.

The computational cost mentioned as a limitation in this research refers to both the processing requirements for training the Machine Learning models and the communication overhead associated with distributed execution. From a training perspective, the use of multiple UAV agents and algorithms such as A3C and PPO increases CPU and memory usage, particularly when multiple workers are executed in parallel. The need to simulate realistic flight dynamics using JSBSim and FlightGear further adds to the computational load, resulting in longer training times and higher resource requirements. In addition to training cost, distributed execution introduces network overhead. When UAV agents are deployed across multiple machines, communication between nodes is required to exchange state information, synchronise learning updates, and maintain real-time interaction with the simulation environment. This clarification provides a more precise understanding of the limitations of the testbed. In addition, GPU resources contribute to the computational cost, as they are used to accelerate neural network training and inference for the learning agents. As the number of UAVs increases, more models need to be trained and updated in parallel, which increases the demand on GPU resources. Limited GPU availability can slow down training and affect overall system performance, while access to multiple GPUs or more powerful hardware can significantly improve scalability and training efficiency.

6.4 Implications and Synthesis of Key Findings

The findings of this research have several important implications for the development of scalable and distributed learning swarm UAV testbeds. One major implication concerns the ability to design testbeds that realistically support multi-agent learning. The results demonstrated that combining FlightGear with the JSBSim FDM creates a reliable and flexible environment for fixed-wing aircraft. This means that future

researchers and system designers can use the same combination to train and evaluate complex UAV behaviours without needing expensive proprietary simulators. The ability to simulate high-fidelity physics within an open-source environment also reduces financial and technical barriers, making UAV research more accessible.

Another important implication is the confirmation that asynchronous multi-agent learning is essential for stable swarm behaviour. The experiments showed that A3C with multiple workers produced more reliable convergence trends, smoother policy updates, and more stable tracking than A2C or PPO (details in section 5.3). This indicates that for real-time swarm coordination, asynchronous learning architectures should be preferred. The strong capability of A3C, especially when combined with ICM, suggests that curiosity-driven asynchronous methods can overcome delayed reward problems and sparse feedback, which are common in large outdoor environments (details in section 5.6). Therefore, future swarm learning frameworks can confidently adopt asynchronous architectures with intrinsic motivation modules to improve training robustness.

The findings also show that a distributed learning architecture is necessary for realistic swarm simulations. The successful use of the Ray framework execution demonstrated that individual agents can run on separate machines while still sharing their states reliably. This result implies that future hardware deployment of swarm systems will benefit from similar architectures, where each aircraft processes its own data but collaborates through lightweight communication protocols. The fact that the distributed learning testbed maintained synchronisation and stable communication throughout all scenarios suggests that this approach can scale to more agents and more complex missions.

The experimental results showing coordinated swarm behaviour have implications for real-world applications. The stable inter-UAV distances in swarm observed in 5.21 and the consistent follower behaviour imply that the policies learned in simulation may be directly adaptable to real fixed-wing UAVs. The realistic flight dynamics generated by JSBSim mean that the learned behaviours take into account aerodynamic forces, making the learned policies closer to something that can be executed on real aircraft.

Finally, the findings highlight the broader potential of the testbed as a foundation for additional ML applications beyond target tracking. The modular design, support for multiple algorithms, distributed learning execution, and stability of the learning framework mean that the system can be extended to applications such as formation

control, collision avoidance, cooperative mapping, and autonomous surveillance. The testbed's performance demonstrates that it can serve as a general platform for future research, providing both a research tool and a stepping stone toward real-world autonomous swarm systems.

The combination of FlightGear and JSBSim provides a realistic, physics-based environment for aerodynamic behaviour. The use of A2C, A3C, and PPO demonstrates that different actor–critic algorithms can be effectively mapped to different roles within a multi-agent system. The introduction of Gaussian-ellipsoidal reward shaping stabilises agent behaviour by providing consistent, meaningful feedback in a three-dimensional environment. Most importantly, the integration of the ICM addresses the delayed reward problem, enabling agents to explore the environment effectively even when extrinsic signals are sparse. Experiments confirm that curiosity-driven agents converge faster, behave more reliably, and maintain stable tracking performance in comparison to non-ICM models.

6.5 Future Work

Although the current implementation of the testbed has demonstrated significant success in enabling distributed learning and scalable UAV coordination, the work presented in this thesis represents only the first step in a broader research agenda. At present, the testbed has been validated using a single use case, namely, target tracking. This use case has proven effective in demonstrating the stability, robustness, and cooperative behaviours of swarm UAVs trained with advanced RL algorithms. Future work will expand the scope of the testbed by incorporating multiple use cases beyond target tracking. These may include formation control, cooperative search and rescue, and multi-objective swarm missions where agents must balance competing goals such as energy efficiency, safety, and task completion. Expanding to multiple use cases will demonstrate the testbed's full flexibility and further establish its generalizability to real-world scenarios.

Another key direction for future research is the implementation of this testbed on physical hardware. While simulation has provided a safe and scalable environment for development and validation, transferring the trained models to real UAVs will allow practical verification under real-world conditions. Hardware deployment will involve integration with onboard sensors, communication modules, and embedded computing

platforms, ensuring that the distributed learning architecture and curiosity-driven RL strategies developed in this project can operate effectively outside the simulated environment.

In summary, future work will extend the testbed from a single use case to a multi-purpose platform, while also bridging the gap between simulation and physical implementation. These directions will strengthen the contribution of this research, demonstrating the practical viability and versatility of the testbed for distributed learning swarm UAV.

6.6 Concluding Insights

In closing, this thesis has demonstrated that a distributed learning, scalable, and intelligent swarm UAV testbed can be designed and implemented using modern simulation tools and ML techniques. By addressing long-standing challenges related to scalability, distribution, delayed rewards, and realistic physics-based evaluation, this work can be useful for future UAV testbeds. The contributions made here not only advance academic understanding but also provide practical tools that can support the development of future autonomous flight systems. Ultimately, this research moves the field one step closer to real-world deployment of cooperative, intelligent, and reliable swarm UAVs. This work reduces the cost and risk of developing autonomous UAV swarms by enabling scalable simulation-based testing for applications such as environmental monitoring, disaster response, and search and rescue operations.

6.7 Summary

This chapter interprets the experimental results in relation to the research questions and demonstrates that the testbed effectively addresses the key limitations identified in existing work. The findings confirm that scalability, distributed learning, and delayed reward challenges can be jointly managed through the integration of A2C, A3C, PPO, and curiosity-driven mechanisms. The success of the testbed is not due to any single algorithm, but to the coordinated interaction between asynchronous learning (A3C), stable policy optimisation (PPO), and exploration enhancement (ICM). This combination enables robust swarm coordination and consistent target-tracking performance in complex scenarios.

The chapter also reflects on limitations, including reliance on simulation and the need for further real-world validation, highlighting areas for future improvement. Overall, it synthesises the contributions of the research by showing how an integrated approach advances the development of intelligent and cooperative swarm UAV systems. This chapter concludes the thesis by synthesising the overall findings and confirming that the developed testbed successfully enables scalable and distributed learning for swarm UAV coordination. It demonstrates that integrating realistic simulation, reinforcement learning algorithms, and curiosity-driven exploration provides an effective solution to challenges such as delayed rewards and policy instability.

A key conclusion is that the strength of the research lies in the unified integration of these components, which allows cooperative and adaptive swarm behaviour to emerge in a controlled yet realistic environment. The testbed is shown to be flexible and extensible, supporting multiple UAVs and heterogeneous learning models. The chapter also highlights limitations, particularly the reliance on simulation, and identifies future directions such as real-world deployment, advanced coordination strategies, and further optimisation of learning models. Overall, the research establishes a strong foundation for developing intelligent swarm UAV systems and demonstrates the importance of combining distributed learning, exploration mechanisms, and high-fidelity simulation within a single framework.

Bibliography

[1] Taha Hamid, Muhammad Zeeshan Babar, Muhammad Uzair Hassan, and Muhammad Fawad Hussain. Pitch attitude control for an aircraft using linear quadratic integral control strategy. In *2019 22nd International Multitopic Conference (INMIC)*, pages 1–6. IEEE, 2019.

[2] Mark Saunders, Philip Lewis, and Adrian Thornhill. *Research methods for business students*. Pearson education, 2009.

[3] Cecilia Milesi and Erika López Franco. Volunteer contributions to building more peaceful, inclusive, just and accountable societies, 2020.

[4] Elina Jääskä, Kirsi Aaltonen, and Jaakko Kujala. Game-based learning in project sustainability management education. *Sustainability*, 13(15):8204, 2021.

[5] Julius Quarshie Azasoo and Kwame Osei Boateng. A retrofit design science methodology for smart metering design in developing countries. In *2015 15th International Conference on Computational Science and Its Applications*, pages 1–7. IEEE, 2015.

[6] Saleh Habib, Mahgul Malik, Shams Ur Rahman, and Muhammad Adil Raja. Nuav-a testbed for developing autonomous unmanned aerial vehicles. In *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, pages 185–192. IEEE, 2017.

[7] Maung K Sein, Ola Henfridsson, Sandeep Purao, Matti Rossi, and Rikard Lindgren. Action design research. *MIS quarterly*, pages 37–56, 2011.

[8] Yiran Wang, Chenshu Liu, Yunfan Li, Sanae Amani, Bolei Zhou, and Lin F Yang. Hyper: Hyperparameter robust efficient exploration in reinforcement learning. *arXiv preprint arXiv:2412.03767*, 2024.

[9] Bradly Stadie, Lunjun Zhang, and Jimmy Ba. Learning intrinsic rewards as a bi-level optimization problem. In *Conference on Uncertainty in Artificial Intelligence*, pages 111–120. PMLR, 2020.

[10] Haoran Sun, Yekun Chai, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Curiosity-driven reinforcement learning from human feedback. *arXiv preprint arXiv:2501.11463*, 2025.

[11] Siyin Wang, Shimin Li, Tianxiang Sun, Jinlan Fu, Qinyuan Cheng, Jiasheng Ye, Junjie Ye, Xipeng Qiu, and Xuanjing Huang. Llm can achieve self-regulation via hyperparameter aware generation. *arXiv preprint arXiv:2402.11251*, 2024.

[12] Yukai Hou, Jin Zhao, Rongqing Zhang, Xiang Cheng, and Liuqing Yang. Uav swarm cooperative target search: A multi-agent reinforcement learning approach. *IEEE Transactions on Intelligent Vehicles*, 2023.

[13] Xiaowei Fu, Jindong Zhu, Zhaoying Wei, Hui Wang, and Sili Li. A uav pursuit-evasion strategy based on ddpg and imitation learning. *International Journal of Aerospace Engineering*, 2022(1):3139610, 2022.

[14] Gonzalo Aguilar Jiménez, Arturo de la Escalera Hueso, and Maria J Gómez-Silva. Reinforcement learning algorithms for autonomous mission accomplishment by unmanned aerial vehicles: A comparative view with dqn, sarsa and a2c. *Sensors*, 23(21):9013, 2023.

[15] Barbara Kitchenham, Stuart Charters, et al. Guidelines for performing systematic literature reviews in software engineering. 2007.

[16] Zhigang Wang and Jianbo Li. 3d simulation of flight control system for quad tilt rotor uav based on flightgear. In *IOP Conference Series: Earth and Environmental Science*, volume 440, page 052056. IOP Publishing, 2020.

[17] Try Susanto, Muhammad Bayu Setiawan, Akhmad Jayadi, Farli Rossi, Afrizal Hamdhi, and Jaka Persada Sembiring. Application of unmanned aircraft pid control system for roll, pitch and yaw stability on fixed wings. In *2021 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE)*, pages 186–190. IEEE, 2021.

[18] Gloria Stenfelt and Ulf Ringertz. Yaw control of a tailless aircraft configuration. *Journal of aircraft*, 47(5):1807–1811, 2010.

[19] Mihai-Alin Stamate, Cristina Pupăză, Florin-Adrian Nicolescu, and Cristian-Emil Moldoveanu. Improvement of hexacopter uavs attitude parameters employing control and decision support systems. *Sensors*, 23(3):1446, 2023.

[20] Yong Zeng and Rui Zhang. Energy-efficient uav communication with trajectory optimization. *IEEE Transactions on wireless communications*, 16(6):3747–3760, 2017.

[21] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *IEEE Robotics & Automation Magazine*, 17(3):56–65, 2010.

[22] Muhammad Adil Raja, Sarmad Ali, and Aisha Mahmood. Simulators as drivers of cutting edge research. In *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pages 114–119. IEEE, 2016.

[23] Murat Yasar, Derek Bridges, Goutham Mallapragada, and Joseph Horn. A simulation test bed for coordination of unmanned rotorcraft and ground vehicles. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 6263, 2006.

[24] Hyuna Kang, Seunghoon Jung, Hakpyeong Kim, Jaewon Jeoung, and Taehoon Hong. Reinforcement learning-based optimal scheduling model of battery energy storage system at the building level. *Renewable and Sustainable Energy Reviews*, 190:114054, 2024.

[25] Jingsha Zhang, Qingbo Geng, and Qing Fei. Uav flight control system modeling and simulation based on flightgear. In *International Conference on Automatic Control and Artificial Intelligence (ACAI 2012)*, pages 2231–2234. IET, 2012.

[26] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *arXiv preprint arXiv:1705.05065*, 2018. Extended for UAV applications with reinforcement learning.

[27] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2020.

[28] Jyotirmoy Vira, Zongkai Liu, Yixuan Wang, Michael Dorothy, and Pratap Tokekar. Swarmrl: A multi-agent reinforcement learning testbed for uav swarms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1215–1222, 2022.

[29] Ibrahim Ahmed, Marcos Quinones-Grueiro, and Gautam Biswas. A high-fidelity simulation test-bed for fault-tolerant octo-rotor control using reinforcement learning. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2022.

[30] Hong-Mei Zhang, Qiang Zhou, and Guang-Yan Xu. Hardware-in-the-loop simulation platform for uav based on dspace. In *2015 International Conference on Computational Science and Engineering*, pages 450–454. Atlantis Press, 2015.

[31] Jun Jet Tai, Jim Wong, Mauro S. Innocente, Nadjim Horri, James Brusey, and Swee King Phang. Pyflyt—uav simulation environments for reinforcement learning research. *IEEE Journal of LaTeX Class Files*, 2023. arXiv:2304.01305.

[32] Jiayu Chen, Chao Yu, Guosheng Li, Wenhao Tang, Shilong Ji, Xinyi Yang, Botian Xu, Huazhong Yang, and Yu Wang. Online planning for multi-uav pursuit-evasion in unknown environments using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 2025.

[33] Alvi Ataur Khalil, Alexander J. Byrne, Mohammad Ashiqur Rahman, and Mohammad Hossein Manshaei. Efficient uav trajectory-planning using economic reinforcement learning. *arXiv preprint arXiv:2103.02676*, 2021.

[34] Ekhlas Sonu and Prashant Doshi. Gatac: A scalable and realistic testbed for multiagent decision making. In *AAMAS*, pages 1507–1508. Citeseer, 2012.

[35] Chao Yan, Xiaojia Xiang, and Chang Wang. Fixed-wing uavs flocking in continuous spaces: A deep reinforcement learning approach. *Robotics and Autonomous Systems*, 131:103594, 2020.

[36] Jie Jiang, Peter M. Atkinson, Jiayi Zhang, Ruhua Lu, Youyan Zhou, Qiang Cao, Yongchao Tian, Yan Zhu, Weixing Cao, and Xiaojun Liu. Combining fixed-wing uav multispectral imagery and machine learning to diagnose winter wheat nitrogen status at the farm scale. *European Journal of Agronomy*, 138:126537, 2022.

[37] Francisco Giral, Ignacio Gomez, and Soledad Le Clainche. Control and motion planning of fixed-wing uav through reinforcement learning. *Results in Engineering*, 23:102379, 2024.

[38] Eivind Bøhn, Erlend M. Coates, Dirk Reinhardt, and Tor Arne Johansen. Data-efficient deep reinforcement learning for attitude control of fixed-wing uavs: Field experiments. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3168–3183, 2024.

[39] Kasper Trolle Borup, Thor Inge Fossen, and Tor Arne Johansen. A machine learning approach for estimating air data parameters of small fixed-wing uavs using distributed pressure sensors. *IEEE Transactions on Aerospace and Electronic Systems*, 56(3):2157–2170, 2019.

[40] Yu Zhao, Jifeng Guo, Chengchao Bai, and Hongxing Zheng. Reinforcement learning-based collision avoidance guidance algorithm for fixed-wing uavs. *Complexity*, 2021:8818013, 2021.

[41] Jinghang Li, Shuting Xu, Yu Wu, and Zhe Zhang. Automatic landing control for fixed-wing uav in longitudinal channel based on deep reinforcement learning. *Drones*, 8:568, 2024.

[42] Aiya Cui, Ying Zhang, Pengyu Zhang, Wei Dong, and Chunyan Wang. Intelligent health management of fixed-wing uavs: A deep-learning-based approach. In *16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2020.

[43] Murat Bronz, Elgiz Baskaya, Daniel Delahaye, and Stéphane Puechmorel. Real-time fault detection on small fixed-wing uavs using machine learning. *IEEE Transactions on Aerospace and Electronic Systems*, 2020.

[44] Francisco Giral, Ignacio Gomez, and Soledad Le Clainche. Control and motion planning of fixed-wing uav through reinforcement learning. *Results in Engineering*, 23:102379, 2024.

[45] Eivind Bøhn, Erlend M. Coates, Signe Moe, and Tor Arne Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019.

[46] AR Perry and C Olson. The flightgear flight simulator history, status and future. In *LinuxTag Conference and Expo*. Citeseer, 2001.

[47] Mario G Perhinschi, Brenton Wilburn, Jennifer Wilburn, Hever Moncayo, and Ondrej Karas. Simulation environment for uav fault tolerant autonomous control laws development. *Journal of Modeling, Simulation, Identification, and Control*, 1(4):164–195, 2013.

[48] Konstantin Metodiev. Unmanned aerial vehicle mission simulation in flightgear.

[49] Andrew Wood, Ali Sydney, Peter Chin, Bishal Thapa, and Ryan Ross. Gymfg: a framework with a gym interface for flightgear. *arXiv preprint arXiv:2004.12481*, 2020.

[50] AW Pahlevi, Erwin Susanto, and MR Rosa. Hypersonic flight control with inverse dynamics approach using flightgear. *JMECS (Journal of Measurements, Electronics, Communications, and Systems)*, 9(2), 2022.

[51] AJ MacLeod, Ampere K Hardraade, Michael Koehne, Steve Knoblock, FDM Server, and FDM Instance. A new architecture for flightgear flight simulator.

[52] B Cervin, C Mills, and BC Wünsche. A 3d interface for an unmanned aerial vehicle. *SE Stage*, 4, 2004.

[53] Calvin Coopmans, Michal Podhradskỳ, and Nathan V Hoffer. Software-and hardware-in-the-loop verification of flight dynamics model and flight control simulation of a fixed-wing unmanned aerial vehicle. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 115–122. IEEE, 2015.

[54] Peirui Cao, Xiaoguang Hu, and Guofeng Zhang. Interface research and flight control based on flightgear. In *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 397–402. IEEE, 2017.

[55] Kamal Sharma, C. Praveen Kumar, Rajesh Prasad Verma, Akhilesh Kumar Khan, Shashi Prakash Dwivedi, and Dinesh Singh. Machine learning for fault detection and diagnosis in mechanical systems. In *6th International Conference on Contemporary Computing and Informatics (IC3I)*, 2023.

[56] Pietro Tosato, Daniele Facinelli, Maurizio Prada, Luca Gemma, Maurizio Rossi, and Davide Brunelli. An autonomous swarm of drones for industrial gas sensing applications. In *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 1–6. IEEE, 2019.

[57] Amin Majd, Mohammad Loni, Golnaz Sahebi, and Masoud Daneshtalab. Improving motion safety and efficiency of intelligent autonomous swarm of drones. *Drones*, 4(3):48, 2020.

[58] Amit Raj, Kapil Ahuja, and Yann Busnel. Ai algorithm for predicting and optimizing trajectory of massive uav swarm. *Robotics and Autonomous Systems*, 186:104910, 2025.

[59] Tae-Won Ban, Kyu-Min Kang, and Bang Chul Jung. Multidimensional swarm flight approach for chasing unauthorized uavs leveraging asynchronous deep learning. *IEEE Transactions on Vehicular Technology*, 2025.

[60] Carles Diaz-Vilor, Angel Lozano, and Hamid Jafarkhani. A reinforcement learning approach for wildfire tracking with uav swarms. *IEEE Transactions on Wireless Communications*, 2024.

[61] Ning Gao, Le Liang, Donghong Cai, Xiao Li, and Shi Jin. Coverage control for uav swarm communication networks: A distributed learning approach. *IEEE Internet of Things Journal*, 9(20):19854–19866, 2022.

[62] Marwan Dhuheir, Emna Baccour, Aiman Erbad, Sinan Sabeeh Al-Obaidi, and Mounir Hamdi. Deep reinforcement learning for trajectory path planning and distributed inference in resource-constrained uav swarms. *IEEE Internet of Things Journal*, 10(9):8185–8197, 2023.

[63] Yahao Ding, Zhaohui Yang, Quoc-Viet Pham, Ye Hu, Zhaoyang Zhang, and Mohammad Shikh-Bahaei. Distributed machine learning for uav swarms: Computing, sensing, and semantics. *IEEE Internet of Things Journal*, 11(5):7447–7461, 2024.

[64] Federico Venturini, Federico Mason, Francesco Pase, Federico Chiariotti, Alberto Testolin, Andrea Zanella, and Michele Zorzi. Distributed reinforcement learning for flexible and efficient uav swarm control. *IEEE Transactions on Cognitive Communications and Networking*, 7(3):955–969, 2021.

[65] Longqian Zhao, Bing Chen, and Feng Hu. Research on swarm control based on complementary collaboration of unmanned aerial vehicle swarms under complex conditions. *Drones*, 9:119, 2025.

[66] Huihui Sun, Hao Yan, Mostafa Hassanalian, Junguo Zhang, and Abdessattar Abdelkefi. Uav platforms for data acquisition and intervention practices in forestry: towards more intelligent applications. *Aerospace*, 10(3):317, 2023.

[67] Falk T Gerpott, Sebastian Lang, Tobias Reggelin, Hartmut Zadek, Poti Chaopaisarn, and Sakgasem Ramingwong. Integration of the a2c algorithm for production scheduling in a two-stage hybrid flow shop environment. *Procedia Computer Science*, 200:585–594, 2022.

[68] Xiao Han, Jing Wang, Qinyu Zhang, Xue Qin, and Meng Sun. Multi-uav automatic dynamic obstacle avoidance with experience-shared a2c. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 330–335. IEEE, 2019.

[69] Qiang Yan, Ye Zhang, and Qi Zhao. Multi-uav path planning based on maa2c. *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[70] Chudi Zhang, Biao Yang, Lei Wang, Wenshuai Ji, Lulu Wang, and Shiyou Xu. A cognitive jamming decision-making method based on heuristic improved a2c algorithm. *IEEE Transactions on Vehicular Technology*, 2024.

[71] Jialin Lu, Jing Yang, Shaobo Li, Yijun Li, Wu Jiang, Jiangtian Dai, and Jianjun Hu. A2c-drl: Dynamic scheduling for stochastic edge-cloud environments using a2c and deep reinforcement learning. *IEEE Internet of Things Journal*, 2024.

[72] Prashanth Choppara and Sudheer Mangalampalli. Reliability and trust aware task scheduler for cloud-fog computing using advantage actor critic (a2c) algorithm. *IEEE Access*, 2024.

[73] Xiongjun Liu, Tianyu Yue, and Bai Yang. Research on multi-target allocation algorithm for uav swarms based on ia2c reinforcement learning. In *2024 8th International Conference on Electrical, Mechanical and Computer Engineering (ICEMCE)*, pages 1846–1849. IEEE, 2024.

[74] Shreshth Tuli, Shashikant Ilager, Kotagiri Ramamohanarao, and Rajkumar Buyya. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks. *IEEE transactions on mobile computing*, 21(3):940–954, 2020.

[75] Miaojiang Chen, Tian Wang, Kaoru Ota, Mianxiong Dong, Ming Zhao, and Anfeng Liu. Intelligent resource allocation management for vehicles network: An a3c learning approach. *Computer Communications*, 151:485–494, 2020.

[76] Jianbo Du, Wenjie Cheng, Guangyue Lu, Haotong Cao, Xiaoli Chu, Zhicai Zhang, and Junxuan Wang. Resource pricing and allocation in mec enabled blockchain systems: An a3c deep reinforcement learning approach. *IEEE Transactions on Network Science and Engineering*, 9(1):33–44, 2021.

[77] Dajeong Lee, Junoh Kim, Kyungeun Cho, and Yunsick Sung. Advanced double layered multi-agent systems based on a3c in real-time path planning. *Electronics*, 10(22):2762, 2021.

[78] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Ga3c: Gpu-based a3c for deep reinforcement learning. *CoRR abs/1611.06256*, 2016.

[79] Alfonso B Labao, Mygel Andrei M Martija, and Prospero C Naval. A3c-gs: Adaptive moment gradient sharing with locks for asynchronous actor–critic agents. *IEEE Transactions on Neural Networks and Learning Systems*, 32(3):1162–1176, 2020.

[80] Zihao Fan, Yang Xu, Yuhang Kang, and Delin Luo. Air combat maneuver decision method based on a3c deep reinforcement learning. *Machines*, 10(11):1033, 2022.

[81] Helin Yang, Jun Zhao, Jiangtian Nie, Neeraj Kumar, Kwok-Yan Lam, and Zehui Xiong. Uav-assisted 5g/6g networks: Joint scheduling and resource allocation based on asynchronous reinforcement learning. *IEEE INFOCOM WKSHPS: DroneCom*, 2024.

[82] M Navaneethakrishnan, P Pushpa, TA Mohanaprakash, Batini Dhanwanth, Faraz Ahmed AS, et al. Design of biped robot using reinforcement learning and asynchronous actor-critical agent (a3c) algorithm. In *2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN)*, pages 1–6. IEEE, 2023.

[83] Kun Liang, Guoqiang Zhang, Jinhui Guo, and Wentao Li. An actor-critic hierarchical reinforcement learning model for course recommendation. *Electronics*, 12(24):4939, 2023.

[84] Xue Bai, Chengxuan Lu, Qihao Bao, Shansheng Zhu, and Shaojie Xia. An improved ppo for multiple unmanned aerial vehicles. *Journal of Physics: Conference Series*, 1757:012156, 2021.

[85] Yue Guan, Sai Zou, Haixia Peng, Wei Ni, Yanglong Sun, and Hongfeng Gao. Cooperative uav trajectory design for disaster area emergency communications: A multiagent ppo method. *IEEE Internet of Things Journal*, 11(5):8848–8862, 2024.

[86] Tuo Chen, Feihong Dong, Hu Ye, Yun Wang, and Bin Wu. Data collection mechanism for uav-assisted cellular network based on ppo. *Electronics*, 12:1376, 2023.

[87] Khalil Chikhaoui, Hakim Ghazzai, and Yehia Massoud. Ppo-based reinforcement learning for uav navigation in urban environments. In *IEEE Midwest Symposium on Circuits and Systems (MWSCAS)*, 2022.

[88] Guang Zhan, Xinmiao Zhang, Zhongchao Li, Lin Xu, Deyun Zhou, and Zhen Yang. Multiple-uav reinforcement learning algorithm based on improved ppo in ray framework. *Drones*, 6:166, 2022.

[89] Li Li, Wei Li, Jun Wang, Xiaonan Chen, Qihang Peng, and Wei Huang. Uav trajectory optimization for spectrum cartography: A ppo approach. *IEEE Communications Letters*, 27(6):1575–1579, 2023.

[90] Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International conference on machine learning*, pages 1331–1340. PMLR, 2019.

[91] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[92] Oleksii Zhelo, Jingwei Zhang, Lei Tai, Ming Liu, and Wolfram Burgard. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv preprint arXiv:1804.00456*, 2018.

[93] Nicolas Bougie and Ryutaro Ichise. Skill-based curiosity for intrinsically motivated reinforcement learning. *Machine Learning*, 109:493–512, 2020.

[94] Jing Li, Xinxin Shi, Jiehao Li, Xin Zhang, and Junzheng Wang. Random curiosity-driven exploration in deep reinforcement learning. *Neurocomputing*, 418:139–147, 2020.

[95] Luca A Thiede, Mario Krenn, Akshat Kumar Nigam, and Alán Aspuru-Guzik. Curiosity in exploring chemical spaces: Intrinsic rewards for molecular reinforcement learning. *Machine Learning: Science and Technology*, 3(3):035008, 2022.

[96] Michal Moran and Goren Gordon. Deep curious feature selection: A recurrent, intrinsic-reward reinforcement learning approach to feature selection. *IEEE Transactions on Artificial Intelligence*, 5(3):1174–1187, 2024.

[97] Yadan Luo, Zi Huang, Zheng Zhang, Ziwei Wang, Jingjing Li, and Yang Yang. Curiosity-driven reinforcement learning for diverse visual paragraph generation. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2341–2349. ACM, 2019.

[98] David E Avison, Francis Lau, Michael D Myers, and Peter Axel Nielsen. Action research. *Communications of the ACM*, 42(1):94–97, 1999.

[99] Matthew T Mullarkey and Alan R Hevner. An elaborated action design research process model. *European journal of information systems*, 28(1):6–20, 2019.

[100] Juhani Iivari and John R Venable. Action research and design science research-seemingly similar but decisively dissimilar. 2009.

[101] Louis Cohen, Lawrence Manion, and Keith Morrison. Action research. In *Research methods in education*, pages 440–456. Routledge, 2017.

[102] Aline Dresch, Daniel Pacheco Lacerda, José Antônio Valle Antunes Jr, Aline Dresch, Daniel Pacheco Lacerda, and José Antônio Valle Antunes. *Design science research*. Springer, 2015.

[103] Abdulrahman Nahhas, Andrey Kharitonov, and Klaus Turowski. Deep reinforcement learning techniques for solving hybrid flow shop scheduling problems: Proximal policy optimization (ppo) and asynchronous advantage actor-critic (a3c). 2022.

[104] Kun Zhou, Wenyong Wang, Teng Hu, and Kai Deng. Application of improved asynchronous advantage actor critic reinforcement learning model on anomaly detection. *Entropy*, 23(3):274, 2021.

[105] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

[106] Yuhui Wang, Hao He, and Xiaoyang Tan. Truly proximal policy optimization. In *Uncertainty in artificial intelligence*, pages 113–122. PMLR, 2020.

[107] Yang Gu, Yuhu Cheng, CL Philip Chen, and Xuesong Wang. Proximal policy optimization with policy feedback. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(7):4600–4610, 2021.

[108] Junwei Zhang, Zhenghao Zhang, Shuai Han, and Shuai Lü. Proximal policy optimization via enhanced exploration efficiency. *Information Sciences*, 609:750–765, 2022.

[109] Zifan Wu, Chao Yu, Deheng Ye, Junge Zhang, Hankz Hankui Zhuo, et al. Coordinated proximal policy optimization. *Advances in Neural Information Processing Systems*, 34:26437–26448, 2021.

[110] Perttu Hämäläinen, Amin Babadi, Xiaoxiao Ma, and Jaakko Lehtinen. Ppo-cma: Proximal policy optimization with covariance matrix adaptation. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2020.

[111] Sanghyun Son, Laura Zheng, Ryan Sullivan, Yi-Ling Qiao, and Ming Lin. Gradient informed proximal policy optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

[112] Yuhu Cheng, Longyang Huang, and Xuesong Wang. Authentic boundary proximal policy optimization. *IEEE Transactions on Cybernetics*, 52(9):9428–9438, 2021.

[113] Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural trust region/proximal policy optimization attains globally optimal policy. *Advances in neural information processing systems*, 32, 2019.

[114] Dong Liushuan. Ellipsoidal method for uavs target tracking and recognition. *IEEE Access*, 10:113935–113944, 2022.

[115] Yi Xia, Xingling Shao, Tianyun Ding, and Jun Liu. Prescribed intelligent elliptical pursuing by uavs: A reinforcement learning policy. *Expert Systems with Applications*, 249:123547, 2024.

[116] Xiao Zhifeng and Jianwang Hong. Ellipsoidal method for uavs target tracking and recognition. *Systems Science & Control Engineering*, 11(1):2135631, 2023.

[117] Dechun Lu, Qingtao Lin, Yu Tian, Xiuli Du, and Qiuming Gong. Formula for predicting ground settlement induced by tunnelling based on gaussian function. *Tunnelling and Underground Space Technology*, 103:103443, 2020.

[118] Gregory G Slabaugh. Computing euler angles from a rotation matrix. *Retrieved on August*, 6(2000):39–63, 1999.

[119] Joost Zwarts. Vectors as relative positions: A compositional semantics of modified pps1. *Journal of semantics*, 14(1):57–86, 1997.

[120] Jawad Mahmood, Muhammad Adil Raja, John Loane, and Fergal McCaffery. A configurable intrinsic curiosity module for a testbed for developing intelligent swarm uavs. *Machine Learning with Applications*, 21:100714, 2025.

[121] Jawad Mahmood, Muhammad Adil Raja, John Loane, and Fergal McCaffery. A multi-objective scheme for collision avoidance, swarm cohesion, and target tracking for smart uavs. *Journal of Information and Intelligence*, 2025.

[122] Gabriel Kabanda, COLLETOR TENDEUKAI CHIPFUMBU, and Tinashe Chingoriwo. A reinforcement learning paradigm for cybersecurity education and training. *Oriental Journal of Computer Science and Technology*, pages 12–45, 2023.

[123] Jinwen Hu, Liyuan Fan, Yifei Lei, Zhao Xu, Wenxing Fu, and Gang Xu. Reinforcement learning-based low-altitude path planning for uas swarm in diverse threat environments. *Drones*, 7(9):567, 2023.

[124] Kenta Takaya, Toshinori Asai, Valeri Kroumov, and Florentin Smarandache. Simulation environment for mobile robots testing using ros and gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101. IEEE, 2016.

[125] Yujing Zhou, Yupeng Yang, Bill Deng Pan, Yongxin Liu, Sirish Namilae, Houbing Herbert Song, and Dahai Liu. Deep reinforcement learning for real-time airport emergency evacuation using asynchronous advantage actor–critic (a3c) algorithm. *Mathematics*, 13(14):2269, 2025.

[126] Hung-Chin Jang, Yi-Chen Huang, and Hsien-An Chiu. A study on the effectiveness of a2c and a3c reinforcement learning in parking space search in urban areas problem. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 567–571. IEEE, 2020.

[127] Hongjian Wang, Wei Gao, Zhao Wang, Kai Zhang, Jingfei Ren, Lihui Deng, and Shanshan He. Research on obstacle avoidance planning for uuv based on a3c algorithm. *Journal of Marine Science and Engineering*, 12(1):63, 2023.

[128] Yang Pan, Weiye Wang, Yanbin Li, Feng Zhang, Yanting Sun, and Dunnan Liu. Research on cooperation between wind farm and electric vehicle aggregator based on a3c algorithm. *IEEE Access*, 9:55155–55164, 2021.

[129] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[130] Alberto Del Rio, David Jimenez, and Javier Serrano. Comparative analysis of a3c and ppo algorithms in reinforcement learning: A survey on general environments. *IEEE Access*, 2024.

[131] Zhihui Liu, Qiwei Zhang, and Yi Su. Ppo-based joint optimization for uav-assisted edge computing networks. *Applied Sciences*, 13(23):12828, 2023.

[132] Zehua Ye, Dan Zhang, Zheng-Guang Wu, and Huaicheng Yan. A3c-based intelligent event-triggering control of networked nonlinear unmanned marine vehicles subject to hybrid attacks. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):12921–12934, 2021.