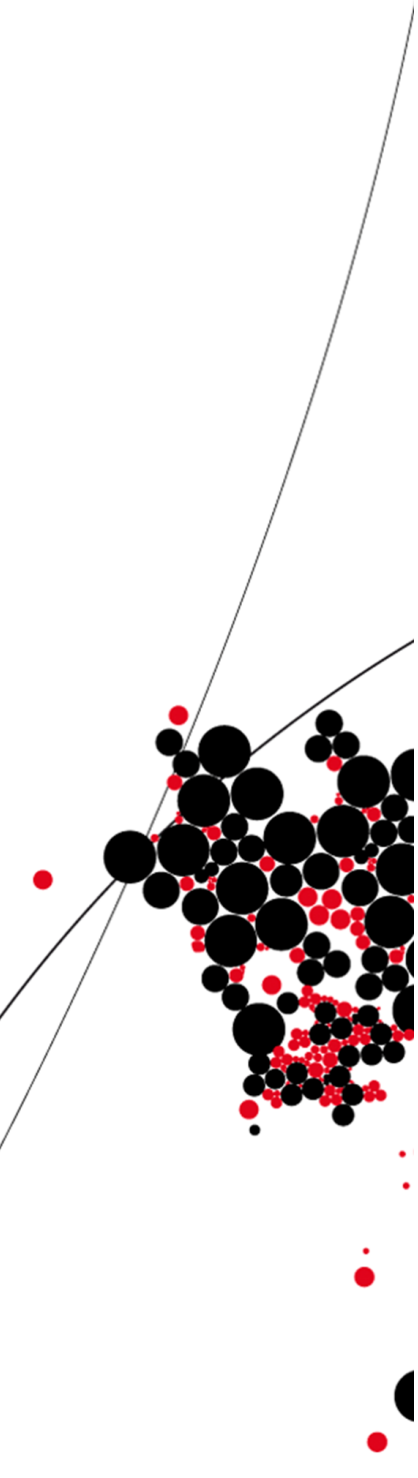
UNIVERSITY OF TWENTE.

Faculty of Geo-Information Science and Earth Observation

Pretraining Deep Reinforcement Learning with Imitation Learning for Autonomous UAV Exploration in Unknown Indoor Environments

Desiree N. Pietersma

M.Sc. Thesis
August 28, 2025



Supervisors:
Prof. Dr. Ing. F.C. Nex, ITC-EOS
B.L.U. Udugama Vithanage MSc, ITC-EOS
Dr. C. Gabellieri PhD, EEMCS-EE-RAM

Earth Observation Science (EOS) Department
Faculty of Geo-Information Science and Earth Observation
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Autonomous Unmanned Aerial Vehicles (UAVs) have great potential for indoor exploration tasks, such as search and rescue, surveillance, and inspection due to their compact size, operational flexibility, and high maneuverability, with autonomy allowing them to operate without the need for human expertise.

Traditional exploration methods struggle with performance and higher computational costs, while Deep Reinforcement Learning (DRL) offers better performance but suffers from instability and struggles with convergence.

This thesis proposes a hybrid approach that integrates Behavior Cloning (BC), a form of Imitation Learning (IL), with the Soft Actor-Critic (SAC) DRL algorithm for continuous control to improve UAV exploration performance, robustness, and generalization. The designed method begins by collecting expert trajectories and using them to pretrain a policy with IL. The pretrained policy is then fine-tuned with DRL in a photorealistic indoor environment in NVIDIA Isaac Sim. The state given to the algorithm includes semantic visual inputs with depth perception to enhance situational awareness. Experiments show that the pretrained IL policy with DRL converges faster and achieves higher coverage and success rates compared to baselines trained purely with DRL or IL. The results also show that incorporating semantic visual inputs enhances exploration performance by providing structured high-level environmental information.

In conclusion, this thesis shows the benefits of combining IL with RL in complex, unknown indoor environments, laying the foundation for deploying autonomous UAVs in mission-critical scenarios.

Contents

List of Acronyms

UAV	Unmanned Aerial Vehicle[s]
SAR	Search and Rescue
DRL	Deep Reinforcement Learning
IL	Imitation Learning
CW	Clock-wise
CCW	Counter Clock-wise
RL	Reinforcement Learning
MDP	Markov Decision Process
DQN	Deep Q-Networks
TRPO	Trust Region Policy Optimization
FEC	Fast Euclidean Clustering
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed DDPG
SAC	Soft Actor-Critic
PPO	Proximal Policy Optimization
RRT	Rapidly exploring Random Trees
NBV	Next Best View
BC	Behavior Cloning
DAgger	Dataset Aggregation
UGV	Unmanned Ground Vehicle
IRL	Inverse Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
CD	Curiosity-Driven learning
RND	Random Network Distillation
EE	Entropy Exploration
IC	Intrinsic Curiosity
ICM	Intrinsic Curiosity Module
EBE	Entropy Based Exploration
USD	Universal Scene Description
MSE	Mean Squared Error
CNN	Convolutional Neural Network

1 Introduction

Search and Rescue (SAR) operations are often races against time. Any delay can result in serious consequences, potentially life-threatening. These missions often occur in complex, confined, and hazardous environments such as disaster scenes, collapsed buildings or underground facilities [1]. In such high-stakes settings, the safety of rescue teams is as vital as the success of the mission itself.

Unmanned Aerial Vehicles (UAVs) are widely used in various fields due to their compact size, operational flexibility, and high maneuverability. These domains include SAR operations, infrastructure inspection, and surveillance. UAVs are particularly valuable in these applications, because they can take on roles too dangerous for humans by entering hazardous and complex environments, collecting critical information without the limitations imposed by ground-based exploration [2].

In SAR operations specifically, UAVs can provide invaluable assistance to rescue teams by providing real-time situational maps in time-sensitive situations and unknown environments. UAVs can autonomously explore the surroundings, locate and identify victims, and detect danger sources, all without endangering the rescue team. This enables the rescue teams to perform their task efficiently and safely. During these operations, UAVs can generate detailed environmental maps, capturing structural layouts, spatial connections, and semantic information. By integrating robot pose and sensor data, these maps can be used for path planning and further exploration to optimally help people [3, 4].

Their use case in SAR operations has already been demonstrated and proven after the 2016 earthquake in Amatrice, Italy. Small drones explored partially collapsed churches. These UAVs collected data that was used to generate 3D structural models, allowing experts to assess collapse risks and plan safe entry for responders. This example highlights the ability of UAVs to navigate hazardous indoor environments and provide critical information in disaster scenarios [5].

Indoor drone exploration also poses several challenges. Indoor environments typically feature both static and dynamic obstacles, such as furniture, moving objects, or people, which require drones to navigate with precision and adaptability. Limited space further complicates maneuvering, while communication latency or signal loss in indoor or disaster environments can disrupt remote control. Additionally, UAVs have restricted battery capacity, requiring rapid decision-making [5, 6].

Traditionally, these tasks require an experienced human pilot. However, relying on human expertise is often impractical, not only during emergencies but also in routine or remote operations where such expertise may not be feasible or readily available. This limitation underscores the need for autonomous drone systems.

Autonomy enables UAVs to address these challenges by performing all computation onboard, allowing them to function even in the absence of stable communication. Furthermore, autonomous exploration algorithms can mimic or exceed expert-level performance without requiring a human expert [6].

1.1 Autonomous Exploration Methods

Autonomous exploration methods in recent articles can be divided into classical and deep learning methods. In classical methods, potential candidates for the next exploration target are evaluated and selected based on a function or other measure. It can be broadly categorized into frontier-based exploration and information-theory based exploration. The first identifies exploration target based on the boundaries between free space and unexplored space (frontiers), while the latter identifies the targets based on maximizing information gain [3]. While these methods are easy to interpret and proven in static environments [3, 7], they are however computationally expensive so it is difficult to perform in real-time scenarios, due to the low update frequency. Often only the short-term information gain is taken into account, therefore making them globally non-optimal and reducing the exploration efficiency [7, 8].

The current state-of-the-art methods shift more towards Deep Reinforcement Learning (DRL) due to better performance [9]. In these methods, UAVs are trained using sensor data to autonomously learn and adapt their exploration strategies through trial and error [10]. This method is gaining more attention since it can adaptively optimize strategies in complex environments due to its learning capabilities and it can also conduct end-to-end learning directly from raw sensor data, which further enhances the robustness of the control strategies. It also is more efficient in exploration [11].

DRL offers a promising solution for the exploration challenges of a UAV, particularly in time-sensitive decision making and in dynamic environments. Drones learn to make quick decisions, maneuver around static obstacles and anticipate moving ones, continuously adjusting their actions to improve performance [4, 9].

However, despite these advantages, DRL algorithms often suffer from sample inefficiency and training instability, with performance heavily dependent on hyperparameter tuning and reward design. Moreover, the balance between exploration and exploitation remains a core challenge. Another problem is the sim-to-real transfer gap. Policies that perform well in simulation often fail when deployed on real drones due to differences in dynamics, noise, and perception [12, 13].

Despite these limitations, DRL remains the more promising approach for autonomous UAV exploration, particularly in dynamic, time-critical, and unstructured environments where classical methods struggle [10]. Recent advancements in literature try to mitigate DRL's limitations by adding techniques such as Hierarchical Learning (breaking tasks into subtasks), Imitation Learning (learning from expert demonstrations), and enhanced exploration strategies [14].

1.2 Goal

Indoor exploration with a UAV is challenging, due to limitations of the drone itself and the complex surroundings. Deep Reinforcement Learning offers an effective approach to address these challenges, allowing the creation of a policy that safely and efficiently explores the indoor surroundings.

The goal of this project is to develop and evaluate a DRL algorithm that integrates both Imitation Learning (IL) and Reinforcement Learning (RL) for improved UAV exploration in unknown indoor environments. By using a hybrid IL-RL approach, this thesis aims to enhance exploration efficiency and adaptability and solve challenges such as generalization and real-world applicability. Isaac Sim, a physics-accurate simulation environment, is used to bridge the gap between the virtual training and the real-world.

1.3 Report Structure

The following section outlines the structure of this report, providing an overview of each chapter and its focus.

Chapter 2 provides background knowledge to understand the dynamics of a drone and the principle of RL, it also covers the state-of-the-art exploration methods, covering both traditional approaches and those based on DRL. The chapter concludes with a consideration on why the IL-RL hybrid method was chosen for this thesis. Chapter 3 explains the problem formulation and the research questions.

Chapter 4 explains the environmental setup in Isaac Sim, here the training and testing environments are discussed, the UAV setup, and the training setup. The methodology of the developed method is explained in Chapter 5. The different stages for the IL-RL hybrid method are explained. The experiments are discussed in Chapter 6.

Chapter 7 presents the outcomes of the experiments, followed by the discussion of the findings and the future directions in Chapter 8. Finally, Chapter 9 concludes the thesis.

2 Background and State-of-the-Art

2.1 Background Knowledge

The following section contains background knowledge for the dynamics of the UAV and for RL.

2.1.1 UAV Dynamics

The algorithm developed in this thesis is designed for deployment on underactuated quadrotors. To effectively work with quadrotors, it is important to understand their underlying dynamics. Figure 2.1 shows a DJI Tello EDU drone. In the image the drone body frame can be seen $F^D = \{O^D, \vec{x}_D, \vec{y}_D, \vec{z}_D\}$ and the propeller frames $F^{R_i} = \{O^i, \vec{x}_i, \vec{y}_i, \vec{z}_i\}$ for $i = 0, 1, 2, 3$. Additionally, a fixed world frame $F^W = \{O^W, \vec{x}_W, \vec{y}_W, \vec{z}_W\}$ is defined. The following equations and conventions are based on the lecture notes from the Aerial Robotics course at the University of Twente [15].

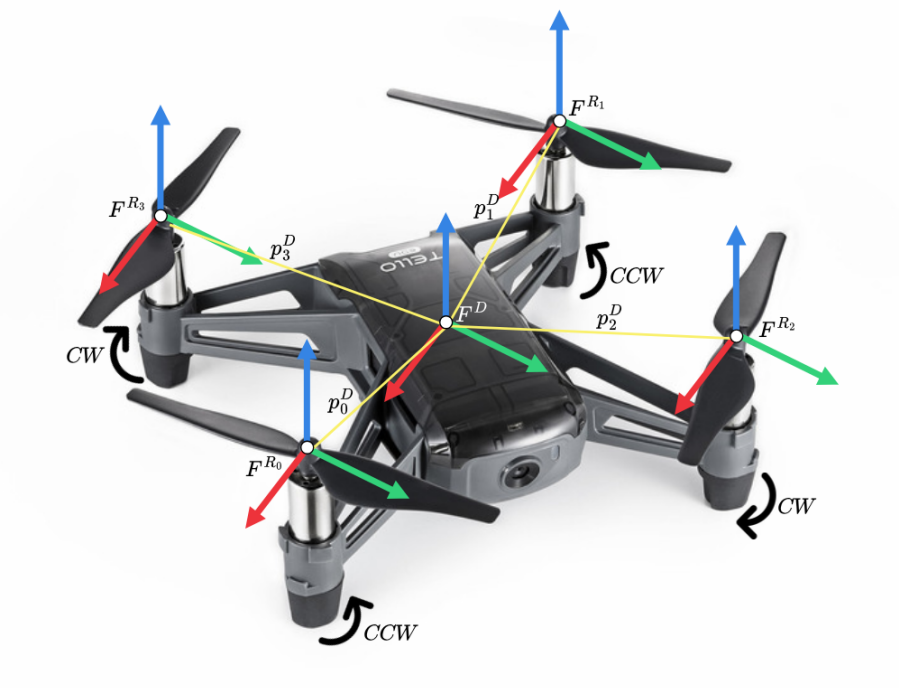


Figure 2.1: Side view of the Tello EDU drone showing the drone-attached reference frames: the body frame F^D and the propeller frames F^{R_i} for $i = 0, 1, 2, 3$. The red, green, and blue colors represent the x-, y-, and z-axes. The yellow lines indicate the distance from the propeller to the center of mass of the drone in the body frame. Each propeller also has an indication for the rotation direction, Clock-wise (CW) or Counter Clock-wise (CCW).

In general, for drones, the combined effect of forces and torques acting on the system is expressed as a wrench. The thrust force generated by propeller i, denoted as $\vec{f}_{P_i}$, is given by:

$$\vec{f}_i^R = c_f \vec{z}_i^R u_{\lambda i}, \tag{2.1}$$

where, i refers to the index of the propeller, c_f is the thrust coefficient and depends on the type and geometry of the propeller, $u_{\lambda i}$ is the velocity input control for propeller i and $\vec{z}_i^R$ is a unit vector indicating the thrust direction of propeller i.

There also exists a drag moment that exerts on the propeller:

$$\vec{m}^R_{\text{drag}_i} = -k_i c_\tau \vec{z}^R_i \tag{2.2}$$

where, c_τ is the drag coefficient and $k_i \in \{-1, 1\}$ indicates the spin direction of the propeller (ascending or descending).

The moment a propeller then generates on the body frame of the drone consists of the drag moment and the moment produced by the thrust force:

$$\vec{m}^D_i = \vec{m}_{\text{drag}_i} + \vec{m}_{\text{thrust}_i} = \left(-k_i c_\tau \vec{z}^D_i + c_f \, \vec{p}^D_i \times \vec{z}^D_i\right) u_{\lambda i}, \tag{2.3}$$

where, $\vec{p}^D_i$ is the position vector from the UAV's center of mass to the i-th propeller.

For a quadrotor, the total wrench $\vec{W^D}$ applied to the system in the drone body frame F^D becomes:

$$\vec{W^D}(\vec{u}) = \begin{bmatrix} \vec{f^D}(\vec{u}) \\ \vec{m^D}(\vec{u}) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N c_{f,i} \vec{z}^D_i(\vec{u}_v) u_{\lambda i} \\ \sum_{i=1}^N \left(-k_i c_{\tau i} \vec{z}^D_i(\vec{u}_v) + c_{f,i} \vec{p}^D_i \times \vec{z}^D_i(\vec{u}_v)\right) u_{\lambda i} \end{bmatrix} = A(\vec{u}_v)\vec{u}_\lambda \tag{2.4}$$

Here, the matrix $A(\vec{u}_v) \in \mathbb{R}^{6\times N}$ maps the control inputs to the resulting wrench $\vec{W}^D(\vec{u})$ in the drone body frame. The input vector is defined as $\vec{u} = [\vec{u}_v, \vec{u}_\lambda]^T \in \mathbb{R}^{K+N}$, where $\vec{u}_v = [u_{v_j}]_{j=0}^{K-1} \in \mathbb{R}^K$ represents the joint positions of the drone's servomotors, and $\vec{u}_\lambda = [u_{\lambda_i}]_{i=0}^{N-1} \in \mathbb{R}^N$ corresponds to the inputs associated with the rotor speeds. Each u_{λ_i} is defined as $u_{\lambda_i} = \omega_i|\omega_i|$, where $\omega_i \in \mathbb{R} > 0$ denotes the angular velocity of the i-th propeller.

In the case of an underactuated quadrotor, the number of propellers is $N = 4$. Furthermore, no servomotors are used for thrust vectoring, so $\vec{u}_v = \emptyset$ and $\vec{u} = \vec{u}_\lambda = [u_{\lambda 0}, u_{\lambda 1}, u_{\lambda 2}, u_{\lambda 3}]^T \in \mathbb{R}^4$.

Consequently, as shown in Equation 2.4, the matrix $A(\vec{u}_v) = A = \frac{d\vec{W}^D(\vec{u})}{d\vec{u}}$ becomes constant and equal to the allocation matrix, as $\vec{W}^D(\vec{u})$ is linearly dependent on $\vec{u}$. Additionally, all the propellers are aligned with the drone's body frame F^D, so that $\forall i : \vec{z}^D_i = \vec{z}^D = [0, 0, 1]^T$.

From Figure 2.1, we observe that propellers 0 and 1 rotate CCW, while propellers 2 and 3 rotate CW. Thus, the spin directions are defined as:

$$k_0 = k_1 = -1, \quad k_2 = k_3 = 1.$$

The wrench can then be rewritten as:

$$\vec{W}^D(\vec{u}_\lambda) = A\vec{u}_\lambda = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_f & c_f & c_f & c_f \\ 0 & 0 & c_f L & -c_f L \\ -c_f L & c_f L & 0 & 0 \\ -c_\tau & -c_\tau & c_\tau & c_\tau \end{bmatrix} \begin{bmatrix} u_{\lambda 0} \\ u_{\lambda 1} \\ u_{\lambda 2} \\ u_{\lambda 3} \end{bmatrix} = \begin{bmatrix} \vec{0}_{2\times 4} \\ \tilde{A} \end{bmatrix} \vec{u}_\lambda = \begin{bmatrix} 0 \\ 0 \\ f \\ m_x \\ m_y \\ m_z \end{bmatrix} \tag{2.5}$$

where $\tilde{A}$ corresponds to the effective actuation matrix for force and moments. L is the distance from a propeller to the center of mass. From this we can define $\vec{\tilde{u}} = (f, m_x, m_y, m_z)^T = \tilde{A}\vec{u}_\lambda$

The first two rows of the force vector are zero because the propellers produce thrust only in the z-direction of the body frame, due to the lack of tilt mechanisms. Rows 4 and 5 correspond to moments m_x and m_y, which are generated by the opposing pairs of propellers positioned along the y- and x-axes, respectively. Specifically, propellers 0 and 1 contribute to m_y since they are placed along the x-axis, while propellers 2 and 3 contribute to m_x since they are along the y axis.

To translate the wrench into motion, the quadrotor must align the thrust force $\vec{f}^D = [0,\ 0,\ f]^T$ with the desired trajectory by generating appropriate moments $\vec{m} = [m_x,\ m_y,\ m_z]^T$. Since horizontal motion requires tilting the drone, it can only be achieved by indirectly steering thrust via body rotations. The full dynamic equations of the quadrotor in the world frame F^W are:

$$\begin{bmatrix} \ddot{\vec{p}}^W \\ \dot{\vec{\omega}}_D^{DW} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \\ -J_D^{-1}(\vec{\omega}_D^{DW} \times J_D \vec{\omega}_D^{DW}) \end{bmatrix} + \begin{bmatrix} \frac{1}{m} R_D^W & 0_{3\times 3} \\ 0_{3\times 3} & J_D^{-1} \end{bmatrix} \begin{bmatrix} \vec{0}_{2\times 4} \\ \tilde{A} \end{bmatrix} \vec{u}_\lambda \tag{2.6}$$

where, $\ddot{\vec{p}}^W$ is the linear acceleration of the drone's center of mass expressed in the world frame, $\dot{\vec{\omega}}_D^{DW}$ is the angular acceleration of the drone body frame F^D, relative to the world frame and expressed in the body frame. g is the acceleration due to gravity, J_D is the moment of inertia matrix of the drone in the body frame, $\vec{\omega}_D^{DW}$ is the angular velocity of the drone body frame relative to the world frame, expressed in the body frame. m is the mass of the drone and R_D^W is the rotation matrix from the drone body frame F^D to the world frame F^W.

Finally, since the system is underactuated and the wrench map is a constant linear relation in the inputs, the partial allocation matrix $\tilde{A}$ can be inverted to retrieve the input $\vec{u}_\lambda$ from a desired $(f, m_x, m_y, m_z)^T$, as long as $c_f \neq 0$, $c_\tau \neq 0$, and $L \neq 0$. This provides a foundation for implementing control strategies.

2.1.2 Reinforcement Learning

RL is a machine learning paradigm where an agent, in this case the UAV, interacts with the environment by observing its state, taking actions, and receiving feedback in the form of rewards [16].

In RL the goal is to find the policy π that maximizes the expected return $J(\pi)$ for a trajectory τ. The return is defined as the expected cumulative reward $R(\tau)$ received along this trajectory [17]:

$$J(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{2.7}$$

With γ the discount factor, this factor determines how much future states are taken into account, r_t is the reward for a given timestep and T is the time horizon. The optimal policy π^* can then be formulated as [17]:

$$\pi^* = \arg\max_\pi J(\pi) = \arg\max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] \tag{2.8}$$

In many RL methods, learning is guided by estimating value functions, which quantify how good it is to be in a certain state or to take a certain action from a state. These value functions are the state-value function and the action-value function.
The state-value function $V^\pi(s)$ is the expected return starting from state s and following policy π [17].

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau) \mid s_0 = s] \tag{2.9}$$

The action-value function $Q^\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π [17]:

$$Q^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau) \mid s_0 = s, a_0 = a] \tag{2.10}$$

RL is described as a Markov Decision Process (MDP). This is a tuple $\mathcal{M} = (S, A, P, R, \gamma)$ [17], where:

- S is the set of all valid states
- A is the set of all valid actions
- P is the transition probability function
- R is the reward function
- γ is the discount factor

RL algorithms can be mainly categorized into value-based methods and policy-based methods:

- **Value-Based Methods:** These methods focus on learning a value function or action-value function that estimates the return from each state. The policy is indirectly derived by choosing actions that maximize the estimated value. Common algorithms are SARSA, Q-learning and Deep Q-Networks (DQN).
- **Policy-Based Methods:** These methods learn the policy $\pi(a|s)$ directly and optimize it by adjusting its parameters to maximize the expected return. Common algorithms are REINFORCE, Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO).

While traditional RL algorithms are powerful, they often struggle with large or continuous state and action spaces. DRL is a deep learning approach to RL and can solve this problem. Through a neural network, the DRL algorithm approximates the functions needed for learning, such as the value function, the action-value function or the policy. This is especially relevant in complex tasks such as autonomous UAV navigation, where the state might include high-dimensional sensor data, and the action space can be continuous [18].

Some methods, known as actor-critic algorithms, combine both approaches by using a policy network (actor) and a value network (critic). A few examples are Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), and Twin Delayed DDPG (TD3) [18].

2.1.3 Types of Deep Reinforcement Learning Algorithms

Many different types of RL methods exist, for example, model-based methods and model-free methods. Model-based methods learn a model of the environment's dynamics. This includes the transition probabilities and rewards, which is then used for planning and decision-making. Model-free methods bypass this modeling step and directly learn policies or value function from experience, so the transition probabilities and rewards are unknown. These methods offers more simplicity and robustness where modeling is difficult, making it more commonly used in practice [18].

RL algorithms can also differ in how they learn the policy, falling into either on-policy or off-policy categories. In on-policy learning, a neural network policy collects data by interacting with the environment using its current behavior, and updates are made using this same data. This leads to stable and unbiased gradient estimates, but the approach is often data inefficient since each sample is used only once. A well-known example of an on-policy method is PPO. Off-policy methods, such as Q-learning and actor-critic algorithms like SAC, DDPG, and TD3, improve sample efficiency by storing experiences in a replay buffer and learning from past interactions. While more efficient, these methods can be less stable and require careful tuning [19].

The most common algorithms in current literature are listed:

- **Proximal Policy Optimization (PPO):** is an on-policy algorithm that updates the policy using data collected from its current behavior. It stabilizes training by limiting how much the policy can change at each update through clipping in the objective function, preventing large, destabilizing steps. PPO also uses Generalized Advantage Estimation to reduce variance in policy gradient estimates, leading to more stable and reliable updates [20].
- **Deep Deterministic Policy Gradient (DDPG):** is an off-policy actor-critic method designed for continuous action spaces. It uses a replay buffer to improve sample efficiency and uses slowly updated target networks (via Polyak averaging) to stabilize learning [21].
- **Twin Delayed DDPG (TD3):** builds on DDPG by using two critic networks instead of one. During updates, it takes the minimum of their value estimates to reduce overestimation bias, which helps improve training stability and policy performance [22].
- **Soft Actor-Critic (SAC):** is also an off-policy actor-critic algorithm but adds an entropy term to the reward, encouraging the policy to explore more by remaining stochastic during training. It also uses two critic networks like TD3 to reduce overestimation bias and improve stability [23].

2.2 State-of-the-Art

This section reviews the current state-of-the-art in autonomous exploration. First, traditional exploration methods are discussed, followed by DRL approaches.

2.2.1 Traditional Exploration Methods

Traditional exploration methods can be broadly categorized into frontier-based exploration and information-theory-based exploration. Hybrid approaches that integrate both methods also exist [3].

Frontier-Based Exploration
In frontier-based exploration, a UAV identifies frontiers. Frontiers are the boundaries between known free space and unexplored space. Figure 2.2 shows a schematic of this process, where frontiers are detected and the UAV navigates toward the selected frontier. Navigation to these frontiers can be achieved through various strategies. One approach involves evaluating a cost function to maximize map coverage [3]. Alternatively, frontier selection can be based on proximity, such as choosing the nearest frontier with Euclidean distance [24] or selecting the centroid of a frontier [7, 25]. To detect frontiers, different types of map making methods are used, ranging from occupancy grid maps [3, 7, 24–27], topological maps [28, 29] and methods based on feature maps [30].

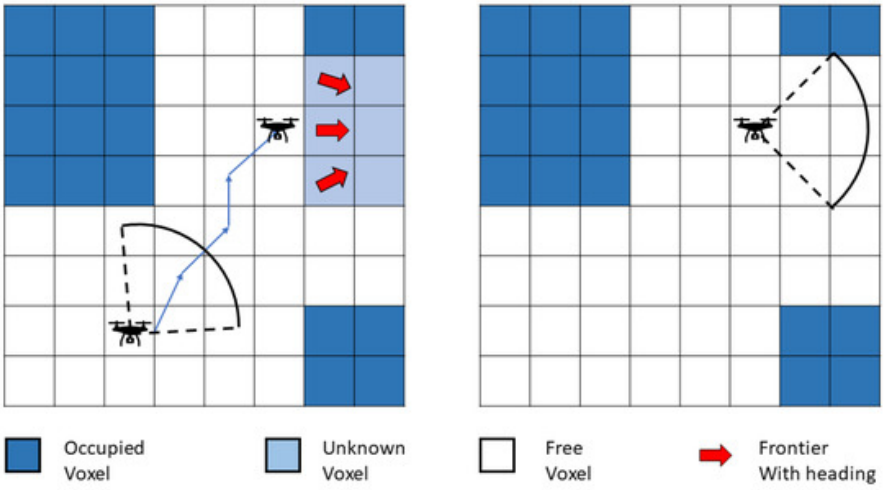


Figure 2.2: Schematic diagram of the frontier-based exploration process. The robot starts exploration by moving to the selected frontier and finishes exploration when there is no new frontier found [31]

However, frontier-based methods have several challenges. Identifying and clustering frontiers is computationally demanding, which can increase planning times and limit real-time responsiveness. This can prevent UAVs from adapting quickly to environmental changes. Additionally, UAVs with this method tend to repeatedly explore the same area due to getting stuck in local minima, which reduces efficiency. Furthermore, most frontier-based methods assume a static environment, which is an unrealistic assumption in real-world applications [24, 26].

To mitigate these limitations, more efficient methods have been proposed. For instance, [27] introduces a technique that avoids processing the entire map, reducing the computational burden as the map grows during exploration. Other researchers have integrated LiDAR data to

enhance mapping accuracy and frontier detection [24, 32]. Hierarchical strategies have been employed to reduce redundant paths by selecting guidance points towards frontiers [25].

As mentioned earlier, different map types have been explored beyond the more common occupancy maps, including topological [28, 29] and feature maps [30]. However, topological maps typically build on occupancy maps, increasing computational demands. The authors of the frontier-based algorithm with the feature map claim higher efficiency and effectiveness, yet their evaluations are limited to simple scenarios, such as a long corridor and a corner, which do not fully capture the complexity of frontier detection [30].

Another example is presented by the authors of [26]. They propose a method that uses UFOMap for frontier detection and Fast Euclidean Clustering (FEC) for real-time scheduling. While this approach is able to achieve a faster response time, it has limitations. It has not been tested in dynamically changing environments, and without additional feature points from ArUco markers in the map, it may struggle in large open areas potentially leading to incomplete exploration.

Despite various efforts to improve the frontier-based exploration method, the challenges still remain [4, 9].

Information-Theory-Based Exploration

Information-theory-based exploration focuses on navigating towards candidate positions that maximize information gain using probabilistic maps to reduce uncertainty or entropy [3]. These candidate positions can be randomly sampled with Rapidly exploring Random Trees (RRT) [33], (RRT*) [34], which is a sampling-based path planning algorithm that efficiently explores high-dimensional spaces by incrementally building a tree through random sampling.

Figure 2.3: Visualization of Rapidly-exploring Random Tree (RRT) expansion (grey) around obstacles (black). The yellow path indicates the selected branch points leading to the Next Best View (NBV) [35]

The candidate position with the highest information gain is then determined, this view is then taken as the position of interest. This process of selecting the best position is called the Next Best View (NBV) method [36, 37]. In the original method, the information gain was determined to be the unmapped volume that can be explored [37]. Figure 2.3 illustrates the RRT branches (grey) and the branch points (yellow) used to navigate the environment with NBV.

While the complexity of planning and sampling in NBV does not scale significantly with environment size, the information gain of each viewpoint requires a computationally intensive evaluation. This causes increased latency in decision-making [8, 38]. Additionally, the random nature of RRT makes the exploration trajectory suboptimal, which renders the overall exploration time unpredictable and extends the computational time. This method produces many invalid viewpoints, which affect the efficiency and exploration [26].

Although information-theoretic approaches typically rely on probabilistic maps for efficiency, 3D semantic maps have also been proposed as an alternative [3]. A method to improve upon the intensive computations has also been tried [36]. Instead of computing the unmapped volume that can be explored, a cuboid-based estimation method has been proposed. This approach evaluates the potential information gain along each edge of the RRT. By assessing broader regions instead of individual points, it reduces planning time, but it still struggles with revisiting regions [36]. A paper by Witting, C. et al. [39] keeps track of its history for potential places to reseed the tree, this is done in order to avoid revisiting places. Another paper [40] tried to improve upon this by keeping and maintaining one tree that they continually update. This appeared to improve the problem of revisiting locations but increased the computational load by keeping track of its history.

Hybrid methods that combine frontier-based and information-theory-based approaches are commonly found in the literature. In [38], the NBV method does not determine candidate targets using the RRT method but instead chooses among detected frontiers. The best frontier is selected based on information gain, travel cost, and proximity to other agents. While promising, the authors noted that there was still much room for more efficient real-time exploration. Lu, L. et al. used a similar method.

A different approach by Tran, D. et al. [41], used the frontiers to guide the RRT trees, this caused a decrease in the number of potential candidates and decreased the computation cost to determine the NBV by enhancing the sampling. In [31], the authors used entropy as the main criterion of the objective function. They defined entropy as the number of unknown cells/voxels in the neighborhood of a candidate point determined through frontiers. Another source [42] also used entropy in the cost function, along with path cost, to minimize fuel consumption.

These hybrid algorithms, however, aim to maximize information gain and go to the next best frontier. This causes them to be globally non-optimal. A method to solve this has been developed by considering all observed candidate points rather than only the most immediate ones [43]. While this strategy improves efficiency, it significantly increases computational cost [7].

2.2.2 Deep Reinforcement Learning

Recent research in autonomous UAV exploration has increasingly focused on the implementation of DRL. The current state-of-the-art in DRL for indoor UAV exploration primarily involves additions to base algorithms, such as PPO[20], DDPG[21], TD3[22], and SAC[23]. Of these algorithms, SAC is often shown to be the most effective and converges the fastest, since it uses entropy for improving exploration [44, 45].

DRL algorithms face challenges in UAV exploration, due to the large state-action space and sparse rewards. Real-world environments are complex and rewards tend to be sparse, often leading to failure in learning. To address these issues, various mechanisms have been introduced to improve DRL performance. These include IL techniques, Hierarchical Reinforcement

Learning (HRL) techniques, and specialized exploration algorithms [14].

Imitation Learning

Imitation learning is a method where a policy is learned directly from expert demonstrations, without relying on rewards. This allows it to rapidly learn an optimal policy. The process starts by collecting expert demonstrations, then feeding the observations into a network. The network predicts an action, and the loss is computed by comparing this prediction to the expert's action [46]. Equation 2.11 shows how imitation learning is a typical a supervised learning problem:

$$\pi_\theta = \arg\min_\theta \sum_{\xi \in \Xi} \sum_{s \in \xi} \mathcal{L}(\pi_\theta(s), \pi^*(s)) \tag{2.11}$$

where π_θ is the policy to be learned. The objective is to minimize a loss function $\mathcal{L}$ that measures the discrepancy between the learned policy π_θ and the expert policy π^* across a set of expert demonstrations. Each expert trajectory ξ belongs to the set Ξ and the inner summation iterates over all the states s within each trajectory. The goal is to minimize this loss across all expert demonstrations, effectively aligning the learned policy with expert behavior [47].

Two of the most common methods for learning policies through imitation are Behavior Cloning (BC) [48] and Dataset Aggregation (DAgger) [49]. BC learns a policy by using supervised learning on observation-action pairs from expert demonstrations. BC is an offline policy and can therefore suffer from distribution shift, where small deviations from the expert policy early on lead to error amplification [46, 48]. Figure 2.4 shows how the policy may suffer large variance due to distribution shift.

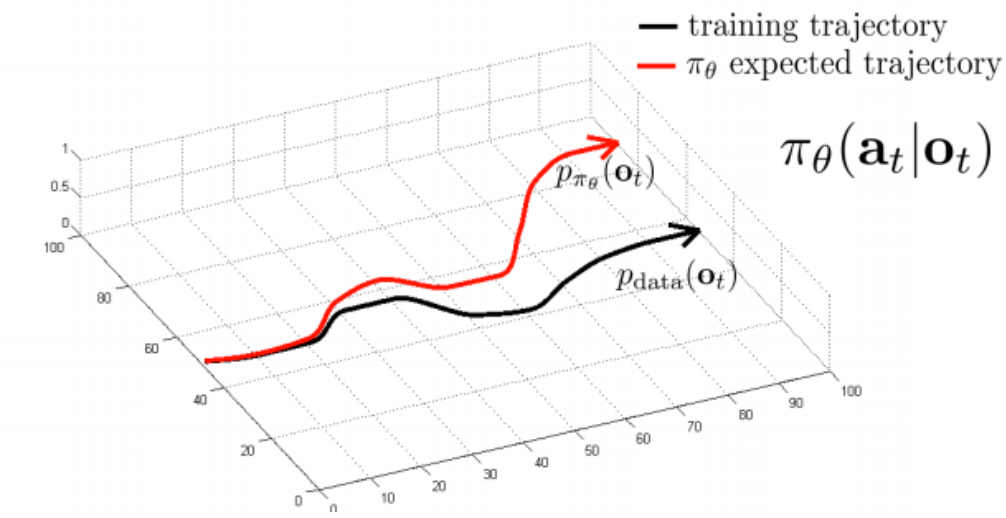


Figure 2.4: Distribution shift problem in BC [46]

DAgger improves on BC by iteratively training on a dataset of expert demonstrations. It then uses the learned policy to create its own data, by querying the expert for actions. The newly labeled observations are then added to the dataset [49]. An online approach such as DAgger avoids error amplification by querying the expert, it does however suffer from frequently visiting sub-optimal states [50]. [48] DAgger while more effective in practice than BC, it requires ongoing expert interaction during training, which is more resource-intensive [51]. Despite their ability to learn an optimal policy, both BC and DAgger have notable drawbacks. They fail to

generalize beyond the expert dataset and are highly dependent on the quality of the expert demonstrations. This makes them ineffective at handling new scenarios [14, 49].
Methods to try to solve this have been tried. For example for exploration with a Unmanned Ground Vehicle (UGV), Damanik, J. et al. [51], added noise to the expert actions to make the algorithm act effectively on various states. In [52], DAgger was applied to a swarm of robots navigating outdoor forest environments in an end-to-end network. The results showed that the performance is better than base DRL algorithms, such as DDPG and SAC. However, the method was not used in new environments, since the training environment and the testing environment were identical. This study thus failed to show generalization for new scenarios. In a paper about autonomous driving, DAgger was slightly adjusted to solve the problem of frequently visiting sub-optimal states. By adding a Bayesian uncertainty estimation method to determine whether a scene is extracted for querying, which they claim to be effective [50].

Hybrid methods exist of IL with RL to fix the poor generalization issues. Two common approaches are Inverse Reinforcement Learning (IRL) [53] and pretraining RL with IL [54].

IRL is a method where the hidden reward function is learned based on the expert demonstration, rather than imitating actions. The policy can then be determined through RL [53]. By extracting the hidden reward function, the algorithm can understand task objectives from the expert demonstrations [55]. This principle can be seen in Figure 2.5, the expert trajectories are taken as the optimal policy and the IRL algorithm learns the underlying reward function that explains the optimal policy [56]. However, IRL suffers from the same problem as IL, that if a new situation does not exist in the expert data, it fails to adapt. To improve generalization, more expert demonstrations are therefore needed [53]. Several variants of IRL have been developed to improve performance. For instance, Kong, W. et al. [57] applied maximum entropy IRL in a sparse reward setting, where incorporating entropy encourages exploration and accelerates convergence during training. A similar approach was used by Phan-Minh, T. et al. for autonomous driving, leveraging maximum entropy to learn effective reward functions in complex driving environments [58]. While both methods show promising results, they typically require thousands of expert trajectories to perform reliably.

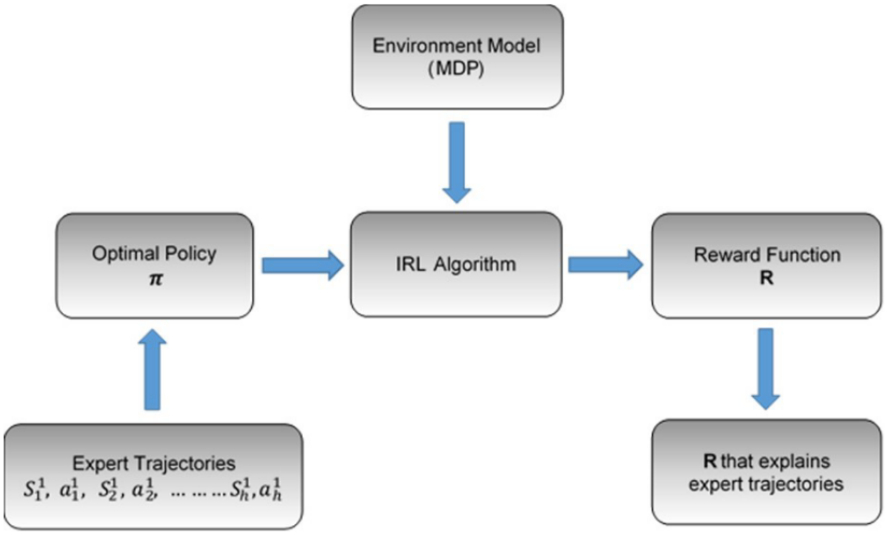


Figure 2.5: Inverse Reinforcement Learning schematic [56].

Another approach is pretraining RL with IL to obtain an initial policy before fine-tuning with

RL. This can be done with static IL, where the initial policy remains unchanged, but also dynamically, where the IL policy is continuously updated based on high-reward episodes from the RL training. By starting with an expert-guided initial policy, the RL algorithm converges faster. Dynamically updating the IL policy allows the policy to adapt to unseen scenarios [54].

Researchers Booher, J. et al. [59] also applied this hybrid method to safe autonomous driving, where they trained an initial policy through IL and fine-tuned it with DRL. To make the driving safe they used a constrained MDP to limit certain actions in certain scenarios. They used BC combined with SAC with discrete actions.

A study by Wang, Z. et al. [54] applied this hybrid approach to a UAV exploring an outdoor riverine environment. For the IL algorithm, BC was used, and PPO for the RL algorithm with discrete actions. Their results showed that PPO with a dynamic BC achieved better performance with more reward and in less steps than PPO, static BC, dynamic BC and PPO with static BC.

In another study on aerial dogfights between drones, [55] a hybrid approach is used of IL and RL. For the RL algorithm PPO is used and DAgger is used for the IL. Using this method shows superior success rates, efficient learning, and robustness while requiring less expert data from normal IL. However, the quality of the final policy depends strongly on the quality of the expert data.

Hierarchical Reinforcement Learning

HRL structures the algorithm into a hierarchy of high-level and low-level tasks. Figure 2.6 illustrates an example where a single high-level task selects among multiple low-level tasks. The high-level task could be making a cup of coffee. The low-level tasks correspond to specific motor skills, such as grabbing a cup, pouring the coffee, or placing the cup on the table [47].

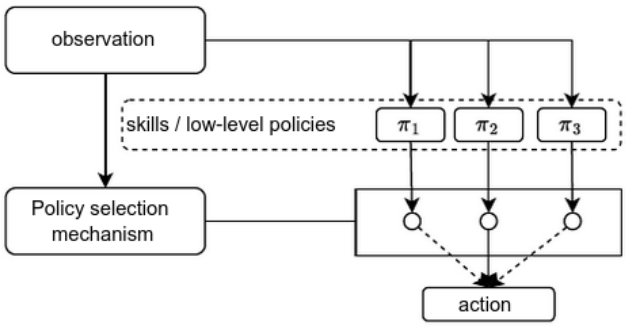


Figure 2.6: Hierarchical Reinforcement Learning schematic [47]

This division allows for better exploration and more efficient learning by reducing the dimensionality. This alleviates the curse of dimensionality, helps in faster convergence, and gives more stability in learning. Different applications of HRL exist, ranging from applying RL to both the high-level and the low-level [60, 61], to applying RL only to the low-level and using a frontier-based or manager-based approach for the high-level [25, 62, 63]. However, HRL has notable drawbacks. With insufficient information, learning meaningful high-level tasks can be challenging. Another limitation is that while the algorithm is able to learn high-level tasks, such as recognizing that passing through a door may lead to better exploration, these high-level policies are often not expressive enough to reason about complex environments or to develop

long-horizon plans. As a result, the algorithm may learn suboptimal behaviors, which can degrade overall learning performance [64]. Another limitation is delayed feedback, since high-level decisions take time to influence the outcome, updates to the high-level policy occur with a lag. This delay can hinder learning efficiency [2, 63].

In an article about aerial dogfighting simulations between drones, an HRL algorithm was applied. This approach used a two-layer hierarchy, where specialized low-level SAC-trained policies executed distinct combat strategies, while a high-level policy selector determined which strategy to activate based on the drone's current state. A problem with this technique is that the rewards of the low-level policy are designed to follow distinct combat strategies, if a maneuver was not determined prior to the training, the system might fail to adapt to new scenarios. Another disadvantage is that the policy selector runs slower than the lower-level, which causes a delay in the feedback. [65]

In [60], a similar HRL approach is applied to UAV exploration. In this framework, a high-level (master) policy selects sub-policies based on the current state, where each sub-policy represents a potential strategy for low-level control. Both the master policy and sub-policies are learned through reinforcement learning, so even after training, it remains unclear what they are precisely doing at each decision point. Unlike in the dogfight scenario, where sub-policy strategies were designed prior to learning, this method initializes sub-policies randomly. Although this increases training time, it enables the system to adapt more flexibly to diverse and previously unseen environments. However, these claims are based on training in very simple environments that will not generalize well to the real world. Furthermore, because a policy must be learned for both the high-level and low-level policies, the method becomes computationally expensive.

The authors of [66] tried it a bit differently for autonomous navigation to a target for a robot. They divided the task up into three layers. The highest layer plans high-level goals by selecting relative target positions based on external observations. The middle layer breaks these goals into reachable subgoals with preferred velocities, focusing on nearby obstacles and progress tracking. The lowest layer directly controls the robot's movements by converting subgoals into motor commands, ensuring safe and collision-free execution. Their results are however not that promising, with a success rate of 50%.

Exploration Algorithms

There are various algorithms designed to enhance the exploration capabilities of robots operating in unknown environments using RL. These approaches include epsilon-greedy exploration [67], upper confidence bound [68], Boltzmann exploration [67], Curiosity-Driven learning (CD) [69], Random Network Distillation (RND) [70], and Entropy Exploration (EE) [14]. Among these, CD learning, RND, and EE are the most effective and recent methods [14, 69, 70].

CD or Intrinsic Curiosity (IC) learning encourages the agent to explore unknown or less-explored areas of the environment. This allows the agent to adapt to new scenarios. This is achieved by integrating curiosity into the reward system of the UAV, typically through an Intrinsic Curiosity Module (ICM), which is easy to integrate [69]. Figure 2.7 shows an example of an ICM. It takes the current state, the next state, and the action as inputs to the encoder. In the ICM the encoded current state and the action are used to predict the next state by a forward model in a learned feature space. It gets a reward based on the error between the predicted next state and the real next state [71].

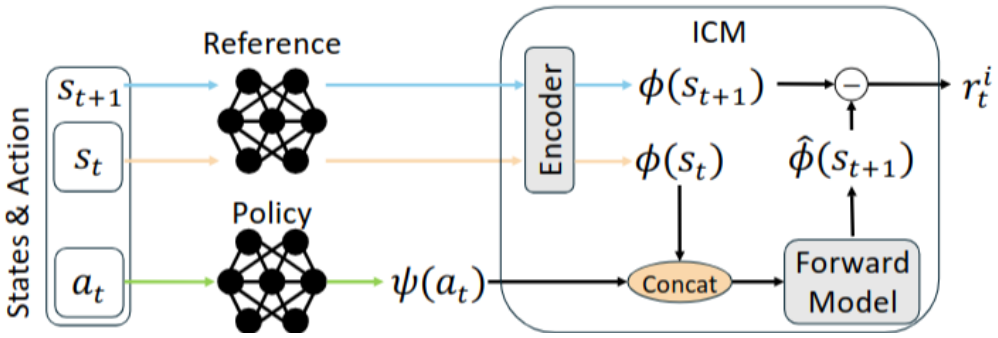


Figure 2.7: Schematic of ICM. The ICM predicts the future state based on the current state and the current action. The reward from the ICM is the error between the predicted future state and the real future state [71].

In [69], a curiosity feature is added to PPO in an outdoor simulation in Unity [72]. The curiosity feature helps a UAV in exploring the surroundings to find a target object. The curiosity feature predicts the degree of surprise or novelty from the agent's actions, this is then added to the reward function. The curiosity feature is trained alongside the PPO algorithm. The results showed that PPO with curiosity slightly outperformed both standard PPO and DDPG but does require careful tuning. CD learning can have problems with derailment and detachment. Derailment is the problem where an agent, driven to explore new states, struggles to return to key states necessary to continue effective learning. Detachment occurs when an agent loses interest in previously visited important states because they no longer provide intrinsic rewards, hindering continued exploration from those states [73].

RND introduces an intrinsic reward by using a network to predict the agent's state. When the network struggles to make accurate predictions, the agent is encouraged to explore those areas further. In [70], the exploration bonus is obtained by calculating the error between the target network and the prediction network. The final policy was able to bridge the sim-to-real gap and avoid dynamically moving obstacles. The results showed that the method with RND had higher success rates and fewer collisions in all environments in comparison to SAC. However, hyperparameter tuning is challenging and highly dependent on the environment, needing expert knowledge to set specific parameters. RND can also suffer from derailment and detachment [73].

EE or Entropy Based Exploration (EBE) is often used in sparse reward scenarios. EE generates intrinsic rewards based on the state (and action) entropy, mitigating the issue of insufficient extrinsic rewards. Entropy is a value that explains how random the state or action is, so the more unknown a state, the higher the entropy. Figure 2.8 showcases this by portraying it next to ϵ-greedy. The more unknown the state, the higher the entropy, and the higher the exploration probability of that state.

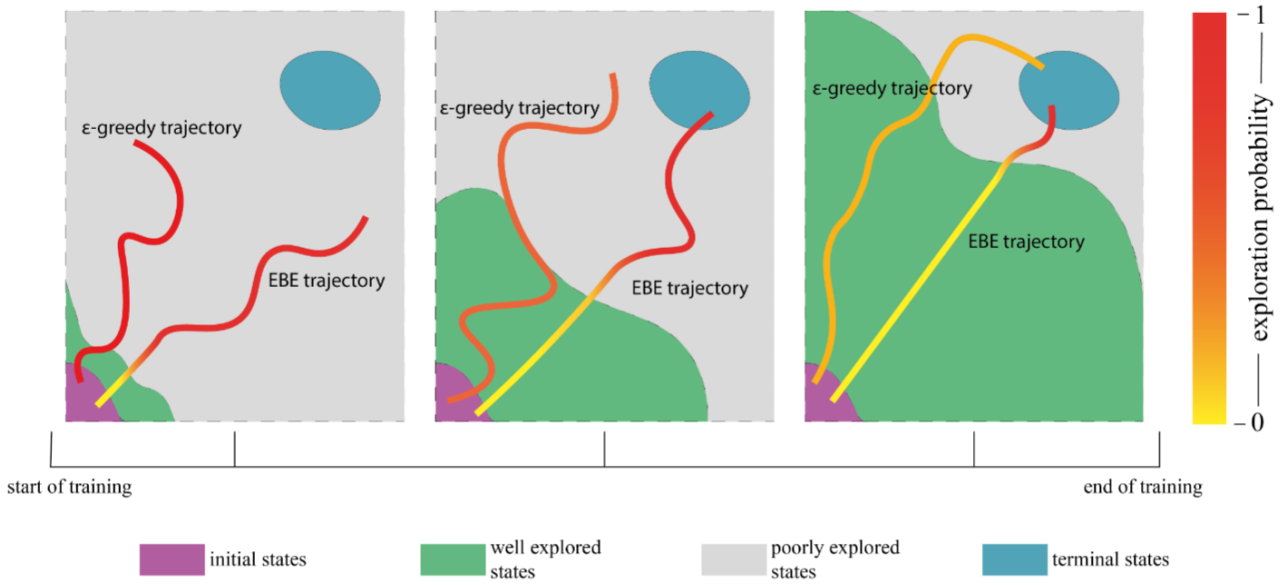


Figure 2.8: EBE overview next to ϵ-greedy exploration. The exploration probability for Entropy Exploration is higher in unknown states, while for ϵ-greedy the probability is the same in all states [74].

In a study by Usama, M. et al. [75], EE was integrated into a DQN framework to improve the navigation of a robotic vehicle. The performance of this entropy-based exploration strategy was compared against the traditional epsilon-greedy method. The results demonstrated that the EE-enhanced agent achieved better exploration efficiency and task performance than the epsilon-greedy baseline. However, it is worth noting that their experiments were conducted in relatively simple and controlled scenarios, which may limit the generalizability of the results to more complex or dynamic environments.

In an article by Lv, H. et al. [14], EE is used on the path planning of a UAV in the context of sparse rewards. The authors integrated this EE into TD3 and tested it in a 3D physics simulation. The entropy for the state is calculated with Rényi entropy [76] to measure the novelty of a state. Rényi entropy extends Shannon entropy by introducing a parameter α, which determines the order of the entropy. The action entropy is determined using Shannon entropy [77]. Their method seemed to perform better than the baseline TD3 algorithm. However, while the exploration of the algorithm improved, the additional entropy explorer increased the number of parameters, complicating the fine-tuning process. This can further conflict with task-specific goals if entropy is overemphasized [78]. Moreover, the computational complexity also increased due to the use of Rényi entropy, despite attempts to estimate the Rényi entropy.

2.3 Overview State-of-the-Art

The following Table 2.1 lists all earlier explained State-of-the-Art DRL methods with a short description and their positives and negatives.

Method	Description	Positives	Negatives
IL [14, 46–52]	Learns a policy by mimicking expert demonstrations, without explicit reward signals.	– Simple to implement – Trains fast – No reward needed – Learns near-optimal policies from good demonstrations	– Poor generalization beyond expert behavior – Suffers from distributional shift – Depends on demo quality and diversity – Requires online expert interaction for DAgger
IRL [53, 55, 57, 58]	Learns reward function from expert demonstrations, then uses RL to learn policy.	– Learns underlying reward function – Improves interpretability	– Fails to generalize to unseen states – Requires large amounts of expert data – Computationally expensive
Hybrid IL-RL [54, 55, 59]	Uses IL for pretraining then fine-tunes with RL.	– Faster convergence – Improves generalization – Combines strengths of IL and RL – Better performance with fewer steps – Reduces expert data requirements	– Requires both expert data and environment access – Performance depends on quality of expert demonstrations – Increased system complexity
HRL [2, 25, 47, 60–66, 79]	Structures policy into high-level and low-level layers to handle complex tasks.	– Efficient learning – Better exploration – Dimensionality reduction	– Can require manual task decomposition – Delayed high-level feedback – Computationally expensive – Hard to interpret – Suboptimal behavior
CD RL [69, 71, 73]	Encourages exploration by rewarding novel states using an intrinsic curiosity signal.	– Encourages exploration of unknown states – Improves adaptability – Increased performance in sparse environments – Easy to integrate	– Requires careful hyperparameter tuning – Prone to derailment – Prone to detachment – Increased training time
RND RL [70, 73]	Uses a prediction error between fixed and learned networks to drive exploration.	– Encourages exploration of novel and uncertain states – Helps bridge sim-to-real gap – works well in sparse-reward environments	– Hyperparameter tuning is difficult and environment-specific – Requires expert knowledge to set good parameters – Prone to derailment – Prone to detachment
EE RL [14, 75]	Adds intrinsic rewards based on state and action entropy to improve exploration in sparse reward settings.	– Improves exploration in sparse reward environments – Improved exploration efficiency – Highly adaptable to many algorithms	– Increases the number of parameters – Can conflict with task goals – Computationally complex

Table 2.1: Comparison of various RL-based methods for improving exploration.

2.4 Considerations

This project will develop a DRL method for indoor exploration with a UAV. DRL was chosen over classical methods due to its superior exploration rates, lower computational costs and faster decision times. These reasons also make it the preferred choice in recent research. Several base DRL algorithms are often used in literature, PPO, DDPG, TD3, and SAC. Among these, SAC appears the most promising for exploration, as its entropy-based intrinsic exploration mechanism enhances performance compared to the others [44].

However, DRL often struggles with convergence due to the large, complex state-action spaces. To solve this, a hybrid approach combining IL with RL fine-tuning was selected. While various other approaches have been researched to solve the limitations of DRL in complex environments, hybrid IL with RL stands out as a more efficient with a faster convergence. It gives a better balance between performance, generalization, learning speed, and implementation complexity compared to methods like IL, IRL, HRL, CD, RND, and EE RL.

While IL with RL does require both expert data and access to a simulation or real environment, the usage of a simulation is standard in most robotics pipelines. Additionally, in this project Isaac Sim is used and with Isaac Sim providing an efficient way to generate expert demonstrations, IL and IRL become appealing solutions. However, pure IL and IRL lack generalization to new, unseen situations and require a large amount of expert demonstrations. Fine-tuning this with RL allows the policy to adapt to new and unforeseen scenarios and decreases the amount of expert data needed. This makes the hybrid method combining the two the preferred approach.

HRL attempts to simplify complex state-action spaces by splitting tasks into high- and low-level layers. While this facilitates efficient exploration and can reduce state-action space dimensionality, it introduces several practical issues: it needs manual task decomposition, it has delayed high-level feedback, it is hard to interpret, it is computationally expensive, and often suffers from suboptimal behavior when the hierarchy is misaligned. In contrast, hybrid IL-RL bypasses the need for manual structuring. It starts with a policy derived from expert demonstrations, which already encodes competent behavior. Fine-tuning with RL then allows the system to adapt, generalize, and explore without reinventing basic skills. This approach avoids the challenges of delayed feedback and manual engineering present in HRL.

Exploration motivation methods like CD RL, RND, and EE aim to solve the exploration problem in sparse-reward environments. They offer improved adaptability and can guide agents toward informative states when extrinsic rewards are scarce. However, they suffer from derailment and detachment, have difficult hyperparameter tuning, have increased training time and system complexity. Hybrid IL-RL avoids these drawbacks by giving the agent a head start, since the initial policy already learns goal-directed behavior through IL. Thus, exploration is grounded in expert knowledge rather than being driven by potentially sub-optimal novelty signals. It reduces the need for intrinsic reward shaping and eliminates the risk of derailment and detachment.

3 Problem Formulation

This thesis proposes a novel hybrid approach that combines IL with RL, using SAC with continuous action spaces, to enable autonomous indoor exploration by UAVs. SAC was selected for its entropy-based exploration mechanism, which offers promising performance in complex, unknown environments. A common limitation of IL is its strong dependence on the quality and quantity of expert training data. However, by combining IL with RL fine-tuning, this limitation can be mitigated. The RL component allows the policy to improve beyond the initial demonstrations, enabling better adaptation to new environments and refinement of suboptimal behaviors learned from imperfect data.

The main objective is to develop and evaluate a learning-based control policy to achieve generalizable, robust, and efficient UAV exploration in unknown indoor environments, while overcoming known limitations of IL and RL when used in isolation. This objective can be broken down into the following key novel contributions:

1. **Hybrid IL-RL for UAV indoor exploration:** A novel integration of IL and SAC-based RL in continuous action spaces is proposed. SAC is chosen for its entropy-driven exploration and stability in high-dimensional continuous control tasks. While SAC has been used in robotics, to the best of my knowledge this hybrid with IL has not been previously applied to indoor UAV exploration.
2. **Use of semantic visual inputs:** The learning framework incorporates semantic segmentation information as part of the state input. None of the previously researched DRL algorithms have included semantic information, even though it can provide structured, high-level understanding of the environment and potentially improve exploration. In this project, the effect of using semantic images will be tested directly. Isaac Sim makes this easy to implement through an extension that allows for automatic object labeling.
3. **Training and testing in a photorealistic simulator:** Many existing algorithms have been developed in highly simplified environments. In contrast, this thesis uses the photorealistic simulator Isaac Sim to accurately capture the physics and reduce the sim-to-real gap.
4. **Focus on generalization:** The project explores how training diversity, expert data quantity, environment variety, and a photorealistic simulator affect the generalization of the learned policy, an aspect often overlooked in prior works.

The proposed system will be evaluated based on its ability to explore at least 80% of the environment, avoid static obstacles, and generalize to previously unseen layouts. Additional investigations that are subject to time constraints will assess its robustness to noisy inputs and its potential to avoid dynamic obstacles in more complex scenarios.

Research Questions

Based on the above objectives, the research question is formulated as follows:

To what degree can a hybrid approach combining imitation learning with reinforcement learning enhance the indoor exploration capabilities of UAVs in unknown environments?
This leads to a number of sub-questions on the different elements of this work:

- What is the optimal number of expert trajectories needed and how does the quality of expert data influence the final performance?
- How many different environments are needed for robust and generalizable training?
- To what extent does the hybrid policy generalize to previously unseen environments with varying obstacles or dynamic changes, such as moving objects or altered terrain?
- Does the use of semantic image inputs improve exploration performance compared to non-semantic visual inputs?

4 Simulation and Training Setup

For this project, the Isaac Sim [80] simulator will be used. Developed by NVIDIA as part of the Omniverse platform, Isaac Sim is a high-fidelity simulator designed for robotics and synthetic data generation. It features RTX-powered rendering and GPU-accelerated physics through the PhysX engine, enabling accurate and realistic simulations. The simulator supports Universal Scene Description (USD) files and integrates with ROS. IsaacLab [81] is a framework built on top of Isaac Sim. It is primarily used for robot learning and aims to simplify common workflows in robotics research, such as RL, learning from demonstrations, and motion planning. The framework's main advantages are its modularity, agility, and openness, which make it easily customizable and allow the community to contribute to and extend its capabilities. Isaac Sim, used in combination with the IsaacLab framework, was originally chosen by a previous student for its physics accuracy, photorealistic rendering, and precise sensor data. The platform has grown much since then and has gained popularity in both academia and in industry. Notably, Disney's new BDX droids from Star Wars were trained using Isaac Sim [82].

4.1 Isaac Lab Setup

The exploration task of the drone can be set-up with two different methods in Isaac Lab: a manager-based system and a more direct approach [81].

The manager-based system divides the environment into modular components (managers) for actions, observations, rewards, and more, making it ideal for prototyping and experimenting. The direct approach uses a single task class to implement all environment logic, offering full control and performance optimization. The manager-based system is preferred in this project because it simplifies development and experimentation. The ability to quickly test different types of observations, reward structures or actions is crucial for experimenting with the UAVs exploration behavior during training, to find the optimal combination. This modular structure of the manager-based system also improves code readability and makes collaboration easier.

A schematic of the interaction of the different managers for the manager-based system is illustrated in Figure 4.1. The full task of exploration with DRL is split into different components. Each manager-based RL system includes the following core managers:

- **Scene Manager**: Responsible for spawning objects, configuring joint articulations, and initializing sensors within the simulation environment.
- **Action Manager**: Receives raw actions from the agent, processes them, and applies them to the environment.
- **Observation Manager**: Collects sensor and state data from scene assets, processes the data, and forwards it to the agent.
- **Event Manager**: Triggers operations based on simulation events, such as domain randomization or environmental resets
- **Reward Manager**: Computes the reward signal for the agent.
- **Termination manager**: Computes the done signals for the agent.

Additional optional managers are:

- **Curriculum Manager**: Facilitates Curriculum RL by gradually increasing task difficulty, enabling the agent to learn more effectively over time.
- **Command Manager**: Supports switching between different command strategies, useful for hierarchical or multi-modal control schemes.

The agent interacts with the environment through these modular managers. For example, it receives processed observations from the Observation Manager and learning signals from the Reward and Termination Managers. Based on this, it outputs actions that are processed and applied by the Action Manager, which interacts with the Scene Manager to affect the simulation.

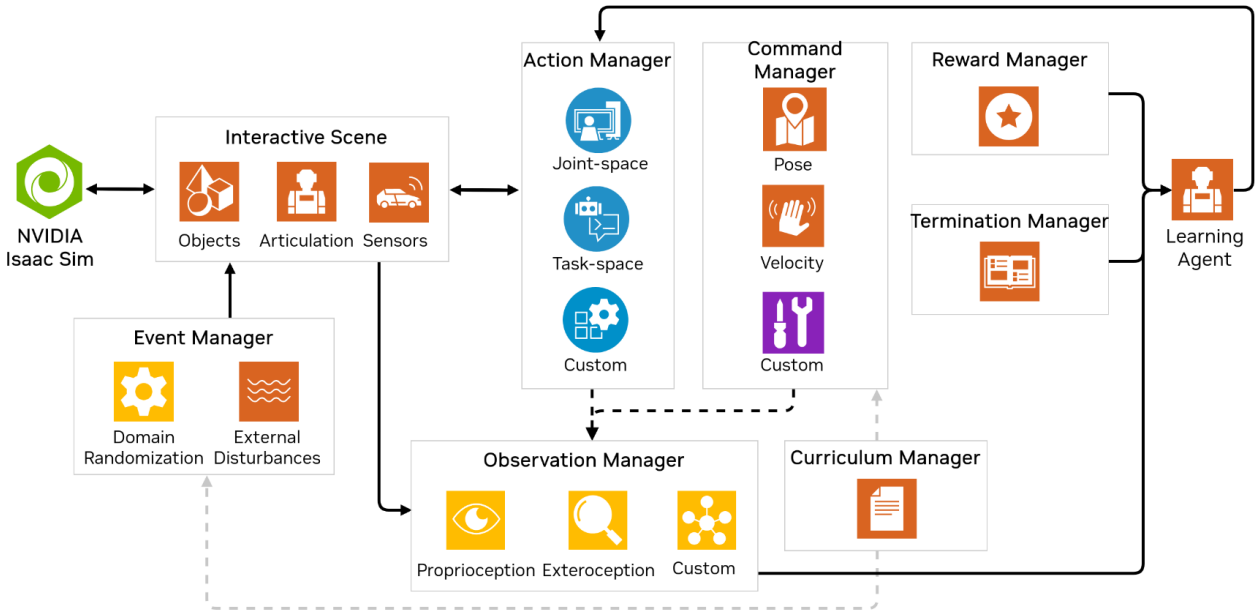


Figure 4.1: Diagram of Manager-Based system in Isaac Lab for RL task [81]

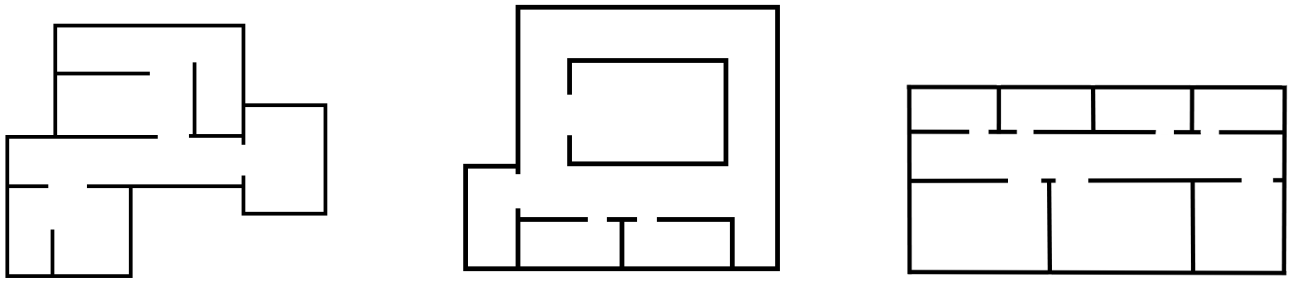
4.2 Environments

To evaluate and train the exploration capabilities of the UAV, two groups of environments are used. The first is a set of training environments created specifically for this project to expose the agent to diverse navigation challenges. The second is a separate set testing environments used exclusively for evaluating generalization to unseen scenarios taken from [83]. This separation ensures that performance metrics reflect the agent's ability to transfer learned behaviors rather than memorizing the layout of a room.

Different office/university type environment are created for training environments. In designing these environments, existing literature was reviewed to find a baseline. However, no consistent standard was found. Many studies, such as [11, 14, 84], employed a simple square room with varying numbers of obstacles. In contrast, the authors of [85] used more complex office-style environments featuring multiple rooms and varying floor structures. This setup closely resembles the intended operational environment of the UAV in this project and therefore served as a key source of inspiration.

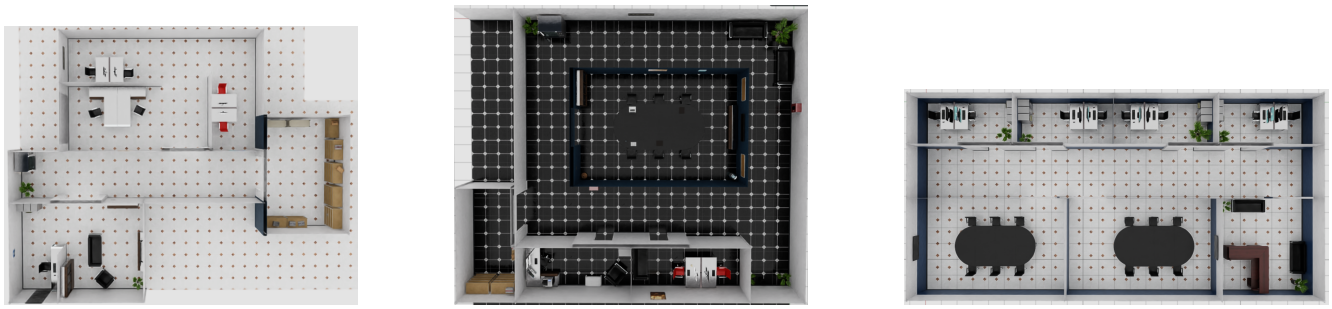
Figures 4.2a, 4.2b and 4.2c show schematic top views of the designed training environments. These environments consists primarily of rooms, doorways, and hallways with diverse shapes and dimensions to promote varied exploration behaviors. Corridors are were made to be between

2-3 meters wide, with lengths varying between 3 and 15 meters. This allows for short transitions between rooms and also longer, straight segments that test the UAVs directional consistencies. Rooms vary in size from smaller 3 by 6 meter offices to larger 9 by 9 meter spaces, these different sizes encourage different navigation and scanning strategies. Doorways range in width from 1 to 2 meters, this introduces variability in access constraints that requires fine controlled movements. These designs stimulate the agent to fully explore its surroundings rather than flying in repetitive loops. All environments include furniture and other interior elements to enhance realism and improve the transfer from the simulation to the real world. Figures 4.3a, 4.3b, and 4.3c show top views of the simulated training environments, including the interior elements.



(a) Environment 1 (b) Environment 2 (c) Environment 3

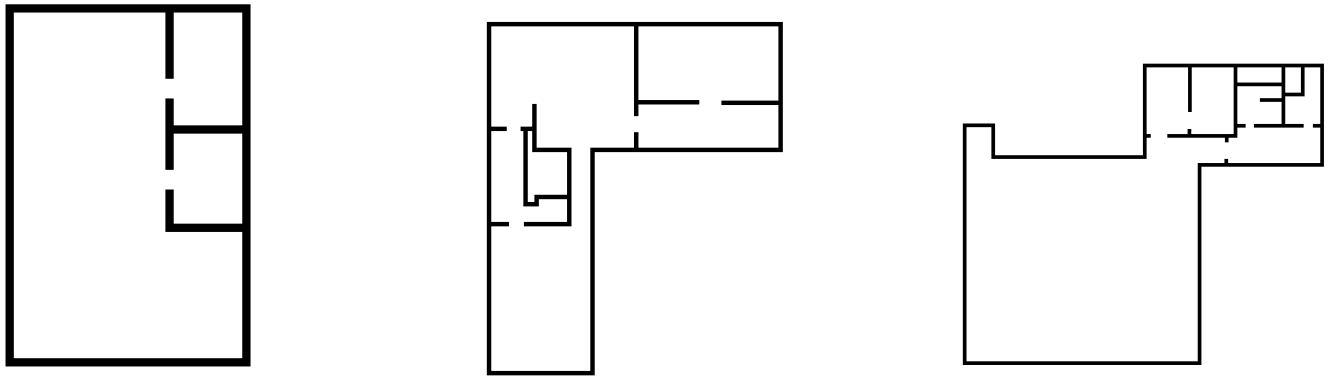
Figure 4.2: Schematic top views of the three different training environments made with Isaac Sim Assets, used for recording expert trajectories and training the models.



(a) Environment 1 (b) Environment 2 (c) Environment 3

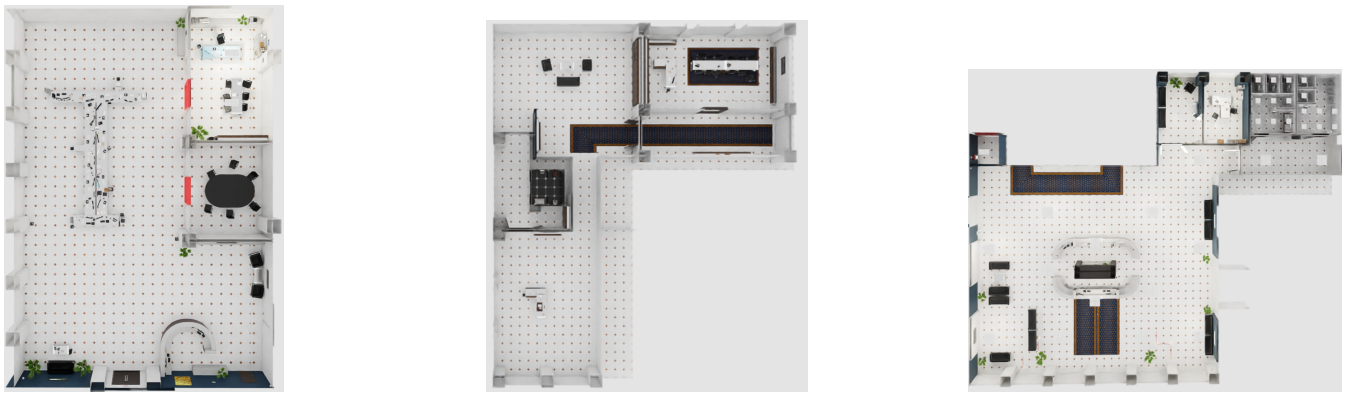
Figure 4.3: Photorealistic top views of the three different training environments made with Isaac Sim Assets, used for recording expert trajectories and training the models.

The testing environments were originally part of a default Isaac Sim office scene, which was divided into four rooms (A, B, C, and D) by Bravo i Forn, A. [83]. In this work, only scenes B, C, and D are used. For simplicity, they will be referred to as environments A, B, and C, respectively. Figures 4.4a, 4.4b and 4.4c show schematic top views of the testing environments. Figures 4.5a, 4.5b and 4.5c show the top view testing environments from the simulation with furniture.



(a) Environment A (b) Environment B (c) Environment C

Figure 4.4: Schematic top views of the three different testing environments from the Isaac Sim office environment, used for evaluating the models.



(a) Environment A (b) Environment B (c) Environment C

Figure 4.5: Photorealistic top views of the three different testing environments from the Isaac Sim office environment, used for evaluating the models.

To provide a better sense of the indoor environment, Figure 4.6 presents a photorealistic view of one of the rooms.

Figure 4.6: Photorealistic view of a room in Isaac Sim

4.3 Configuration Details

The simulation environment runs at 100 Hz, while the policy is updated at 4 Hz. This means each policy decision governs 25 environment updates, ensuring smoother control and preventing overly reactive behavior. The sensors are updated at the same frequency as the policy (4 Hz), since higher update rates would be redundant and increase computational load without providing any benefits.

Training was performed on a server with an Intel Xeon Silver 4216 CPU (16 cores, 32 threads, 2.10GHz) and an NVIDIA A40 GPU (48 GB VRAM), running Ubuntu 22.04.5 LTS. The simulation environment used Isaac Sim version 4.5.0 in combination with Isaac Lab version 2.1.0.

To accelerate training, 10 environments run in parallel, this strikes a balance between training speed and computational feasibility. Using more than 10 environments caused excessive strain on the GPU and led to decreased performance due to the large number of environments being loaded into memory.

During training, the UAV is randomly spawned in one of the rooms of the three training environments at a random position and with a random orientation. The UAVs were spawned at least 30 cm from any wall, this constraint ensured that the UAV does not start too close to a wall and immediately crashes. Corridors were excluded as starting locations due to their lack of diversity and can bias early exploration behavior.

4.4 UAV Setup

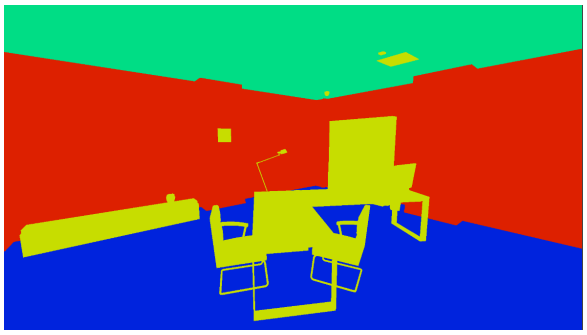
The UAV in the simulation is a model made from the 3DR Iris quadcoptor [86]. This UAV was chosen by a student working on a similar project [83]. The model has a width and depth of 550mm, a height of 100mm and a weight of 1.3kg.

Figure 4.7: 3DR Iris UAV used in Isaac Sim [86]

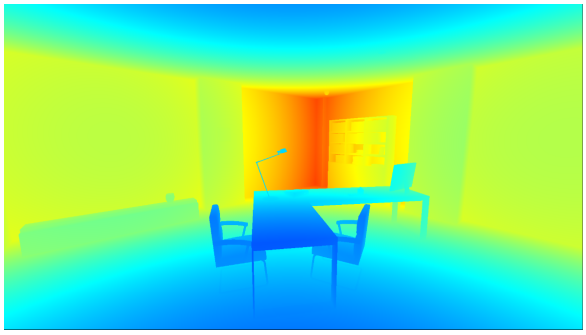
Isaac Sim can simulate ground-truth perception and physics-based sensing, and it provides a library of realistic sensor models. Sensors all inherit from the SensorBase abstract class that provides the core functionality of sensors. This class handles periodic scene measurements and supports vectorized environments. Due to this base class, Isaac Sim is able to handle many sensor types such as cameras, IMUs, contact sensors, and ray-caster sensors.

For this thesis, the UAVs are equipped with a camera that captures both depth and semantic images. A contact sensor is also attached to detect collisions by measuring external forces

exerted on the UAV when it comes into contact with other objects. In the semantic images, only doors, walls, floors, and ceilings are considered, all other objects are labeled as unknown. This is done to reduce the number of features the drone needs to learn and to focus only on the most relevant structural elements. Both the depth and semantic images are limited to a range of 4 meters to simplify the information available to the drone. In Isaac Sim, semantic labels can be assigned to objects (prims) using the Semantics Schema Editor. Figure 4.8 shows representative examples of semantic and depth images.



(a) Semantic segmentation image: red indicates walls, blue represents the floor, green denotes the ceiling, and yellow represents unlabeled regions.



(b) Depth image, with depth values ranging from dark blue (0 meters) to red (4 meters).

Figure 4.8: Semantic and depth images captured from the wide-lens onboard camera in Isaac Sim.

The simulated camera uses the intrinsic parameters of the ZED X One wide-lens camera [87]. The camera has a focal length of 2.208 mm, a focus distance of 28.0 mm, a horizontal aperture of 5.76 mm, and a vertical aperture of 3.24 mm. The camera resolution will be downscaled to 48 by 64 pixels to decrease the number of parameters for the network. This specific camera was selected because it matches the one used by the intended UAV for this project.

4.4.1 UAV Controller

The action manager in Isaac Sim obtains actions from the agent or manually. These actions v_x, ω_z are then passed to a controller, only forward velocity in the range [0, 1] and angular velocity along the z-axis in [-1, 1] are considered. This constraint ensures the drone moves only in directions it can see, aiding in collision understanding since collisions occur within its field of view. The high-level actions generated by the agent are passed to a low-level controller.

A previous student working on a similar project implemented both an ideal controller and a more realistic non-linear controller [83].

For this thesis, the ideal controller was selected due to difficulties encountered with the non-linear controller when the UAV is commanded to move forward and rotate simultaneously. The non-linear controller exhibited instability in such scenarios, leading to unpredictable UAV behavior and trajectory deviations. Addressing these issues was beyond the scope of this work. This instability is likely caused by coupling effects between translational and rotational dynamics, which the controller can not handle. Due to time constraints, there was insufficient opportunity to address and resolve these issues, and the ideal controller was used instead to ensure stable and consistent training. This is not expected to significantly impact the policy,

as the policy outputs remain unchanged but might impact generalizability. A summary of the technical details of the non-linear controller is included here for completeness and to provide a foundation for future work. Future students building upon this work may also benefit from this controller description when addressing the stability issues or extending the controller design.

Ideal Controller
The ideal controller provides a simplified method to apply actions directly to the UAV without simulating physics or control dynamics. It takes the 2D action vector from the agent that is in the UAV's body frame, scales it to maximum velocity values, and transforms it to the world frame based on current orientation. The next position and orientation of the UAV are computed using Euler integration and applied directly in the simulation. This ensures ideal motion but lacks physical realism.

Non-Linear Controller
For completeness, the non-linear controller is briefly summarized here. To control the UAV, the controller computes the appropriate control inputs $\vec{\hat{u}} = (f, m_x, m_y, m_z)^T = (\hat{u}_0, \hat{u}_1, \hat{u}_2, \hat{u}_3)$ that was defined in the dynamics equations in Section 2.1.1.

It computes the desired force $\vec{F}_{\text{des}}$ as:

$$\vec{F}_{\text{des}} = -K_p\vec{e}_p - K_v\vec{e}_v - K_i\vec{e}_i + mg\vec{z}_w + m\ddot{\vec{r}}_{\text{des}}, \tag{4.1}$$

where K_p, K_v, and K_i are gain matrices, $\vec{e}_p, \vec{e}_v, \vec{e}_i$ are position, velocity and integral errors, $\vec{z}_w = [0, 0, 1]^\top$ denotes the gravity direction, and $\ddot{\vec{r}}_{\text{des}}$ is the desired acceleration.

The full control law is then:

$$\vec{\hat{u}} = \begin{bmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{bmatrix} = \begin{bmatrix} \vec{F}_{\text{des}} \cdot \vec{z}_D \\ -K_R\vec{e}_R - K_\omega\vec{e}_\omega \end{bmatrix}, \tag{4.2}$$

where $\vec{z}_D$ is the body z-axis, $\vec{e}_R$ the orientation error, and $\vec{e}_\omega$ the angular velocity error. K_R and K_ω are diagonal gain matrices for orientation and angular velocity control. Full derivations and implementation details can be found in the report of Bravo i Forn, A [83].

4.5 Reinforcement Learning Setup

The MDP for the RL algorithm is defined through the configuration of observations, rewards, and termination criteria as follows.

Observations
At each policy update step, the agent receives observations that represent the state of the environment in a compact and structured format. Depending on the configuration, different combinations of the following observation types are used:

- **Egocentric Occupancy Map:** A 2D top-down occupancy map centered and aligned with the agent's heading, encoded as a one-hot tensor. The map helps the agent understand spatial layout relative to its current orientation. An example can be seen in Figure 4.9. Each color represents a different class in the image, yellow is the agent itself, green is occupied space, light blue is the unoccupied space and dark purple is the unknown space.

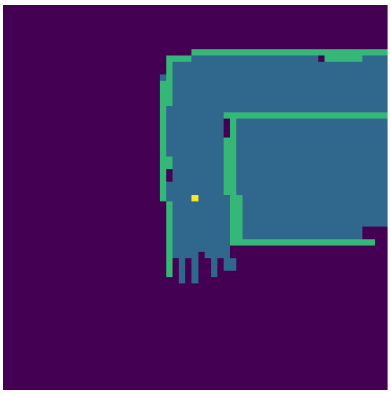


Figure 4.9: Egocentric Occupancy Map.

- **Depth Image:** A depth image clipped at a threshold of 4.0m. This offers dense spatial information at the cost of increased dimensionality.
- **Depth Line:** A single scanline from the center row of the depth image, clipped at a threshold of 4.0m. This provides a condensed representation of the depth directly in front of the UAV.
- **Semantic Image:** A semantic image using one-hot encoding for 6 classes. This representation offers rich semantic context for each pixel but has increased complexity.
- **Semantic Line:** A single scanline from the center row of the semantic image. This representation offers rich semantic context for each pixel but has increased complexity. This provides a condensed representation of semantic categories directly in front of the UAV.
- **Linear Velocity:** Vector of the linear velocities of the UAV in m/s.
- **Angular Velocity:** Vector of the angular velocities of the UAV in rad/s.

Rewards
The reward function combines multiple terms to balance exploration, efficiency and safety. The reward terms include:

- **Area Coverage Reward** ($r_{\text{cov},t}$)**:** Encourages the agent to explore new areas by rewarding the amount of new grid cells in the occupancy map. Returns the number of new cells found at time t.
- **Collision Penalty** ($r_{\text{col},t}$)**:** Penalizes collisions, to discourage unsafe actions. Returns a binary value: 1 if a collision occurred at time t, and 0 otherwise.
- **Exploration Completion Reward**, ($r_{\text{fin},t}$)**:** Gives a high reward when the agent successfully visits a sufficient number of unique cells from the occupancy map, signaling completion of the exploration task. Returns a binary value of 1 if the total number of explored cells is higher than 80% of total cells, and 0 otherwise.
- **Room Entry Reward** ($r_{\text{entry},t}$)**:** Provides a reward each time the UAV enters a new room through a doorway, encouraging high-level exploration behavior. Returns a binary value: 1 if an agent has gone through a new door at time t, and 0 otherwise.

- **Idle Penalty** ($r_{\text{idle},t}$)**:** Penalizes low motion behavior to discourage the agent from remaining stationary. Returns a binary value: 1 if the agent is moving slower than the threshold of 0.1 for both actions, and 0 otherwise.

The reward equation for timestep t then becomes:

$$R_t = \alpha_1 \cdot r_{\text{cov},t} - \alpha_2 \cdot r_{\text{col},t} + \alpha_3 \cdot r_{\text{fin},t} + \alpha_4 \cdot r_{\text{entry},t} - \alpha_5 \cdot r_{\text{idle},t} \tag{4.3}$$

Here, α_i are scalar weights for the rewards. These weights control the relative importance of all rewards. Through trial and error, the final weights were in balance with the following values:

- $\alpha_1 = 0.1$
- $\alpha_2 = 100$
- $\alpha_3 = 150$
- $\alpha_4 = 10$
- $\alpha_5 = 0.01$

This combination encourages the agent to actively explore new areas and enter new rooms while strongly discouraging collisions. A small penalty for idle behavior ensures the agent keeps moving, reducing the chance that the drone learns to stop completely when near obstacles.

Terminations
The following terms indicate when an episode will end:

- **Timeout:** The episode is terminated after N amount of timesteps.
- **Collision:** The episode is terminated when the drone has a collision.
- **Area Fully Covered:** If the UAV has covered a specific total number of cells, then the exploration has been completed and the episode is terminated.

The specific parameter values used in this thesis are as follows: the timeout is set to 1500 seconds, a collision terminates the episode immediately upon contact, with a collision force threshold set at 0.01 N, and the area is considered fully covered when the UAV has visited 80% of the total cells, so this number varies between scenes. While this termination condition does not guarantee 100% coverage, setting the threshold at 80% ensures that the UAV explores most of the environment while allowing episodes to terminate in a reasonable timeframe. Empirical testing showed that higher thresholds (95-100%) were rarely reached, which hindered effective training.

4.6 Network Setup

For the RL, the SAC algorithm is used. This algorithm consists of one actor and two critics, as introduced in Section 2.1.2. The actor is the policy network that maps states (observations) to actions, implemented as a fully connected neural network with two hidden layers of 256 and 128 nodes, respectively. The critics are value networks that estimate the expected return (Q-value) for a given state-action pair. The two critics are trained independently, each implemented as a fully connected network with two hidden layers of 256 and 128 nodes. During training, the minimum of the two values is used for learning, this reduces the overestimation bias typically observed with a single critic and leads to more stable learning [23].

SAC makes use of an experience replay buffer that stores past transitions collected from the agent's interactions with the environment. During training, SAC samples random batches of experiences from this buffer, ensuring the samples are uncorrelated, which improves learning stability. By repeatedly learning from diverse past experiences, SAC can better generalize and converge faster [23].

The observations will not be directly given to the SAC algorithm due to the high dimensionality of images, this will be inefficient and often leads to poor performance. Figure 4.10 shows the network structure used in this thesis.

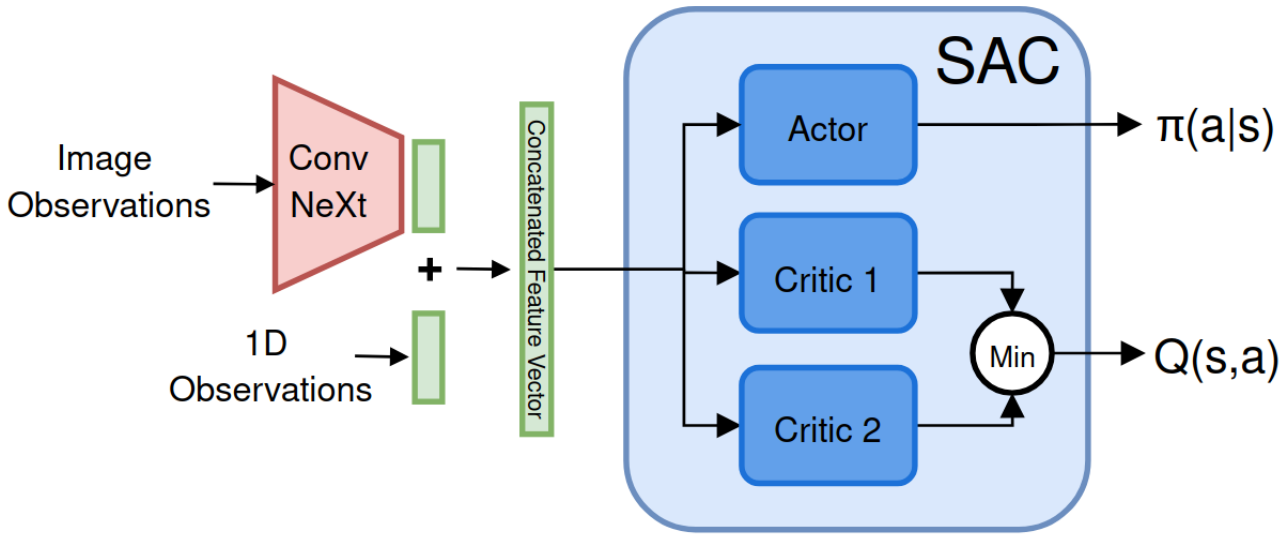


Figure 4.10: The network architecture used in this thesis processes image observations through ConvNeXt to extract features, which are then concatenated with 1D observations. This combined feature vector is fed into the SAC network, comprising an actor and two critic networks. The actor outputs the policy, while the two critics estimate action values, with the minimum of the two values used for learning to reduce overestimation bias [23].

Input images are first passed through a pretrained Convolutional Neural Network (CNN) backbone. A CNN is chosen over a transformer network due to its smaller size and faster processing, although it is less accurate. The one used in this project is ConvNeXt [88]. This algorithms was chosen because it combines the strengths of classical CNNs with the design innovations of transformers. ConvNeXt compresses the input images into a compact feature representation. It outputs a 1D feature vector of size 128, which serves as a low-dimensional feature vector of the visual input. Any non-image observations, such as velocity, position, orientation, as well as the depth line and semantic line, that are already 1D are concatenated directly with the visual features and passed into the SAC network without further processing.

This architecture is designed to support various combinations of observations. Since input images are processed into a fixed-size feature vector and concatenated with other 1D inputs, the network can easily work with different sets of observations without changing the architecture, while also avoiding the high computational cost and instability of training directly from image pixels. This design enables experimentation with observation configurations while keeping the training pipeline the same.

5 Methodology

For the indoor exploration of a UAV, a hybrid method of IL with DRL will be used. For this the RL network will be pretrained with IL to obtain an initial policy which will be fine-tuned with RL. This process is structured into three main stages, illustrated schematically in Figure 5.1. The first stage is the gathering of expert trajectories from simulation. In the second stage, the model is pretrained on this data with BC. The third and final stage applies RL to fine-tune the model for optimal performance.

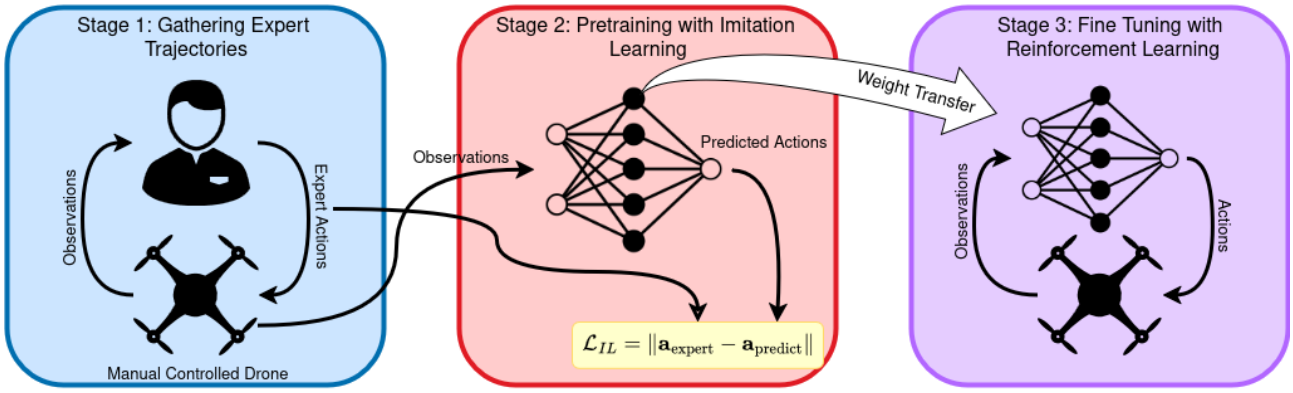


Figure 5.1: Schematic diagram of the three stages of the project. The first stage is gathering expert trajectories. The second step takes the observations from the expert trajectories predicts its own action and the loss is determined through BC. The final stage is the fine-tuning with RL of the transferred weights from the IL stage.

5.1 Stage 1: Gathering Expert Trajectories

Expert demonstrations were collected in simulation using Isaac Sim. The simulation trajectories were recorded in the training environments previously described as environments 1, 2, and 3. The use of a physics-based simulator and diverse environments was to improve robustness and support transfer to the real world by mitigating the sim-to-real gap. Figure 5.2 illustrated a schematic of how the expert trajectories were collected.

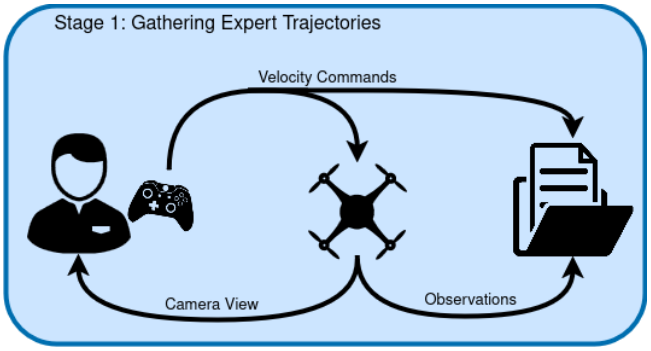


Figure 5.2: Schematic of first stage of methodology: Gathering Expert Trajectories.

The data was gathered manually, this ensures that the learned policy exhibits human-like behavior.

The simulation expert trajectories were recorded in the same environments where the RL agent was trained. The expert controlled the drone using an Xbox controller with two joysticks, one joystick controlled translation while the other controlled rotation, this gave the ability for smooth and continuous actions, including simultaneous rotation and forward movement.

The expert's view is an RGB image from the drone's camera, this view is limited to a depth of 4 meters to match the drone's actual sensing range.

To improve the coverage and effectiveness of the training data, five expert trajectories included more challenging scenarios, such as flying close to walls or narrowly avoiding doorways. This exposes the IL algorithm to more scenarios and difficult maneuvers, improving its ability to handle such situations during autonomous flight.

The recorded expert trajectories save the actions, depth images, egocentric occupancy maps, semantic images, angular velocities and linear velocities. This allowed for experimentation with different combinations of observations when training the IL model. Although all observation types were recorded, only a smaller subset was used during training which will be explained further in the experiment explanation in Section 6. A total of 30 simulation expert trajectories were recorded.

Figure 5.3 shows an occupancy map of the expert data of training scene 3.

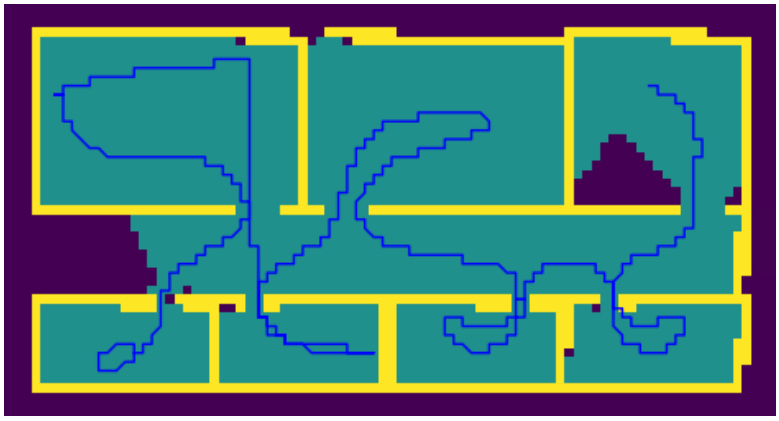


Figure 5.3: Occupancy map of expert trajectory through environment 3.

5.1.1 Data Augmentation

In order for the data to generalize better, the data was augmented to also include flipped datasets. This means that images were flipped and that expert actions were the inverse. Also noise was added to the expert actions according to [51] to help with unforeseen scenarios.

5.2 Stage 2: Pretraining with Imitation Learning

For pretraining with IL, the BC method is used. BC is a supervised learning approach that approaches IL as a regression problem. An agent tries to learn the mapping from observations to expert actions from expert demonstrations. It tries to minimize the difference between its predicted action and the expert action at each timestep, using a loss function [48]. Figure 5.4 shows the imitation learning process. Expert observations are provided as inputs to the network, which predicts corresponding actions. A loss is then calculated by comparing the predicted actions with the expert actions. By minimizing this loss over many training iterations,

the network learns to mimic the expert behavior. The resulting trained weights are saved for transfer to stage 3.

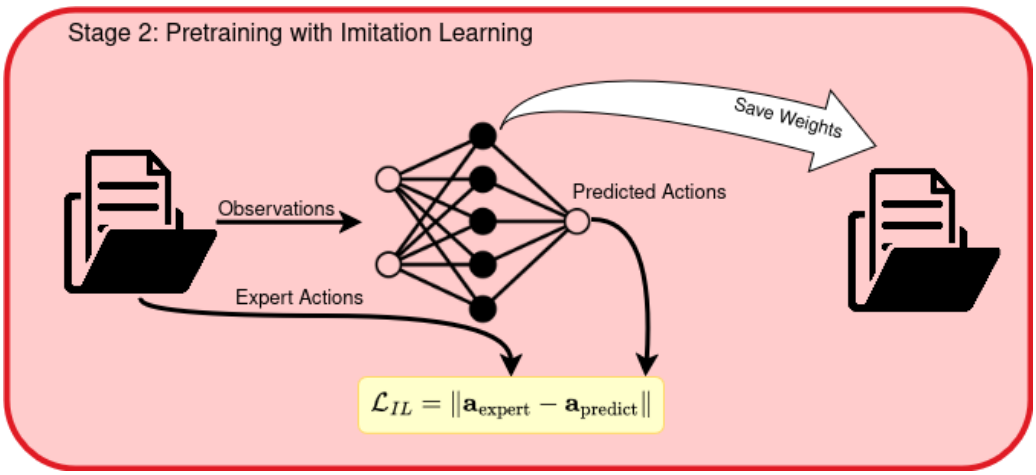


Figure 5.4: Schematic of second stage of methodology: Pretraining with Imitation Learning.

BC is an offline IL method, which does not require real-time querying of the expert during training. This method was chosen above DAgger because the initial plan involved using real-world expert data, which makes interactive expert access difficult. BC remained the practical choice for pretraining due to its simplicity, lower time requirements, and ease of implementation.

The same neural network architecture for BC will be used as for the RL part. This allows for seamless transfer of the pretrained weights from the IL phase to the RL phase, effectively initializing the RL policy with expert pretrained weights. While BC provides a good initialization, it suffers from error amplification. This is mitigated in Stage 3, where the policy is further refined using RL. [48]

During training, the IL model receives states (observations) from the demonstration dataset and predicts corresponding actions. These predictions are compared to the expert actions using the L1 loss. Equation 5.1 shows that the difference between the predicted action $\vec{a}_{pred}$ and the real action $\hat{\vec{a}}_{real}$ will be calculated and then averaged to obtain the L1 loss. L1 loss is preferred over Mean Squared Error (MSE) because it penalizes small errors less aggressively. This is desirable in continuous tasks, where minor deviations from expert actions may not significantly impact performance.

$$\text{L1 Loss} = \frac{1}{N} \sum_{i=1}^{N} \left| \vec{a}_{pred} - \hat{\vec{a}}_{real} \right| \tag{5.1}$$

As shown in Algorithm 1, the dataset of expert demonstrations is first loaded, and a neural network based on the SAC architecture with a ConvNeXt encoder is initialized. The algorithm then iterates for a predefined number of training steps. In each iteration, a random trajectory is selected from the dataset, and a sliding window of fixed sequence length is used to extract batches of observations and corresponding expert actions. For each batch, the observations are passed through the network to generate predicted actions. The L1 loss is then computed between the predicted and expert actions. The previously accumulated gradients are cleared, backpropagation is performed to compute new gradients, and the weights are updated accordingly.

Preliminary experiments explored various values for the learning rate, batch size, sequence length, and number of training iterations. Final hyperparameters were selected based on prior literature, empirical testing, and practical considerations. In the literature on BC, learning rates typically range between 1×10^{-4} and 3×10^{-3}, and batch sizes between 64 and 256 [89, 90]. Based on tests within these ranges, the learning rate was set to 1×10^{-4} as it provided stable convergence. A sequence length of 600 steps and a batch size of 200 were chosen to balance the learning signal per update with computational efficiency. The model was trained for a total of 2000 iterations, after which performance plateaued. More iterations were tested, but they did not lead to further improvement.

Algorithm 1 Pretraining with Behavior Cloning

```
datasets = load(demo_directory)
model = SACNetConvNext(action_dimension, observation_space)
for n_iterations do
    for random trajectory in datasets do
        obs_seq, expert_actions = trajectory
        for start in range(0, traj_len, seq_len) do
            obs = get_observations(obs_seq[start:start+seq_len])
            demo_actions = expert_actions[start:start+seq_len]
            pred_action, pi_std = model(obs, hidden_state)
            loss = L1_loss(pred_action, demo_actions)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        end for
    end for
end for
```

After training the IL network, its weights are transferred to initialize the RL policy. The experience replay buffer is then initially populated with trajectories generated by the pretrained policy, without updating the network during this phase. This ensures that the buffer contains meaningful, expert-aligned experiences from the outset, which helps stabilize early RL training and accelerates convergence.

5.3 Stage 3: Fine-tuning with Reinforcement Learning

The fine-tuning of the network is done with SAC. Figure 5.5 illustrates the fine-tuning process. The network is initialized with pretrained weights. It then determines an action based on the weights and the action is given to the UAV. The UAV interacts with the environments collects observations and passes its state and reward to the experience replay buffer. A batch is taken from the replay buffer to update the weights of the network. Over many training iterations, the network fine-tunes its policy and the final trained weights are saved to a file.

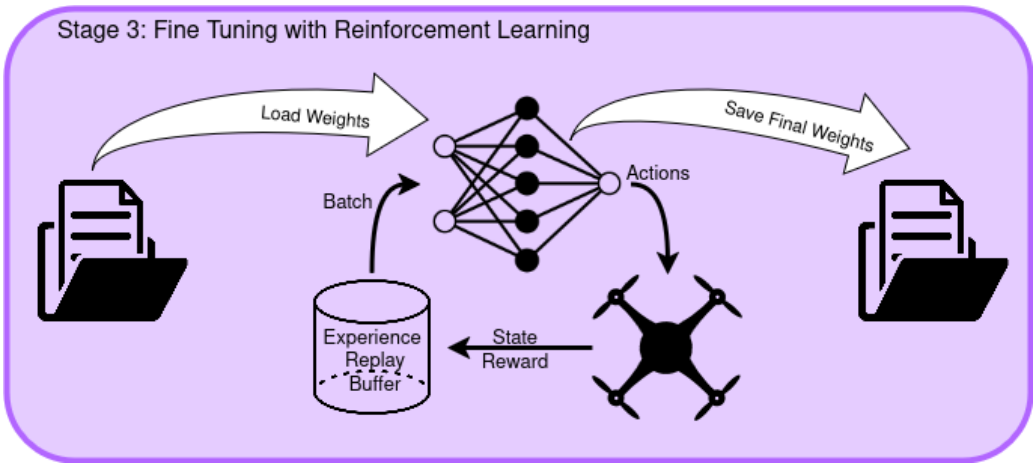


Figure 5.5: Schematic of third stage of methodology: Fine-tuning with Reinforcement Learning.

The SAC algorithm makes use of entropy regularization. In this type of algorithm, the actor gets an extra reward proportional to the entropy of the policy in that time step. This entropy quantifies the uncertainty in the policy's actions, the less predictable the action for a given state, the higher the entropy. It helps in the exploration of the state-action space [23]. The entropy H of the policy π given the state s_t is indicated as: $H(\pi(\cdot|s_t))$. SAC further learns a policy π_θ and two Q functions Q_{ϕ_1} and Q_{ϕ_2}. Two Q functions are used to avoid exploitation of overestimation bias, this is done by taking the minimum value of the two Q functions.

The RL problem from Equations 2.7 and 2.8 to find the optimal policy with the maximum expected return can be adjusted to include the entropy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\right)\right], \tag{5.2}$$

where π^* denotes the optimal policy, τ is a trajectory of states and actions sampled from π, $\gamma \in (0, 1]$ is the discount factor, $R(s_t, a_t, s_{t+1})$ is the reward obtained after taking action a_t in state s_t and transitioning to s_{t+1}, and $\alpha > 0$ is the entropy coefficient. This parameter controls the explore-exploit tradeoff, with a higher value leading to more exploration and a lower value leading to more exploitation.

In SAC, the action-value function $Q^\pi(s, a)$ is also modified to account for the entropy bonuses, except for the first timestep, since entropy does not affect the initial action a_0. Using Equation 2.10, the modified Q-function becomes:

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H\left(\pi(\cdot|s_t)\right) \,\middle|\, s_0 = s, a_0 = a\right] \tag{5.3}$$

with all other terms as defined above.

Algorithm 2 shows pseudocode for how the fine-tuning of the RL algorithm is performed. First, the SAC network is initialized by loading the pretrained weights obtained from the IL stage. These pretrained weights are used as a starting point and are gradually updated during SAC training. These weights are not overwritten by the fine-tuning but further enhanced to handle scenarios outside of the expert demonstrations.

After this, the experience replay buffer is filled with transitions generated by the current policy, which at the start corresponds to the IL pretrained model. At each step during this filling process, the current observation, next observation, action, reward, and done signal are added to the buffer.

After the buffer is filled, the training loop starts. In each training iteration, a batch of experiences is sampled randomly from the replay buffer to update the SAC network's Q-functions and policy through gradient steps. The current observation is then used to generate a new action with the updated policy. This new action is used by the agent in the environment. The resulting transition is added to the replay buffer. This cycle of training and interaction with the environment and adding to the buffer continues until a predefined number of training steps is reached. This gradually fine-tunes the pretrained policy to better generalize beyond the expert data.

Algorithm 2 Fine-tuning SAC with IL Pretraining and Replay Buffer Prefill

```
model = SACNetConvNext(action dimension, observation space)
pretrained_net = load(SACNetConvNext_pretrained)
model_filled = transfer(model, pretrained_net)
obs = env.reset()

Filling Buffer
for step = 1 to n_prefill do
    action = model.predict(obs)
    next_obs, reward, done = env.step(action)
    replay_buffer.add(obs, action, reward, next_obs, done)
    if any done then
        env.reset_done(done)
    end if
    obs = next_obs
end for

Training
for train_step = 1 to n_timesteps do
    batch = replay_buffer.sample(batch_size)
    update_q_functions(batch)
    update_policy(batch)
    adjust_entropy_coefficient()
    action = model.sample_action(obs)
    next_obs, reward, done = env.step(action)
    replay_buffer.add(obs, action, reward, next_obs, done)
    if any done then
        env.reset_done(done)
    end if
    obs = next_obs
end for
```

To validate the impact of pretraining with IL, a comparison will be done between RL with and without pretraining with IL. For this, two distinct sets of parameters are used. When

training the network with only RL, a higher learning rate is needed, because the network must learn all behavior from random initialization, therefore a learning rate of $3.0 \cdot 10^{-4}$ was used. A higher learning rate helps by making larger updates early on in training, which is beneficial when there is no prior knowledge. In contrast, the pretrained network already contains useful knowledge from IL, so a lower learning rate is preferred to allow gradual fine-tuning without overwriting previously learned behaviors. To preserve this knowledge while still allowing the agent to adapt and improve, a lower learning rate of $1.0 \cdot 10^{-5}$ was used. This helps maintain a balance between learning new information and keeping the benefits of pretraining.

Entropy regularization is also handled differently between the two cases. For the only RL case, the entropy coefficient term is initially set at 0.2 and then decays automatically over time. This encourages a high degree of exploration early on in the training, which is needed when the agent has no prior behavioral knowledge. Slowly during training the entropy is reduced, allowing the policy to stabilize.

For the IL pretrained network, the entropy coefficient starts at 0.01. Since the transferred weights already guide the agent towards reasonable behavior, less exploration is needed. The entropy coefficient also decays automatically to stabilize the policy after a while.

Other hyperparameters are kept the same with and without pretraining. A batch size of 256 was chosen to balance training stability and computational load. The discount factor was set to 0.99 to focus on long-term reward accumulation, since the most reward can be obtained through completing exploration. The experience replay buffer is set to a size of 500,000 steps, providing a diverse set of experiences for learning. Before training begins, the buffer is prefilled with 10,000 transitions from the IL policy. In the RL-only setup, this initial data comes from taking random actions, while in the pretrained setup, the buffer is filled with trajectories generated by the IL policy. This ensures that training starts with a range of behaviors, including both successful and less effective trajectories learned by the IL algorithm.

6 Experiments

The following section explains the experiments conducted to validate the proposed methodology. The experiments are divided into two categories: Imitation Learning and Reinforcement Learning. The IL experiments investigate the influence of different observation combinations and different trajectory numbers on the IL policy performance. The RL experiment compares standard RL with discrete and continuous actions to RL with IL pretraining with continuous actions.

6.1 Imitation Learning Experiments

The aim of these experiments is to evaluate the impact that different types of observations, as well as the amount and type of expert trajectories, have on IL performance. A key focus is improving the generalizability.

6.1.1 Testing Observation Combinations

Various types of observations will be tested to research their impact on IL performance. The goal is to identify what works well for IL, so that it does not overfit to any single given observation. The observations that will be used are the linear and angular velocity, the egocentric occupancy map, a semantic image, a depth line and a semantic line. A depth image was initially considered, but preliminary tests indicated that a depth line provided sufficient depth information to the agent while requiring fewer parameters. The combinations of observations used are:

- O_1: Occupancy Map and Depth Line
- O_2: Occupancy Map, Depth Line, Linear Velocity and Angular Velocity
- O_3: Occupancy Map, Depth Line and Semantic Image
- O_4: Occupancy Map, Depth Line and Semantic Line
- O_5: Occupancy Map, Depth Line, Semantic Image, Linear Velocity and Angular Velocity
- O_6: Occupancy Map, Depth Line, Semantic Line, Linear Velocity and Angular Velocity

These observation combinations are designed to evaluate the contribution of different sensory inputs to the performance of the IL policy. All combinations include depth information and an egocentric occupancy map, as these were shown in the previous work by Bravo i Forn, A [83] to be effective for autonomous exploration. The depth provides spatial awareness by helping the agent detect nearby objects, and the occupancy map tracks where the agent has and has not explored, this is a guide for efficient coverage of the environment. Adding to this baseline are semantic observations, these are added to provide a higher-level understanding of the environment. This gives the agent the opportunity to recognize meaningful features such as doorways or open spaces, which can indicate the potential for discovering new areas and more rewards. Another observation is the addition of linear and angular velocity data. This gives the agent information about its previous movement and can help in maintaining consistent rotational directions to avoid obstacles and adapt motion based on its current velocity. For this experiment 20 expert trajectories were used that were recorded in the Isaac Sim simulation.

6.1.2 Testing Number of Trajectories

IL often struggles with generalization, so the effect of the number of expert trajectories on the performance will be investigated. The observation combination with the best performance will be used for this experiment. The IL algorithm will be trained with varying amounts of expert trajectories, with the number of trajectories ranging from 5 to 30.

After training, each policy is evaluated across three previously unseen environments, the three testing environments A, B and C, to assess generalization beyond the training distribution. These environments are earlier discussed in the Environment setup in Section 4. The agent is spawned in 15 different locations throughout the scene with 4 different orientations resulting in 60 different starting positions for each scene.

Each trained policy will be evaluated on the following metric:

- **Area Coverage:** The average percentage of total area covered in a scene.
- **Time:** The average number of seconds until an episode is terminated either due to collision, timeout or successful completion.
- **Collision Rate:** The percentage of episodes that end in collision.
- **Success Rate:** The percentage of successful explorations from all episodes. In this report, success is defined as achieving at least 80% area coverage.

6.2 Reinforcement Learning Experiment

The goal of this experiment is to evaluate the effectiveness of pretraining with IL in improving RL performance, convergence speed, sample efficiency and generalization. The two configurations that will be compared are:

- A baseline SAC agent trained with only RL.
- A hybrid agent that is first pretrained using IL and then fine-tuned with RL.

As an additional point of comparison, the implementation from a previous student's project is used as a baseline [83]. That student implemented exploration on a UAV with the PPO algorithm with a discrete action space.

The SAC agents are trained for an equal number of timesteps to ensure a fair comparison, using the observation combination that achieved the best performance in the IL experiments. For the PPO baseline, the best exploration results reported in [83] are used.

The same evaluation metrics are used here as in the IL experiments.

7 Results

This section presents the outcomes of the experiments. The results are divided into two main parts: Imitation Learning Results and Reinforcement Learning Results.

7.1 Imitation Learning Results

All IL algorithms were trained for 2000 training steps with the parameters described in Section 5.2. The expert datasets used for training the IL algorithms were recorded in the training environments 1, 2, and 3. For validation, the trained policies were deployed in the testing environments A, B, and C.

7.1.1 Results for Observation Combinations

The results from the IL experiment with the different observation combinations over the testing scenes A, B and C can be seen in Table 7.1, 7.2 and 7.3, respectively. These tables show the area coverage, time, collision rate, and success rate for the different combinations of observations. The exact definitions of these metrics are explained in the Experiments Section 6.

For each scene, the area coverage and time are averaged over 15 different starting positions with 4 orientations each, and the corresponding standard deviations are reported. The collision rate and success rate are computed as the percentage of episodes that end in collision or meet the success criterion across all positions and orientations, so there is no mean or standard deviation. For this experiment, 20 expert trajectories were used.

The observations used are:

- O_1: Occupancy Map and Depth Line
- O_2: Occupancy Map, Depth Line, Linear Velocity and Angular Velocity
- O_3: Occupancy Map, Depth Line and Semantic Image
- O_4: Occupancy Map, Depth Line and Semantic Line
- O_5: Occupancy Map, Depth Line, Semantic Image, Linear Velocity and Angular Velocity
- O_6: Occupancy Map, Depth Line, Semantic Line, Linear Velocity and Angular Velocity

Table 7.1: Testing results from trained IL algorithms with different observations in scene A.

Scene A				
Observations	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
O_1	41.8±26.9	547.7±384.8	84.3	15.7
O_2	10.8±5.2	637.8±474.5	93.0	0
O_3	16.7±8.9	226.2±241.8	98.1	0
O_4	55.7±28.4	643.9±541.7	69.3	30.7
O_5	23.9±16.9	571.7±462.3	92.2	0
O_6	17.6±2.2	1371.3±430.1	21.6	0

In scene A, O_4 achieved the highest area coverage and success rate, while most other observation combinations resulted in high collision rates and low success. Especially the observations containing linear and angular velocity had a low area coverage and success rates.

Table 7.2: Testing results from trained IL algorithms with different observations in scene B.

Scene B				
Observations	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
O_1	31.9±15.2	288.9±156.4	100	0
O_2	19.5±5.3	447.7±334.8	100	0
O_3	12.7±5.9	140.9±94.2	100	0
O_4	40.1±22.8	364.2±202.5	91.9	8.1
O_5	18.4±10.2	482.8±329.6	100	0
O_6	17.0±7.2	494.3±356.2	100	0

In scene B, O_4 again outperformed the other combinations, although the overall success rate is very low.

Table 7.3: Testing results from trained IL algorithms with different observations in scene C.

Scene C				
Observations	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
O_1	27.5±18.4	483.0±375.1	94.1	1.9
O_2	11.5±4.7	904.8±626.8	56.8	0
O_3	11.7±6.5	207.8±215.0	100	0
O_4	50.5±24.8	1306.2±762.0	87.3	12.6
O_5	16.7±13.4	505.2±365.6	96.2	0
O_6	10.6±4.5	994.3±656.2	37.3	0

In scene C, O_4 still remains the best performing combination with a success rate of 12.6%. O_1 is the second best performing combination with a success rate of 1.9%.

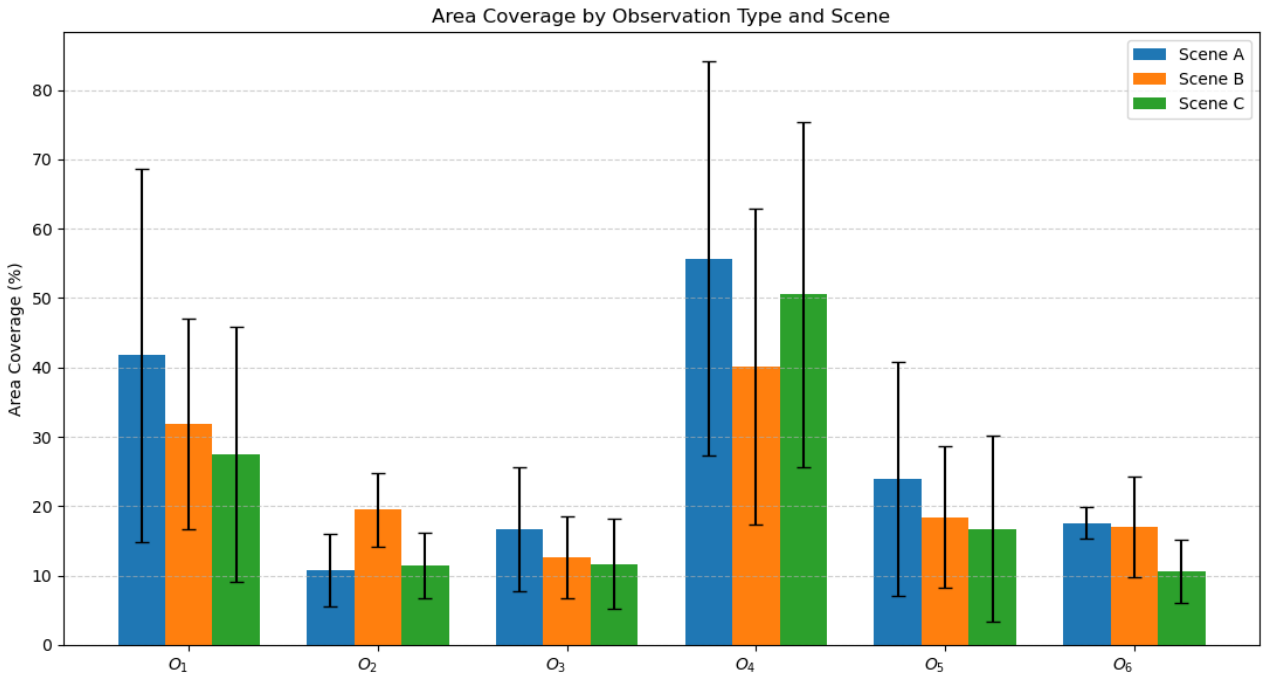


Figure 7.1: Bar chart of area coverage by observation combinations and scenes.

Across all testing scenes, O_4, the combination with an occupancy map, depth line, and semantic line achieved the highest success rates compared to all other observation combinations, with

30.7%, 8.1%, and 12.6% in scenes A, B, and C, respectively. This combination also achieved relatively high area coverage metrics compared to the other combinations. Figure 7.1 shows a bar chart of all area coverage percentages across all scenes. It further illustrates that O_4 was the best observation combination with O_1 second. It also portrays that adding velocity (O_2, O_5, O_6) decreases performance.

7.1.2 Results for Number of Trajectories

The results from the IL experiment with different numbers of trajectories across testing scenes A, B, and C are shown in Table 7.4, 7.5, and 7.6, respectively.

In scene A, increasing the number of trajectories from 5 to 15 led to noticeable improvements in both area coverage and success rate, with the success rate rising from 10.8% at 5 trajectories to 52.9% at 15 trajectories. Beyond 15 trajectories, this steep increase in performance plateaued. The model trained with 25 trajectories achieved the highest area coverage of 73.9% and success rate of 58.1%, while the model trained with 30 trajectories showed a decrease in success rate. Collision rates showed a decreasing trend between 10 and 25 trajectories.

Table 7.4: Testing results from trained IL algorithms with different numbers of trajectories in scene A.

Scene A				
Number of Trajectories	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
5	45.7±25.4	576.2±416.2	89.2	10.8
10	33.4±21.5	315.6±260.8	97.3	2.7
15	69.5±26.2	801.8±360.9	47.1	52.9
20	55.7±28.4	643.9±541.7	69.3	30.7
25	73.9±23.7	912.5±385.4	41.9	58.1
30	62.4±23.9	819.9±406.0	71.6	28.4

In scene B, the effect of additional trajectories was less noticeable, but the success rate increased with more trajectories, from 4.2% and 12.2% for 5 and 30 trajectories, respectively. Area coverage showed a gradual increase, from 34.5% with 5 trajectories to 45.9% with 30 trajectories. However, collision rates stayed high across all experiments (ranging from 87.8% to 95.7%).

Table 7.5: Testing results from trained IL algorithms with different numbers of trajectories in scene B.

Scene B				
Number of Trajectories	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
5	34.5±22.0	354.7±214.7	95.7	4.2
10	28.3±23.9	254.8±356.8	90.5	8.1
15	42.5±23.5	432.4±280.1	90.5	9.5
20	40.1±22.8	364.2±202.5	91.9	8.1
25	42.3±22.9	482.4±348.9	94.6	5.4
30	45.9±25.6	512.6±289.6	87.8	12.2

In scene C in Table 7.6, increasing the number of trajectories from 5 to 15 led to improvements in both area coverage and success rate, with coverage rising from 40.6% to 52.5% and the success rate rising from 5.4% to 24.3%. At 10 trajectories, however, performance dropped sharply, with coverage decreasing to 23.7% and no successful episodes. Beyond 15 trajectories,

the performance did not increase anymore, with area coverage remaining around 50-53% and success rates between 12.6% and 27.1%.

Table 7.6: Testing results from trained IL algorithms with different numbers of trajectories in scene C.

Scene C				
Number of Trajectories	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
5	40.6±23.8	954.6±862.1	94.6	5.4
10	23.7±18.3	618.9±550.3	100	0
15	52.5±27.3	1479.6±891.2	75.7	24.3
20	50.5±24.8	1306.2±762.0	87.3	12.6
25	52.7±29.8	1490.4±956.8	72.9	27.1
30	50.4±28.0	1616.8±980.9	75.7	24.3

Figure 7.2 presents the bar chart of area coverage by number of trajectories for scenes A, B, and C. It clearly illustrates the patterns earlier discussed. An increasing trend between 5 and 15 trajectories can be seen. A drop in performance is observed at 10 trajectories. Beyond 15 trajectories, the performance stabilized across scenes B and C. Scene A had more variation, achieving the highest coverage at 25 trajectories.

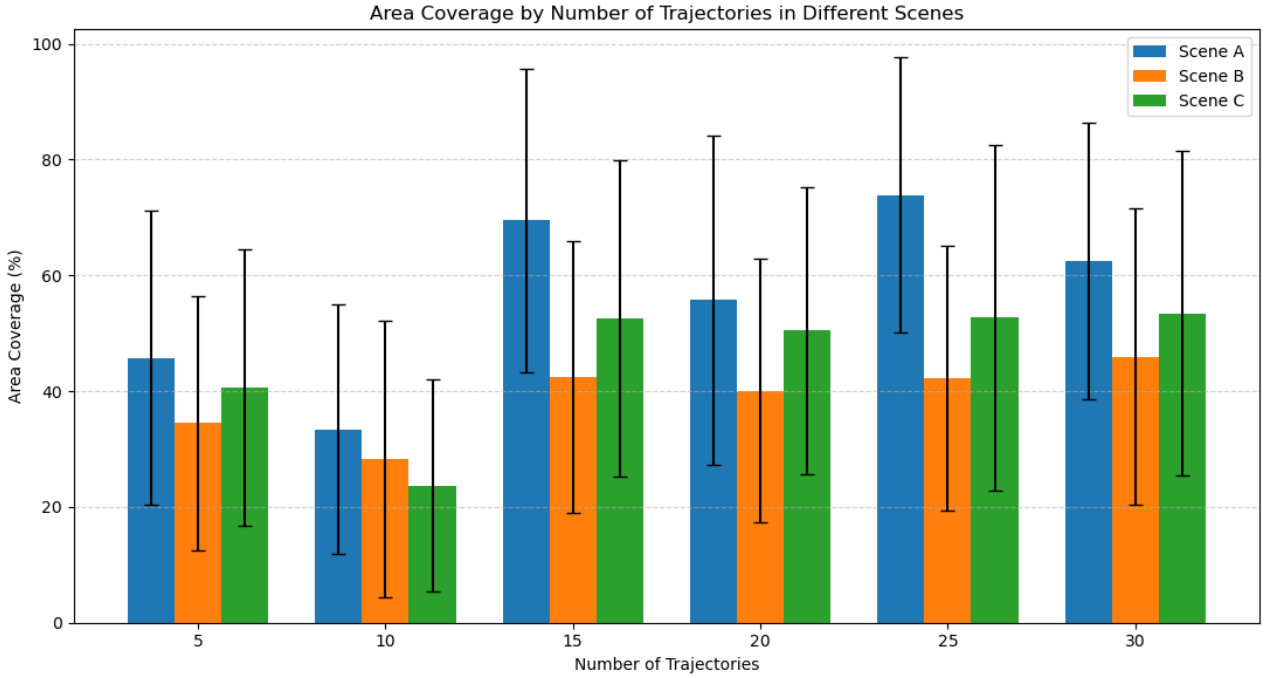


Figure 7.2: Bar chart of area coverage by different numbers of trajectories and scenes.

7.2 Reinforcement Learning Results

This section presents the results of the RL experiment. Three configurations were evaluated:

1. SAC pretrained with BC with continuous actions,
2. SAC with continuous actions, and
3. PPO with discrete actions from Bravo i Forn, A. [83].

The objective of this experiment was to assess whether pretraining the RL policy with IL improves performance and convergence speed. The same evaluation setup was used as in the previous experiments: 15 starting positions with 4 different orientations across the three testing environments A, B, and C with the same performance metrics.

For training, both the hybrid BC-SAC and the pure SAC agents were set to run for a maximum of 10 million steps. This choice was motivated by prior work, where 10 million steps were used for a relatively simple 2D UAV exploration task in a square room [91]. However, the policy converged earlier, after approximately 7 million steps, at which point a checkpoint was saved and used for evaluation. For comparison, the pure SAC algorithm was trained for the same 7 million steps. This setup allows for direct comparison of whether BC-SAC achieves faster convergence under the same training time.

The following Tables 7.7, 7.8 and 7.9 show the results of the RL experiments of PPO with discrete actions, BC with SAC, and SAC with continuous actions in the three testing environments A, B and C, respectively. In each table, the algorithms are compared on area coverage, exploration time, collision rate, and success rate. Representative occupancy maps of the hybrid IL-SAC agent are also shown for each scene.

Table 7.7: Testing results from trained RL algorithms in scene A.

Scene A				
Algorithm	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
PPO with discrete actions [83]	70	2613.7	18	0
BC with SAC continuous actions	88.3±12.2	784.9±373.9	7.0	87.7
SAC continuous actions	65.1±24.8	1853.3±852.7	2.2	39.1

For the results of the hybrid method of BC with SAC in scene A, it seems that the results are fairly consistent with a much lower standard deviation than with only BC. It also has a high success rate of 87.7% and high average area coverage of 88.3%. The hybrid method performs better than the policies trained purely with DRL, with higher area coverage, less time, less collisions, and a higher success rate. The pure SAC agent reached a moderate success rate of 39.1% with similar coverage to the PPO agent, while this agent achieved reasonable coverage but a 0% success rate.

Figure 7.3 shows what the average occupancy map looks like for scene A with the hybrid BC-SAC. The agent started in the middle of the scene and ended in the top right room.

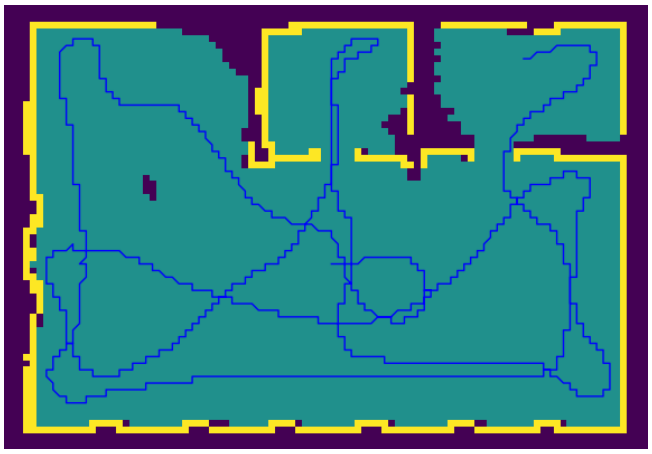


Figure 7.3: Occupancy map of the hybrid method of IL and RL in testing scene A

Table 7.8: Testing results from trained RL algorithms in scene B.

Scene B				
Algorithm	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
PPO with discrete actions [83]	61	2432.2	29	5
BC with SAC continuous actions	88.2±19.3	1255.1±812.4	16.2	83.8
SAC continuous actions	72.6±23.4	1798.9±686.1	0	74.0

In scene B (Table 7.8), the hybrid method again performed well, with 83.8% success and 88.2% coverage. Performance was similar to scene A, although exploration time was longer. The pure SAC policy achieved a comparable success rate of 78.0% but lower coverage, while the PPO agent had a much lower success rate of 5%. The PPO agent also had considerably more exploration time than the other two policies. Figure 7.4 shows an average occupancy map for scene B with the hybrid BC-SAC. The agent started in the top left room of the scene and ended on the right side of the environment.

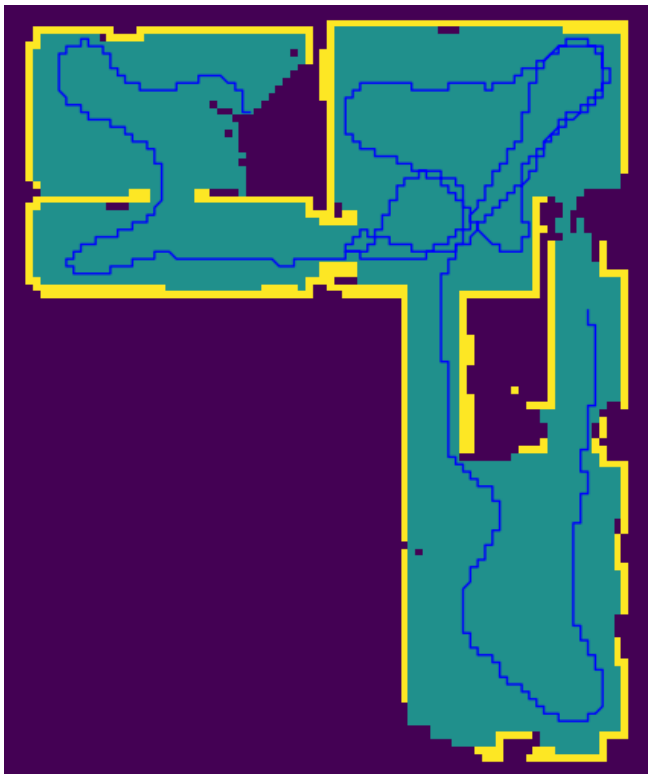


Figure 7.4: Occupancy map of the hybrid method of IL and RL in testing scene B

Table 7.9: Testing results from trained RL algorithms in scene C.

Scene C				
Observations	**Area Coverage (%)**	**Time (s)**	**Collision Rate (%)**	**Success Rate (%)**
PPO with discrete actions [83]	98	582.4	6	100
BC with SAC continuous actions	88.6 ±14.6	1849.0±939.6	10.4	89.6
SAC continuous actions	49.3±12.5	1477.7±1000.3	34.3	8.4

In scene C the PPO with discrete actions performs the best. However this comparison is not fair, since that algorithm was trained in environment C, while the BC with SAC and pure SAC have not been trained in that scene. Despite this, the hybrid method still achieves strong performance, with no collisions, a success rate of 89.6%, and an average area coverage of 88.6%.

By contrast, pure SAC performs poorly in this scene, with a low success rate of 8.4% and reduced coverage of 49.3%. A look at the occupancy maps showed that the agent often became stuck circling in the middle of the room, failing to explore efficiently.

Figure 7.5 shows what the average occupancy map looks like for scene C with BC and SAC. The agent started in the bottom right of the scene.

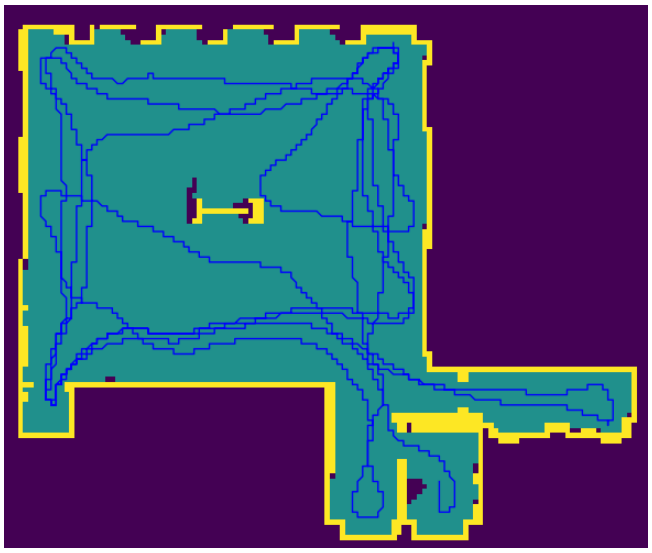


Figure 7.5: Occupancy map of the hybrid method of IL and RL in testing scene C

Figure 7.6 shows a bar chart of area coverage across all three testing scenes for the trained RL algorithms. The bar chart clearly illustrates that the hybrid BC-SAC agent consistently achieves the highest area coverage in scenes A and B, with averages of 88.3% and 88.2%, respectively, outperforming both pure SAC with continuous actions and PPO with discrete actions. From this chart it can also be seen that the hybrid method of BC-SAC has a consistent average value across all scenes.

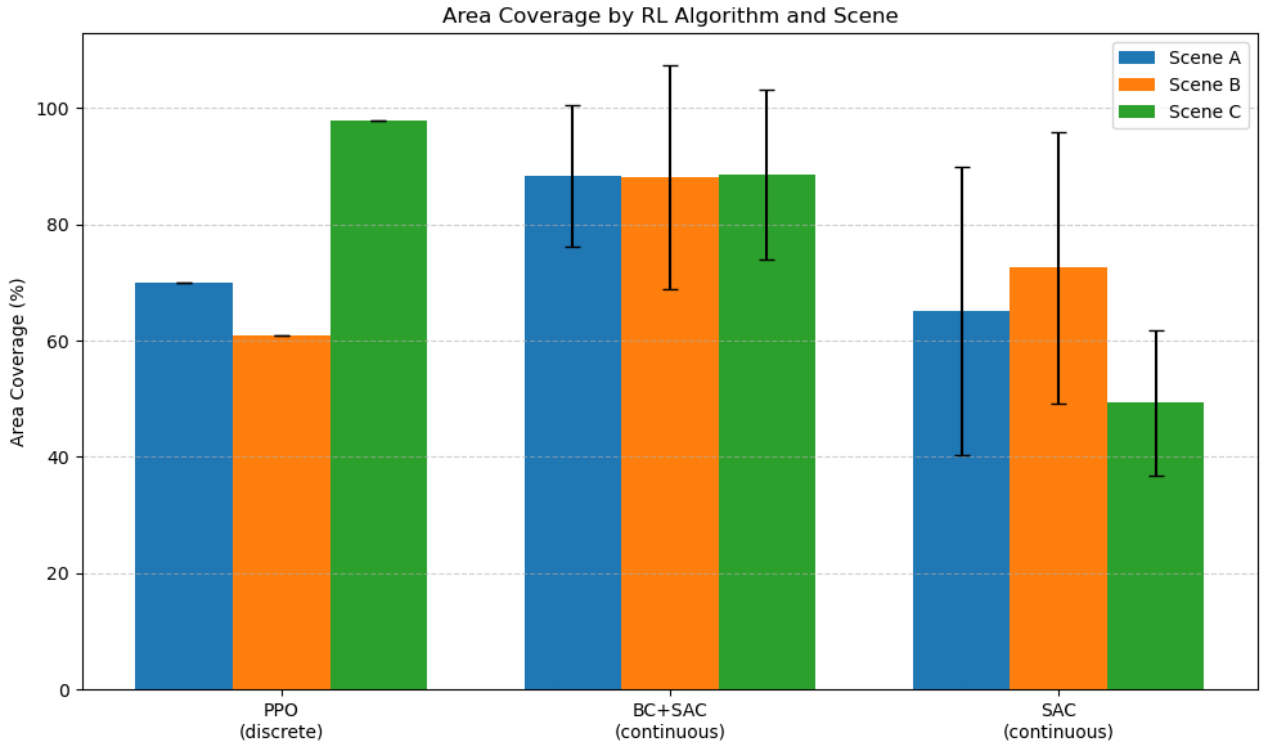


Figure 7.6: Bar chart of area coverage for the trained RL algorithms in testing scenes.

8 Discussion

The discussion is divided into multiple sections. These sections consist of the imitation learning experiment results, the reinforcement learning experiment results, the research questions, and the suggestions for future work.

8.1 Imitation Learning

Two experiments were performed to test different aspects of the IL policies. These experiments included testing different combinations of observations and testing different amounts of expert trajectories.

From the results of the first experiment, it could be concluded that the combination of the occupancy map, depth line, and semantic line was the most effective combination across all testing scenes. This combination had the highest success rates of 30.7%, 8.1%, and 12.6% in scenes A, B, and C, respectively. This combination also had relatively high area coverage metrics compared to the other combinations. This suggests that semantic information contributes to a more meaningful understanding of the environment.

In contrast, combinations that included the full semantic image or linear and angular velocities underperformed across all scenes. The full semantic image resulted in lower area coverage and success rates, along with higher collision rates, despite the expectation that semantic information would enhance scene understanding. This was true for the semantic line observations, but not for the full semantic image, which is likely due to the excessive complexity and the increased difficulty in training caused by the high-dimensional input. This led to a degradation in policy performance. Similar challenges are noted in the literature: learning directly from high-dimensional visual representations is difficult because of the large input space, their diversity, and the limited generalization to visual perturbations [92]. In this case, although the semantic image is first encoded into a feature vector using ConvNeXt before being passed to the RL agent, the resulting representation is still substantially higher-dimensional than the compact semantic line input. This likely explains why the semantic line gives more stable and effective policies.
Additionally, the inclusion of velocity observations did not lead to effective exploration. Since the angular and linear velocities reflect the previous actions, the network may have learned to replicate the input velocity as its own output action, leading to poor learning of the policy. It was expected that knowledge of the previous action would help in choosing the new action, but in literature it has been found that recurrent or history based imitation hurts performance and causes misidentification [93, 94]. This behavior was evident in the occupancy maps and the collision rate. The agent often flew in circles during the whole episode, especially in Scenes A and C for O_6 due to the large open space. There the collision rate is at 21.6% and 37.3% respectively with a success rate of 0% and a low area coverage percentage, indicating that the drone flew in circles. Another noteworthy result from the IL tests is that the performance is not very consistent and often shows a high standard deviation, more than half of the average value. This indicates that when the IL algorithm encounters a scenario from the expert data, it performs well. However, even slight deviations from the training distribution lead to poor performance. This pattern reflects a common limitation of IL, sensitivity to distributional shift and poor generalization beyond the demonstrations [46, 48]. Overall, these results suggest that more compact, structured observation combinations with semantic information give more reliable policy performance.

From the IL experiments with an increasing number of expert trajectories, an observation can be made that more trajectories do not lead to consistent improvements in performance. In scene A and C, in the beginning with more trajectories the area coverage and success rates improved, but between 15 and 30 trajectories this improvement plateaued. In scene B, more trajectories had minimal effect on the success rate and on coverage above 20 trajectories.

This suggests that the quality and diversity of expert demonstrations are more important than quantity. Initially, adding new trajectories introduces new variations in the environment, such as alternative paths or unique obstacle encounters, which significantly improve learning. However, once these variations are captured, additional trajectories tend to be repetitive and contribute less to policy improvement. This aligns with findings in the IL literature, where informative and representative demonstrations are essential for generalization and effective policy learning [14, 49].

A limitation of the IL policies is that they are learned through BC and not with a more advanced method such as DAgger. This choice was initially made because there was an idea to incorporate real-world trajectories into the IL algorithm to help with the sim-to-real gap. However, BC struggles with distributional shift, meaning it performs poorly in scenarios not well represented in the expert demonstrations. For example, the expert demonstrations contained almost no situations where the UAV flew close to walls. As a result, the policy trained with BC failed to generalize to such cases and often crashed when encountering them. This problem was mitigated during the fine-tuning process with RL but did limit the IL performance.

Another problem with IL is that there is no history given to the algorithms, only the egocentric occupancy map from which it has to determine which way to rotate. This can sometimes cause issues for the agent, where it gets stuck in a loop of rotating clockwise and counter-clockwise. It gets stuck because it has no memory of its previous actions.

8.2 Reinforcement Learning

From the results of the RL experiments, it can be concluded that using IL as pretraining and fine-tuning with RL using continuous actions gives better results than the IL policies, the PPO with discrete actions, and the SAC with continuous actions. Not only are the trajectories more efficient, as observed from the lower completion times, but the success rate is also much higher with increased area coverage in scenes A and B. In literature, it was also found that a hybrid method of IL with RL showed superior performance over only RL [54, 55]. Adding continuous actions to this is assumed to further have enhanced the performance, as previous studies have shown that continuous actions outperform discrete ones by providing more precise control and better adaptability [95, 96].

This demonstrates that pretraining with IL provides a strong initial policy that guides the agent toward reliably accomplishing full exploration, while RL fine-tuning refines the policy to handle variations and obstacles more effectively. Continuous action methods allow for finer control of the agent's movements, enabling smoother adjustments in speed and direction compared to the coarser, stepwise decisions of discrete actions. The occupancy maps created through the IL-RL hybrid also generated efficient maps, close to optimal, with minimal redundancy from revisiting the same areas.

A big difference that can be found between the policies of IL with SAC and purely SAC is that

convergence happened much earlier, leading to faster training and a better policy with the same number of training steps. The hybrid method was able to learn from the expert demonstrations how to exit a room efficiently. In contrast, the pure SAC policy often got stuck looping inside rooms, requiring longer exploration and achieving lower coverage. While SAC can eventually learn this behavior, it does so at the cost of significantly more training time.

Another noteworthy comparison is between the best performing IL policy of 25 trajectories and pure SAC. Both achieve similar area coverage and comparable success rates across all scenes. However, SAC exhibits almost no collisions but requires significantly more time, suggesting that it has learned to avoid walls but follows suboptimal trajectories. In contrast, the 25 trajectory IL policy shows a much higher collision rate but has a significantly lower time while achieving the same coverage. This indicates that the IL policy is much more optimal in its trajectory but has not learned to avoid walls.

There are, however, a few problems worth noting with the IL-RL algorithm. Due to limited training time and server constraints, the hyperparameters are not optimal for the scenario. No parameter sweep could be performed, which means the current results are likely sub-optimal. With more careful tuning, performance could be improved. Another point to note is that, although it happens rarely, the UAV can get stuck in a loop in smaller rooms. Because of the layout, it sometimes searches in a certain pattern that keeps the door hidden, causing the UAV to circle along the walls.

In the training environment, smaller doorways were used compared to the testing environments. These testing environments were chosen for direct comparison with a similar algorithm in Isaac Sim by a previous student [83]. During training, these narrow doorways caused difficulties for the drone since the doorways are 90cm wide while the drone is 55 cm wide, leaving little room for error. As a result, the UAV often failed near these doorways, avoided them, or came to almost a complete stop, unsure of how to proceed. By adding a penalty for idling and a reward for passing through doorways, this behavior was mitigated.

Scene C was also somewhat more challenging for the IL-RL hybrid due to the lack of training environments with large open spaces. This caused some difficulty in finding an optimal path. In this scene, the path is less optimal compared to the other rooms. Its behavior goes through doors and moves along walls but does not efficiently move through open spaces, it rotated along the walls multiple times before it came into a scenario where it went more along the middle of the room. While it was able to explore the open area, there was much more variation in the time taken to fully explore it, resulting in a higher standard deviation. A stark contrast can be seen in the behaviors of the hybrid and pure SAC policies in this environment. The hybrid method did explore the room albeit, taking a bit more time. The pure SAC policy was unable to effectively explore the big open space and often got itself stuck circling the same area. The policy could not generalize its exploration strategy to this novel scenario.

8.3 Research Questions

This section provides answers to the research questions introduced in Section 3.

The main research question was:
To what degree can a hybrid approach combining imitation learning with reinforcement learning enhance the indoor exploration capabilities of UAVs in unknown environments?

Based on the results of this thesis, it can be concluded that combining IL with RL significantly

improves UAV indoor exploration performance in unknown indoor environments. In environment A, the hybrid had a success rate almost 50% higher compared to RL with SAC alone and almost 90% higher than PPO. In environment B, the BC-SAC policy had an improvement of almost 80% over PPO with discrete actions and an improvement of 10% over SAC. The hybrid method also outperforms the SAC policy in scene C and achieved only a 10% lower success rate than the PPO policy, despite the latter being trained in scene C.

- **What is the optimal number of expert trajectories needed and how does the quality of expert data influence the final performance?:** The results show that 15 to 25 trajectories provide the best performance for the used training and testing environment, achieving high area coverage and success rates with relatively low collisions. Surprisingly, using 30 trajectories did not improve results, likely due to inconsistencies or suboptimal behaviors in the additional data. This indicates that more trajectories are not always better. Quality of expert data matters more than quantity.
- **How many different environments are needed for robust and generalizable training?:** In this thesis, the IL-RL policy was trained on three different environments and tested on three unseen environments. While this is a limited number of training environments, the final policy achieved high coverage and success rates on the test environments, suggesting generalizability. However, it is difficult to determine the minimum number needed for truly robust and generalizable training. Future work could investigate this by systematically increasing the number and diversity of training environments to better understand how it affects generalization performance.
- **To what extent does the hybrid policy generalize to previously unseen environments with varying obstacles or dynamic changes, such as moving objects or altered terrain?:** The final IL-RL policy demonstrated strong generalization to unseen layouts in the Isaac Sim office environment. Across all testing scenes, it achieved approximately 88% area coverage with an average success rate of 87%, indicating that the policy can effectively handle new layouts and altered terrain, even though it was not trained on certain features such as large open spaces. Other scene types are expected to have similar performance, because the agent only receives semantic information about walls, doors, ceilings, floors, unlabeled areas, and unknown space. Different furniture should not affect the policy, as these objects are classified as unlabeled. Additionally, since the drone does not have an RGB image input and only uses depth information, variations in color or lighting should not influence the policy's performance.

  Regarding dynamic obstacles, no conclusions can be drawn, as the policy was not tested in such scenarios. This is an area for future work, where the robustness of the IL-RL policy against moving objects or dynamic changes could be tested and potentially improved.
- **Does the use of semantic image inputs improve exploration performance compared to non-semantic visual inputs?:** From the IL experiments with different observation combinations, it can be concluded that incorporating semantic information can improve exploration performance. Specifically, as shown in Tables 7.1, 7.2, and 7.3, using semantic line observations resulted in superior area coverage, fewer collisions, and a higher success rate compared to observations without semantic information.

  In contrast, using the full semantic image led to lower area coverage, reduced success rates, and more collisions. This was likely due to the increased input complexity and higher dimensionality, which made training more difficult and caused a degradation in

policy performance. These results suggest that carefully chosen semantic representations, such as semantic lines, can enhance exploration, whereas overly complex semantic inputs may hinder learning.

8.4 Future Work

The methodology was originally designed to be able to incorporate real world expert datasets. This was why an offline method like BC was chosen over an online method such as DAgger. Ultimately, there was some trouble with obtaining the real-world trajectories in time so this approach was not used in the experiments and for the final IL-RL policy. In future work, this can be an area of research, to see the influence on generalizability by using real-world data with the sensor noise and latency.

To further test the generalizability of the learned IL-RL policies, they can be tested on a physical drone to evaluate their performance under real-world conditions, including sensor noise, latency, and unmodeled dynamics.

Additionally, the policies for IL and RL were trained and tested in static environments only. Future work could look into the robustness of IL-RL hybrids in dynamic setting. This further tests generalizability.

Another line of improvement is with optimization. No extensive parameter sweep was performed on the algorithm, so testing additional hyperparameter combinations for the IL and RL algorithms can enhance performance.

Finally, introducing memory to the algorithm through Long Short-Term Memory, Gated Recurrent Units, or Transformers can help in reducing issues like looping or indecision in complex environments.

9 Conclusion

This thesis presented a novel hybrid approach that combined Imitation Learning (IL) with Soft Actor-Critic (SAC) based Deep Reinforcement Learning (DRL) to enable autonomous indoor exploration by UAVs with continuous actions.

This hybrid method was chosen to use the strength of both approaches. IL provides a strong initial policy mimicking expert action through BC, while SAC fine-tuning allows the policy to adapt and generalize beyond the demonstrations. SAC with its entropy term, further enhances the exploration capabilities of the algorithm. Furthermore, the hybrid framework mitigates common limitations of pure IL, such as high dependence on expert data, and overcomes difficulties in RL, including sparse rewards, sample inefficiency, and unstable training.

Across three unseen indoor test environments, the designed method achieved an average coverage of 88.4 $\pm$15.4%, with faster convergence, smoother navigation, and improved robustness compared to baseline DRL methods and IL methods. It successfully navigated large state-action spaces, avoided obstacles, and adapted to new layouts, confirming that the hybrid method enhances exploration efficiency and generalization.

Key novel contributions of this work include:

1. First application of a hybrid IL-RL approach with continuous actions for indoor UAV exploration. Combining IL with continuous-action SAC provided a strong initial policy from expert trajectories and improved robustness through RL fine-tuning.
2. Integration of semantic segmentation inputs to enhance spatial understanding. Using compact semantic representations allowed the agent to better understand the environment while keeping the input simple for effective learning.
3. Use of photorealistic Isaac Sim environments to reduce the sim-to-real gap. The simulator provides realistic visual inputs and accurate physics. This led to effective training and the simulator had automatic semantic labeling.
4. Systematic analysis of generalization factors, including environment variety and expert data quality. This thesis investigated how training diversity and the quality of expert trajectories influenced performance in unseen environments.

The findings show that incorporating semantic visual inputs enhances exploration performance by providing structured high-level environmental information. The experiments further show how diverse training environments affect generalizability and demonstrate that the quality of expert demonstrations is more important than quantity, by testing the UAV's ability to explore unknown indoor layouts. Furthermore, the use-case of continuous actions provides the agent with finer control of its movement, enabling smoother adjustments in speed and direction compared to discrete actions.

While promising, the framework requires further validation in real-world conditions, including handling dynamic obstacles, sensor noise, and latency. Future work will focus on dynamic environments and physical UAV tests to confirm sim-to-real transfer.

Overall, this thesis showcases that hybrid IL-RL provides a promising framework for autonomous UAV exploration in unknown indoor environments. By overcoming the challenges typically influencing DRL frameworks, it is a step towards implementing autonomous drone systems in applications such as search-and-rescue, inspection, and security.

References

[1] Sonia Waharte and Niki Trigoni. "Supporting Search and Rescue Operations with UAVs". In: *2010 International Conference on Emerging Security Technologies*. 2010, pp. 142–147. DOI: 10.1109/EST.2010.31.

[2] Zhengjun Wang et al. "Path Planning for Unmanned Aerial Vehicle via Off-Policy Reinforcement Learning With Enhanced Exploration". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 8.3 (2024), pp. 2625–2639. DOI: 10.1109/TETCI.2024.3369485.

[3] Reem Ashour et al. "Exploration for Object Mapping Guided by Environmental Semantics using UAVs". In: *Remote Sensing* 12.5 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12050891. URL: https://www.mdpi.com/2072-4292/12/5/891.

[4] Yumin Zhao, Jianlei Zhang, and Chunyan Zhang. "Deep-learning based autonomous-exploration for UAV navigation". In: *Knowledge-Based Systems* 297 (2024), p. 111925. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2024.111925. URL: https://www.sciencedirect.com/science/article/pii/S0950705124005598.

[5] Hartmut Surmann et al. "Small Commercial UAVs for Indoor Search and Rescue Missions". In: *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*. 2021, pp. 106–113. DOI: 10.1109/ICARA51699.2021.9376551.

[6] Daniel Schleich et al. "Autonomous Flight in Unknown GNSS-denied Environments for Disaster Examination". In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2021, pp. 950–957. DOI: 10.1109/ICUAS51884.2021.9476790.

[7] Khawaja Ghulam Alamdar et al. "Frontier Exploration for Disaster Management UAVs". In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2022, pp. 1098–1103. DOI: 10.1109/ROBIO55434.2022.10011693.

[8] Zichen Wang et al. "Efficient Autonomous UAV Exploration Framework With Limited FOV Sensors for IoT Applications". In: *IEEE Internet of Things Journal* 12.1 (2025), pp. 713–725. DOI: 10.1109/JIOT.2024.3467396.

[9] Zhefan Xu et al. *NavRL: Learning Safe Flight in Dynamic Environments*. 2024. arXiv: 2409.15634 [cs.RO]. URL: https://arxiv.org/abs/2409.15634.

[10] Andreas Seel et al. "Deep Reinforcement Learning Based UAV for Indoor Navigation and Exploration in Unknown Environments". In: *2022 8th International Conference on Control, Automation and Robotics (ICCAR)*. 2022, pp. 388–393. DOI: 10.1109/ICCAR55106.2022.9782602.

[11] Shuyuan Liu et al. "Autonomous navigation of UAV in complex environment : a deep reinforcement learning method based on temporal attention". In: *Applied Intelligence* 55.5 (2025). DOI: 10.1007/s10489-024-06036-2.

[12] Yanfei Zhu et al. "UAV Path Planning Based on Random Obstacle Training and Linear Soft Update of DRL in Dense Urban Environment". In: *Energies* 17.11 (2024). ISSN: 1996-1073. DOI: 10.3390/en17112762. URL: https://www.mdpi.com/1996-1073/17/11/2762.

[13] Bhaskar Joshi, Dhruv Kapur, and Harikumar Kandath. *Sim-to-Real Deep Reinforcement Learning based Obstacle Avoidance for UAVs under Measurement Uncertainty*. 2023. arXiv: 2303.07243 [cs.RO]. URL: https://arxiv.org/abs/2303.07243.

[14] Hui Lv et al. "Improve exploration in deep reinforcement learning for UAV path planning using state and action entropy". In: *Measurement Science and Technology* 35.5 (2024), p. 056206. DOI: 10.1088/1361-6501/ad2663. URL: https://dx.doi.org/10.1088/1361-6501/ad2663.

[15] C. Gabellieri and A. Franchi. *Lecture notes: Aerial Robotics 2024–2025*. Lecture notes. Master in Robotics of the University of Twente. 2025.

[16] Yinjie Jiang et al. "Artificial Intelligence for Retrosynthesis Prediction". In: *Engineering* 25 (2023), 3250. DOI: 10.1016/j.eng.2022.04.021.

[17] OpenAI. *Part 1: Key Concepts in RL - Spinning Up documentation*. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html.

[18] Elinor Ginzburg-Ganz et al. "Reinforcement Learning Model-Based and Model-Free Paradigms for Optimal Control Problems in Power Systems: Comprehensive Review and Future Directions". In: *Energies* 17.21 (2024), p. 5307. DOI: 10.3390/en17215307.

[19] Shixiang Gu et al. "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning". In: *CoRR* abs/1706.00387 (2017). arXiv: 1706.00387. URL: http://arxiv.org/abs/1706.00387.

[20] OpenAI. *Proximal Policy Optimization*. URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[21] OpenAI. *Deep Deterministic Policy Gradient*. URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html.

[22] OpenAI. *Twin Delayed DDPG*. URL: https://spinningup.openai.com/en/latest/algorithms/td3.html.

[23] OpenAI. *Soft Actor-Critic*. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html.

[24] Yuxi Sun and Chengrui Zhang. "Efficient and Safe Robotic Autonomous Environment Exploration Using Integrated Frontier Detection and Multiple Path Evaluation". In: *Remote Sensing* 13.23 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13234881. URL: https://www.mdpi.com/2072-4292/13/23/4881.

[25] Linzhen Shi et al. "Enhancing Adaptability: Hierarchical Frontier-Based Path Planning for Navigation in Challenging Environments". In: *IEEE Robotics and Automation Letters* 9.10 (2024), pp. 8611–8618. DOI: 10.1109/LRA.2024.3448137.

[26] Hong Zhang et al. "EFP: Efficient Frontier-Based Autonomous UAV Exploration Strategy for Unknown Environments". In: *IEEE Robotics and Automation Letters* 9.3 (2024), pp. 2941–2948. DOI: 10.1109/LRA.2024.3363531.

[27] Matan Keidar and Gal A. Kaminka. "Robot exploration with fast frontier detection: theory and experiments". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. AAMAS '12. Valencia, Spain: International Foundation for Autonomous Agents and Multiagent Systems, 2012, 113120. ISBN: 0981738117.

[28] Chaoqun Wang et al. "Efficient Autonomous Exploration With Incrementally Built Topological Map in 3-D Environments". In: *IEEE Transactions on Instrumentation and Measurement* 69.12 (2020), pp. 9853–9865. DOI: 10.1109/TIM.2020.3001816.

[29] Lin Li et al. "Improving Autonomous Exploration Using Reduced Approximated Generalized Voronoi Graphs". In: *Journal of Intelligent Robotic Systems* 99 (July 2020). DOI: 10.1007/s10846-019-01119-6.

[30] Haiming Gao et al. "Autonomous Indoor Exploration Via Polygon Map Construction and Graph-Based SLAM Using Directional Endpoint Features". In: *IEEE Transactions on Automation Science and Engineering* 16.4 (2019), pp. 1531–1542. DOI: 10.1109/TASE.2018.2883587.

[31] Liang Lu, Carlos Redondo, and Pascual Campoy. "Optimal Frontier-Based Autonomous Exploration in Unconstructed Environment Using RGB-D Sensor". In: *Sensors* 20.22

(2020). ISSN: 1424-8220. DOI: 10.3390/s20226507. URL: https://www.mdpi.com/1424-8220/20/22/6507.

[32] Ana Batinovic et al. "A Multi-Resolution Frontier-Based Planner for Autonomous 3D Exploration". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4528–4535. DOI: 10.1109/LRA.2021.3068923.

[33] Steven M. LaValle. *Rapidly-exploring Random Trees: A New Tool for Path Planning*. Tech. rep. TR 98-11. http://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf. Ames, IA, USA: Department of Computer Science, Iowa State University, 1998.

[34] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761. eprint: https://doi.org/10.1177/0278364911406761. URL: https://doi.org/10.1177/0278364911406761.

[35] Jin-Gu Kang, Yong-Sik Choi, and Jin-Woo Jung. "A Method of Enhancing Rapidly-Exploring Random Tree Robot Path Planning Using Midpoint Interpolation". In: *Applied Sciences* 11.18 (2021). ISSN: 2076-3417. DOI: 10.3390/app11188483. URL: https://www.mdpi.com/2076-3417/11/18/8483.

[36] Ana Batinovic et al. "A Shadowcasting-Based Next-Best-View Planner for Autonomous 3D Exploration". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2969–2976. DOI: 10.1109/LRA.2022.3146586.

[37] Andreas Bircher et al. "Receding Horizon "Next-Best-View" Planner for 3D Exploration". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1462–1468. DOI: 10.1109/ICRA.2016.7487281.

[38] Andr Ribeiro and Meysam Basiri. "Efficient 3D Exploration with Distributed Multi-UAV Teams: Integrating Frontier-Based and Next-Best-View Planning". In: *Drones* 8.11 (2024). ISSN: 2504-446X. DOI: 10.3390/drones8110630. URL: https://www.mdpi.com/2504-446X/8/11/630.

[39] Christian Witting et al. "History-aware Autonomous Exploration in Confined Environments using MAVs". In: *CoRR* abs/1803.10558 (2018). arXiv: 1803.10558. URL: http://arxiv.org/abs/1803.10558.

[40] Lukas Schmid et al. *An Efficient Sampling-based Method for Online Informative Path Planning in Unknown Environments*. Sept. 2019. DOI: 10.48550/arXiv.1909.09548.

[41] Dong Huu Quoc Tran et al. *An Enhanced Sampling-Based Method With Modified Next-Best View Strategy For 2D Autonomous Robot Exploration*. 2023. arXiv: 2305.04576 [cs.RO]. URL: https://arxiv.org/abs/2305.04576.

[42] Chaoqun Wang et al. "Towards autonomous exploration with information potential field in 3D environments". In: July 2017, pp. 340–345. DOI: 10.1109/ICAR.2017.8023630.

[43] Boyu Zhou et al. "FUEL: Fast UAV Exploration Using Incremental Frontier Structure and Hierarchical Planning". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 779–786. DOI: 10.1109/LRA.2021.3051563.

[44] Yanjie Liu et al. "A Soft Actor-Critic Deep Reinforcement-Learning-Based Robot Navigation Method Using LiDAR". In: *Remote Sensing* 16.12 (2024). ISSN: 2072-4292. DOI: 10.3390/rs16122072. URL: https://www.mdpi.com/2072-4292/16/12/2072.

[45] Ramanjeet Singh, Jing Ren, and Xianke Lin. "A Review of Deep Reinforcement Learning Algorithms for Mobile Robot Path Planning". In: *Vehicles* 5.4 (2023), pp. 1423–1451. ISSN: 2624-8921. DOI: 10.3390/vehicles5040078. URL: https://www.mdpi.com/2624-8921/5/4/78.

[46] Baihan Lin. "Behavioral Cloning and Imitation Learning". In: *Reinforcement Learning Methods in Speech and Language Technology*. Cham: Springer Nature Switzerland, 2025,

pp. 63–67. ISBN: 978-3-031-53720-2. DOI: 10.1007/978-3-031-53720-2_7. URL: https://doi.org/10.1007/978-3-031-53720-2_7.

[47] Christopher Mutschler, Georgios Kontes, and Sebastian Rietsch. "Learning from Experience". In: *Unlocking Artificial Intelligence*. Springer Nature, July 2024, pp. 49–76. ISBN: 978-3-031-64831-1. DOI: 10.1007/978-3-031-64832-8_3.

[48] Dylan J. Foster, Adam Block, and Dipendra Misra. *Is Behavior Cloning All You Need? Understanding Horizon in Imitation Learning*. 2024. arXiv: 2407.15007 [cs.LG]. URL: https://arxiv.org/abs/2407.15007.

[49] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudk. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 627–635. URL: https://proceedings.mlr.press/v15/ross11a.html.

[50] Changquan Wang and Yun Wang. "Uncertainty-Driven Data Aggregation for Imitation Learning in Autonomous Vehicles". In: *Information* 15.6 (2024). ISSN: 2078-2489. DOI: 10.3390/info15060336. URL: https://www.mdpi.com/2078-2489/15/6/336.

[51] Joshua Julian Damanik et al. "LiCS: Navigation Using Learned-Imitation on Cluttered Space". In: *IEEE Robotics and Automation Letters* 10.2 (Feb. 2025), 20002007. ISSN: 2377-3774. DOI: 10.1109/lra.2024.3518078. URL: http://dx.doi.org/10.1109/LRA.2024.3518078.

[52] Yu Wan, Jun Tang, and Zipeng Zhao. "Imitation Learning of Complex Behaviors for Multiple Drones with Limited Vision". In: *Drones* 7.12 (2023). ISSN: 2504-446X. DOI: 10.3390/drones7120704. URL: https://www.mdpi.com/2504-446X/7/12/704.

[53] Stephen Adams, Tyler Cody, and Peter A. Beling. "A survey of inverse reinforcement learning". In: *Artificial Intelligence Review* 55.6 (2022), 43074346. DOI: 10.1007/s10462-021-10108-x.

[54] Zihan Wang, Jianwen Li, and Nina Mahmoudian. "Synergistic Reinforcement and Imitation Learning for Vision-driven Autonomous Flight of UAV Along River". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 9976–9982. DOI: 10.1109/IROS58592.2024.10801487.

[55] Siyuan Li et al. *An Imitative Reinforcement Learning Framework for Autonomous Dogfight*. 2024. arXiv: 2406.11562 [cs.LG]. URL: https://arxiv.org/abs/2406.11562.

[56] Syed Ihtesham Hussain Shah and Giuseppe De Pietro. "An Overview of Inverse Reinforcement Learning Techniques". In: June 2021. ISBN: 9781643681863. DOI: 10.3233/AISE210097.

[57] Weiren Kong et al. "UAV Autonomous Aerial Combat Maneuver Strategy Generation with Observation Error Based on State-Adversarial Deep Deterministic Policy Gradient and Inverse Reinforcement Learning". In: *Electronics* 9.7 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9071121. URL: https://www.mdpi.com/2079-9292/9/7/1121.

[58] Tung Phan-Minh et al. *Driving in Real Life with Inverse Reinforcement Learning*. 2022. arXiv: 2206.03004 [cs.RO]. URL: https://arxiv.org/abs/2206.03004.

[59] Jonathan Booher et al. *CIMRL: Combining IMitation and Reinforcement Learning for Safe Autonomous Driving*. 2024. arXiv: 2406.08878 [cs.LG]. URL: https://arxiv.org/abs/2406.08878.

[60] Zun Liu et al. "A Hierarchical Reinforcement Learning Algorithm Based on Attention Mechanism for UAV Autonomous Navigation". In: *IEEE Transactions on Intelligent*

Transportation Systems 24.11 (2023), pp. 13309–13320. DOI: 10.1109/TITS.2022.3225721.

[61] Kai Kou et al. "Autonomous Navigation of UAV in Dynamic Unstructured Environments via Hierarchical Reinforcement Learning". In: (2022), pp. 1–5. DOI: 10.1109/ICARCE55724.2022.10046655.

[62] Jie Fan, Xudong Zhang, and Yuan Zou. "Hierarchical path planner for unknown space exploration using reinforcement learning-based intelligent frontier selection". In: *Expert Systems with Applications* 230 (2023), p. 120630. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120630. URL: https://www.sciencedirect.com/science/article/pii/S0957417423011326.

[63] Mahmut Nedim Alpdemir. "A Hierarchical Reinforcement Learning Framework for UAV Path Planning in Tactical Environments". In: *Turkish Journal of Science and Technology* 18 (Mar. 2023). DOI: 10.55525/tjst.1219845.

[64] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latr. "Hierarchical Reinforcement Learning: A Survey and Open Research Challenges". In: *Machine Learning and Knowledge Extraction* 4.1 (2022), pp. 172–221. ISSN: 2504-4990. DOI: 10.3390/make4010009. URL: https://www.mdpi.com/2504-4990/4/1/9.

[65] Adrian P. Pope et al. *Hierarchical Reinforcement Learning for Air-to-Air Combat*. 2021. arXiv: 2105.00990 [cs.LG]. URL: https://arxiv.org/abs/2105.00990.

[66] Christopher Gebauer, Nils Dengler, and Maren Bennewitz. "Sensor-Based Navigation Using Hierarchical Reinforcement Learning". In: *Intelligent Autonomous Systems 17*. Ed. by Ivan Petrovic, Emanuele Menegatti, and Ivan Marković. Cham: Springer Nature Switzerland, 2023, pp. 546–560. ISBN: 978-3-031-22216-0.

[67] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 1st. Cambridge, MA, USA: MIT Press, 1998.

[68] Peter Auer. "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research* 3 (2002), pp. 397–422.

[69] Rupayan Das, Angshuman Khan, and Gunjan Paul. "A proximal policy optimization with curiosity algorithm for virtual drone navigation". In: *Engineering Research Express* 6.1 (2024), p. 015057. DOI: 10.1088/2631-8695/ad1f14. URL: https://dx.doi.org/10.1088/2631-8695/ad1f14.

[70] Van Manh Tran and Gon-Woo Kim. "Cooperative Deep Reinforcement Learning Policies for Autonomous Navigation in Complex Environments". In: *IEEE Access* 12 (2024), pp. 101053–101065. DOI: 10.1109/ACCESS.2024.3429230.

[71] Haoran Sun et al. *Curiosity-Driven Reinforcement Learning from Human Feedback*. 2025. arXiv: 2501.11463 [cs.CL]. URL: https://arxiv.org/abs/2501.11463.

[72] Unity Technologies. *Unity*. Version 6000.1. Game engine. 2025. URL: https://unity.com/.

[73] Jiong Li and Pratik Gajane. *Curiosity-driven Exploration in Sparse-reward Multi-agent Reinforcement Learning*. 2023. arXiv: 2302.10825 [cs.AI]. URL: https://arxiv.org/abs/2302.10825.

[74] Muhammad Usama and Dong Eui Chang. *Learning-Driven Exploration for Reinforcement Learning*. 2020. arXiv: 1906.06890 [cs.LG]. URL: https://arxiv.org/abs/1906.06890.

[75] Muhammad Usama and Dong Eui Chang. "Robotic Navigation using Entropy-Based Exploration". In: *CoRR* abs/1906.06969 (2019). arXiv: 1906.06969. URL: http://arxiv.org/abs/1906.06969.

[76] Alfréd Rényi. "On Measures of Entropy and Information". In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. University of California Press, 1961, pp. 547–561.

[77] Yeliz Karaca and Majaz Moonis. "Chapter 14 - Shannon entropy-based complexity quantification of nonlinear stochastic process: diagnostic and predictive spatiotemporal uncertainty of multiple sclerosis subgroups". In: *Multi-Chaos, Fractal and Multi-Fractional Artificial Intelligence of Different Complex Systems*. Ed. by Yeliz Karaca et al. Academic Press, 2022, pp. 231–245. ISBN: 978-0-323-90032-4. DOI: https://doi.org/10.1016/B978-0-323-90032-4.00018-3. URL: https://www.sciencedirect.com/science/article/pii/B9780323900324000183.

[78] Dongyoung Kim et al. "Accelerating reinforcement learning with value-conditional state entropy exploration". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[79] Zhengjun Wang et al. "Path Planning for Unmanned Aerial Vehicle via Off-Policy Reinforcement Learning With Enhanced Exploration". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 8.3 (2024), pp. 2625–2639. DOI: 10.1109/TETCI.2024.3369485.

[80] NVIDIA. *Isaac Sim*. Accessed: 2025-05-29. 2025. URL: https://developer.nvidia.com/isaac/sim.

[81] NVIDIA. *Isaac Lab*. Accessed: 2025-05-29. 2023. URL: https://isaac-sim.github.io/IsaacLab/main/index.html.

[82] Diana Wolf Torres. *NVIDIA Expert Explains Why These Star Wars Robots Are Like Nothing You've Ever Seen*. Accessed: 2025-05-29. Apr. 2025. URL: https://droids.substack.com/p/nvidia-expert-explains-why-these.

[83] Andrea Bravo i Forn. *Deep Reinforcement Learning for Autonomous Indoor Exploration with UAVs*. 2025. URL: http://essay.utwente.nl/105419/.

[84] Reinis Cimurs, Il Hong Suh, and Jin Han Lee. "Goal-Driven Autonomous Mapping Through Deep Reinforcement Learning and Planning-Based Navigation". In: *CoRR* abs/2103.07119 (2021). arXiv: 2103.07119. URL: https://arxiv.org/abs/2103.07119.

[85] Yuezhan Tao et al. *Learning to Explore Indoor Environments using Autonomous Micro Aerial Vehicles*. 2023. arXiv: 2309.06986 [cs.RO]. URL: https://arxiv.org/abs/2309.06986.

[86] ArduPilot. *3DR Iris*. Accessed: 2025-08-07. URL: https://www.arducopter.co.uk/iris-quadcopter-uav.html.

[87] Stereolabs. *ZED X One Stereo Camera*. https://www.stereolabs.com/en-nl/products/zed-x-one. Accessed: 2025-07-24.

[88] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV]. URL: https://arxiv.org/abs/2201.03545.

[89] Felipe Codevilla et al. *End-to-end Driving via Conditional Imitation Learning*. 2018. arXiv: 1710.02410 [cs.RO]. URL: https://arxiv.org/abs/1710.02410.

[90] Hao Lin et al. "Taking complementary advantages: Improving exploration via double self-imitation learning in procedurally-generated environments". In: *Expert Systems with Applications* 238 (2024), p. 122145. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.122145. URL: https://www.sciencedirect.com/science/article/pii/S0957417423026477.

[91] Myoung Hoon Lee and Jun Moon. "Deep reinforcement learning-based model-free path planning and collision avoidance for UAVs: A soft actorcritic with hindsight experience

replay approach". In: *ICT Express* 9.3 (2023), pp. 403–408. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2022.06.004. URL: https://www.sciencedirect.com/science/article/pii/S2405959522000935.

[92] Alexandre Brown and Glen Berseth. *SegDAC: Segmentation-Driven Actor-Critic for Visual Reinforcement Learning*. 2025. arXiv: 2508.09325 [cs.CV]. URL: https://arxiv.org/abs/2508.09325.

[93] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. *Causal Confusion in Imitation Learning*. May 2019. DOI: 10.48550/arXiv.1905.11979.

[94] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. *ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst*. Dec. 2018. DOI: 10.48550/arXiv.1812.03079.

[95] Nan Jiang, Yansha Deng, and Arumugam Nallanathan. "Deep Reinforcement Learning for Discrete and Continuous Massive Access Control optimization". In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–7. DOI: 10.1109/ICC40277.2020.9149055.

[96] Benchun Zhou et al. "Vision-based Navigation of UAV with Continuous Action Space Using Deep Reinforcement Learning". In: *2019 Chinese Control And Decision Conference (CCDC)*. 2019, pp. 5030–5035. DOI: 10.1109/CCDC.2019.8832593.