UNIVERSITY OF TWENTE.

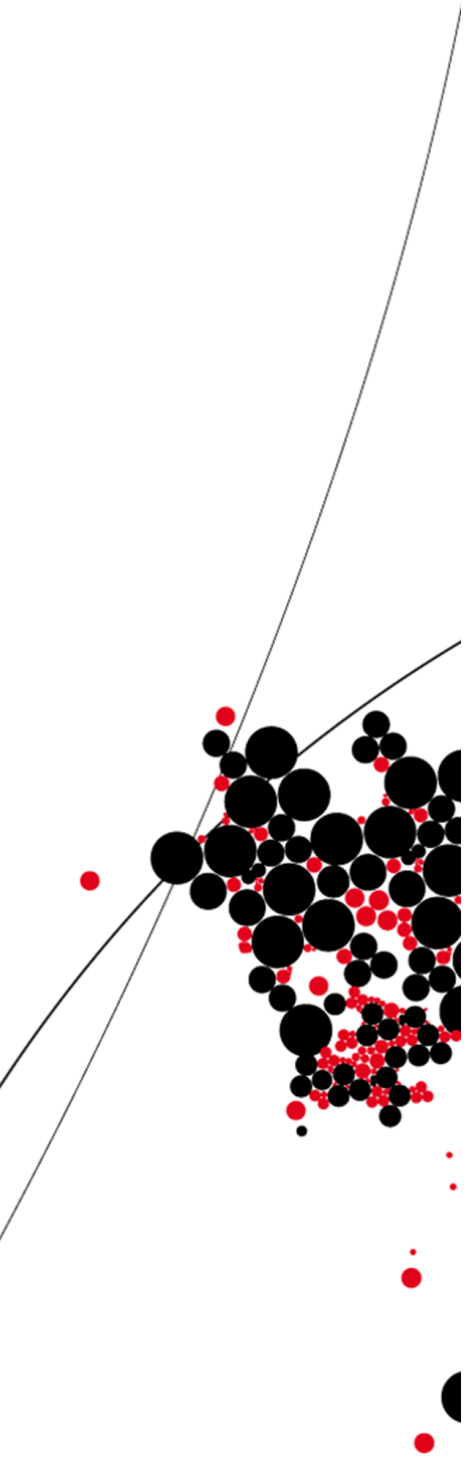
Faculty of Geo-Information Science and Earth Observation

Deep Reinforcement Learning for Autonomous Indoor Exploration with UAVs

Andrea Bravo i Forn

M.Sc. Thesis

March 11, 2025



Supervisors:
Prof. Dr. Ing. Francesco Nex
PhD student Bavantha Udugama

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

Summary

This master's thesis addresses the challenge of autonomous exploration in unknown, office-like indoor environments using an Unmanned Aerial Vehicle (UAV). The primary objective is to train a UAV to explore such environments through Deep Reinforcement Learning (DRL) techniques within a physics-accurate, photorealistic simulation framework.

To achieve this, a modular and adaptable pipeline was designed, integrating three core components: a high-fidelity simulation environment, a high-level DRL policy for decision-making, and a low-level controller to ensure the drone accurately tracks the given commands. Given its versatility, the pipeline can be adapted to other DRL tasks beyond exploration, such as navigation or search tasks, with minimal adjustments. During implementation, various simulation tools, interaction libraries, and DRL algorithms were rigorously evaluated, and the most suitable ones were selected based on requirements.

The pipeline was validated through two distinct tasks. First, a preliminary navigation task served as a proof of concept, where the drone had to navigate toward a goal while avoiding obstacles. Second, the main exploration task was evaluated, focusing on the drone's ability to autonomously map unknown environments. For each task, the Reinforcement Learning (RL) task specifications—including the action space, observation space, and reward function—were clearly defined. Using these specifications, agents were trained, and the resulting policies were evaluated.

Contents

Summary ii

List of acronyms v

1 Introduction 1
- 1.1 Preliminary technical notions . . . 2
- 1.2 General Background . . . 3
- 1.3 Thesis Problem Formulation . . . 6
- 1.4 Report Organization . . . 7

2 State of the art 8
- 2.1 State Of The Art . . . 8
  - 2.1.1 Geometric approaches . . . 8
  - 2.1.2 Dynamic approaches . . . 9
  - 2.1.3 Approaches for tasks complementary to autonomous exploration 11

3 Tools and Methods 13
- 3.1 Exploration Pipeline Overview . . . 13
- 3.2 Choice of Tools . . . 15
  - 3.2.1 Simulator Choice . . . 15
  - 3.2.2 Choice of Framework to Interact with Isaac Sim . . . 16
  - 3.2.3 Choice of DRL Library . . . 18
- 3.3 Environment Setup . . . 18
  - 3.3.1 Environment workflow . . . 18
  - 3.3.2 Scene Setup . . . 19
- 3.4 Low-level Non-linear Controller Module . . . 24
  - 3.4.1 Non-linear Control Law . . . 24
  - 3.4.2 Implementation specifications . . . 25
- 3.5 High-level DRL Controller . . . 26
  - 3.5.1 Selected Model-free DRL algorithm . . . 27
  - 3.5.2 General DRL architecture . . . 27

4 Experiments **29**
4.1 Experimental Setup for Controller Validation 29
4.2 Experimental Setup for DRL Tasks 29
4.2.1 Task 1: Collision-Free Target Navigation 29
4.2.2 Task 2: Autonomous Exploration 31

5 Results **38**
5.1 Performance Results for Controller Validation 38
5.2 Results of DRL Tasks 40
5.2.1 Task 1: Collision-Free Target Navigation 40
5.2.2 Task 2: Autonomous Exploration 41

6 Conclusions and recommendations **45**
6.1 Conclusions 45
6.2 Future Work 46

References **47**

Appendices

A State of the art summary table **54**

B Available Simulation Scenes **57**

List of acronyms

EOS	Earth Observation Science
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
VIO	Visual-Inertial Odometry
DNN	Deep Neural Network
RNN	Recurrent Neural Network
A-SLAM	Active Simultaneous Localization and Mapping
POMDP	Partially Observable Markov Decision Process
MDP	Markov Decision Process
IT	Information Theory
TOED	Theory of Optimal Experimental Design
KLD	Kullback–Leibler Divergence
DWA	Dynamic Window Approach
MI	Mutual Information
DQN	Deep Q Network
NN	Neural Network
CNN	Convolutional Neural Network
MLP	MultiLayer Perceptron
SAC	Soft Actor Critic
PPO	Proximal Policy Optimization

TRPO Trust Region Policy Optimization

DDPG Deep Deterministic Policy Gradient

ML Maximum Likelihood

SB3 Stable Baselines 3

OIGE Omniverse-Isaac-Gym-Envs

CW Clock-wise

CCW Counter Clock-wise

USD Universal Scene Description

FLU Front (x), Left (y), Up (z)

UAV Unmanned Aerial Vehicle

Chapter 1

Introduction

Optimal autonomous exploration of unknown indoor environments remains an open problem in the robotics community, drawing significant attention due to its broad range of applications. These include inspecting deteriorating confined spaces, such as old buildings, mines, or ballast water tanks; conducting search and rescue missions, like navigating burning buildings locating victims, or assisting police in hazardous operations; and exploring areas beyond human reach, such as ventilation systems and pipelines that are too narrow for a human to enter.

Despite considerable efforts to address this challenge, it is far from solved. A major obstacle is the inability to perform global path planning effectively, due to the inherent uncertainty in the structure of the environment.

This thesis was conducted at the Department of Earth Observation Science (EOS) at the University of Twente and is linked to the research of PhD candidate Bavantha Udugama. His previous work focuses on developing a real-time spatial perception system deployed on a drone, which uses a monocular camera and IMU data to create a hierarchical 3D scene graph — environment representation that compacts geometric and high-level semantic information into a graph [1]. The system uses deep learning models to predict depth and semantic information from RGB images. By integrating these models with a Visual-Inertial Odometry (VIO) algorithm, 3D scene graphs can be generated [2]. Currently, the drone is manually controlled during the graph construction process. His next step is to train a DRL policy to enable autonomous 3D scene graph generation.

This work marks the first step in that direction, initiating a new research line within the EOS Department. While complementary to Bavantha's research, it remains a standalone, independent project. A DRL exploration pipeline has been developed from scratch, including the selection of a simulator, interaction libraries, and the es-

tablishment of a general codebase. The resulting pipeline is highly modular, making it valuable not only for Bavantha's research but also for other projects that don't focus on exploration or use DRL. Additionally, significant insights gained during this project will contribute to future work in this research line.

Assuming that this pipeline will eventually be integrated with Bavantha's spatial perception system, we consider the drone to have access to depth and semantic images without addressing how these are generated. Furthermore, we assume the UAV has access to precise pose information from the simulator, and will not use Bavantha's VIO algorithm. Considering pose uncertainty and sensor noise is left as future work to keep the thesis within a manageable scope.

1.1 Preliminary technical notions

RL provides a framework for solving problems where an agent learns to make decisions to achieve a specific goal by actively and sequentially interacting with its environment while optimizing a feedback reward signal.

This type of interaction is formally described by a Markov Decision Process (MDP). At each timestep t, the agent selects an action $a_t \in \mathcal{A}$ from its current state $s_t \in \mathcal{S}$, according to its policy $\pi(s_t|a_t)$. The environment then transitions to a new state $s_{t+1} \in \mathcal{S}$ based on the transition probability function $p(s_{t+1}|s_t, a_t)$, and provides a reward signal $r_t = R(s_t, a_t)$. The agent's goal is to find a policy $\pi(a_t|s_t, \theta)$, where θ represents the policy parameters, that maximizes the expected cumulative reward $E[\sum_{t=0}^{T} \gamma^t r_t]$, with being γ the discount factor and T the time horizon. [3]

In Equation 1.1, the value function $v_\pi(s)$ represents the expected cumulative reward when starting form state s and acting under policy π. Similarly, the Q-function $q_\pi(s, a)$ represents the expected cumulative reward when starting from state s, taking action a and then following policy π. These functions measure how beneficial it is for the agent to be in a given state (or take a specific action) under the given policy. Notice that, for a deterministic policy, $v_\pi(s) = q_\pi(s, \pi(s))$.

$$v_\pi(s) = E[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s] \quad q_\pi(s, a) = E[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a] \tag{1.1}$$

Model free DRL approaches use a Deep Neural Network (DNN) to solve RL problems when $p(s_{t+1}|s_t, a_t)$ and $R(s_t, a_t)$ are unknown. We can classify these approaches into value-based and policy-based methods. In value-based methods, the DNN estimates $v_\pi(s)$ or $q_\pi(s, a)$ for all possible actions in a discrete action

space, given state s_t inferred from the observation. Well-known examples of this approach are Deep Q-Learning methods. In policy-based methods, the DNN directly parametrizes policy $\pi(a, s)$. An example within this category is the Proximal Policy Optimization (PPO) method. Actor-critic methods, where an actor network updates the policy and a critic network evaluates the actions, combine both approaches.

1.2 General Background

The Active Simultaneous Localization and Mapping (A-SLAM) problem aims to optimize the exploration of unknown environments, by proposing navigation strategies that generate future goals or actions for a robot to minimize uncertainty in its map and pose estimates. Acting as an "envelope" of already existing SLAM systems, A-SLAM enables fully autonomous exploration. A key challenge is balancing environmental exploration (maximizing coverage) and exploitation (revisiting areas to improve map and pose accuracy). This thesis focuses on the exploration aspect of the problem, as assuming exact pose knowledge prevents drift, makes loop closure straightforward, and ultimately removes the need for exploitation.

A-SLAM can be seen as a sequential decision-making process under actuation and observation uncertainties. Thus, it can be modeled as a Partially Observable Markov Decision Process (POMDP), formally described by the tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{Z}, \xi_s, \xi_z, R, \gamma\}$. Here, $\mathcal{S}$ is the state space -set of all possible true states s of the environment (e.g., exact robot's pose and true geometry of the environment, with landmarks and dynamic objects)-, $\mathcal{A}$ is the action space that contains any action a that the agent can take (e.g., discrete velocity commands or goal positions) and $\mathcal{Z}$ is the observation space - set of all possible observations z the agent can make (e.g., raw sensor measurements). Moreover, $\xi_s = p(s_{t+1}|s_t, a_t)$ is the transition model (accounting for actuation uncertainty), $\xi_z = p(z_{t+1}|s_{t+1}, a_t)$ is the observation model (accounting for measurement uncertainty), $R(s_t, a_t)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor (balancing the importance of immediate and future rewards).

In POMDPs the agent cannot access the true state s_t. Instead, it maintains an internal belief of it given by $b_t(s_t) = p(s_t|z_{1:t}, a_{1:t-1})$, a posterior probability function over the states at time t. Additionally, we define the control policy $\pi_t^\theta(b_t) = p(a_t|b_t)$ as a mapping from the agent's belief to actions (parametrized by θ) that determines the agent's behaviour. Since the belief is a sufficient statistic, optimal policies for the original POMDP may be found by solving the optimization problem in equation 1.2, where $\rho(b_t, \pi(b_t)) = \int_{\mathcal{S}} b_t(s_t) R(s_t, a_t) ds_t$ is the expected reward for belief state b_t [4].

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \rho(b_t, \pi(b_t))] \tag{1.2}$$

This optimization is computationally very expensive. To manage its complexity, classical approaches typically divide the implementation of an A-SLAM method into three sub-modules. First, collision-free waypoints are identified within the available map estimate. Next, a utility function based on Information Theory (IT) or Theory of Optimal Experimental Design (TOED) evaluates the potential information gain of visiting each waypoint candidate. With this, a goal position is chosen. Finally, a path-planning method guides the robot toward the selected goal generating a trajectory [5]. In contrast, dynamic approaches —primarily DRL-based— aim to directly predict the control inputs the agent must follow to generate collision-free trajectories that maximize coverage, while minimizing state uncertainty.

Traditional methods have been extensively studied in the literature and, while they perform well, DRL-based methods offer several advantages:

- **Reduced computational complexity during inference**. Unlike classical approaches that require intensive computations at runtime, DRL-based methods transfer the computational burden to the DNN training phase, requiring only feed-forward propagation during evaluation [6].
- **End-to-end optimality**: Classical approaches decouple optimal goal selection from path planning, leading to suboptimal solutions of the exploration problem. DRL optimizes these tasks jointly, enabling more globally efficient solutions by directly maximizing exploration rewards.
- **Flexibility in input modalities**. Classical approaches rely on specialized sensors to observe geometry, whereas learning-based approaches can directly process diverse input types (eg. can infer geometry from raw RGB images) [7].
- **Better generalization to unseen environments**. DRL-based methods can effectively leverage structural regularities of the real world, leading to more efficient behaviour in previously unseen environments [7].
- **Improved robustness to uncertainty**. These methods can handle errors in state estimation [7].

Given these advantages, we adopt a DRL-based approach to address the challenge of autonomous exploration in 3D environments. However, it is important to acknowledge the limitations of this approach. A critical concern is the design of the action and observation spaces, as high sample complexity can lead to extremely

large search spaces during training and non-convergence. To overcome this, previous work (such as [8]) uses imitation learning to guide the exploration. Yet, it still underperforms compared to classical approaches [7]. Notice that imitation learning might not be the best solution, as it is labor-intensive and susceptible to bias [9].

After reviewing the state-of-the-art (Section 2.1), several domain gaps have been identified in DRL-based autonomous exploration approaches:

1. There is a notable lack of research on DRL methods for optimal indoor exploration using visual input (e.g., depth, semantic, or RGB images) that generalize well to unseen environments without imitation learning. End-to-end trained methods are entirely absent.

2. To the best of my knowledge, no existing works employing DRL for indoor exploration are specifically tailored for UAVs. A key challenge in this context is the limited payload capacity of UAVs, which limits computational power, making it impractical to process highly detailed maps. The reviewed studies that have been tested in realistic environments typically use grid cells of 5 cm × 5 cm. Furthermore, enabling 3D exploration —where the UAV can move vertically— adds another layer of complexity to the problem compared to ground robots, which are constrained to 2D motion.

3. None of the reviewed works explicitly incorporate semantic information as part of the observations or utilize it in reward shaping. Some methods attempt to learn semantic cues from RGB images using a Convolutional Neural Network (CNN) trained for object classification ([8], [10]).

4. Many approaches have been trained and tested in highly simplified environments. Some DRL methods leveraging visual data are trained in more advanced simulators (e.g., Habitat-Sim [11]), but these simulators are relatively outdated and lack advanced physics models. Instead, a photorealistic simulator with advanced physics should be used to accurately capture the robot's dynamics and minimize the sim-to-real gap, enhancing the policy's transferability to real-world deployment.

5. Episode termination conditions often depend on prior knowledge of the environment's map (e.g., covering a certain percentage of the area). Developing map-agnostic termination criteria remains largely unexplored.

1.3 Thesis Problem Formulation

The main goal of this thesis is formulated as follows:

Train a UAV using DRL to autonomously and efficiently explore unknown indoor office-like environments relying on visual input. The UAV's movement will be constrained to a 2D plane, maintaining a fixed flight altitude. Training and evaluation will be conducted in a physics-accurate, photorealistic simulator.

In addition, subject to time constrains, the following extended goals were defined:

- *Evaluate the impact of semantic information on exploration performance.*
- *Design termination conditions independent of prior environment knowledge.*

To tackle these goals, the following research questions were posed:

1. Which simulator and libraries should be used to set up the exploration pipeline?
2. How should the architecture of the exploration pipeline be? What modules should it include?
3. Which DRL policy, DNN architecture, action and observation spaces should be used?
4. How can we exploit lightweight low-resolution environment representations in a safe way to navigate without collisions?
5. What cues from the environment, observations or map estimate can be used to determine when an episode should end?

A key challenge in this thesis was designing and implementing the exploration pipeline to train such a model from scratch. Crucial design choices —such as the pipeline architecture and the software tools used- had to be made from the very beginning. The most promising software libraries were chosen and, while later on they became the standard for DRL frameworks, at the time this project began, they had minimal documentation and were under active development with frequent updates. To maximize utility beyond this work, the pipeline was designed with modularity as a core principle, enabling the reuse of individual components for other applications. The final pipeline is intended for release as open-source software to support reproducibility and benefit the research community.

1.4 Report Organization

The remainder of this report is organized as follows. Chapter 2 provides a review of the state of the art in autonomous exploration. Chapter 3 gives an overview of the designed exploration pipeline, discusses the selected software tools and libraries, and details the pipeline modules. Chapter 4 presents the experiments conducted to validate the exploration pipeline, including tests for the low-level controller, preliminary DRL tasks designed to verify system readiness before tackling the full exploration task, and the specifications of the exploration task itself (the main goal). Chapter 5 presents the results of these experiments. Finally, Chapter 6 summarizes the thesis's conclusions and insights and discusses potential directions for future work.

Chapter 2

State of the art

2.1 State Of The Art

Many approaches have been proposed to tackle the A-SLAM problem. A recent survey [5] categorizes these methods into geometric and dynamic approaches.

2.1.1 Geometric approaches

Geometric approaches seek optimal robot trajectories that minimize uncertainty in both the robot's map and localization. As mentioned in Section 1.2, these methods typically follow three steps. First, potential goals or paths are identified within the estimated map. Next, candidates are ranked using a utility function that favors goals/paths maximizing the uncertainty reduction in the robot's trajectory and map. Ideally, the utility function would quantify how much each candidate sharpens the joint probability distribution of the map and robot's pose. However, this is computationally expensive. Instead, most methods propose a cost function that leverages IT and TOED notions to separately assess the uncertainty reduction in the robot's pose and map estimates associated to each candidate. Finally, a path planner is used to navigate towards the selected goal or follow the selected path.

To select candidate goals, [12] introduces the concept of frontiers, borders between known and unknown map regions, where the center of each frontier serves as a goal candidate. In this work, the nearest frontier is chosen as the next exploration target. Other works also using frontier-based exploration are [13], [14], [15] and [16].

Information-driven utility functions rank candidates based on information gain, for instance checking which candidate minimizes the robot's pose and map entropy. Both [13] and [14] use Rao-Blackwellized Particle filter SLAM and IT-based util-

ity functions. The former compares three cost functions (joint trajectory and map entropy, expected map mean and Kullback–Leibler Divergence (KLD)-based information gain) and finds them computationally prohibitive. The latter uses A* [17] to compute trajectories to candidate frontiers and selects the one with the highest map-segment covariance, with Dynamic Window Approach (DWA) [18] as a local planner.

Task-driven utility functions use TOED optimality criteria to map the pose-graph covariance matrix (from graph-based SLAM, quantifying the uncertainty in the estimated robot's trajectory) into a scalar value for ranking candidates. [15] computes trajectories to all candidate frontiers with Djikstra's algorithm [19] (global planner). For each trajectory, a hallucinated pose-graph is created from the last pose-graph of the SLAM algorithm, and D-optimality is used to rank candidate trajectories. [16] extends this by augmenting the utility function with a path entropy term, combining IT and TOED notions.

Other approaches not relying on frontier-based exploration include the method in [20], which uses Topological Feature Graphs as environment representations, samples reachable goals from the graph and selects the one maximizing landmark entropy reduction. Work in [21] grows an RRT* tree [22] within the known region of a map estimate, and selects the candidate path that minimizes joint path and map entropy change per distance traveled. This work highlights the link between map entropy and coverage, as well as path entropy and accuracy.

2.1.2 Dynamic approaches

Dynamic approaches frame A-SLAM as a sequential decision-making problem, where the robot selects a control action at each step to minimize map and pose estimation uncertainty. These methods typically employ DRL, where a learned policy maps sensor and state inputs to actions. Based on the reward function, these approaches are sorted as uncertainty-based, area-coverage-driven or a hybrid of both.

Uncertainty-based DRL methods use a reward function that encourages uncertainty minimization in state estimation, addressing both exploration and exploitation. Many approaches shape rewards by mimicking the utility functions of geometric methods. For instance, [6] and [23] use TOED-based reward functions, employing the D-optimality and A-optimality criteria, respectively. Meanwhile, [24], [25], and [26] adopt IT-based rewards: the first two leverage Mutual Information (MI) gain to determine the next best view, while the latter uses entropy reduction. Other meth-

ods quantify uncertainty more loosely and are often referred to as curiosity-driven approaches. For instance, [27] rewards actions that move the robot beyond a certain distance from previously visited waypoints, while [28] encourages visiting states that are difficult to predict in a learned feature space. Slightly outside the scope of DRL, [29] employs supervised deep learning with an IT-based loss function.

Area-coverage-driven DRL methods use a reward function based on the increase in explored area at each step, overlooking exploitation. All reviewed approaches in this category rely on imitation learning for generalization. Work in [8] uses a classical mapping module to generate an occupancy grid, which, along with RGB images, is fed into a DRL policy to predict velocity commands. Similarly, [7] proposes a hierarchical approach combining learning-based and classical modules. A Neural SLAM module [30], trained jointly with a DRL-based global policy, processes RGB and odometry data to generate an estimate of the map and agent's pose. The global policy receives these estimates to set long-term goals. A geometric planner then computes a trajectory and samples short-term goals, which a local imitation learning policy uses with RGB images to generate velocity commands. Work in [10] enhances [7] by incorporating spatial and channel attention mechanisms.

Hybrid DRL methods combine uncertainty-based and area-coverage reward terms to balance exploration and exploitation. For example, [31] uses a coverage reward term equal to [8] alongside a curiosity-driven term similar to [27]. Likewise, [32] adopts a modular hierarchical approach akin to [7], with a reward function comprising three components: a coverage term, a term encouraging loop closures for map consistency, and a term based on ORB feature points to ensure accuracy.

An extensive summary table of the reviewed papers on autonomous exploration is provided in A.

Notably, all DRL policies rely on environment representations (either learned or provided as input) to avoid overfitting to training environments. Moreover, none of the reviewed works explicitly incorporates semantic information in observations or reward functions. The closest approaches, [8], [7] and [10], attempt to implicitly learn semantic cues from RGB images via encoders. In addition, exploration methods that rely solely on visual inputs are rare, as many approaches use range measurements.

2.1.3 Approaches for tasks complementary to autonomous exploration

DRL methods for tasks directly related to autonomous exploration have also been reviewed, as they can provide valuable insights and inspiration.

Some works focus on collision avoidance in unknown environments without explicitly encouraging exploration. All reviewed approaches for this task use reward functions with a term heavily penalizing collisions, along with extra terms encouraging behaviors like staying alive, moving fast, or avoiding rotations. Q-learning-based methods include [33] and [34], both using a Deep Q Network (DQN) [35] to process depth images and generate 2D velocity commands. The former rewards survival, while the latter encourages speed and minimal turning. UAV-specific methods include [36] and [37]. The first uses a Soft Actor Critic (SAC) policy [38], with a Neural Network (NN) architecture featuring an encoder extracted from an autoencoder trained to predict depth images from RGB inputs. This setup allows the policy to process raw RGB images, and generate continuous linear and angular velocity commands for high-level UAV control. Meanwhile, [37] uses a DQN that processes stacked depth images (for temporal awareness) and is trained with a reward function that incentivizes maintaining distance from obstacles. Finally, [3] compares Deep Q-Learning and PPO [39] methods across different observation and action spaces. Notably, none of the reviewed approaches for collision avoidance leverage a map of the environment.

A large body of research focuses on target navigation, guiding robots to specific coordinates while avoiding obstacles. All reviewed works for this task use reward functions that heavily reward reaching the target and strongly penalize collisions. Since the reward for reaching the target is sparse, many works introduce intrinsic reward terms to provide more continuous feedback, improving training stability. A common approach is to give continuous rewards based on the robot's distance and heading difference from the target. This method is used in [40] and [41], both employing a Deep Deterministic Policy Gradient (DDPG) [42] DRL algorithm. The first work uses an observation space consisting of laser readings, the previous action (2D velocity command), and the target's position in the robot's frame, while the second is tailored for UAVs. When navigating towards faraway targets in unknown environments, an exploration component is also needed in the reward function. For this reason, works in [43] and [9] include, on top of the previously mentioned reward terms, a continuous curiosity term (similar to the one in [28]) that encourages visiting states difficult to predict in a learned feature space. Moreover, work in [44]

optimizes the navigation policy with auxiliary tasks (e.g., depth prediction from RGB images), although the policy does not generalize well, as shown in [45]. Finally, some methods leverage semantic information for improved navigation. For example, [46] incorporates object-based reward terms to maintain safe distances from humans and other robots, while [47] proposes a semantic-aware DRL algorithm for UAVs that minimizes SLAM drift by avoiding low-information regions, such as flat surfaces with limited semantic variety.

Other DRL methods address the active localization problem, estimating an agent's pose in an environment by guiding it toward locations that provide more informative observations. The works reviewed for this task use IT-based reward functions that account for the Maximum Likelihood (ML) of the robot being in a state, measuring belief accuracy. Examples include [48], using laser scanners and a map as policy observations, and [49], using RGB images and a map as observations.

Finally, some methods focus on exploring unknown environments to locate object instances. Notably, all reviewed works leverage semantics. Examples include [50], where a DRL policy maps 3D scene graph [1] observations to motion primitives; [51], which uses joint visual-semantic embeddings as policy inputs; and [52], which infers a complete semantic map (including unobserved regions) from partial RGB-D observations and employs a DQN policy to select actions based on this map and target object class.

Chapter 3

Tools and Methods

3.1 Exploration Pipeline Overview

The developed exploration pipeline had to meet several requirements: *i)* the used simulator should be photorealistic, physics-accurate, and should support easy access to depth and semantic images from modeled cameras within the scene, *ii)* trained DRL policies should consider the dynamics of the drone, *iii)* the pipeline should enable scalability for parallel multi-agent training, and *iv)* the pipeline architecture should ensure modularity to facilitate reusability.

Figure 3.1 shows an overview of the proposed pipeline. In continuous-line boxes we find it's four main modules: the simulation environment (in blue), a classical low-level controller (in green), modules to process simulator data (in purpule), and a high-level DRL controller (in orange). Sensor data and environmental information are collected and processed to generate crafted observations, termination signals, and rewards for the DRL controller. The high-level policy outputs velocity commands, which serve as a reference for the low-level controller. The latter then computes the appropriate torque to be applied to the UAV's body in the simulator.

In this pipeline, requirement *ii)* is met thanks to the low-level controller that accounts for the drone's dynamics. If this requirement wasn't in place, a simpler approach (similar to works using ground robots reviewed in Section 2.1) could have been taken, where the drone is spawned in the simulator at a target position given by the high-level controller, without physics considerations. By using a low-level controller, we ensure that the high-level policy learns to generate actions that construct realistic, trackable trajectories. Another option considered was having the DRL policy directly output drone torques. However, this was discarded due to the added training complexity of learning both the drone's dynamics (how to hover and move) and the exploration task simultaneously.

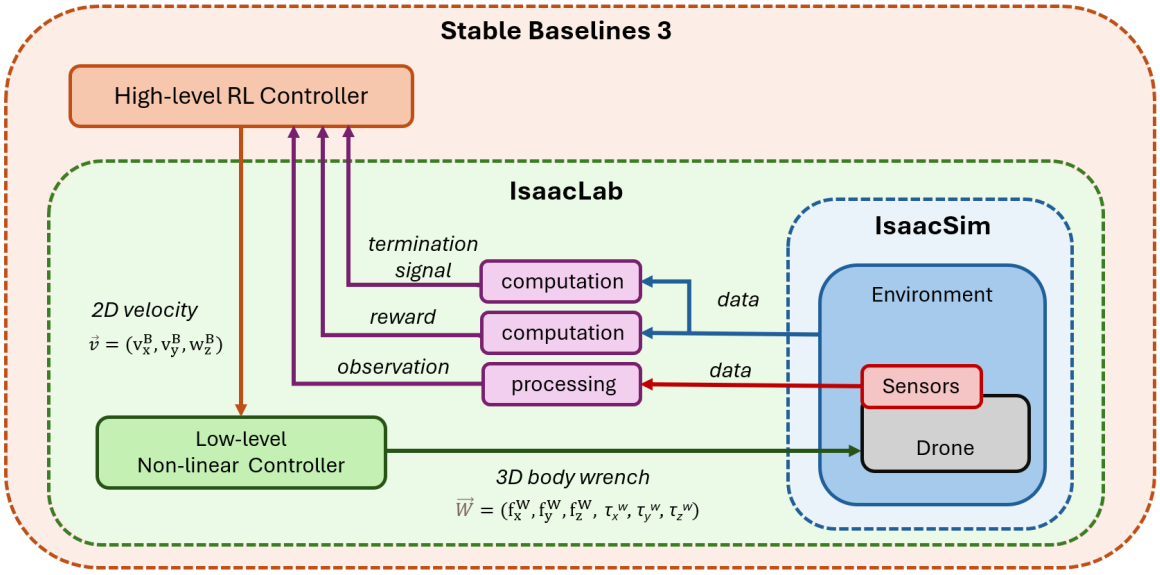


Figure 3.1: General pipeline scheme. Continuous-line boxes indicate modules, while dashed-line boxes represent frameworks. Modules: simulated environment (blue), low-level controller (green), data processing (purple), and high-level controller (orange). Frameworks: Isaac Sim simulator (blue), IsaacLab (green), and SB3 DRL library (orange).

The pipeline's modular design also satisfies requirement *iv)*. As shown in Figure 3.1, the framework's core (inside the green dashed box) consists of the low-level controller, the simulator, and sensor processing modules providing access to collected data (e.g., RGB/depth images or IMU data). This core component, on its own, can be used for other applications beyond DRL, for example, to test SLAM algorithms by having the drone follow a predefined trajectory or driving it manually via keyboard. The created codebase also supports interchangeable low-level controllers for UAVs, enabling their testing in simulation. The DRL modules act as a wrapper around the core component, making the pipeline adaptable to various DRL tasks beyond exploration, such as goal navigation, collision avoidance, object search, and active localization. Overall, the framework is highly versatile and extendable to other research applications.

The pipeline development process consisted of three stages. First, a suitable simulator was selected (see Section 3.2.1). Next, libraries to interact with the simulator were chosen and the pipeline core was set up (see Section 3.2.2). Finally, a DRL library was integrated (see Section 3.2.3). After evaluating the available options, we chose Isaac Sim 4.2.0 [53] as the simulator, Isaac Lab 1.2.0 [54] as the interaction library, and Stable Baselines 3 (SB3) 2.3.2 [55] as the DRL library.

3.2 Choice of Tools

3.2.1 Simulator Choice

The first step in the development process was selecting a simulator that met requirement *i)*, ensuring physics accuracy, photorealistic rendering, and precise sensor data, including seamless depth and semantic images from scene cameras. Table 3.1 compares the considered options.

Gazebo [56] is a widely used simulator designed for robotics research. It utilizes the DART [57] rigid-body physics engine for accurate dynamics modeling. While flexible and well-integrated with ROS, it lacks photorealism, though it benefits from strong community support. Habitat [11], developed by Facebook AI, is tailored for Embodied AI research. It is mainly used to train AI agents for navigation tasks in 3D photorealistic environments, relying on 3D scans to generate realistic scenes. Its physics engine, Bullet [58], while adequate for navigation applications, is not optimized for precise robot dynamics. Simulators built on top of game engines, such as AirSim [59] and Flightmare [60] (based on Unreal Engine and Unity), offer photorealistic rendering and custom physics engines that enhance standard game physics for robotics applications. However, they are difficult to use and extend, with high API complexity, and lack strong community support compared to Gazebo or Isaac Sim.

Isaac Sim [53], part of NVIDIA's Omniverse, is a modern simulator tailored for robotics applications and multi-modal data generation. It features a high-fidelity RTX rendering engine and the PhysX GPU-accelerated physics engine, providing precise simulations. While it has demanding system requirements, it supports Universal Scene Description (USD) files, integrates with ROS, and offers extensions for reinforcement learning and domain randomization. Isaac Sim was chosen as it best meets the project's requirements. It is a rapidly growing platform, gaining popularity in both academia and industry, and shows strong potential for advanced robotics simulation.

Simulator	Physics Accuracy	Photorealism	Depth & Semantics	API Complexity
Gazebo [56]	***	×	✓	**
Habitat [11]	*	✓	✓	**
Unreal Engine/Unity	**	✓	✓	****
Isaac Sim [53]	****	✓	✓	**

Table 3.1: Comparison of simulators based on requirement *i)*. Some entries follow a progressive qualitative rating scale, where more * symbols indicate a higher degree of fulfillment.

3.2.2 Choice of Framework to Interact with Isaac Sim

Once Isaac Sim was selected as simulator, the next step was to choose a framework to interface with it and implement the core of the exploration pipeline. As discussed in Section 3.1, this core is responsible for spawning an office-like environment in the simulator scene, along with a UAV, and enables the drone to navigate and gather scene data using a low-level controller that receives velocity commands (from the keyboard or DRL policy).

The chosen framework needed to be compatible with a DRL library (i.e., have an *environment* class that interfaces with Isaac Sim, and includes a *step* method accepting actions from the DRL algorithm and returning observations and rewards). A key implementation challenge stemmed from the non-standard way in which the DRL policy and the drone interacted in the proposed pipeline: instead of directly controlling rotor speeds or body torque, the high-level DRL controller provides set-points for the low-level controller to track. Since most Isaac Sim RL frameworks are designed to handle direct joint torques or velocities, they lack native support for this hierarchical control structure. Thus, we had to integrate the low-level controller into the default workflow of the chosen framework, bridging the gap between DRL outputs and environment inputs.

Several frameworks were considered, but none stood out as an ideal solution, either because they did not fully meet our requirements or were not well-established at the time this thesis started. See Table 3.2 for a comparison. One option was Pegasus Simulator [61], a framework built on top of Isaac Sim to simulate multi-rotor dynamics. It integrates with the PX4-Autopilot controller [62] and supports custom Python controllers, making it a strong candidate to handle the low-level control portion of the pipeline. However, its codebase offers limited flexibility for adding sensors or processing sensor data and, most importantly, lacks integration with DRL libraries.

At the time this thesis began, two frameworks — Omniverse-Isaac-Gym-Envs (OIGE) [63] and Orbit [64] — were competing to become the standard for DRL in Isaac Sim. Neither was designed for UAVs, as both focused on manipulators and legged robots. Additionally, as mentioned earlier, neither natively supported hierarchical control structures.

On one hand, OIGE [63], developed by NVIDIA, was a well-established RL framework for robotics applications, optimized for fast, parallel training. At the time, it was the go-to choice for DRL in Isaac Sim, integrating Isaac Gym [65] functionalities and providing a set of example implementations. While effective for legged

robots due to its precise mechanical simulations, it was less optimized for photorealistic rendering and image-based tasks. Its lack of modularity also made integrating a low-level controller challenging. On the other hand, Orbit [64] was a modular and flexible framework for robot learning built on top of Isaac Sim. Unlike OIGE, Orbit was specifically designed to simplify common workflows in robotics research. Its modular structure facilitated interaction with Isaac Sim for RL tasks and made it a better candidate for integrating a custom low-level controller. In addition, it provided better rendering capabilities than OIGE, making it more suitable for applications requiring high-fidelity visuals.

A fourth option, Omnidrones [66], was specifically designed for RL tasks with drones in Isaac Sim. The default action space allowed DRL policies to directly command the target thrust of each rotor. While its documentation mentioned the option of integrating a high-level controller, this feature was poorly supported. Additionally, Omnidrones was less mature and modular than Orbit, with limited documentation and a smaller community, making it a riskier choice.

Ultimately, Orbit was chosen for its modularity, flexibility, and high-fidelity physics and rendering. This decision was validated later on when NVIDIA officially merged Orbit into Omniverse and rebranded it as Isaac Lab [54], solidifying its role as the standard framework for RL with Isaac Sim. Throughout this entire thesis, the framework remained in constant evolution, with major updates occurring at least once a month. Keeping up with these changes was essential, as new features were regularly introduced, further enhancing Isaac Lab's capabilities and reinforcing its position as the best choice for this work.

Framework	DRL compatibility	Modularity	Low-Level Controller Integration	Native UAV Support
Pegasus [61]	×	*	-	✓
OIGE [63]	✓	**	**	×
Orbit/ Isaac Lab [64]	✓	****	****	×
Omnidrones [66]	✓	**	***	✓

Table 3.2: Comparison of frameworks to interact with Isaac Sim. Some entries follow a progressive qualitative rating scale, where more * symbols indicate a higher degree of fulfillment. The symbol - means non-applicability.

3.2.3 Choice of DRL Library

After selecting Isaac Sim and Isaac Lab as the simulator and interaction framework, the next step was to choose a suitable DRL library. Isaac Lab provides wrappers for four RL libraries: RL-Games [67], RSL-RL [68], SKRL [69], and SB3 [55]. For a comparison of these libraries, visit Isaac Lab's webpage [70].

RL-Games and RSL-RL offer a limited range of DRL algorithms, while SKRL and SB3 provide a much broader selection. With this, the options were narrowed down to the latter two libraries, as they offer greater flexibility. Furthermore, RL-Games, RSL-RL, and SKRL have minimal documentation and small support communities, although Isaac Lab maintainers have progressively released more examples using these frameworks. In contrast, SB3 benefits from a large, active community, comprehensive documentation, and a model zoo filled with examples. This extensive support ultimately made SB3 the preferred choice.

SB3 supports vectorized environments during training, allowing multiple independent environments to be stacked into one. This enables training on n environments simultaneously, aligning with the scalability requirement (*iii)*). To use this feature, the environment passed to SB3 must inherit from its inner *VecEnv* class. Without using a wrapper provided by Isaac Lab, SB3 defaults to its *DummyVecEnv* class, which wraps the environment sequentially, disabling parallel training.

3.3 Environment Setup

3.3.1 Environment workflow

Isaac Lab uses a manager-based system to set up and run simulation environments. With this, the environment is broken down into individual components (or managers) that handle different simulation aspects, such as spawning objects in the scene, computing observations, and applying actions. The user defines a configuration class for each manager, and an *environment* class coordinates them, calling the functions specified in the configurations. Figure 3.3.1 (from Isaac Lab's website) illustrates how these managers interact. The *environment* class has a *step* method that receives actions, advances the simulation by one time step, and returns environment information such as observations, rewards, and done signals.

A basic environment consists of a Scene Manager (spawning objects, robots,

and sensors in the virtual world), an Observation Manager (extracting observations from the simulation state and sensor data), and an Action Manager (converting raw actions into low-level commands). The Non-linear controller for the UAV was integrated into the Action Manager. RL environments introduce additional managers: an Event Manager (triggering operations based on simulation events, such as randomizing object properties at the beginning of an episode), Reward and Termination Managers (computing reward and done signals), a Command Manager (switching between command strategies), and a Curriculum Manager (gradually increasing task difficulty to stabilize learning).

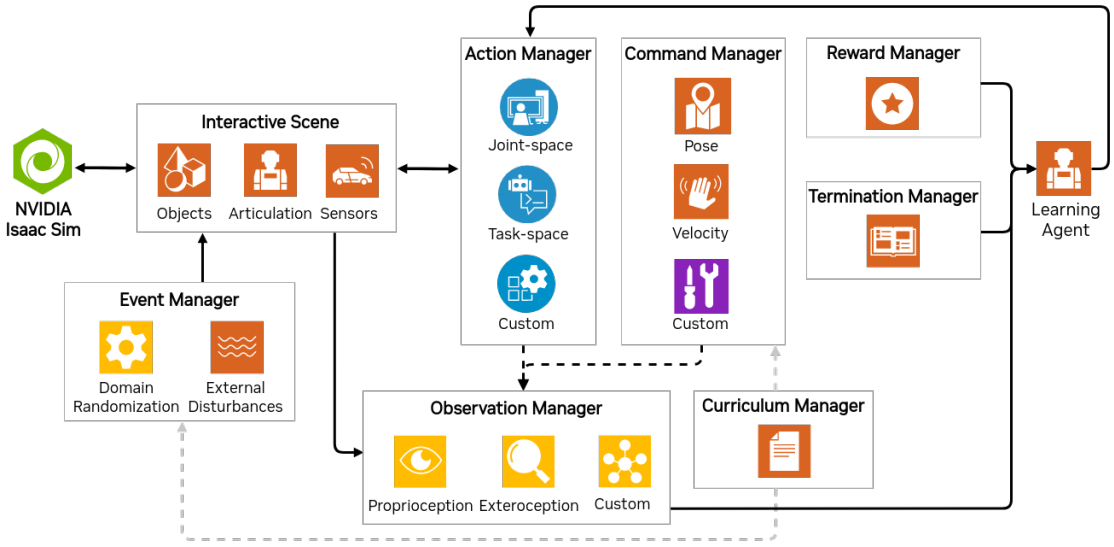


Figure 3.2: Diagram showing the interaction between managers in Isaac Lab (extracted from Isaac Lab's website [54]).

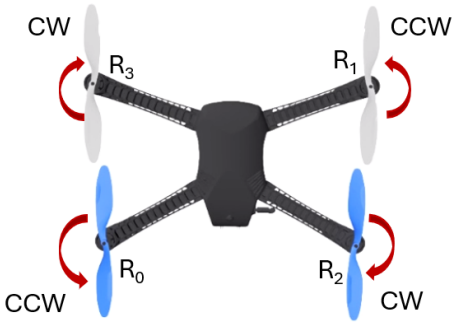
3.3.2 Scene Setup

In this Section, the components handled by the Scene Manager when setting up the simulation environment are described in detail.

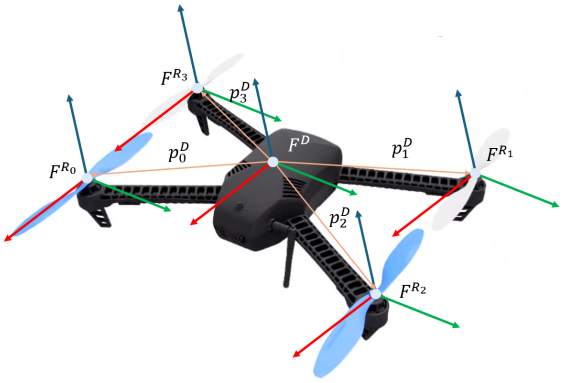
Drone Model

For this project, the Iris UAV model shown in Figure 3.3 has been used. This quadrotor features two descending-chord propellers (R_0, R_1) and two ascending-chord propellers (R_2, R_3), as shown in Figure 3.3a. The drone was described to the Isaac Sim simulator using a USD file. More detailed drone specifications can be found in [71].

The following frames have been used throughout the project: the world frame $F^W = \{O^w, \vec{x}_w, \vec{y}_w, \vec{z}_w\}$, with $\vec{z_w}$ pointing upwards, the drone body frame $F^D = \{O^D, \vec{x}_D, \vec{y}_D, \vec{z}_D\}$, and the propeller frames $F^{R_i} = \{O^i, \vec{x}_i, \vec{y}_i, \vec{z}_i\}$ for $i = 0, 1, 2, 3$. Notice that the drone-attached frames, displayed in Image 3.3b, follow the Front (x), Left (y), Up (z) (FLU) convention.



(a) Top view of the Iris drone with rotors' specifications. Counter Clock-wise (CCW) and Clock-wise (CW) correspond to descending and ascending chord propellers respectievly.



(b) Side view of the Iris drone with drone-attatched frames (body frame F^D and propeller frames F^{R_i} for $i = 0, 1, 2, 3$). The red, green and blue colors correspond to the x, y and z axis.

Figure 3.3: Iris quadrotor

- **Dynamics model of an underactuated quadrotor**

As shown in [72], the wrench $\vec{W}^D(\vec{u}) \in \mathbb{R}^6$ generated on the drone's body by the movement of the N propellers (expressed in frame F^D) is given by Equation 3.1.

$$\vec{W}^D(\vec{u}) = \begin{pmatrix} \vec{f}^D(\vec{u}) \\ \vec{m}^D_{drag}(\vec{u}) + \vec{m}^D_{thrust}(\vec{u}) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^N c_{fi}\vec{z}_i^D(\vec{u_v})u_{\lambda i} \\ \sum_{i=1}^N (-k_i c_{\tau i}\vec{z}_i^D(\vec{u_v}) + p_i^D \times c_{fi}\vec{z}_i^D(\vec{u_v}))u_{\lambda i} \end{pmatrix} = A(\vec{u_v})\vec{u_\lambda} \tag{3.1}$$

In this equation, $\vec{f}^D(\vec{u}) \in \mathbb{R}^3$ and $\vec{m}^D(\vec{u}) = \vec{m}^D_{drag}(\vec{u}) + \vec{m}^D_{thrust} \in \mathbb{R}^3$ are the total thrust force and total moment on the drone's body, both expressed in frame F^D. Notice that the total moment includes contributions from both thrust forces and drag moments acting on the N propellers.

Matrix $A(\vec{u_v}) \in \mathbb{R}^{6\times N}$ relates the wrench $\vec{W}^D(\vec{u})$ and the input vector $\vec{u} = [\vec{u_v}, \vec{u_\lambda}]^T \in \mathbb{R}^{K+N}$. This vector consists of: $\vec{u_v} = [u_{vj}]_{j=0}^{K-1} \in \mathbb{R}^K$, denoting the positions of the servomotors in the drone's joints, and $\vec{u_\lambda} = [u_{\lambda i}]_{i=0}^{N-1} \in \mathbb{R}^N$, inputs related to the rotor velocities. Specifically, $u_{\lambda i} = w_i|w_i|$ being $w_i \in \mathbb{R} > 0$ the angular velocity of the i^{th} rotor. The parameters c_{fi} and $c_{\tau i}$ are the thrust and drag coefficients for the i^{th}

propeller, assumed to be identical for all propellers. Moreover, the free vector $\vec{z_i^D}(\vec{u_v})$ represents the z-axis of each propeller frame F^{R_i} (expressed in frame F^D), while $p_i^D = (p_{xi}, p_{yi}, p_{zi})$ is the position of the i^{th} propeller origin O^i relative to the origin O^D of F^D (see Figure 3.3b).

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 3.2, 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 $\forall i\ \vec{z_i^D} = \vec{z_D^D} = [0, 0, 1]^T$.

$$\vec{W}^D(\vec{u}) = A\vec{u} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_f & c_f & c_f & c_f \\ c_f p_{y0} & c_f p_{y1} & c_f p_{y2} & c_f p_{y3} \\ -c_f p_{x0} & -c_f p_{x1} & -c_f p_{x2} & -c_f p_{x3} \\ -c_\tau & -c_\tau & c_\tau & -c_\tau \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0_{2\times 4} \\ \bar{A} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ f \\ m_x \\ m_y \\ m_z \end{pmatrix} \quad (3.2)$$

As shown in Equation 3.2, $rank(A) = rank(\bar{A}) = 4$. Thus, $\bar{A} \in \mathbb{R}^{4\times 4}$ is a bijective linear map that uniquely relates $\vec{u} = (u_1, u_2, u_3, u_4)^T$ and $\hat{\vec{u}} = (f, m_x, m_y, m_z)^T$ via $\hat{\vec{u}} = \bar{A}\vec{u}$. Since the first two rows of A are zero, no input $\vec{u}$ can generate a force along the x- or y-axis of frame F^D. To move the drone in these directions, the UAV must first reorient to align the thrust force $\vec{f}^D(\vec{u}) = (0, 0, f)^T$ with the desired direction, by generating the necessary torque $\vec{m}^D(\vec{u}) = (m_x, m_y, m_z)^T$. This principle underlies the controller described in Section 3.4.

Using the 3^{rd} Newton's Law and Euler's rotation equations, the dynamics of a generic UAV can be described as in Equation 3.3. Here, $\vec{p_D^W}$, $\dot{\vec{p}}_D^W$, $\ddot{\vec{p}}_D^W \in \mathbb{R}^3$ represent the drone's position, velocity, and acceleration relative to frame F^w, while $R_D^W \in \mathbb{R}^{3\times 3}$ is the rotation matrix of the drone with respect to F^w. The angular velocity and acceleration of the drone with respect to F^w (expressed in frame F^D) are $\vec{w_D^{DW}}$, $\dot{\vec{w}}_D^{DW} \in \mathbb{R}^3$, respectively. $J_D \in \mathbb{R}^{3\times 3}$ is the drone's inertia matrix, g and $m \in \mathbb{R}$ are the gravitational constant and drone mass, and $0_{3\times 3}$, $Id_{3\times 3} \in \mathbb{R}^{3\times 3}$ are the zero and identity matrices. Note that the translational dynamics is expressed in the F^w, while the rotational dynamics is in frame F^D.

$$\begin{pmatrix} m\ddot{\vec{p}}_D^W \\ J_D\dot{\vec{w}}_D^{DW} \end{pmatrix} = -\begin{pmatrix} mg\vec{z}_w \\ \vec{w}_D^{DW} \times J_D\vec{w}_D^{DW} \end{pmatrix} + G\vec{W}^D(\vec{u}), \quad G = \begin{pmatrix} R_D^W & 0_{3\times 3} \\ 0_{3\times 3} & Id_{3\times 3} \end{pmatrix} \in \mathbb{R}^{6\times 6} \quad (3.3)$$

In the case of a quadrotor, we can re-write Expression 3.3 as in Equation 3.4 using $G\vec{W}^D(\vec{u}) = [\vec{z}_D^W f, m_x, m_y, m_z]^T \in \mathbb{R}^6$, being $\vec{z}_D^W$ the z-axis of frame F^D (expressed in frame F^W). With this, $\hat{\vec{u}}$ can be seen as the input of the dynamical system (and its corresponding $\vec{u}$ can be retrieved with map $\vec{u} = \bar{A}^{-1}\hat{\vec{u}}$).

$$\begin{pmatrix} m\ddot{\vec{p}}_D^W \\ J_D\dot{\vec{w}}_D^{DW} \end{pmatrix} = -\begin{pmatrix} mg\vec{z}_w \\ \vec{w}_D^{DW} \times J_D\vec{w}_D^{DW} \end{pmatrix} + \bar{G}\vec{u}^*, \quad \bar{G} = \begin{pmatrix} \vec{z}_D^W & 0_{1\times3} \\ 0_{3\times1} & Id_{3\times3} \end{pmatrix} \in \mathbb{R}^{6\times4} \tag{3.4}$$

As shown in [73], the quadrotor dynamics with the four inputs ($\vec{u}^*$) is differentially flat, meaning that we can track trajectories in a carefully selected four-dimensional space of flat outputs. In our case, we select as outputs: $(p_{D_x}, p_{D_y}, p_{D_z}, \psi)$ = $(\vec{p_D}, \psi)$ — the drone's position and yaw.

Sensors

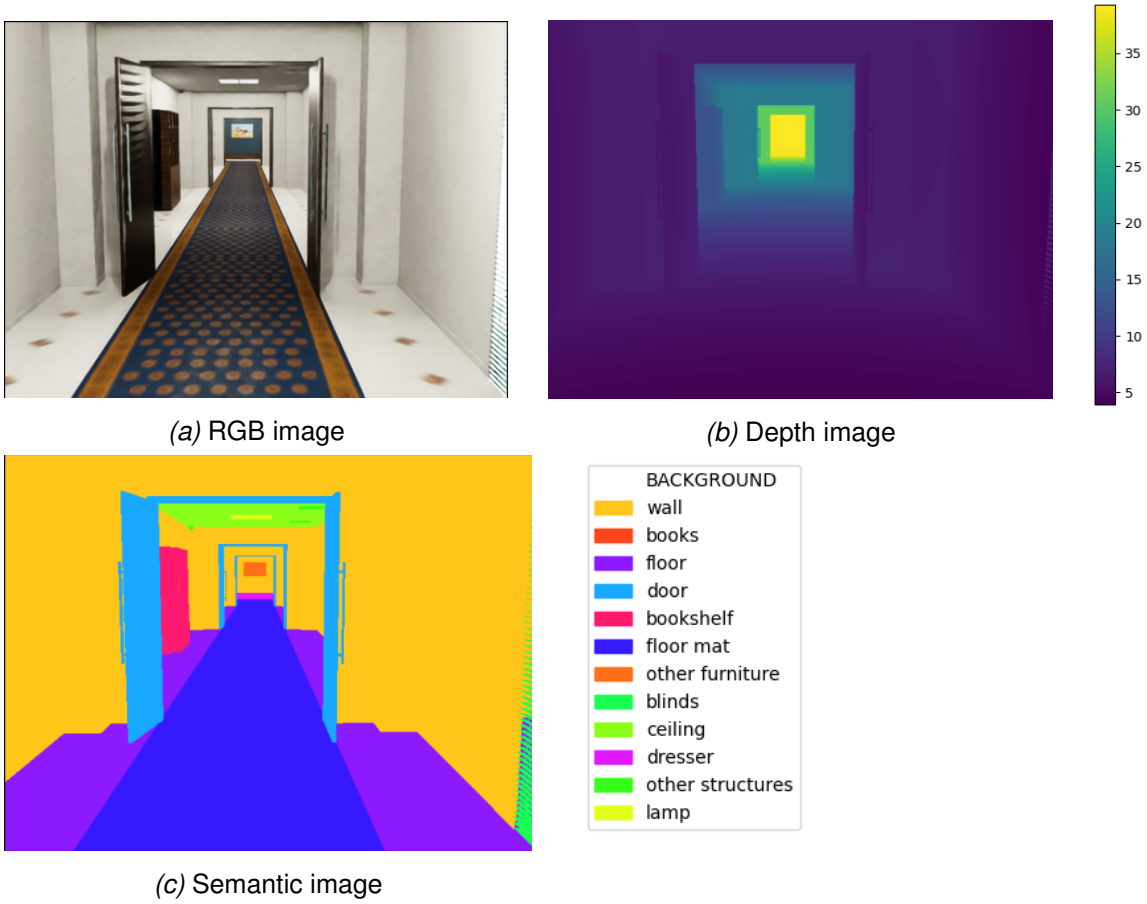
Isaac Lab supports various sensor types (including cameras, IMUs, contact sensors and ray caster sensors). Through its APIs, users can configure and spawn sensors within the simulation scene, and can acess their measurements during runtime.

In this thesis, the drones instantiated in the virtual world were equipped with onboard cameras (capturing RGB, depth, and semantic images from the camera's viewport), and a contact sensor that measures the net force acting on the drone's body (enabling collision detection). Examples of the RGB, depth, and semantic images captured by the onboard camera are provided in Figure 3.4. Data from these sensors, along with real-time scene information, are used to design the observation spaces, reward functions, and termination conditions for both the preliminary target navigation RL task and the main exploration task.

While Isaac Sim natively provides access to depth and semantic images from modelled cameras, the semantic images were limited to pre-labeled scene objects. To address this limitation, all prims in the scene were semi-automatically annotated with semantic labels (see Section 3.3.2). This ensured that all relevant objects within the camera's field of view could be detected.

Environment

The autonomous exploration task addressed in this project is set in an office-like environment. Isaac Sim provides a default office environment described in a USD file, see in Figure 3.5b an image this environment from the drone's onboard camera. To



(a) RGB image

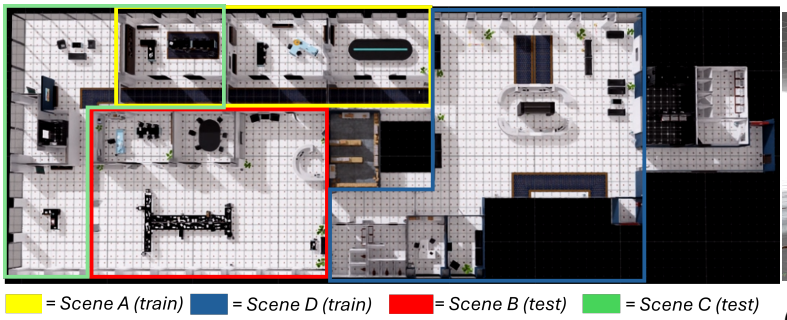
(b) Depth image

(c) Semantic image

Figure 3.4: Images captured by the drone's onboard camera in an office environment.

ensure robust training and testing of the DRL policies, a diverse set of environments was required. To achieve this, the default office environment was divided into four distinct scenes, as illustrated in Figure 3.5a. Scenes D and B feature large rooms with attached smaller rooms, while scenes A and C consist of medium-sized rooms connected by corridors, creating a maze-like structure.

Additionally, the original USD files were modified to simplify the environment and reduce potential sources of error. All windows were removed, and semantic labels were semi-automatically added to all office objects, with labels extracted from the NYUDv2 dataset. Furthermore, adjustments were made to ensure that doors, entrances, and corridors were clearly visible on a low-resolution map, with grid cells sized at 0.4 x 0.4 meters. As a result, all doorways and hallways were designed to be at least twice the size of a grid cell to maintain navigability.



(a) Office environment map, with different areas highlighted.

(b) View of one office area in the environment.

3.4 Low-level Non-linear Controller Module

The low-level controller, implemented within Isaac Lab´s Action Manager, receives an action vector $\vec{v} = (v_x, v_y, w_z)$ from the high-level DRL policy or from the keyboard. This vector specifies 2D linear and angular velocity commands (in the drone's body frame F^D) to move the drone at a fixed height. The controller module computes the body torque to be applied to the UAV's body to track these velocity commands.

3.4.1 Non-linear Control Law

The implemented controller, proposed by [73], is designed to track trajectories in the 4D space $(\vec{p}_D^W(t), \psi(t))$, being $\vec{p}_D^W \in \mathbb{R}^3$ and $\psi \in \mathbb{R}$ the drone's 3D position and yaw in F^W. To achieve this, it computes the appropriate control inputs $\hat{\vec{u}}(t) = (f, m_x, m_y, m_z)^T$, being f the thrust force and $\vec{m}^D = (m_x, m_y, m_z)^T$ the torque on the UAV's body.

First, the controller computes the desired force $\vec{F}^d$ the UAV should exert to follow the desired trajectory, as in Equation 3.5. In this expression, $\vec{e_p} = \vec{p_D}^W - \vec{p_D}^d$ and $\vec{e_v} = \dot{\vec{p}}_D^W - \dot{\vec{p}}_D^d$ are the position and velocity errors (being $\vec{p_D}^d$ and $\dot{\vec{p}}_D^d$ the reference position and velocity values). Additionally, $\vec{e_i} = \int \vec{e_p} dt$ is the integral error (added as in the custom controller of [61]). With this, we can compute $f = \vec{F}^d \cdot \vec{z}_D^W$, which projects $\vec{F}^d$ onto F^D z-axis.

$$\vec{F}^d = -K_p\vec{e_p} - K_v\vec{e_v} - K_i\vec{e_i} + mg\vec{z_w} + m\ddot{\vec{p}}_D^d \tag{3.5}$$

As mentioned in Section 3.3.2, for the drone to move in the desired direction marked by $\vec{F}^d$, it must reorient itself so that $\vec{z}_D^W$ aligns with $\vec{F}^d$. Based on this, we compute the drone's desired orientation $R_D^d = [\vec{x}_D^d, \vec{y}_D^d, \vec{z}_D^d] \in \mathbb{R}^{3\times3}$ as in Equation 3.6, with $\vec{x}_{aux}^d = [cos(\psi^d)], sin(\psi^d), 0]^T$ being ψ^d the desired drone yaw.

$$\vec{z}_D^d = \frac{\vec{F}^d}{\|\vec{F}^d\|} \quad \vec{y}_D^d = \frac{\vec{z}_D^d \times \vec{x}_{aux}^d}{\|\vec{z}_D^d \times \vec{x}_{aux}^d\|} \quad \vec{x}_D^d = \vec{y}_D^d \times \vec{z}_D^d \tag{3.6}$$

Finally, the body torque $\vec{m}^D$ to be applied to the drone to achieve orientation R_D^d can be computed as in Equation 3.7.

$$\vec{m}^D = -K_r\vec{e_r} - K_w\vec{e_w} \tag{3.7}$$

In this expression, $\vec{e_r} = \frac{1}{2}({R_D^d}^T R_D^W - {R_D^W}^T R_D^d)^\vee$ is the orientation error (where $\vee$ is the vee map sign) and $\vec{e_w} = \vec{w}_D^{DW} - \vec{w}_D^d$ is the angular velocity error. The desired angular velocity $\vec{w}_D^d$ is computed as in Equation 3.8, where $\dot{\psi}^d$ is the desired yaw rate, as in the custom controller of [61].

$$\vec{w}_D^d = [-\vec{h}_w \cdot \vec{y}_D^d, \vec{h}_w \cdot \vec{x}_D^d, \dot{\psi}^d \vec{z}_D^d]^T \quad \vec{h}_w = \frac{m}{f}(\dddot{\vec{p}}_D^d - (\vec{z}_D^d \cdot \dddot{\vec{p}}_D^d)\vec{z}_D^d) \tag{3.8}$$

See Figure 3.6 to get an overview of the workflow inside the controller module.

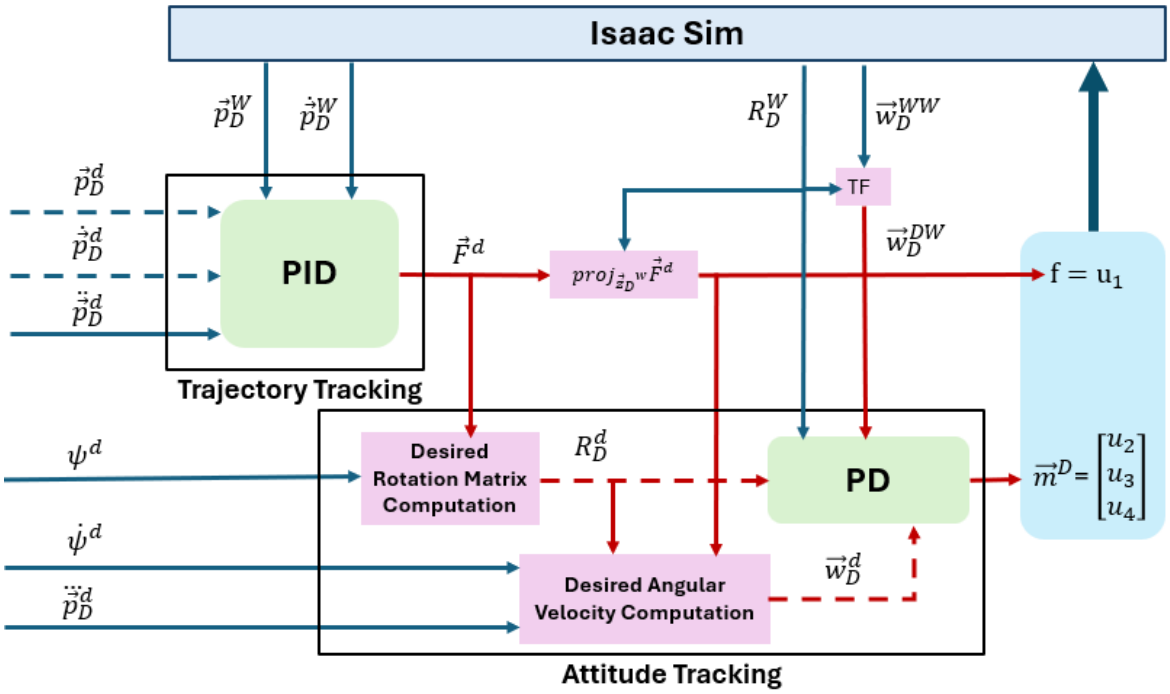


Figure 3.6: Diagram of the drone controller workflow

3.4.2 Implementation specifications

We set the reference values $\vec{p_D}^d$, $\dot{\vec{p}}_D^d$, $\ddot{\vec{p}}_D^d$, $\dddot{\vec{p}}_D^d$, ψ^d and $\dot{\psi}^d$ from the action vector $\vec{v} = (v_x, v_y, w_z)$. The vector components $v_x, v_y, w_z \in [-1, 1]$ and can have a discrete or continuous nature. Here, v_x and v_y specify the desired x and y linear velocity

components for the drone in F^D, while w_z specifies the desired yaw rate (also in F^D).

With this, the reference linear velocity is computed as $\dot{\vec{p}}_D^d = M_{lv} R_D^W [v_x, v_y, 0]^T$, where $M_{lv} \in \mathbb{R}$ scales the velocity components to the drone's maximum achievable linear velocity, and $\dot{p_z}_D^d = 0$ is enforced. The reference position is obtained by integrating the velocity $\vec{p_D}^d = \int \dot{\vec{p}}_D^d dt$ and and enforcing $p_{z_D}^d = H$, where H is a fixed height value. We compute the reference yaw rate as $\dot{\psi}^d = M_{av} w_z$, where $M_{av} \in \mathbb{R}$ scales the yaw rate to it's maximum system value. Finally, we can obtain the reference yaw value by integrating the yaw rate $\psi^d = \int \dot{\psi}^d dt$.

As for the controller outputs, once $\hat{\vec{u}} = (f, m_x, m_y, m_z)^T$ is computed, a real quadrotor would use the allocation matrix A to map this controls to adequate rotor velocities $\vec{u} = (u_1, u_2, u_3, u_4)^T$, where $u_i = w_i |w_i|$ for $i \in [0, 4]$ (see Section 3.3.2). However, due to simulator limitations — specifically, the inability to accurately model the physics of rotating propellers generating thrust and moments on the rotors - we directly apply the computed body wrench to the UAV's body. This approach bypasses the need for drone-specific thrust and drag coefficients c_f, c_τ, requiring only the UAV's mass m and the linear and angular velocity limits defined by M_{lv} and M_{av}.

There is still significant room for improvement in the implementation of this controller. For example, looking at the controller modules in Figure 3.6, it would be desirable for the attitude tracking to operate at a higher rate than the trajectory tracking. However, in the current implementation, both components run at the same rate, which is sufficient for our application.

3.5 High-level DRL Controller

The DRL high-level controller, implemented with SB3, receives observations, rewards, and done signals from the simulation environment, to generate adequate action vectors $\vec{v} = (v_x, v_y, w_z)$ for the RL task at hand.

As outlined in Section 3.1, the designed pipeline is highly versatile and can be adapted to various RL tasks by modifying the MDP specifications (e.g., observation space, action space, reward function and termination conditions), as well as the used DRL algorithm and NN architecture. The implemented pipeline was first tested on a simple goal navigation task before being applied to the more complex problem of autonomous exploration in unknown environments. This section provides a general overview of the DRL controller, detailing its basic structure and the common

elements shared across all tasks. For detailed task-specific configurations, refer to Section 5.2.

3.5.1 Selected Model-free DRL algorithm

The Proximal Policy Optimization (PPO) [39] method was chosen for both the preliminary navigation task and the main exploration task. The reasoning behind this choice is that PPO is specifically designed to ensure training stability and is computationally less costly than other policy-based methods, such as Trust Region Policy Optimization (TRPO) [74]. PPO achieves this through a clipping mechanism on the loss function, which limits how much the policy can change during each optimization step. Additionally, its ability to handle both continuous and discrete action spaces provided the flexibility we needed for our tasks.

PPO, being a policy-based method, learns a parametrized policy $\pi_\theta(b_t) = p(a_t|b_t)$ through a DNN. The network takes observations as inputs and learns a probability distribution over the action space, from which actions are sampled. Additionally, it is trained to maximize the expected cumulative reward over an entire episode while avoiding large, destabilizing policy updates. The loss function defined in Equation 3.9 is used, where $A^{\pi_\theta}(s,a) = q_{\pi\theta}(s,a) - v_{\pi\theta}(s)$ is the advantage function measuring how much better or worse action a is in comparison to the expected action from policy π. Also, ϵ is a hyperparameter which roughly says how far away the new policy is allowed to go from the old [75].

$$L(\theta, \theta_{old}) = E[min[rA^{\pi_{\theta old}}, clip(r, 1-\epsilon, 1+\epsilon)A^{\pi_{\theta old}}]] \quad r = \frac{\pi_\theta(a|s)}{\pi_{\theta old}(a|s)} \tag{3.9}$$

In the SB3 implementation of the PPO algorithm, rollouts are collected from n agents interacting in parallel with the environment. The interaction data is stored in a buffer and later used to update the DNN using the loss function described above.

3.5.2 General DRL architecture

Although each considered task utilized a different DNN, all networks shared the same underlying structure, pictured in Figure 3.7. As shown, the network takes observations as inputs, which can include depth images, semantic images, velocities, and other relevant data. For each type of observation, the DNN employs a dedicated feature extractor (CNNs for image-based observations and an MultiLayer Perceptron (MLP) for vector-based observations). The extracted features are then concatenated and passed to two separate networks: a critic network (an MLP) that

estimates the value of the drone's state (used to compute the advantage for the PPO loss function), and an actor network (also an MLP) that parametrizes the policy. Notably, the concatenated features can be interpreted as the agent's belief state b_t at time t.

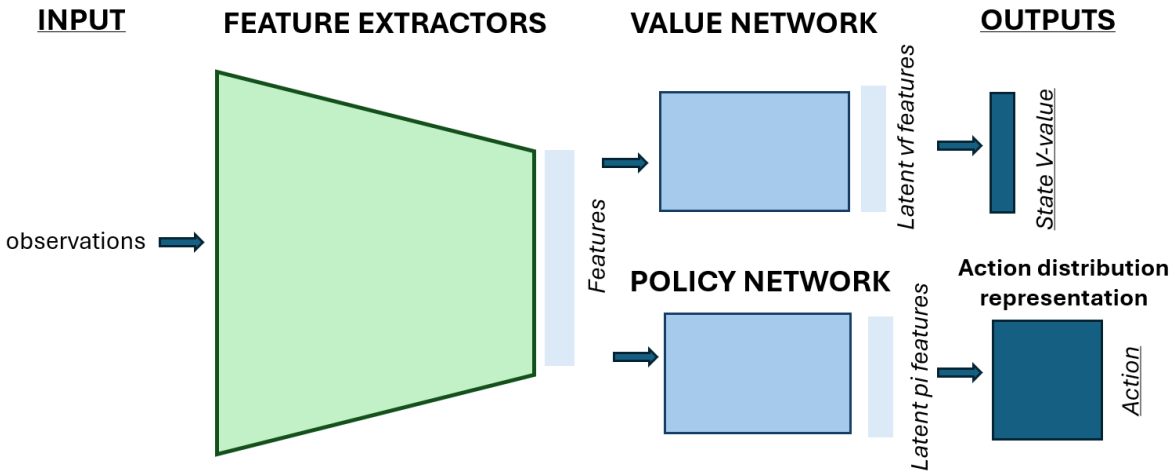


Figure 3.7: Underlying structure of the networks used for both the preliminary navigation task and the main exploration task

Chapter 4

Experiments

4.1 Experimental Setup for Controller Validation

To test the low-level controller module, a simple experiment was conducted in which the drone was spawned in an empty simulator environment and manually controlled using keyboard inputs. The drone was commanded to perform a sequence of basic maneuvers: move forward, backward, to the right, and to the left, followed by a full rotation to the right and another to the left.

During the experiment, reference values for the drone's position $\vec{p_D}^d$, velocity $\dot{\vec{p}}_D^d$, acceleration $\ddot{\vec{p}}_D^d$, yaw angle ψ^d and yaw rate $\dot{\psi}^d$ were recorded. Additionally, the corresponding measured state values — position $\vec{p_D}^W$, velocity $\dot{\vec{p}}_D^W$, acceleration $\ddot{\vec{p}}_D^W$, yaw angle ψ^D and yaw rate $w_z{}_D^{DW}$ — were tracked to assess the controller's ability to follow the reference commands. Section 5.1 presents an evaluation of the controller's tracking performance over time.

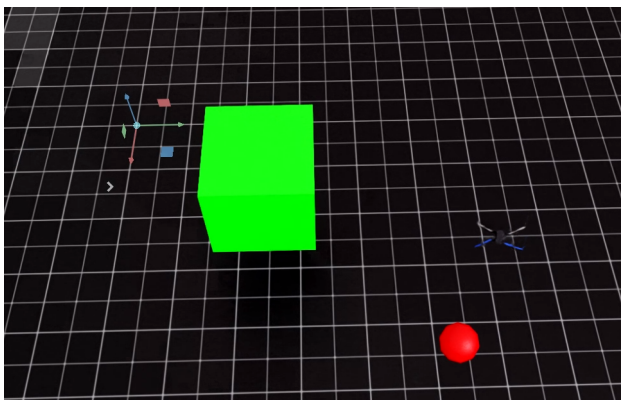
4.2 Experimental Setup for DRL Tasks

4.2.1 Task 1: Collision-Free Target Navigation

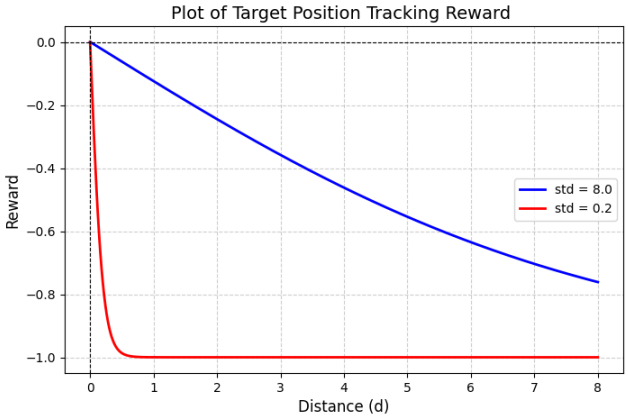
For this task, the drone had to navigate toward a fixed goal while avoiding an obstacle positioned in the middle of its path. To set up the simulation, the drone was spawned at the initial position $\vec{p_D}^W = (0, 0, 1.8)$m, a small target sphere of radius $r = 0.25$m (with collision properties deactivated) was placed at $\vec{p}_{target}^W = (5.0, 5.0, 1.8)$m, and a large cubic obstacle with a side length of 1.5m was positioned at $\vec{p_{obstacle}}^W = (2.5, 2.5, 1.8)$m. See in Figure 4.1a a picture of the scene. No randomization was applied to the initial positions of the drone, target, or obstacle; all episodes began with the same configuration.

As mentioned in Section 3.3.2, the drone was equipped with an onboard camera of resolution 480×640 providing access to both depth and semantic images, as well as a contact sensor. Using this setup, two different observation spaces (with and without semantics) were compared:

- O_1: Downsampled depth and semantic images from the camera (reduced to a size of 48×64 and stacked), the drone's linear velocity $\dot{\vec{p}}^D$ in frame F^D, and the goal position $\vec{p}^D_{target}$ and heading error $\Delta\psi_{target}$ in the drone's frame F^D.
- O_2: Downsampled depth images from the camera (reduced to a size of 48×64), the drone's linear velocity $\dot{\vec{p}}^D$ in frame F^D, and the goal position $\vec{p}^D_{target}$ and heading error $\Delta\psi_{target}$ in the drone's frame F^D.



(a) Setup of the Navigation Task



(b) Plot of $R = -\tanh(\frac{d}{std})$ for $std = 0.2, 8.0$

Figure 4.1: Target Navigation task: drone, target (in red) and obstacle (in green).

The reward function for this task consists of five terms, and is defined as:

$$R = w_{cgt}R_{cgt} + w_{fgt}R_{fgt} + w_{ht}R_{ht} + w_{rg}R_{rg} + w_cR_c \tag{4.1}$$

The first two terms, R_{cgt} and R_{fgt}, share the same mathematical expression $R = -\tanh(\frac{d}{std})$, where d represents the distance between the drone and the goal, and std is a parameter that controls the steepness of the function around zero. Specifically, R_{cgt} used $std = 0.2$, and R_{fcgt} uses $std = 8.0$. These terms encourage the drone to move toward the goal, but with different influences: as illustrated in Figure 4.1b, R_{fgt} (in blue) has a longer-range effect, guiding the drone from further distances, whereas R_{cgt} (in red) provides a stronger incentive when the drone is near the target, acting as a final push. Moreover, the term $R_{ht} = \|\Delta\psi_{target}\|$ encourages the drone to orient itself toward the goal. The term R_{rg} is a sparse reward, where

$R_{rg} = 1$ if the drone successfully reaches the goal and 0 otherwise, reinforcing successful task completion. Finally, R_c penalizes collisions, taking a value of $R_c = -1$ when a collision is detected by the contact sensors and 0 otherwise.

Additionally, the termination conditions dictate that an episode can end in one of three ways: the drone either crashes, reaches the fixed goal, or the episode times out. The maximum episode duration is set to $16s$. Regarding the action space, the output action vectors $\vec{v} = (v_x, v_y, w_z)$ have continuous components, where $v_x, v_y, w_z \in [-1, 1]$. In the controller, the linear and angular velocity limits are set to $M_{lv} = 1.5$m/s, $M_{av} = 1$rad/s, respectively, while the drone's height remains fixed at $H = 1.8$m.

In Isaac Lab, we differentiate between the simulation step dt and the environment step $dt_{env} = decimation * dt$. The first one is the timestep used by the physics engine to update the state of the simulation, while the later one determines how frequently the agent receives observations from the environment and makes a new decision. For this task, we chose $dt = 0.004$s, and $dt_{env} = 0.016$s. As a result, the high-level controller sending an action vector at each environment step, operates at a frequency of 62.5Hz, while the classical low-level controller runs at 250Hz. In hindsight, this was a design flaw, as both frequencies were excessively high and contributed to significant convergence issues during the policy training. A more in-depth discussion on this topic is provided in Section 5.2.2.

The architecture of the trained DNN follows the structure outlined in Section 3.5.2. Each observation has its own custom feature extractor: vector observations are simply flattened, while image observations are processed through a CNN. Both the actor and critic networks are implemented as MLPs with two hidden layers containing 128 and 64 neurons, respectively. The learned distribution for v_x, v_y, w_z consists of independent Gaussian distributions.

4.2.2 Task 2: Autonomous Exploration

Mapping module

As discussed in the literature review (Section 2.1), all the revised autonomous exploration approaches rely on environment representations, as these are essential for preventing overfitting to specific scenarios and for distinguishing between visited and unvisited regions. To tackle the exploration task, we integrate a mapping module into the designed pipeline. In the general scheme of Figure 3.1, this module

belongs to the set of computation modules (in purple) responsible for crafting observations for the DRL controller using sensor data and simulator information. Notably, the module is implemented within Isaac Lab.

At each environment timestep, a 3D PointCloud of the environment (expressed in world coordinates F^W) is generated using the drone's pose and the depth image collected at that moment. To achieve this, the extrinsic camera matrix is computed based on the drone's position and orientation. Then, depth image pixels are transformed into 3D points using the pinhole camera model, the depth values corresponding to each pixel, the previously computed extrinsic matrix, and the intrinsic calibration matrix, which is camera-dependent. This procedure follows standard computer vision techniques, details of which can be found in any fundamental Computer Vision course book [76]. The obtained PointCloud is refined by removing 3D points that exceed a predefined distance threshold from the camera, or correspond to pixels with invalid depth readings.

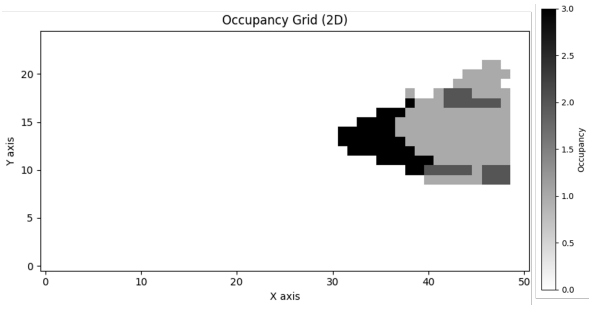
The drone maintains a global grid map estimate that is continuously updated as the UAV explores the environment. At each timestep, a local grid map is generated from the newly computed PointCloud and subsequently merged into the global grid map. To ensure consistency, the current PointCloud is first projected onto a 2D grid, and if any points fall outside the existing global map boundaries, the global map is resized accordingly to accommodate the new data.

In this mapping module, we are constructing a traversability map for the drone. At each timestep, the 3D points from the PointCloud that lie at the same height as the drone (within a specified tolerance) are filtered. Cells in the local grid map where these points are projected are marked as non-traversable, indicating obstacles or areas that the drone cannot navigate. These obstacle cells are also marked in the global grid map in a conservative manner, meaning that once an obstacle is detected, it cannot be removed. Next, ray tracing is applied to the local grid map within the drone's field of view to determine hovered traversable regions. This is necessary because, since the drone flies at a fixed height, areas beneath it that are not directly visible remain classified as unknown (unless an obstacle is explicitly detected), even though they are traversable in reality. Finally, the updated (seen and hovered) traversable regions are added to the local map are merged into the global map.

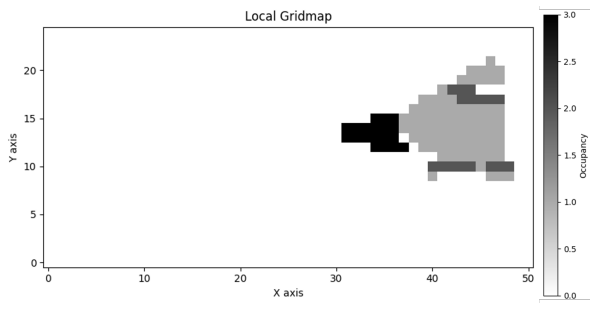
Figures 4.2a, 4.2b, and 4.2c show the global grid map, local grid map and generated PointCloud at a specific time t. Notice how each map cell is assigned a value

depending on its state: 0 indicates unknown, 1 indicates traversable, and 2 indicates an obstacle. In the grid maps shown in Figures 4.2a and 4.2b, cells from the hovered traversable region (added through ray tracing) are differentiated from the seen traversable regions for better clarity. Normally, this distinction is not made.

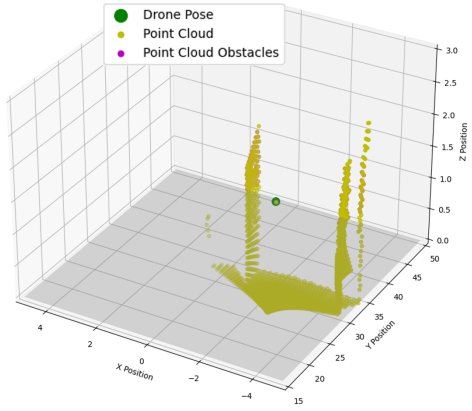
The trained DRL policies for autonomous exploration used a mapping module that built local and global grid maps with a resolution of 0.4×0.4m. This is significantly lower than the 0.05×0.05m resolution used in the reviewed works of Section 2.1. Initially, this choice was driven by system limitations, as the available server struggled with the computational cost of high-resolution maps, leading to extremely slow training. However, this constraint proved beneficial, as UAVs have limited payload capacity and cannot carry the powerful computers required to process such detailed maps efficiently. Nevertheless, the reduced resolution introduced challenges, particularly in collision avoidance, due to the increased mapping errors associated with lower resolution. Moreover, the 3D points in the PoinCloud whose distance from the camera was greater than 4m were filtered out.



(a) Global estimated grid map



(b) Local grid map at a specific time t



(c) PointCloud at a specific time t

Figure 4.2: Global grid map, local grid-map and PointCloud at a specific time t. For the occupancy grids: 0 -Unknown space, 1 -Seen traversable space, 2 -Obstacles, 3 - Hovered traversable space.

First test on a small environment

To tackle the autonomous exploration problem progressively, the agent was initially trained to explore a simplified, small-scale environment consisting of a single room with four columns, instead of the larger office environment that we are aiming for. This smaller environment is shown in Figure 4.3. For each episode, randomization was applied to the drone's initial pose by sampling from a set of 20 possible initial poses. With this, we aim to enhance the agent's generalization capabilities.

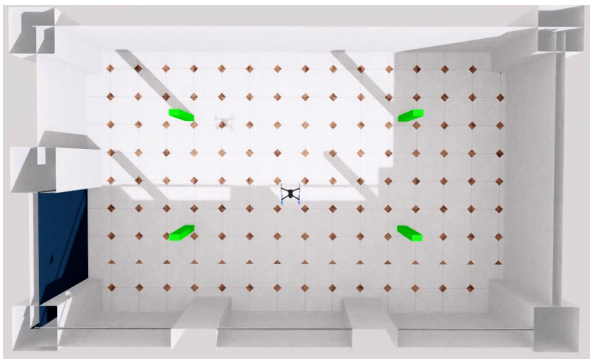


Figure 4.3: Simple scene consisting of one room and fours columns (in green) for the "small exploration" task.

As in the previous task, the drone was equipped with an onboard camera (this time with a resolution of 48×64), which provided access to both depth and semantic images, along with a contact sensor. Using this setup, the proposed observation space consisted of semantic images from the camera, and an egocentric map of size 41×41 cropped from the global grid map maintained by the drone.

The reward function for the "small exploration" task includes three terms:

$$R = w_a R_a + w_{ca} R_{ca} + w_c R_c \tag{4.2}$$

The first term, $R_a = A_{t+1} - A_t$, where A_t represents the number of known cells in the global map at time t, encourages exploration by maximizing area coverage. The second term, R_{ca}, is a sparse reward, where $R_{ca} = 1$ if the area covered in the global map exceeds a specified threshold, and 0 otherwise. Note that this term requires access to prior knowledge of the environment. Finally, R_c penalizes collisions, taking a value of $R_c = -1$ when a collision is detected by the contact sensors and 0 otherwise.

As for the termination conditions, we imposed that an episode can end in one of three ways: the drone either crashes, maps a certain amount of area (successfully

finishing exploration), or reaches the maximum episode time of 210s. Regarding the action space, the DRL policy outputs a discrete value $a \in [0, 5]$, with each possible value corresponding to a predefined motion primitive represented by an action vector $\vec{v} = (v_x, v_y, w_z)$. Specifically, $a = 0$ moves the drone forward with $\vec{v} = (1, 0, 0)$, $a = 1$ moves it backward with $\vec{v} = (-1, 0, 0)$, $a = 2$ moves it right with $\vec{v} = (0, 1, 0)$, and $a = 3$ moves it left with $\vec{v} = (0, -1, 0)$. For rotations, $a = 4$ makes the drone turn right with $\vec{v} = (0, 0, 1)$ and $a = 5$ makes it turn left with $\vec{v} = (0, 0, -1)$.

When concluding the navigation task (Section 4.2.1), we hypothesized that both the high-level and low-level controllers were operating at excessively high frequencies. This caused the drone to receive commands too rapidly, leaving it insufficient time to engage with its actions or observe their consequences, which hindered training convergence. To test this hypothesis, we bypassed the low-level controller and directly updated the drone's position and yaw in the simulator using:

$$\vec{p_D}^W(t+1) = \vec{p_D}^W(t) + dt\dot{\vec{p}}_D^d \quad \psi_D^W(t+1) = \psi_D^W(t) + dt w_{z_D}^d \tag{4.3}$$

In this equation, dt is the simulation step (equal to the environment step), $\dot{\vec{p}}_D^d = M_{lv} R_D^W [v_x, v_y, 0]^T$ (as when setting the reference for the low-level controller), and $w_{z_D}^d$ is the third component of vector $\dot{\vec{w}}_D^d = M_{av} R_D^W [0, 0, w_z]^T$. As in Section 3.4.2, we enforce $\dot{p_z}_D^d = 0$ and $p_{z_D}^d = H$ with $H = 1.8$m. Two different models with $dt = 0.1$s and $dt = 0.25$s were trained and compared.

As for the DNN architecture, we employed two pretrained ResNet18 networks as feature extractors for the image observations (semantic images and egocentric maps). As in the navigation task, both the actor and critic networks were implemented as MLPs with two hidden layers, containing 128 and 64 neurons, respectively. The learned distributions for vx, vy, wz were modeled as Categorical distributions.

Final experiment on big training and testing environments

For the final task, we trained a UAV to autonomously explore an office environment. As mentioned in Section 3.3.2, the drone was trained in two distinct environments: Scene A and Scene D. Scene A consists of a narrow corridor connecting three separate rooms, introducing navigation challenges related to constrained spaces and doorways. In contrast, Scene D features a spacious central area with multiple adjoining smaller rooms, requiring the drone to adapt to open spaces. This setup maximized environmental diversity, ensuring the drone was exposed to a wide range of navigation scenarios. For each episode, randomization was applied to the drone's

initial pose by sampling from a set of 128 possible initial poses, with half belonging to Scene A and the other half to Scene D. This ensured the drone had an equal chance of being spawned in either environment.

As in the ”small exploration” task, the drone was equipped with an onboard camera with a resolution of 48×64 providing both depth and semantic images, along with a contact sensor. Using this setup, two different observation spaces (with and without semantics) were compared:

- O_1: Depth images of size 48×64 (truncated for depth values higher than 4m), and a one-hot-encoded egocentric map of size $4 \times 61 \times 61$, cropped from the global grid map in the mapping module. In the egocentric map, the first dimension represented distinct classes, similar to how cells are marked with specific values in the global grid map of the mapping module. The four classes included: unknown space, traversable space, non-traversable space, and the drone's current position.
- O_2: Same observations as in O_1, along with one-hot-encoded semantic images of size num classes $\times\ 48 \times 61$. One model was tested using the 41 labeled classes present in the scene, while another model was tested with only four key classes: door, wall, ceiling, and floor. We refer to these observation spaces as O_2^{full} for the model using all 41 classes, and $O_2^{filtered}$ for the model using the four key classes.

The reward function for the main exploration task was practically the same as the one for the ”small exploration task” (see Equation 4.2), with an added term R_r that takes a value of -1 at each step to encourage the drone to explore more quickly.

$$R = w_a R_a + w_{ca} R_{ca} + w_c R_c + w_r R_r \tag{4.4}$$

The termination conditions were the same as those for the ”small exploration task.” With this, an episode could end in one of three ways: the drone either crashes, successfully maps a predefined area (completing the exploration), or reaches the maximum episode time of 700s.

Regarding the action space, two options were considered:

- A_1: Identical to the one used in the ”small exploration task,” where the DRL policy outputs a discrete value $a \in [0, 5]$. Each value corresponds to a predefined motion primitive represented by an action vector $\vec{v} = (v_x, v_y, w_z)$. With these motion primitives, the drone can move forward, backward, left, right, and rotate left or right.

- A_2: The DRL policy outputs a discrete value $a \in [0, 3]$, with each value corresponding to a predefined motion primitive represented by an action vector $\vec{v} = (v_x, v_y, w_z)$. In this case, the drone can only move forward and rotate left or right. Since its movement is restricted to areas visible in the depth image, this action space was tested to assess whether it improved collision avoidance performance.

When training the DRL policy for the "large exploration" task, the low-level controller was also bypassed and the drone was spawned at the adequate positions following Equation 4.3 with $dt = 0.25$s.

The basic architecture of the trained DNN was practically the same as in the "small exploration task". Pretrained ResNet18 networks were used as feature extractors for the image observations, with a separate network for each. The actor-critic networks followed the same architecture as for the two previous tasks. Additionally, we tested incorporating a Recurrent Neural Network (RNN) immediately after the feature extractors and before the actor-critic networks, processing the concatenated extracted features. This approach, commonly used in the reviewed papers in Section 2.1, aimed to leverage memory mechanisms that could benefit the agent. For example, a RNN could help retain crucial information when using local maps where exploration borders might disappear from view, when dealing with noisy maps, or when navigating environments with dynamic obstacles.

For this task, a total of five models were trained combining the different action and observation space configurations: A_1+O_1, A_1+O_2^{full}, A_2+O_1, A_2+$O_2^{filtered}$ and A_2+O_1+ with an added RNN.

Chapter 5

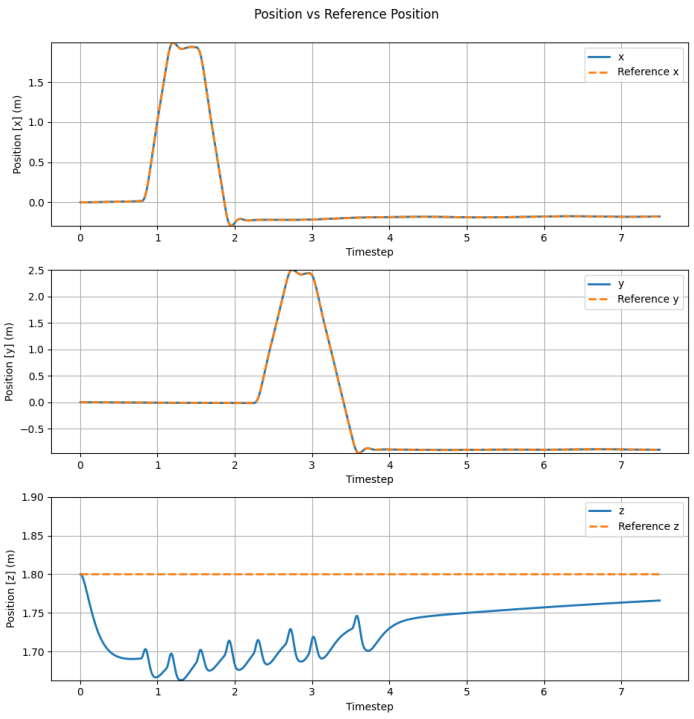
Results

5.1 Performance Results for Controller Validation

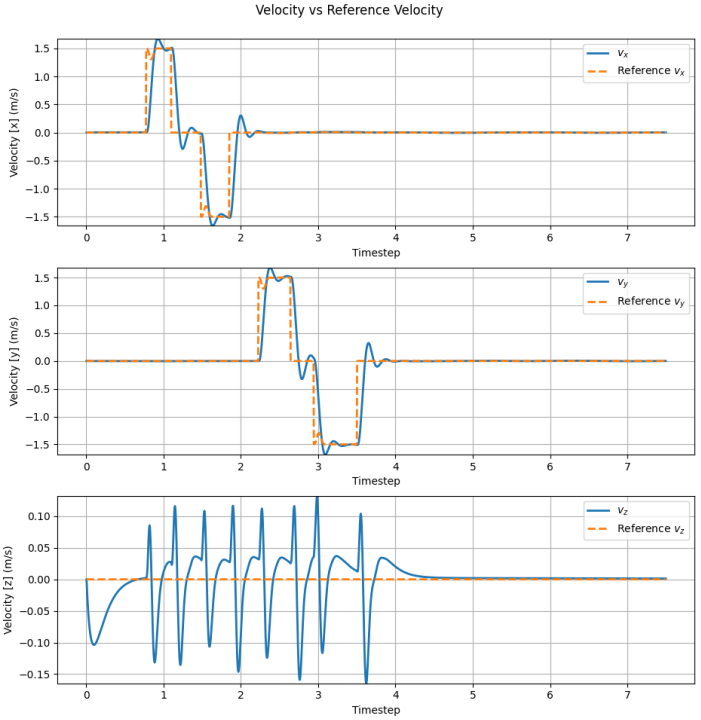
The plots in Figures 5.1a, 5.1b, and 5.2 illustrate the performance of the low-level controller in tracking position $\vec{p_D}^d$, velocity $\dot{\vec{p}}_D^d$, yaw ψ^d and yaw rate $\dot{\psi}^d$ references provided via keyboard input. These results were recorded during a series of basic maneuvers designed to test the controller's capabilities: moving forward, backward, to the right, and to the left, followed by full rotations to the right and left.

The controller demonstrates excellent tracking performance in the x and y components of both position and velocity, accurately following the commanded references. However, the z component exhibits some strange behaviour. While the reference for this component is constant for both the position ($p_{z_D}^d = 1.8$m) and the velocity ($\dot{p_z}_D^d = 0$m/s), the corresponding states $p_{z_D}^W$, $\dot{p_z}_D^W$ show noticeable spikes over time. Additionally, the altitude $p_{z_D}^W$ is maintained around 1.7 m instead of the desired 1.8 m, indicating a steady-state error. In contrast, the yaw and yaw rate references are perfectly tracked. These observations suggest that we need to further refine the gains for the z component.

We also observed that the drone sways significantly when the reference changes abruptly. To mitigate this, we experimented with adjusting the derivative gain of the controller to make it less reactive. Alternatively, interpolating the reference during abrupt changes could help smooth the response.



(a) Low-level controller position tracking: x-component (top), y-component (middle), z-component (bottom).



(b) Low-level controller velocity tracking: x-component (top), y-component (middle), z-component (bottom).

Figure 5.1: Position and velocity tracking.

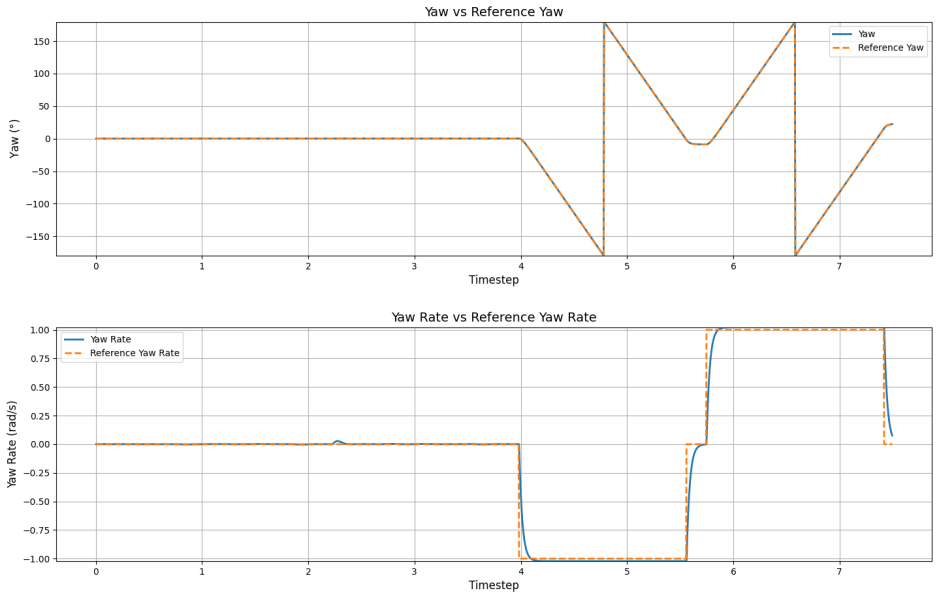
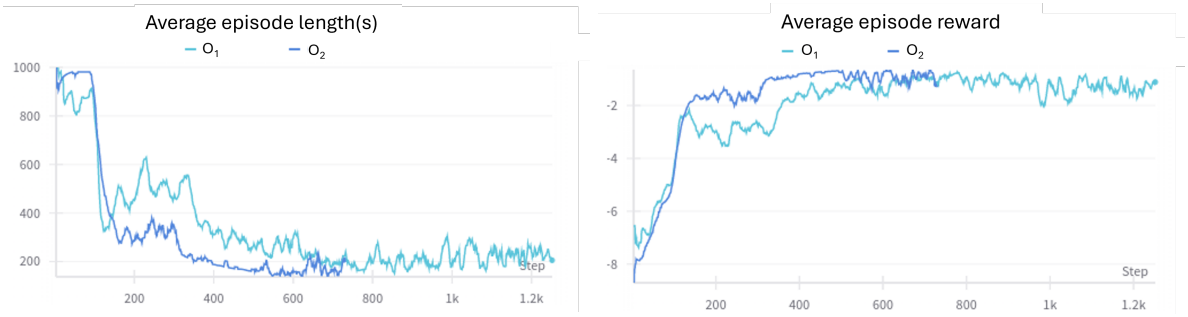


Figure 5.2: Low-level controller yaw tracking (top) and angular velocity tracking (bottom).

5.2 Results of DRL Tasks

5.2.1 Task 1: Collision-Free Target Navigation

Figures 5.3a and 5.3b show the average episode length and average episode reward during the training of an agent for the navigation task, using the MDP specifications outlined in Section 4.2.1. The light blue curve represents the model with observation space O_1, which includes both semantic and depth images, while the darker blue curve corresponds to the model with observation space O_2, which excludes semantic images. As the plots indicate, there is little difference in training convergence between the two models. This is likely because, in such a simple environment, the obstacle and target are already clearly visible in the depth images, and the semantic labels don't add any information.



(a) Average episode length during training

(b) Average episode reward during training

Figure 5.3: Average episode length and reward when training the Navigation task.

This task served primarily as a proof of concept for the pipeline and was not the main focus of the study. A qualitative assessment was conducted by playing back checkpoints, and we observed that the drone was able to successfully navigate around the obstacle and reach the target. However, there is a very high possibility that the policy is overfitting, as the training did not include randomization for the drone's initial pose, the obstacle position, or the target location. If we had more time, the policy could have been trained with randomized initial conditions of these quantities to ensure better generalization. Under such conditions, the inclusion of semantic information would likely have played a more significant role in improving the agent's performance and robustness.

After training multiple models with slight tweaks for the navigation task, a significant concern arose: the training process took an excessively long time—up to four days—for a task that is in theory simple. Additionally, the drone's behavior was often suboptimal; instead of moving directly toward the goal, it sometimes appeared to sway on the spot or exhibit erratic movements. These observations led us to think

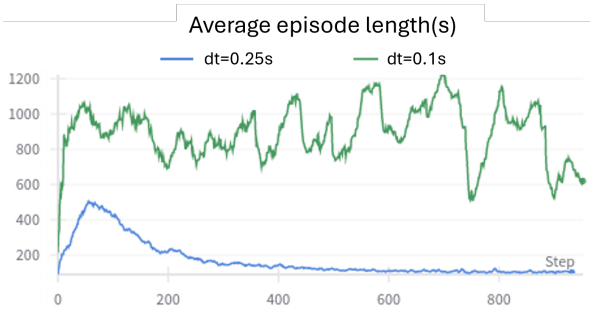
that the high-level controller frequency was improperly set, causing convergence issues. Specifically, we suspected that the frequency was too high, resulting in the drone receiving commands at an excessive rate. This left the UAV with insufficient time to fully execute its actions or observe their consequences, ultimately hindering the training convergence.

A possible reason why some policies converged, despite the potential issue with the frequency, is that extended training allowed the drone to ”memorize” the path to the goal. Testing this would have required randomizing the drone's initial pose, obstacle position, and target location, but due to time constraints, we moved on to the next task—the ”small exploration task.” For this task, we tested an alternative approach: bypassing the low-level controller and directly controlling the drone with a larger environment step. This significantly reduced the high-level controller frequency, helping us evaluate if the convergence issues were tied to the command rate.

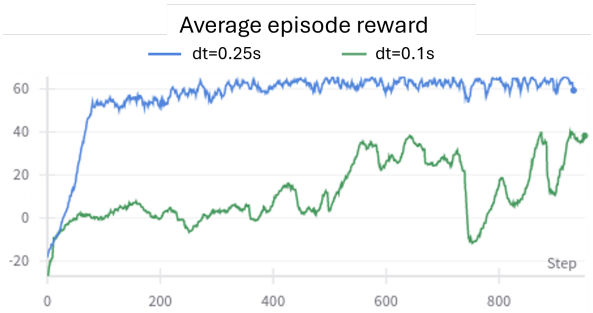
5.2.2 Task 2: Autonomous Exploration

First test on a small environment

For this task, two different models were trained with $dt = 0.1$s and $dt = 0.25s$. Figures 5.4a and 5.4b show the average episode length and average episode reward during training. As the plots indicate, the model with $dt = 0.25s$ (blue line) converges very quickly, while the model with $dt = 0.1s$ fails to converge at all. These results confirm our suspicions regarding the issue with the high-level controller frequency.



(a) Average episode length during training



(b) Average episode reward during training

A qualitative assessment of the model's performance was conducted by playing back checkpoints from the model that converged. The drone successfully covered the majority of the room area, moving in a straight ”up and down” pattern, as illustrated by the red trajectory in Figure 5.5.

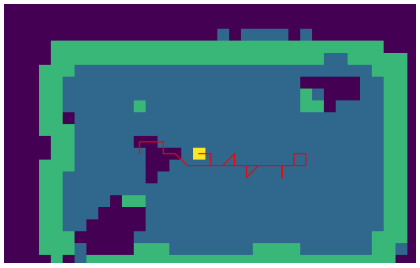


Figure 5.5: Global grid map created during training of the ”small exploration” task. The drone's trajectory appears in red.

Final experiment on big training and testing environments

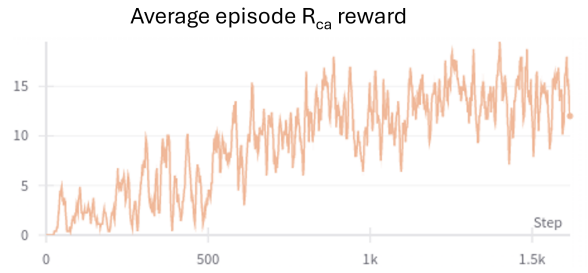
For this final task, a total of five models were trained, combining the action and observation spaces detailed in Section 4.2.2: A_1+O_1, A_1+O_2^{full}, A_2+O_1, A_2+$O_2^{filtered}$ and A_2+O_1+ with an added RNN. To summarize:

- A_1: Action space allowing the drone to move forward, backward, left, right, turn left, and turn right.
- A_2: Action space limiting the drone's movement to the region visible in the camera's field of view (FOV), enabling only forward, turn left, and turn right.
- O_1: Observation space composed of depth image observations and an egocentric map.
- O_2: Observation space using semantic images on top of O_1.In particular, +O_2^{full} uses all 41 semantic labels during training, while $O_2^{filtered}$ uses only four key class labels.

Notice that the semantic images are one-hot encoded and fed into a feature extractor (ResNet). As a result, the number of trainable parameters increases with the number of labels.

All models successfully converged during training. Figure 5.6 illustrates the training process for model $A2$+$O1$. As shown, the average episode contribution of the reward term R_{ca} (a sparse reward triggered when the entire scene is mapped) increases over time. This indicates that, as training progresses, more agents successfully explore the full environment. Notably, training time for $O_2^{filtered}$ is significantly shorter than for O_2^{full}. A crucial factor in achieving convergence was dramatically increasing the optimization batch size. This adjustment allowed the drone to gather a more diverse set of interactions per optimization step, reducing the risk of overfitting to a specific batch of data. Specifically, the batch size was increased from 256 to 3500 interactions.

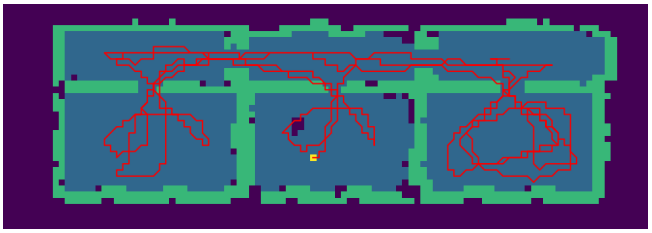
(a) Average episode reward during training



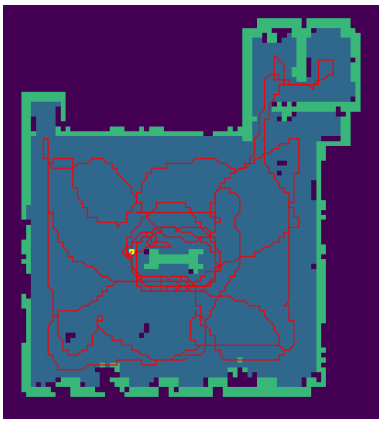
(b) Contribution of R_{ca} to the average episode reward during training

Figure 5.6: Average episode length and reward when training the full exploration task with model $A2$+$O1$

A qualitative assessment of the models' performance was conducted by playing back checkpoints. For those checkpoints further along in training, the drone was able to cover nearly all of Scene A and Scene D, as shown in Figure 5.7. Notably, the drone seems to follow a strategy aimed at time-efficient mapping. In the trajectory (shown in red) for Scene D (which features a very large room), the drone moves in circles to maximize area coverage.



(a) Global grid map created during training in Scene A



(b) Global grid map created during training in Scene D

Figure 5.7: Global grid maps created during training

To test the generalization capabilities of the trained policies, a drone following each policy was spawned in all available scenes, both seen (during training) and unseen, from various initial poses (268 in total, counting all scenes). While the drone interacted with the environment, key data was recorded, such as area covered, timestep, episode ID, and the reason for episode termination. Based on this data, the tables in Figure 5.8 and 5.9 were created, detailing for each scene and model: average episode time (s), average area coverage ($\%$), success rate ($\%$), and collision rate ($\%$). The success rate indicates how many of the spawned drones

cover at least an 80% of the total complete map of the corresponding scene. Note that, to measure area coverage percentage and success rate, an exact ground truth of each scene's area (the number of cells in a complete map) was not available. We approximated it from the scene geometric map, although it is not exact. Model A_2+$O_2^{filtered}$ could not be evaluated due to time constraints

Scene	A				D			
Model	Area coverage (%)	Average episode time(s)	Collision rate (%)	Success rate (%)	Area coverage (%)	Average episode time(s)	Collision rate (%)	Success rate (%)
$O_1 + A_1$	69	1978,11	21	57	93	406	28	92
${O_2}^{full} + A_1$	59	1023,86	77	47	81	257,07	56	78
$O_1 + A_2$	72	2417,95	28	65	98	582,4	6	100
$O_1 + A_2 + RNN$	73	2400,86	26	64	95	597,25	19	94

Figure 5.8: Table comparing the generalization capabilities for each policy and scene pair, for training scenes A and D

Scene	B				C			
Model	Area coverage (%)	Average episode time(s)	Collision rate (%)	Success rate (%)	Area coverage (%)	Average episode time(s)	Collision rate (%)	Success rate (%)
$O_1 + A_1$	57	1814.16	43	0	54	1264.16	74	4
${O_2}^{full} + A_1$	58	1942.13	58	0	31	871.72	84	1
$O_1 + A_2$	70	2613.68	18	0	61	2432.22	29	5
$O_1 + A_2 + RNN$	70	2627.38	12	0	59	2160.89	45	5

Figure 5.9: Table comparing the generalization capabilities for each policy and scene pair, for testing scenes B and C

There doesn't appear to be a policy that stands out significantly from the others. Performance seems to be more influenced by the scene rather than the policy itself. In the training scenes, performance in Scene D is notably better than in Scene A, which may be due to Scene A being more challenging with its smaller spaces. Although there is a slight drop in performance between the training and testing environments, it is not particularly significant, especially when comparing scenes of the same type (e.g., A vs. C or D vs. B).

Chapter 6

Conclusions and recommendations

6.1 Conclusions

In this work, we have addressed the challenge of autonomous exploration in unknown, office-like indoor environments using a UAV trained with DRL. The primary contributions of this thesis are as follows:

- **Development of an Exploration Pipeline:** A comprehensive DRL-based exploration pipeline was developed from scratch, integrating high-level decision-making with low-level control to enable autonomous exploration. This pipeline is modular and adaptable, making it suitable for a wide range of UAV tasks beyond exploration.
- **Semantically Labelled Simulation Environments:** Realistic, semantically labelled environments were created from the default office environment in NVIDIA Isaac Sim, providing a robust and versatile platform for training and testing exploration agents.
- **Low-Level Controller Implementation:** A low-level controller for UAVs was implemented in Isaac Lab, making it one of the first of its kind for multirotors in this framework. This controller can work independently or be easily integrated into the DRL pipeline, providing flexibility for future use.
- **Bridging Gaps in Autonomous Exploration Literature:** This work identifies and addresses several gaps in the field of autonomous exploration
  - It is one of the first approaches to avoid imitation learning, relying solely on DRL for training.
  - It uses low-resolution maps that reduce computational costs, making the proposed approach suitable for UAVs with limited onboard processing capabilities.

- It uses a photorealistic, physics accurate simulator.
- It provides initial insights into the use of semantic information as part of the observation space.

6.2 Future Work

1. **Low-level controller integration for exploration**. Combine the low-level and high-level controllers to train the drone for the exploration task, accounting for the robot's dynamics. Use reference interpolation to reduce swaying.
2. **Implement termination conditions without prior knowledge**. Avoid the need to know the environment's area for termination criteria. Instead, try to detect ”open ends” in the map.
3. **Improve generalization**. Create a module in Isaac Lab (using the Event Manager) to randomly place walls and obstacles in the environment, ensuring the drone encounters different layouts each time it's launched.
4. **Benchmark against other methods**. Compare the proposed method with other exploration strategies to understand its strengths and weaknesses. Possible benchmarks include geometry-based approaches or those in [8].
5. **Account for uncertainty**. Currently, the drone assumes perfect data for pose estimation. Future work should evaluate the policy's performance with realistic conditions by adding drift and noise to the odometry sensor data.
6. **Real-world deployment**.Test the trained policy on a real drone. Since the policy uses depth and semantic images, the sim-to-real gap should be small.
7. **Use semantics in reward shaping**. Currently, semantic information is part of the observation space, but its potential for improving the reward shaping is unexplored.
8. **3D exploration**. Extend the approach to 3D actions. Develop methods to represent and navigate in 3D spaces, such as volumetric mapping.

Bibliography

[1] N. Hughes, Y. Chang, S. Hu, R. Talak, R. Abdulhai, J. Strader, and L. Carlone, "Foundations of spatial perception for robotics: Hierarchical representations and real-time systems," 2023.

[2] U. V. B. L. Udugama, G. Vosselman, and F. Nex, "Mono-hydra: Real-time 3d scene graph construction from monocular camera input with imu," 2023.

[3] P. Wenzel, T. Schön, L. Leal-Taixé, and D. Cremers, "Vision-based mobile robotics obstacle avoidance with deep reinforcement learning," 2021.

[4] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," 2022.

[5] M. F. Ahmed, K. Masood, V. Fremont, and I. Fantoni, "Active slam: A review on last decade," *Sensors*, vol. 23, no. 19, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/19/8097

[6] J. A. Placed and J. A. Castellanos, "A deep reinforcement learning approach for active slam," *Applied Sciences*, vol. 10, no. 23, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/23/8386

[7] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," 2020.

[8] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," 2019.

[9] C. Oh and A. Cavallaro, "Learning action representations for self-supervised visual exploration," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 5873–5879.

[10] Y. Wu, N. Chen, G. Fan, D. Yang, L. Rao, S. Cheng, X. Song, and Y. Ma, "Navs: A neural attention-based visual slam for autonomous navigation in unknown 3d environments," *Neural Process. Lett.*, vol. 56, p. 61, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267961555

[11] X. Puig, E. Undersander, A. Szot, M. D. Cote, R. Partsey, J. Yang, R. Desai, A. W. Clegg, M. Hlavac, T. Min, T. Gervet, V. Vondrus, V.-P. Berges, J. Turner, O. Maksymets, Z. Kira, M. Kalakrishnan, J. Malik, D. S. Chaplot, U. Jain, D. Batra, A. Rai, and R. Mottaghi, "Habitat 3.0: A co-habitat for humans, avatars and robots," 2023.

[12] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997, pp. 146–151.

[13] J. Du, L. Carlone, M. Kaouk, B. Bona, and M. Indri, "A comparative study on active slam and autonomous exploration with particle filters," pp. 916–923, 07 2011.

[14] D. Trivun, E. Salaka, D. Osmankovic, J. Velagic, and N. Osmic, "Active slam-based algorithm for autonomous exploration with mobile robot," *Proceedings of the IEEE International Conference on Industrial Technology*, vol. 2015, pp. 74–79, 06 2015.

[15] J. A. Placed and J. A. Castellanos, "Fast autonomous robotic exploration using the underlying graph structure," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6672–6679.

[16] M. F. Ahmed, V. Frémont, and I. Fantoni, "Active slam utility function exploiting path entropy," in *2023 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, 2023, pp. 1–7.

[17] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/tssc.1968.300136

[18] X. Yan, R. Ding, Q. Luo, C. Ju, and D. Wu, "A dynamic path planning algorithm based on the improved dwa algorithm," in *2022 Global Reliability and Prognostics and Health Management (PHM-Yantai)*, 2022, pp. 1–7.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[20] B. Mu, M. Giamou, L. Paull, A. akbar Agha-mohammadi, J. Leonard, and J. How, "Information-based active slam via topological feature graphs," 2015.

[21] J. Vallvé and J. Andrade-Cetto, “Active pose slam with rrt*,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2167–2173.

[22] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” 2011.

[23] F. Chen, J. D. Martin, Y. Huang, J. Wang, and B. Englot, “Autonomous exploration under uncertainty via deep reinforcement learning on graphs,” 2020.

[24] F. Chen, S. Bai, T. Shan, and B. Englot, “Self-learning exploration and mapping for mobile robots via deep reinforcement learning,” *AIAA Scitech 2019 Forum*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:68149666

[25] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, “Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[26] H. Li, Q. Zhang, and D. Zhao, “Deep reinforcement learning based automatic exploration for navigation in unknown environment,” 2020.

[27] N. Botteghi, R. Schulte, B. Kallfelz Sirmacek, M. Poel, and C. Brune, “Curiosity-driven reinforcement learning agent for mapping unknown indoor environments,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. V-1-2021, pp. 129–136, 06 2021.

[28] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” 2017.

[29] S. Bai, F. Chen, and B. Englot, “Toward autonomous mapping and exploration for mobile robots through deep supervised learning,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2379–2384.

[30] J. Zhang, L. Tai, M. Liu, J. Boedecker, and W. Burgard, “Neural slam: Learning to explore with external memory,” 2017.

[31] S. Zhao and S.-H. Hwang, “Exploration- and exploitation-driven deep deterministic policy gradient for active slam in unknown indoor environments,” *Electronics*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:268328434

[32] W. Chen, W. Li, A. Yang, and Y. Hu, “Active visual slam based on hierarchical reinforcement learning,” *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7155–7162, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:266195456

[33] L. Tai and M. Liu, “Towards cognitive exploration through deep reinforcement learning for mobile robots,” 2016.

[34] K. Wu, M. A. Esfahani, S. Yuan, and H. Wang, “Depth-based obstacle avoidance through deep reinforcement learning,” in *Proceedings of the 5th International Conference on Mechatronics and Robotics Engineering*, ser. ICMRE’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 102–106. [Online]. Available: https://doi.org/10.1145/3314493.3314495

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” 2013.

[36] J. C. de Jesus, V. A. Kich, A. H. Kolling, R. B. Grando, R. da Silva Guerra, and P. L. J. D. Jr, “Depth-cuprl: Depth-imaged contrastive unsupervised prioritized representations in reinforcement learning for mapless navigation of unmanned aerial vehicles,” 2022.

[37] D.-G. Thomas, D. Olshanskyi, K. Krueger, T. Wongpiromsarn, and A. Jannesari, “Interpretable uav collision avoidance using deep reinforcement learning,” 2021.

[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” 2018.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017.

[40] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” 2017.

[41] L. He, N. Aouf, J. F. Whidborne, and B. Song, “Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7491–7497.

[42] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2015.

[43] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, “Curiosity-driven exploration for mapless navigation with deep reinforcement learning,” 2018.

[44] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," 2016.

[45] V. Dhiman, S. Banerjee, B. Griffin, J. M. Siskind, and J. J. Corso, "A critical investigation of deep reinforcement learning for navigation," 2018.

[46] L. Kästner, C. Marx, and J. Lambrecht, "Deep-reinforcement-learning-based semantic navigation of mobile robots in dynamic environments," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 1110–1115.

[47] L. Bartolomei, L. Teixeira, and M. Chli, "Semantic-aware active perception for uavs using deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3101–3108.

[48] S. Krishna, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull, "Deep active localization," 2019.

[49] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov, "Active neural localization," 2018.

[50] Z. Ravichandran, L. Peng, N. Hughes, J. D. Griffith, and L. Carlone, "Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks," 2021.

[51] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi, "Visual semantic navigation using scene priors," 2018.

[52] Y. Liang, B. Chen, and S. Song, "Sscnav: Confidence-aware semantic scene completion for visual semantic navigation," 2020.

[53] "Web site of nvidia isaac sim." [Online]. Available: https://developer.nvidia.com/isaac/sim

[54] "Web site of nvidia isaac lab." [Online]. Available: https://isaac-sim.github.io/IsaacLab/main/index.html

[55] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[56] "Web site of gazebo simulator." [Online]. Available: https://gazebosim.org/home

[57] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, Feb 2018. [Online]. Available: https://doi.org/10.21105/joss.00500

[58] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[59] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," 2017.

[60] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on Robot Learning*, 2020.

[61] M. Jacinto, J. Pinto, J. Patrikar, J. Keller, R. Cunha, S. Scherer, and A. Pascoal, "Pegasus simulator: An isaac sim framework for multiple aerial vehicles simulation," in *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2024, pp. 917–922.

[62] "Web site of px4 autopilot." [Online]. Available: https://docs.px4.io/main/en/flight_controller/

[63] "Web site of omniverse isaac gym environments." [Online]. Available: https://github.com/isaac-sim/OmniIsaacGymEnvs

[64] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," 2023.

[65] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[66] B. Xu, F. Gao, C. Yu, R. Zhang, Y. Wu, and Y. Wang, "Omnidrones: An efficient and flexible platform for reinforcement learning in drone control," 2023.

[67] "Github of rl-games." [Online]. Available: https://github.com/Denys88/rl_games

[68] "Github of rsl-rl." [Online]. Available: https://github.com/leggedrobotics/rsl_rl

[69] "Web site of skrl." [Online]. Available: https://skrl.readthedocs.io/en/latest/

[70] "Web site of nvidia isaac lab (drl library comparison)." [Online]. Available: https://isaac-sim.github.io/IsaacLab/main/source/overview/reinforcement-learning/rl_frameworks.html

[71] “Web site of the sellers of iris in uk.” [Online]. Available: https://www.arducopter.co.uk/iris-quadcopter-uav.html

[72] C. Gabellieri and A. Franchi, “Lecture notes: Control for uavs 2022-2023,” Faculty of Electrical Engineering, Mathematics Computer Science, University of Twente, May 2023.

[73] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[74] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” 2015.

[75] “Ppo explanation in openai website.” [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[76] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010.

Appendix A

State of the art summary table

Reference	Approach	Candidate Actions/ Action Space (AS)	Cost Function/ Reward	Sensors/ Observation Space (OS)	Environment Map/ SLAM	Validation	Stopping Criteria	Robot	Code Available
Yamauchi, B [12]	Geometric	Frontier-based goal selection	Distance-based	Laser, sonar, infrared	Occupancy grid-map	Real	No candidate frontier poses	Wheeled robot	×
Jingjing et al. [13]	Geometric	Frontier-based goal selection	IT-based: Joint Entropy, Expected Map Mean & KLD	Laser	Occupancy grid-map (Particle SLAM)	Sim (MobileSim)	No candidate frontier poses	-	×
Trivun et al. [14]	Geometric	Frontier-based goal selection	IT-based: map-segment covariance	Laser	Occupancy grid-map (Particle SLAM)	Sim (Stage) & Real	No candidate frontier poses	Wheeled robot	×
Placed et al [15]	Geometric	Frontier-based goal selection	TOED-based: D-optimality	Lidars	Occupancy grid-map (Graph SLAM)	Sim (Gazebo)	No candidate frontier poses	Wheeled robot	✓
Muhammad Farhan et al [16]	Geometric	Frontier-based goal selection	IT & TOED hybrid: D-optimality & Path entropy	Lidars	Occupancy grid-map (Graph SLAM)	Sim (Gazebo)	No candidate frontier poses	Turtlebot	×
Beipeng et al. [20]	Geometric	Candidate goals in reachable area	IT-based: Landmark entropy reduction	Laser	TFG	Sim (Gazebo) & Real	-	Turtlebot	×

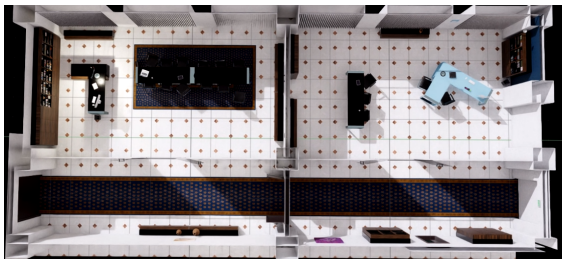
Reference	Approach	Candidate Actions/ Action Space (AS)	Cost Function/ Reward	Sensors/ Observation Space(OS)	Environment Map/ SLAM	Validation	Stopping Criteria	Robot	Code Available
Vallvé et al. [21]	Geometric	RRT* path	IT-based: Joint map & pose Entropy change/distance	Laser	Occupancy grid-map (Pose SLAM)	Sim	-	-	×
Placed et al [6]	DRL (DQN)	AS: Discrete (Go forward, Turn Left, Turn Right)	Collision avoidance term + Uncertainty term (TOED-based: D-optimality)	OS: Laser readings	Occupancy grid-map (Particle SLAM)	Sim (Gazebo)	Collision/ Fixed steps	Turtlebot	✓
Fanfei et al [24]	DRL (DQN)	AS: Discrete (Quasi-random poses within a fixed radius of the robot)	Collision avoidance term + Uncertainty term (IT-based: MI gain)	Range sensor / OS: Local map estimate	Occupancy grid-map	Sim	Map entropy = 0	-	✓
Fanfei et al [23]	DRL (DQN/ A2C)	AS: Discrete (Graph frontier nodes)	Uncertainty term (TOED-based: A-optimality) + Distance penalty term	Range sensor/ OS: Pose graph	Pose Graph (Graph SLAM)	Sim	map % covered	-	✓
Farzad et al [25]	DRL (A2C)	AS: Discrete (candidate frontier poses)	Uncertainty term (IT-based: MI gain) + Distance penalty term	Laser/ OS: map, robot pose, frontier poses	Occupancy grid-map (Particle SLAM)	Sim (Stage) & Real	-	Turtlebot	×
H. Li et al [26]	DRL (DQN)	AS: Discrete (uniformly sampled poses in map)	Uncertainty term (IT-based: Entropy gain) + Distance penalty term + Collision penalty term	LiDAR/ OS: Partial map + robot trajectory	Occupancy grid-map (Karto SLAM)	Sim(Stage)	map % covered	-	×
Botteghi, et al [27]	DRL (DDPG)	AS: continuous linear & angular velocity setpoints	Curiosity term (go far from visited poses) + Collision penalty term + Complete map term	OS: Lidar readings, root pose, % covered map, time left, previous action	Occupancy grid-map (Particle SLAM)	Sim (Gazebo)	% covered map or fixed steps	-	×
Pathak, et al [28]	DRL (A3C)	AS: Discrete (move forward, left, right, stop / 14 $\neq$ Nintendo button presses)	Curiosity term (error in next feature prediction)	Game display	-	Sim (VizDoom, Mario Bros Game)	Fixed steps, Game end	-	✓
Bai et al [29]	Supervised DL	AS: Discrete (36 poses around the robot, excluding non-known free space)	IT-based Loss function: MI gain	Range sensor / OS: Local map	Occupancy grid-map	Testing dataset of generated maps	Dead end reached	-	×

Reference	**Approach**	**Candidate Actions/ Action Space (AS)**	**Cost Function/ Reward**	**Sensors/ Observation Space(OS)**	**Environment Map/ SLAM**	**Validation**	**Stopping Criteria**	**Robot**	**Code Available**
Tao Chen et al [8]	DRL (PPO)	AS: Discrete (move forward, backward, left or right; turn left or right)	Collision penalty + Coverage term	RGB-D Camera, bump sensor/ OS: fine and coarse local maps, RGB image	Occupancy grid-map	Sim (House3D)	Fixed steps	Mobile robot	✓
Chaplot et al [7]	DRL (PPO)	AS: long-term goal	Coverage term	RGB Camera, pose sensor/ OS: map, robot pose estimates	Probabilistic maps (Neural SLAM)	Sim (Habitat)	Fixed steps	Mobile robot	✓
Yu Wu et al [10]	DRL (PPO)	AS: long-term goal	Coverage term	RGB Camera, pose sensor/ OS: map, robot pose estimates	Probabilistic maps (Neural SLAM)	Sim (Habitat)	Fixed steps	Mobile robots	×
Shengmin et al [31]	DRL (DDPG)	AS: Continuous (linear velocity $\in$ [0,1] to go forward, angular velocity $\in$ [-1,1] to turn	Complete map term+ Collision penalty + Coverage term + Curiosity term	LiDAR, IMU/ OS: Map, LiDAR data, previous action	Occupancy grid-map (Karto SLAM)	Sim (Gazebo)	Collision, fixed steps	Turtlebot	×
Wensong et al [32]	DRL (PPO)	AS: Discrete (long-term goal position)	Coverage term + SLAM Accuracy term + Loop closure term	RGB-D Camera/ OS: Global and Local map, robot yaw	Occupancy grid-map (ORB-SLAM2)	Sim (Habitat simulator) & Real	Fixed steps	Wheeled robot	×

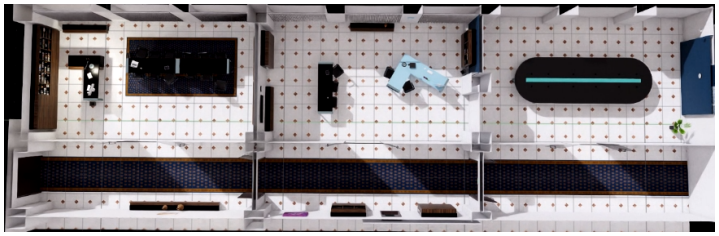
Appendix B

Available Simulation Scenes

This appendix shows images of the office scenes created in IsaacLab. Starting from an existing, unique office environment (3.5a), sections were cropped to form distinct scenes. For visualization purposes, the ceiling, lights, and other details suspended from the roof have been removed. Additionally, all windows from the original map have been covered, and the walls fully enclose the scenes, with no open doors or glass panels, ensuring a sealed scene.



(a) Scene A (small). Corridor with two rooms.



(b) Scene A (big). Corridor with three rooms.

Figure B.1: Scene A

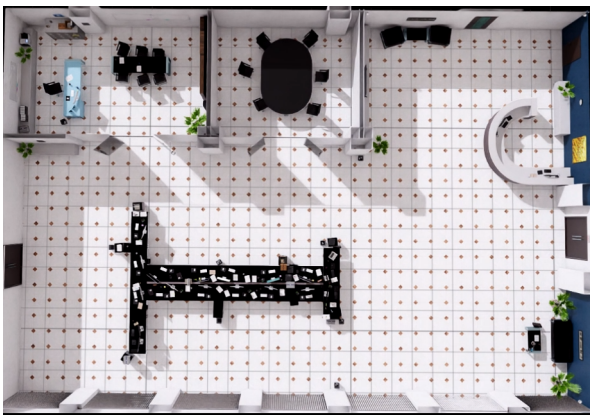
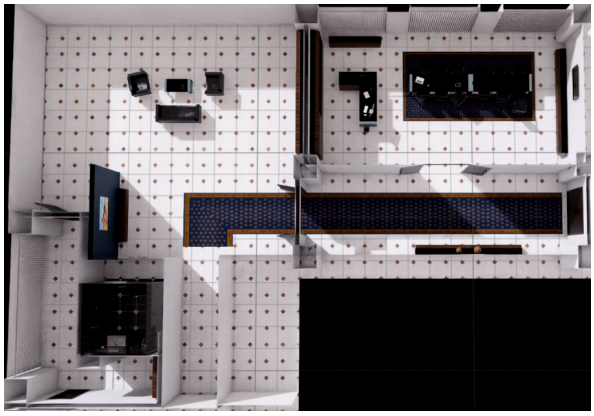
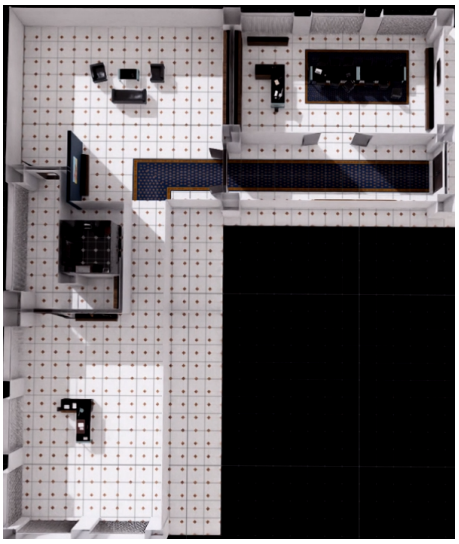


Figure B.2: Scene B. Big room with two adjacent small rooms.



(a) Scene C (small). L-shaped scene with 3 corridors and 3 rooms.



(b) Scene C (big). L-shaped scene with 3 corridors and 4 rooms.

Figure B.3: Scene C

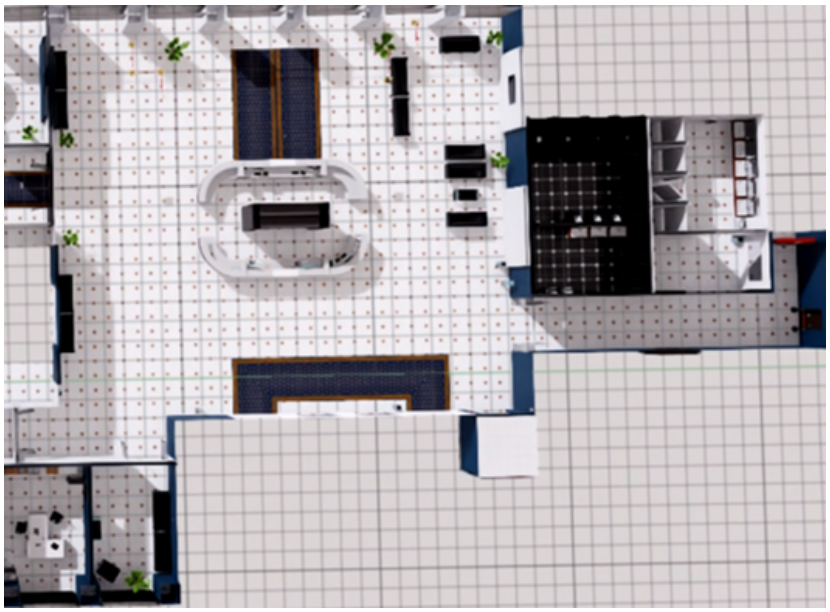


Figure B.4: Scene D. Big room with two adjacent small rooms.