Deep Reinforcement Learning for Autonomous Satellite Guidance and Control

Abdulla Tammam

Supervisor: Prof. Nabil Aouf

This dissertation is submitted for the degree of
Doctor of Philosophy

City St George's, University of London

Department of Engineering, School of Science & Technology

October 2025

Abstract

As autonomous satellite operations such as docking, inspection, and debris removal become more common, there is a growing demand for onboard control systems that can operate reliably under uncertainty, intermittent communication, and limited opportunities for ground intervention. Traditional model-based control methods have demonstrated strong performance in well-characterised environments, but can face challenges when extended to close-proximity operations involving modelling errors, unmodelled disturbances, degraded sensing or actuation, and nonlinear operational constraints. This thesis aims to develop and evaluate deep reinforcement learning (DRL) control architectures for autonomous six-degree-of-freedom (6-DoF) spacecraft close-proximity operations that are robust, scalable, and suitable for real-time deployment.

The thesis develops a suite of DRL-based controllers for 6-DoF guidance and control tasks, progressing from architectural baselines to safety, sparse-reward learning, and uncertainty-aware decision making. A modular control design is adopted throughout, decoupling translational and rotational control to improve training stability and reduce interference between objectives. This design choice is first evaluated through a comparative study of centralised and decentralised TD3-based controllers for a 1U CubeSat approaching and aligning with a passive, non-cooperative target. Across 100 randomised trials, the decentralised controller achieved more stable and precise behaviour, producing lower position and orientation errors, reduced control effort, and more consistent convergence than the centralised formulation.

Building on this baseline, the thesis introduces a hybrid safe-learning framework that combines adaptive domain randomisation (ADR) with relaxed control barrier functions (CBFs) to improve robustness under uncertainty while maintaining operational constraints during both training and deployment. In a 6-DoF rendezvous task with a progressive actuator and sensor degradation cascade, the Baseline controller diverged rapidly, with position error exceeding 45m and reaching up to 1000m across

trials. In contrast, the Safe controller consistently bounded position error within 5 m and maintained a controlled attitude response, while also remaining within the $0.5\mathrm{m\,s^{-1}}$ linear velocity constraint under nominal conditions, at the cost of slower positional convergence.

To address sparse rewards and long-horizon planning without dense, hand-crafted reward shaping, a goal-conditioned hierarchical deep reinforcement learning (HDRL) framework is developed. The controller decomposes control into high-level subgoal generation and low-level actuation, enabling learning from binary terminal rewards using hindsight experience replay and subgoal relabelling. In simulation, across 100 Monte Carlo trials, the hierarchical controller achieved mission-compliant terminal accuracy with a final position error of approximately 2.75 m and final orientation error below 1°, comparable to PID baseline performance with similar peak translational speeds. Real-time feasibility is evaluated using a custom hardware-in-the-loop (HIL) testbed combining a reaction-wheel-actuated CubeSat mock-up on a spherical air-bearing with a 6-DoF robotic arm for translational emulation. Both HDRL and TD3-based controllers maintained closed-loop stability under real-time sensing, actuation, and communication conditions, reducing position error from approximately 110 m to below 5 m and regulating attitude within 5°, while the PID controller was unable to stabilise the vehicle in the same HIL setting.

Finally, the thesis addresses decision making under uncertainty through an uncertainty-aware distributional RL architecture based on a novel Uncertainty-Aware Implicit Quantile Network (UA-IQN). By modelling the full return distribution and using variance-penalised action selection, UA-IQN supports risk-sensitive control by favouring actions with more predictable outcomes. Applied to a 6-DoF final-approach docking task, UA-IQN achieved task success comparable to a baseline decentralised TD3 controller, with sub-centimetre lateral accuracy at the docking threshold (mean off-track 0.004 m, mean cross-track 0.006 m) and sub-degree attitude errors at docking (mean roll/pitch/yaw 0.263°/0.175°/0.277°), while exhibiting more consistent and conservative translational approach behaviour across diverse initial conditions in simulation and HIL evaluation.

Together, these results demonstrate that modular DRL architectures can provide accurate spacecraft guidance and control while improving robustness to uncertainty, degraded sensing and actuation, and operational constraints, with validation in simulation and hardware-in-the-loop. By integrating architectural modularity, safety mechanisms, sparse-reward learning, and uncertainty-aware decision making, this thesis

provides a pathway towards deployable DRL-based control for autonomous spacecraft close-proximity operations.

Acknowledgements

First, I would like to express my sincere thanks to my supervisor, Professor Nabil Aouf, for his mentorship, insight, and steady guidance throughout this thesis. His thoughtful feedback challenged and refined my thinking, helping shape the direction of this work. I have learned a great deal from his approach to research and problem-solving, which will continue to influence how I think and work moving forward.

I am grateful to the team at the Robotics, Autonomy and Machine Intelligence Group at City, St George's for their camaraderie and collaboration throughout this journey, offering not only motivation but also good company when it was most needed. Special thanks to Dr. Maxwell Hogan and Dr. Ziwei Wang for their valuable input and technical support during the final stage of the project.

I would also like to recognise my friends and family back home, especially my parents, Adam and Najla, for their belief in me and unwavering support throughout the PhD. Their confidence in me has been a constant source of motivation. I am grateful to my grandmother, Donia, for her lifelong encouragement, for always standing behind me, and for the wisdom and comfort she offered whenever I needed it, and to my aunt, Dr. Thaira, whose example first made a PhD feel possible, and whose genuine enthusiasm for my progress meant a great deal. I would also like to thank Youssef, whose friendship, perspective, and well-timed distractions helped me keep my balance.

Finally, I owe special thanks to my wife, Rubi, for walking beside me through every step of this journey and being my greatest supporter. Her endless love, patience, and encouragement made this work possible.

Contents

List of Tables

List of Figures

Abbreviations

6-DoF	Six-degree-of-freedom
ADCS	Attitude Determination and Control System
ADR	Adaptive domain randomisation
AI	Artificial Intelligence
ANNs	Artificial Neural Networks
AOCS	Attitude and Orbit Control System
CBFs	Control barrier functions
CMGs	Control moment gyroscopes
CW	Clohessy-Wiltshire equations
DCM	Direction cosine matrix
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DPG	Deterministic Policy Gradient
DRL	Deep reinforcement learning
ECI	Earth-Centered Inertial
FQF	Fully Parameterized Quantile Function
FTC	Fault Tolerant Control
GEO	Geostationary Earth orbit
GNC	Guidance, Navigation, and Control
HAC	Hierarchical Actor-Critic
HAT	Hindsight Action Transitions
HDRL	Hierarchical reinforcement learning
HER	Hindsight Experience Replay
HIL	Hardware-in-the-loop
IQN	Implicit Quantile Networks
ISAM	In-Orbit Servicing, Assembly and Manufacturing
LEO	Low Earth orbit

LQR	Linear-quadratic regulators
LTI	Linear time-invariant
LVLH	Local-Vertical Local-Horizontal
MDP	Markov Decision Process
MIB	Minimum impulse bit
ML	Machine Learning
MPC	Model predictive control
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
PPO	Proximal Policy Optimization
PWM	Pulse-width modulation
QP	Quadratic Program
QR-DQN	Quantile Regression Deep Q-Network
ReLU	Rectified Linear Unit
RL	Reinforcement learning
ROS	Robotic Operating System
RSI	Robot Sensor Interface
SGD	Stochastic Gradient Descent
SMC	Sliding Mode Control
TD	Temporal-Difference learning
TD3	Twin Delayed DDPG
UA-IQN	Uncertainty-Aware Implicit Quantile Network
UADRL	Uncertainty-aware deep reinforcement learning
URDF	Unified Robot Description Format
UVFA	Universal Value Function Approximator

Chapter 1

Introduction

1.1 Motivation

The role of onboard autonomy is expanding in modern space missions, especially in applications involving close-proximity operations, defined here as operations conducted at relative ranges of less than 100 m, such as on-orbit servicing, active debris removal, inspection, and docking. These tasks require satellite to operate at short range with high precision, intermittent communication and limited opportunities for ground intervention. As close-proximity missions become more frequent and complex there is an increasing demand for satellite systems capable of making reliable, time-sensitive decisions independently. Scalable autonomy is becoming essential not only to reduce operational workload but also to enable behavior that goes beyond pre-scripted automation [9].

Proximity operations involve a range of technical challenges, including accurate relative navigation, control in dynamic and partially known orbital environments, and robustness to unmodeled disturbances. Developing control algorithms that can maintain performance across such conditions, while avoiding over-reliance on hand-tuned models or rigid heuristics, remains an open challenge in satellite autonomy.

Traditional control methods such as linear-quadratic regulators, model predictive control, and other optimization-based approaches have been widely studied in this context [10, 11, 12]. While effective under well-modelled conditions, these methods typically rely on accurate system dynamics and may perform poorly when confronted with model mismatches, unmodeled disturbances, or system faults. Furthermore, many such approaches are not easily adaptable to tasks involving nonlinear constraints, unmodeled dynamics, or unexpected failures without retuning or redesign. State ma-

chines or other rule-based systems can be robust in predefined scenarios but lack the flexibility to generalize to novel or unexpected situations [9].

Reinforcement learning (RL) has emerged as a data-driven alternative for control synthesis, with the potential to learn policies directly from interaction with the environment. In particular, deep reinforcement learning (DRL) has shown success in solving complex, high-dimensional control problems in robotics, multi-agent systems, and autonomous vehicles [13, 14, 15]. Its potential application to space systems lies in its ability to operate without relying on explicit system models, adapt through experience, and optimise behavior. Despite this potential, the adoption of DRL for satellite GNC remains limited. Concerns about reliability, interpretability, and generalisation beyond the training environment have constrained the transition from research to application.

To date, a limited number of studies have applied DRL methods to satellite control problems [9]. Among these, only a small subset explores advanced DRL techniques that address key challenges such as uncertainty, degraded sensing or actuation, or limited model fidelity [16, 17]. Hardware-in-the-loop (HIL) testing remains absent in this domain, leaving most approaches unvalidated under realistic sensing and actuation conditions.

This thesis addresses these gaps by developing and evaluating a set of DRL-based control architectures tailored to the requirements of autonomous six-degree-of-freedom (6-DoF) spacecraft proximity operations. Because a free-flying spacecraft must control both relative position and attitude, proximity operations are inherently 6-DoF, and the thesis adopts this formulation throughout.

The aim of this thesis is to develop and evaluate deep reinforcement learning control architectures for autonomous 6-DoF spacecraft close-proximity operations that are robust, scalable, and suitable for real-time deployment. To achieve this, the objectives are to:

- Develop learning-based control architectures that achieve accurate, reliable 6-DoF spacecraft guidance and control.
- Ensure controllers maintain performance under uncertainty, modelling errors, disturbances, and degraded sensing or actuation.
- Maintain safe behaviour by respecting operational constraints throughout execution.

- Produce architectures that are computationally efficient, scalable, and suitable for real-time implementation.

1.2 Thesis Outline

This thesis is organized into four technical chapters following the introduction and background, each building on the previous to progressively develop, evaluate, and validate a suite of deep reinforcement learning architectures for autonomous satellite proximity operations. What follows is a chapter-wise overview of the thesis structure and contributions.

Chapter 2 establishes the technical foundation necessary for understanding the rest of the work. It begins with a high-level overview of reinforcement learning, covering core principles, and key algorithms that underpin modern deep reinforcement learning architectures. The second part of the chapter shifts focus to satellite modeling, presenting the kinematic and dynamic formulations used to simulate six-degrees-of-freedom (6-DoF) motion. It details the reference frames, coordinate systems, and models used throughout the thesis to provide a realistic simulation environment for learning and evaluation.

Chapter 3 presents a comparative study of two control architectures: Decentralised TD3 (D-TD3) and Centralised TD3 (C-TD3). These controllers are tasked with guiding a 1U CubeSat to approach and align with a passive target. This chapter establishes a performance baseline for learning-based satellite control and examines the trade-offs between decentralised and centralised control spaces. The results demonstrate that even standard DRL methods can achieve reliable performance in satellite guidance and control, serving as a strong foundation for the more advanced methods developed in subsequent chapters.

Chapter 4 expands the control architecture to address fault tolerance and safety, critical requirements for autonomous satellite operations. It introduces a novel DRL framework that combines adaptive domain randomisation (ADR) with relaxed control barrier functions (CBFs). This hybrid system is designed to train policies that generalise across uncertain conditions while embedding safety constraints directly into the learning process. Three controller variants (Baseline, Adaptive, and Safe) are trained and evaluated in a 6-DoF CubeSat rendezvous task under nominal and degraded actuator/sensor conditions. The results show significant improvements in robustness, highlighting the feasibility of DRL in fault-prone environments.

Chapter 5 tackles the challenge of sparse rewards and long-horizon planning through a goal-conditioned hierarchical reinforcement learning (HDRL) framework. Departing from prior approaches that rely on dense, hand-crafted reward functions, this chapter proposes a layered control structure that decomposes the guidance and control problem into high-level goal generation and low-level actuation. The architecture leverages hindsight experience replay and subgoal relabelling to learn from binary terminal rewards. The controller is validated in simulation and compared against flat DRL and PID baselines, showing competitive performance. Additionally, a modular hardware-in-the-loop (HIL) testbed was developed to further evaluate real-world feasibility, integrating a spherical air-bearing for rotational motion with a 6-DoF tabletop robotic arm for translation.

Chapter 6 addresses decision-making under uncertainty through the development of an uncertainty-aware reinforcement learning architecture based on distributional RL. A novel Uncertainty-Aware Implicit Quantile Network (UA-IQN) framework is introduced, enabling risk-sensitive action selection by modeling the full distribution of expected returns and using return variance to guide decisions. This architecture is applied to a 6-DoF final-approach for docking task, where it demonstrates stable and performant behavior across a range of initial conditions. Real-time performance was compared against a baseline DRL controller and validated using a state-of-the-art hardware-in-the-loop testing facility.

Chapter 7 concludes the thesis by discussing key insights, architectural trade-offs, limitations of the proposed methods, and practical considerations for deployment within real-world satellite systems. The chapter also outlines several promising directions for future work.

1.3 Original Contributions

The central goal of this thesis is to develop learning-based satellite control architectures that are modular, robust, and deployable, capable of operating under uncertainty, with minimal supervision, and within real-world safety constraints. This work contributes to the field of autonomous space systems by addressing four key research directions: modular control architecture design, safety and robustness in reinforcement learning, sparse-reward learning via hierarchical structures, and risk-aware decision-making through uncertainty modeling. The contributions are summarized below.

1. **Modular Learning Architectures for 6-DoF satellite Control** This work

proposes a structured design methodology for building deep reinforcement learning (DRL) controllers tailored to the 6-DoF dynamics of satellite. In contrast to conventional DRL systems that learn monolithic policies over the entire action space, this thesis investigates modular architectures that separate translational and rotational control into distinct agents. This decomposition mirrors the structure of traditional GNC pipelines while introducing the flexibility of learning-based control.

Through systematic evaluation, it is shown that modular controllers improve training stability and operational robustness, particularly under system constraints. A comparative study between decentralised and centralised variants of TD3 reveals how control architecture directly impacts learning efficiency and performance in high-dimensional, multi-objective environments. These findings contribute to a more principled understanding of how modularity in RL architectures can enhance policy interpretability, reliability, and integration into real-world systems.

2. **Safe Reinforcement Learning under Uncertainty and Degradation** Another major contribution is a hybrid control framework that embeds safety and robustness into the training loop of reinforcement learning agents. By integrating control barrier functions (CBFs) with adaptive domain randomisation (ADR), the thesis presents a method for shaping policy learning around constraint satisfaction rather than filtering unsafe behavior post hoc. This creates agents that inherently respect physical safety boundaries during operation.

   In addition, a structured degradation testing pipeline is introduced to evaluate controller resilience under a spectrum of actuator and sensor faults. This includes failure scenarios typically omitted from standard benchmarks but critical for spaceflight. The result is a reproducible framework for assessing fault tolerance in DRL-based control, bridging the gap between theoretical learning algorithms and the high-assurance demands of real-world space systems.

3. **Learning from Sparse Rewards via Hierarchical Abstraction** To address the limitations of DRL in sparse-reward, long-horizon tasks, the thesis introduces a goal-conditioned hierarchical reinforcement learning (HRL) framework. This architecture incorporates temporal abstraction, subgoal discovery, and hindsight relabelling to enable learning in environments with minimal feedback — a common condition in autonomous space missions.

The approach eliminates the need for dense, hand-engineered reward signals and instead learns complex control behaviors from binary success indicators. This is achieved through a layered decision-making structure that divides the task into high-level goal setting and low-level actuation. The practical viability of this system is demonstrated through deployment on a hardware-in-the-loop testbed, showing that HRL policies can operate in real-time on embedded satellite platforms. To the best of the author's knowledge, this constitutes the first hardware-in-the-loop validation of a novel hierarchical deep reinforcement learning guidance and control framework reported in the literature.

4. **Risk-Sensitive Control via Distributional Reinforcement Learning** Finally, this thesis extends the capabilities of DRL controllers by incorporating uncertainty modeling directly into the decision-making process. Standard RL optimizes for expected return without accounting for variance in outcomes — a limitation in safety-critical domains. To address this, a novel Uncertainty-Aware Implicit Quantile Network (UA-IQN) is developed, enabling agents to reason over full return distributions.

   By modeling both the mean and variance of expected outcomes, the UA-IQN architecture supports risk-sensitive control policies that adapt their behavior based on confidence and uncertainty. This advancement makes it possible to tune decision-making behavior under risk and enables a richer notion of policy robustness. The integration of distributional RL into the satellite guidance and control loop represents a novel step toward safer, more interpretable learning-based autonomy. This work further presents, for the first time in literature, a hardware-in-the-loop evaluation of an uncertainty-aware distributional DRL controller, demonstrating its feasibility and robustness under realistic on-board execution conditions.

In summary, this thesis contributes a set of learning-based control architectures that advance the state of the art in autonomous satellite systems. These methods push DRL closer to operational viability by introducing modularity, embedding safety, enabling sparse-reward learning, and modeling uncertainty at execution time. While the focus is on satellite control, the underlying techniques generalize to a broader class of autonomous systems that demand high reliability and robustness in complex, uncertain environments.

1.4 Publications

The following publications have resulted from the work presented in this thesis.

1. A. Tammam, A. Zenati, and N. Aouf, "Deep reinforcement learning for rendezvous and attitude control of cubesat class satellite," in *2023 7th International Conference on Automation, Control and Robots (ICACR)*, pp. 140–146, 2023

2. A. Tammam and N. Aouf, "Hierarchical deep reinforcement learning for cubesat guidance and control," *Control Engineering Practice, an IFAC Journal*, vol. 156, p. 106213, 2025

3. A. Tammam and N. Aouf, "Safe spacecraft guidance using adaptive domain randomization and relaxed control barrier functions," in *2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2025. To appear

4. A. Tammam and N. Aouf, "Uncertainty-aware distributional reinforcement learning for satellite final-approach and rendezvous," *Submitted to IEEE Transactions on Aerospace and Electronic Systems*, 2025

Chapter 2

Background

2.1 Machine Learning and Reinforcement Learning

2.1.1 Introduction to Machine Learning and Neural Networks

Foundations of Machine Learning

Artificial Intelligence (AI) is the broad field of building machines or software that exhibit intelligent behavior, performing tasks that typically require human intelligence. This encompasses capabilities such as reasoning, problem-solving, perception, and natural language understanding [22]. Machine Learning (ML) is a core subfield of AI that focuses on algorithms enabling computers to learn from data and experience. A widely cited definition describes ML as the study of algorithms that "improve their performance P at some task T with experience E" [23].

Early AI research in the 1950s-1970s was dominated by symbolic AI and expert systems, where humans encoded knowledge as logical rules. While effective in constrained domains, these systems lacked adaptability. In the 1990s, statistical machine learning emerged as a more flexible, data-driven approach, using methods such as decision trees and support vector machines. The late 2000s and 2010s witnessed the deep learning revolution [24], where deep neural networks began outperforming existing ML models in computer vision and speech recognition, driven by improved training algorithms, large datasets, and GPU acceleration [25].

Since 2015, progress has accelerated with the development of transformer-based architectures [26], large-scale pretraining [27], and foundation models capable of integrating multimodal inputs. Current research emphasizes scalability, robustness, and safe deployment across domains such as scientific discovery, creative generation, and

autonomous control.

Machine Learning Paradigms and the Learning Pipeline

Machine learning tasks are typically categorized by the form of supervision available and the structure of the learning problem. The four major paradigms, supervised, unsupervised, self-supervised, and reinforcement learning, differ in how the learning signal is derived and what objective is pursued.

Supervised learning uses labeled data, consisting of input–output pairs $(\mathbf{x}, \mathbf{y})$, to train models that predict outputs given new inputs. The objective is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that minimizes prediction error on unseen data. This paradigm dominates most applied ML systems, with widespread use in classification (e.g., image recognition, disease diagnosis) and regression (e.g., financial forecasting, sensor calibration).

Unsupervised learning operates on unlabeled datasets $\{\mathbf{x}_i\}$ and seeks to uncover hidden patterns or intrinsic structure. Common objectives include clustering similar samples (e.g., customer segmentation), reducing dimensionality (e.g., via PCA or autoencoders), or estimating data distributions. Since no labels are provided, evaluation is often task dependent and qualitative.

Self-supervised learning constructs surrogate supervision signals directly from the input data. For example, a model might be trained to predict missing parts of an input (e.g., masked language modeling) or determine whether two views of the same data point correspond (contrastive learning). These proxy tasks enable the learning of transferable representations, especially valuable in settings where labeled data is scarce.

Reinforcement learning (RL) differs fundamentally: it involves an agent that learns by interacting with an environment. At each time step, the agent observes a state, takes an action, and receives a scalar reward. Learning proceeds by trial and error to maximize long-term reward. RL has achieved state-of-the-art performance in domains such as robotic control (e.g., motor skills [28]), video games (e.g., Atari, StarCraft II [29, 30]), and autonomous driving (e.g., lane control, policy learning [15]).

Across all paradigms, the typical ML pipeline involves five key stages:

1. Data Acquisition and Preprocessing: Any ML project begins with gathering relevant data. The data is preprocessed to improve quality: e.g. removing noise and outliers, handling missing values, normalizing or scaling features, and

splitting into training, validation, and test sets.

2. Feature Extraction and Engineering: In traditional ML, substantial effort is devoted to designing informative features from raw data. Features are measurable attributes used as model inputs. In contrast, deep learning automates feature discovery by learning hierarchical representations directly from raw inputs.

3. Model Selection and Architecture: Once data are prepared, an appropriate model type (e.g., linear regression, decision tree, or neural network) is selected. For neural networks, this involves choosing an architecture (number of layers, units, and connections) suited to the task.

4. Training and Validation: The chosen model is then trained on the training dataset by adjusting its parameters to fit the data. Most ML algorithms define a loss or cost function that quantifies the error between the model's predictions and the true targets. Training is an optimization process to minimize this loss.

5. Testing and Deployment: After training the final model is evaluated on an independent test set to estimate its true performance. If satisfactory, the model can be deployed for use on new data.

This pipeline underpins both classical and modern machine learning methods, with neural networks playing an increasingly central role in complex domains.

Neural Networks: Structure, Training, and Applications

Artificial Neural Networks (ANNs) are a family of models loosely inspired by biological neurons, and they form the backbone of modern deep learning. At a high level, a neural network is a parameterized function $f_{\boldsymbol{\theta}}(\mathbf{x})$ that takes an input $\mathbf{x}$ and produces an output (prediction). The parameters $\boldsymbol{\theta}$ are learned from data.

The fundamental unit of a neural network is a neuron (or node) which computes a weighted sum of its inputs and then applies a nonlinear activation function. In mathematical terms, for a neuron j,

$$z_j = \sum_i w_{ji} x_i + b_j,$$

and the output is

$$a_j = \sigma(z_j),$$

where w_{ij} are weights on inputs x_i, b_j is a bias term, and $\sigma(\cdot)$ is an activation function.

Neurons are typically organized into layers where each layer's output becomes the input to the next layer. A network with many layers is called a deep network. The layered structure allows complex functions to be built up from simpler components: lower layers might learn simple features which higher layers combine into more abstract features. Fully Connected Networks, also called multilayer perceptrons (MLPs) have every unit in a layer connected to every unit in the next.

Nonlinear activation functions enable neural networks to approximate complex functions. Modern networks often use the Rectified Linear Unit (ReLU), defined as

$$\text{ReLU}(z) = \max(0, z),$$

which is simple and effective at mitigating vanishing gradients in deep networks [31]. ReLU units output 0 for negative inputs and are linear for positive inputs, promoting sparse activation and easier optimization.

To learn the parameters of a neural network, we define a loss function $L(\boldsymbol{\theta})$ that quantifies the error between the network's predictions and the true targets on the training data. Training the network means adjusting $\boldsymbol{\theta}$ to minimize $L(\boldsymbol{\theta})$.

This is done via gradient-based optimization. The primary algorithm is backpropagation [32] combined with gradient descent. Backpropagation efficiently computes the gradient of the loss with respect to every weight in the network by recursively applying the chain rule from the output layer back to the inputs. Given the gradients, an optimizer like Stochastic Gradient Descent (which in practice means updating weights incrementally using the gradient on mini-batches of data) is used. Advanced optimizers such as SGD with momentum, RMSProp, or Adam [33] adapt the learning rates during training and generally converge faster.

Neural networks are universal function approximators: with sufficient width, even shallow networks can approximate any continuous function on compact domains [34]. In practice, deep architectures enable efficient approximation of highly structured, high-dimensional mappings. Their ability to learn directly from raw inputs with minimal human-designed features makes them central to modern AI. Neural networks power state-of-the-art systems in image recognition, speech translation, robotics, and game playing, and they serve as the foundation for deep reinforcement learning, as discussed in later sections.

2.1.2 Reinforcement Learning Foundations

Learning Through Interaction

Reinforcement Learning (RL) is a learning paradigm centered on goal-directed behavior through interaction with an environment [1]. The core idea is often summarized as learning by trial-and-error. An RL agent must discover actions that yield high cumulative rewards by experimenting in its environment. Each action affects subsequent states and rewards, creating a feedback loop that guides learning.

This learning process mimics how humans and animals learn many behaviors: through experience, by exploring different actions and observing the outcomes (rewards or punishments). Over time, beneficial actions are reinforced. In RL, the feedback signal is the reward: a scalar value that the environment provides to the agent to indicate the immediate desirability of the agent's last action or state. The agent's objective is to maximize the long-term sum of rewards (the return). Because rewards may be delayed or sparse, the agent faces a credit assignment problem: determining which past actions were responsible for obtaining a reward.

A useful contrast is with supervised learning. In supervised learning, each training example provides the correct answer (e.g. a label), and the learning algorithm adjusts the model to better produce those answers. In reinforcement learning, there are no explicit correct answers for each situation. There is only a reward signal, which might not explicitly tell the agent what the best action was, only how good the outcome turned out. Moreover, rewards can be delayed: an agent might make a sequence of many actions with no reward until the end, when it finally gets a positive reward, that reward must be credited appropriately to those actions that truly contributed. This delayed and evaluative feedback makes RL problems particularly challenging and interesting from a learning perspective.

Another key challenge in RL is the exploration-exploitation dilemma: an agent must exploit what it has learned (choose actions it believes yield high reward) but it also must explore new actions to discover if they might lead to even higher rewards in the long run. Exclusively exploiting can cause getting stuck in suboptimal behavior; too much exploration wastes time on suboptimal actions. Balancing these is a central issue in RL algorithms.

The Markov Decision Process

The standard framework for reinforcement learning is the Markov Decision Process (MDP). In this formalism, an agent interacts with an environment over discrete time steps $t = 0, 1, 2, \ldots$. At each step:

- The agent observes the current state $S_t \in \mathcal{S}$.
- It selects an action $A_t \in \mathcal{A}$ according to its policy.
- The environment transitions to a new state $S_{t+1} \in \mathcal{S}$ and emits a reward $R_{t+1} \in \mathbb{R}$.

This interaction generates a trajectory $(S_0, A_0, R_1, S_1, A_1, R_2, \ldots)$ and continues either indefinitely or until a terminal state is reached in episodic tasks.

The agent's behavior is governed by a policy $\pi(a \mid s)$. The goal of reinforcement learning is to find an optimal policy π^* that maximizes the agent's expected long-term return.

Usually, we consider tasks in which the agent eventually terminates an episode. An episode is a sequence from a starting state to some terminal state (for example, one game from start to finish). Episodic tasks naturally reset to a start after ending.

The agent–environment interaction assumes the Markov property: the environment's dynamics depend only on the current state and action, not on the full history. The process is modeled as a Markov Decision Process defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$: set of states,
- $\mathcal{A}$: set of actions,
- $P(s' \mid s, a)$: transition probability of reaching state s' from s after taking action a,
- $R(s, a)$: expected immediate reward received after taking action a in state s,
- $\gamma \in [0, 1)$: discount factor weighting future rewards relative to immediate ones.

The discount factor γ expresses the agent's preference for immediate versus future rewards. A value close to 1 encourages long-term planning, whereas a value near 0 focuses on immediate outcomes.

Figure 2.1 illustrates the canonical agent-environment loop in reinforcement learning, the interaction structure used throughout this thesis to formulate spacecraft guidance and control as sequential decision-making problems. At each time step, the agent's action affects the environment, and the environment in turn provides the agent with a new state and reward.

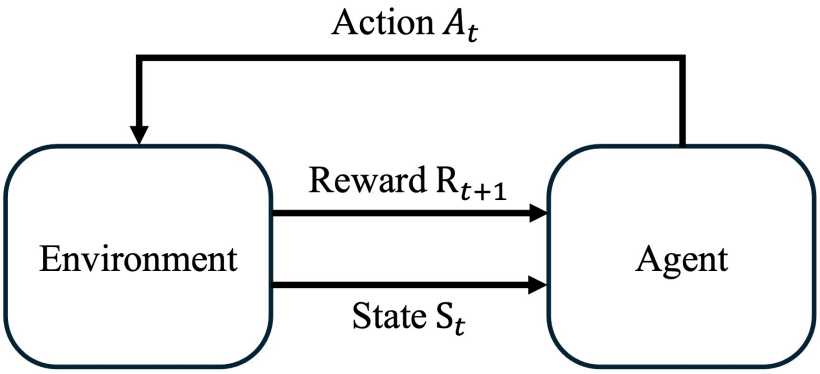


Figure 2.1: Agent–environment interaction in reinforcement learning. At time t, the agent observes state S_t, selects action A_t, receives reward R_{t+1}, and transitions to state S_{t+1}. The agent uses this feedback to update its policy to maximize long-term return [1].

Value Functions and Bellman Equations

The objective in a Markov Decision Process is to maximize the expected return, defined as the total discounted sum of rewards that an agent receives over time. For an episode starting at time t, the return is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.1}$$

Value functions estimate how good it is for the agent to be in a given state or to take a specific action in that state under a particular policy π. The state-value function (or simply, value function) under policy π is:

$$V^\pi(s) = \mathbb{E}_\pi[\, G_t \mid S_t = s \,], \tag{2.2}$$

which represents the expected return starting from state s and following policy π thereafter. It measures the long-term desirability of state s under π.

The action-value function (or Q-function) under policy π is:

$$Q^\pi(s,a) = \mathbb{E}_\pi[\, G_t \mid S_t = s,\, A_t = a \,], \tag{2.3}$$

which gives the expected return obtained by taking action a in state s and then following policy π.

The value functions satisfy recursive relationships known as the Bellman equations [35]. The Bellman expectation equation for the state-value function is:

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} P(s' \mid s,a)\big[\, R(s,a,s') + \gamma\, V^\pi(s') \,\big]. \tag{2.4}$$

This expresses that under policy π, the value of state s equals the expected immediate reward $R(s,a,s')$ plus the discounted value of the next state s', averaged over all actions a and next states s' drawn from the policy and environment dynamics.

Similarly, the Bellman equation for the action-value function is:

$$Q^\pi(s,a) = \sum_{s'\in\mathcal{S}} P(s' \mid s,a)\big[\, R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a' \mid s') Q^\pi(s',a') \,\big]. \tag{2.5}$$

Equations (2.4) and (2.5) are linear in the value functions and have unique solutions for any fixed policy π. They form the foundation for policy evaluation algorithms in reinforcement learning.

Of particular interest are the optimal value functions, which correspond to the maximum achievable return from each state (or state-action pair), assuming the agent acts optimally thereafter. The optimal state-value function is defined as:

$$V^*(s) = \max_\pi V^\pi(s), \tag{2.6}$$

and the optimal action-value function as:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a). \tag{2.7}$$

There exists at least one policy π^* such that $V^{\pi^*}(s) = V^*(s)$ and $Q^{\pi^*}(s,a) = Q^*(s,a)$ for all states and actions; such a policy is called an optimal policy.

The optimal value functions satisfy:

$$V^*(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s' \mid s,a)\big[\, R(s,a,s') + \gamma\, V^*(s') \,\big], \tag{2.8}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \big[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \big]. \tag{2.9}$$

Intuitively, these equations state that the optimal value of a state (or state-action pair) equals the expected reward from the best possible immediate action plus the discounted value of following the optimal policy thereafter. The Bellman optimality equations are nonlinear and form the basis for dynamic programming methods such as value iteration and policy iteration [35].

Temporal Difference Learning and Q-Learning

Reinforcement learning algorithms can be organized along several key dimensions, which reflect different assumptions and design goals:

- **Model-Free vs. Model-Based:**
  - *Model-free* methods learn directly from experience without modeling the environment's dynamics.
  - *Model-based* methods explicitly learn or use a model of transition probabilities and rewards to plan or simulate outcomes. These are often more sample-efficient but harder to scale.
- **Value-Based, Policy-Based, and Actor-Critic:**
  - *Value-based* methods estimate state or action values (e.g., $V(s)$, $Q(s,a)$) and derive policies from them.
  - *Policy-based* methods directly parameterize and optimize the policy (e.g., $\pi_{\boldsymbol{\theta}}(a \mid s)$) via gradient ascent on expected return [36].
  - *Actor–Critic* methods combine both: the actor updates the policy, while the critic estimates value functions to guide learning. This hybrid structure is used in many modern algorithms such as DDPG and TD3.
- **On-Policy vs. Off-Policy:**
  - *On-policy* methods (e.g., SARSA) learn from data generated by the current policy.
  - *Off-policy* methods (e.g., Q-learning) learn about a target policy using data collected from a different (often more exploratory) behavior policy.

Many reinforcement learning algorithms aim to estimate the optimal value functions without access to a full model of the environment. Temporal-Difference (TD) learning provides a method to do this by updating value estimates based on observed transitions and previously learned estimates [37]. It incrementally improves predictions by using the difference between consecutive estimates as a learning signal, without waiting for final outcomes.

The simplest version is TD(0), where the value of the current state is updated using the immediate reward and the estimated value of the next state:

$$V(s_t) \leftarrow V(s_t) + \alpha\big[\, R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \,\big], \tag{2.10}$$

where α is the learning rate. The term in brackets is the TD error; it quantifies the discrepancy between the current estimate and the better-informed estimate using the one-step reward and next state's value.

TD methods extend naturally to action-value functions, forming the basis of algorithms such as SARSA (on-policy) and Q-learning (off-policy).

Q-learning [38] is one of the most influential off-policy TD control algorithms. It seeks to directly approximate the optimal action-value function $Q^*(s, a)$ by iteratively applying the Bellman optimality update. At each step, the agent observes a transition $(S_t, A_t, R_{t+1}, S_{t+1})$ and updates its estimate according to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[\, R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \,\big], \tag{2.11}$$

This update uses the maximum estimated future reward (over actions a') in the next state, regardless of the policy currently being followed. Hence, Q-learning is off-policy: it learns the value of the optimal policy independently of the behavior policy used to generate the data.

Over time, with sufficient exploration and under appropriate conditions (e.g., decaying learning rate, visiting all state-action pairs), Q-learning is guaranteed to converge to Q^*.

These temporal-difference methods form the foundation of many deep reinforcement learning algorithms, where function approximation (e.g., neural networks) is used to scale value estimation to large or continuous state spaces. This extension will be discussed in depth in the next section.

2.1.3 Deep Reinforcement Learning

Motivation and Challenges

Deep Reinforcement Learning (DRL) integrates the representation learning capabilities of deep neural networks with the sequential decision-making framework of reinforcement learning. This combination enables agents to operate directly on high-dimensional inputs such as images, sensor streams, or raw audio, eliminating the need for manual feature engineering. Traditional RL methods, which rely on tabular representations or shallow function approximators, fail to scale in large or continuous state spaces. Deep RL addresses this limitation by using neural networks to approximate policies or value functions, allowing agents to learn from raw data and generalize to unseen states.

The impetus for deep RL stems from several core limitations of classical RL, especially in real-world domains:

- **Curse of Dimensionality:** Many environments have exponentially large state spaces. For instance, an Atari game screen consists of over 100,000 pixels. Representing a value for each possible state is infeasible without function approximation.
- **Automatic Feature Learning:** Deep networks extract task-relevant features directly from raw observations, obviating the need for domain-specific feature engineering.
- **Generalization:** Neural networks can generalize value estimates or policies to novel states that share structural similarity with previously encountered states.

The potential of deep RL was first demonstrated in Tesauro's TD-Gammon [39], which achieved expert-level backgammon play using a neural network trained via temporal-difference learning. The field advanced substantially with the Deep Q-Network (DQN) [29], which learned to play Atari games directly from raw pixels. Subsequent breakthroughs, including AlphaGo [40] and advanced robotic control systems, have established deep RL as a powerful framework for complex, high-dimensional decision-making.

However, the integration of deep learning into RL introduces several challenges not present in supervised learning or classical RL. These arise from the interaction between bootstrapped value estimation, policy-dependent data, and the non-stationary nature of the learning signal:

- **Instability and Divergence:** The data distribution depends on the evolving policy, making training non-stationary. Small updates to the network can destabilize learning by changing future data distributions.
- **Temporal Credit Assignment:** Deep models amplify the difficulty of associating delayed rewards with the specific actions or network parameters responsible for them.
- **Sample Inefficiency:** Deep RL typically requires millions of interactions to learn competent policies, posing practical challenges in real-world applications.
- **Overestimation Bias:** In value-based methods, the use of max operations with noisy function approximators can lead to systematic overestimation of Q-values [41].
- **Hyperparameter Sensitivity:** Deep RL algorithms are notoriously sensitive to learning rates, reward scaling, and exploration parameters, with results varying significantly across runs [42].

To address these issues, various stabilization strategies have been developed. These include experience replay [29], target networks, normalization techniques [43], and entropy regularization [44]. Replay buffers mitigate correlation in data by randomly sampling past experiences. Target networks reduce non-stationarity in bootstrapped targets by decoupling action evaluation from policy updates. Additionally, reward clipping, batch normalization, and ensemble methods help control variance and improve training robustness.

Deep Q-Network (DQN)

The Deep Q-Network algorithm marked a pivotal breakthrough in deep reinforcement learning by successfully combining Q-learning with deep neural networks at scale [29]. DQN achieved human-level performance on many Atari 2600 video games using only raw pixel inputs and game scores, without access to environment dynamics or handcrafted features. Crucially, it provided the first convincing evidence that value-based RL could be stabilized using deep function approximators, unlocking the potential of RL in high-dimensional domains.

DQN is a model-free, off-policy, value-based algorithm. It uses a deep neural network $Q(s, a; \theta)$ to approximate the optimal action-value function $Q^*(s, a)$. For

environments with visual input, the network typically consists of convolutional layers followed by fully connected layers. The network takes a stack of frames representing the current state as input and outputs a scalar Q-value for each discrete action. The learning objective is to minimize the temporal difference (TD) error by adjusting the network parameters θ to approximate the Bellman target.

To overcome instability associated with nonlinear function approximation in Q-learning, DQN introduced several key innovations:

1. **Experience Replay:** Transitions (s, a, r, s') observed during interaction are stored in a replay buffer $\mathcal{D}$. During training, mini-batches of transitions are sampled uniformly at random from $\mathcal{D}$. This breaks temporal correlations between consecutive transitions and improves data efficiency by allowing past experiences to be reused.

2. **Target Network:** DQN maintains two networks: an online network with parameters $\boldsymbol{\theta}$ and a target network with parameters $\boldsymbol{\theta}^-$. The TD target is computed using the target network:

$$y_{\text{DQN}} = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-), \tag{2.12}$$

   while the loss minimized is:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(y_{\text{DQN}} - Q(s, a; \boldsymbol{\theta})\big)^2 \Big]. \tag{2.13}$$

   The target network is updated periodically by copying the weights from the online network, thereby stabilizing the targets and mitigating oscillations in learning.

3. **Network Architecture:** In the Atari implementation, the input state was composed of four consecutive grayscale frames resized to 84×84 pixels. The network consisted of three convolutional layers followed by two fully connected layers, with the final output layer predicting Q-values for the available discrete actions. This architecture learned visual features end-to-end without prior knowledge of the environment.

The impact of DQN was substantial. It achieved human-level or better performance on over half of the 49 Atari games in the benchmark suite [29], using a single

architecture and minimal domain-specific tuning. The agent learned directly from raw pixels and game scores, without any knowledge of game rules or object semantics. This generality and performance led to widespread interest in deep RL and motivated numerous extensions.

Notable extensions to DQN include:

- **Double DQN** [45]: Mitigates overestimation bias by decoupling action selection and evaluation in the TD target.
- **Prioritized Experience Replay** [46]: Samples transitions with high TD error more frequently to accelerate learning.
- **Dueling Network Architecture** [47]: Separates value and advantage estimation to improve learning efficiency.
- **Multi-Step Targets** [1]: Uses n-step returns to balance bias and variance in TD updates.
- **Distributional Q-Learning** [8]: Models the distribution over returns instead of the mean, enhancing expressiveness and uncertainty estimation.

These refinements improved DQN's sample efficiency and robustness, but the algorithm remains primarily suited for discrete action spaces. In continuous control domains, computing $\max_a Q(s, a)$ becomes intractable, motivating a shift toward policy-based and actor–critic methods.

Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm was one of the first deep reinforcement learning methods to effectively handle continuous action spaces [43]. It combines the actor-critic architecture with several key stabilization techniques adapted from DQN, allowing scalable learning in high-dimensional, continuous control domains.

DDPG is an off-policy, model-free, actor-critic algorithm built on the Deterministic Policy Gradient (DPG) theorem [48]. Unlike stochastic policy gradient methods, DDPG uses a deterministic policy $\mu(s; \theta^\mu)$ that directly maps states to actions. The corresponding policy gradient is given by:

$$\nabla_{\boldsymbol{\theta}^\mu} J \approx \mathbb{E}_{s\sim\mathcal{D}} \left[\nabla_a Q(s, a; \boldsymbol{\theta}^Q)\big|_{a=\mu(s)} \nabla_{\boldsymbol{\theta}^\mu} \mu(s; \boldsymbol{\theta}^\mu) \right], \tag{2.14}$$

where $Q(s, a; \theta^Q)$ is the critic network estimating the action-value function.

The DDPG algorithm consists of two neural networks:

- The **actor** $\mu(s; \theta^\mu)$, which outputs a continuous action for each state.
- The **critic** $Q(s, a; \theta^Q)$, which evaluates the actor's actions by estimating expected returns.

Both networks are trained using experience replay and target networks, analogous to DQN. Transitions (s, a, r, s') are stored in a replay buffer, and updates are performed on mini-batches. The critic is trained to minimize the temporal-difference error:

$$L(\boldsymbol{\theta}^Q) = \Big(r + \gamma Q(s', \mu(s'; \boldsymbol{\theta}^{\mu^-}); \boldsymbol{\theta}^{Q^-}) - Q(s, a; \boldsymbol{\theta}^Q)\Big)^2, \tag{2.15}$$

where θ^{Q^-} and θ^{μ^-} denote the parameters of slowly updated target networks.

These target networks are updated using soft updates:

$$\boldsymbol{\theta}^{Q^-} \leftarrow \tau \boldsymbol{\theta}^Q + (1 - \tau)\boldsymbol{\theta}^{Q^-}, \tag{2.16}$$

$$\boldsymbol{\theta}^{\mu^-} \leftarrow \tau \boldsymbol{\theta}^\mu + (1 - \tau)\boldsymbol{\theta}^{\mu^-}, \tag{2.17}$$

with $\tau \ll 1$ (e.g., $\tau = 0.001$), to ensure stability.

The actor is updated via the deterministic policy gradient, pushing it to produce actions that maximize the critic's estimated return. Unlike stochastic policy gradient methods, DDPG's deterministic policy requires external noise for exploration. During training, noise is added to the actor's output:

$$a_t = \mu(s_t; \theta^\mu) + \mathcal{N}_t, \tag{2.18}$$

where $\mathcal{N}_t$ is typically drawn from an Ornstein–Uhlenbeck process, producing temporally correlated noise suitable for physical control domains.

DDPG was demonstrated to solve a range of benchmark tasks in the MuJoCo and OpenAI Gym environments, including locomotion, manipulation, and pendulum swing-up [43]. It was also extended to visual domains, though learning from pixels proved significantly more difficult and data-intensive.

Despite its impact, DDPG is known to be sensitive and brittle:

- It may suffer from **Q-value overestimation**, where inaccurate critics provide overly optimistic values, misleading the actor.

- **Exploration** in high-dimensional action spaces is inefficient with Gaussian or OU noise alone.
- **Hyperparameter tuning** is critical: learning rates, target update rates, and noise parameters all strongly affect performance.

Nevertheless, DDPG introduced a framework that became the basis for subsequent advances in continuous control. Its modular use of actor-critic structure, off-policy updates, experience replay, and target networks laid the foundation for algorithms like TD3 and SAC, which improved performance and stability.

Twin Delayed DDPG (TD3)

Twin Delayed DDPG (TD3) extends the DDPG framework with targeted improvements that enhance learning stability and reduce value overestimation [49]. While retaining DDPG's core actor-critic structure with target networks, replay buffer, and off-policy learning, TD3 incorporates three techniques designed to address known sources of variance and bias in value-based deep RL.

Clipped Double Q-Learning: TD3 maintains two separate critic networks $Q_1(s, a)$ and $Q_2(s, a)$, each with its own target network. When computing the target for the Bellman update, TD3 uses the minimum of the two critics:

$$y = r + \gamma \min_{j=1,2} Q_j(s', \mu(s'; \theta^{\mu^-}); \theta^{Q_j^-}). \tag{2.19}$$

This conservative estimate reduces overestimation bias by lowering the likelihood that both critics simultaneously overestimate a given value. The approach is inspired by Double Q-learning [45] and provides more reliable training signals for the actor.

Delayed Policy Updates: Unlike DDPG, which updates the actor and critic at every step, TD3 updates the actor network and its target less frequently—typically once every two or more critic updates. This delay allows the critic to better approximate the value function before the actor adapts to it, reducing the risk of compounding approximation errors.

Target Policy Smoothing: When computing the target Q-value, TD3 perturbs the target policy's action with clipped noise:

$$a' = \mu(s'; \theta^{\mu^-}) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c). \tag{2.20}$$

This smoothing regularizes the critic by making it less sensitive to minor changes in

action values and prevents the actor from overfitting to sharp peaks in the Q-function that may arise due to approximation artifacts.

These modifications significantly improve training stability and sample efficiency. TD3 achieved strong performance across a range of continuous control benchmarks, including HalfCheetah, Hopper, and Walker2d, often exceeding DDPG in both average return and robustness across random seeds [49]. It remains a widely used baseline for continuous-action RL tasks.

TD3 retains the off-policy and model-free nature of DDPG, which makes it sample-efficient relative to on-policy methods, but potentially less efficient than model-based approaches. It does not substantially alter the exploration mechanism, still relying on action noise. The use of two critics introduces a modest computational overhead, but the trade-off is generally favorable due to the improved reliability of the value estimates.

In summary, TD3 preserves the strengths of DDPG while addressing specific sources of error through simple yet effective modifications. Its design exemplifies the iterative development of deep RL algorithms, where empirical performance is improved through careful adjustment of architectural and algorithmic components.

2.2 Dynamics and Modeling

2.2.1 Reference Frames and Coordinate Systems

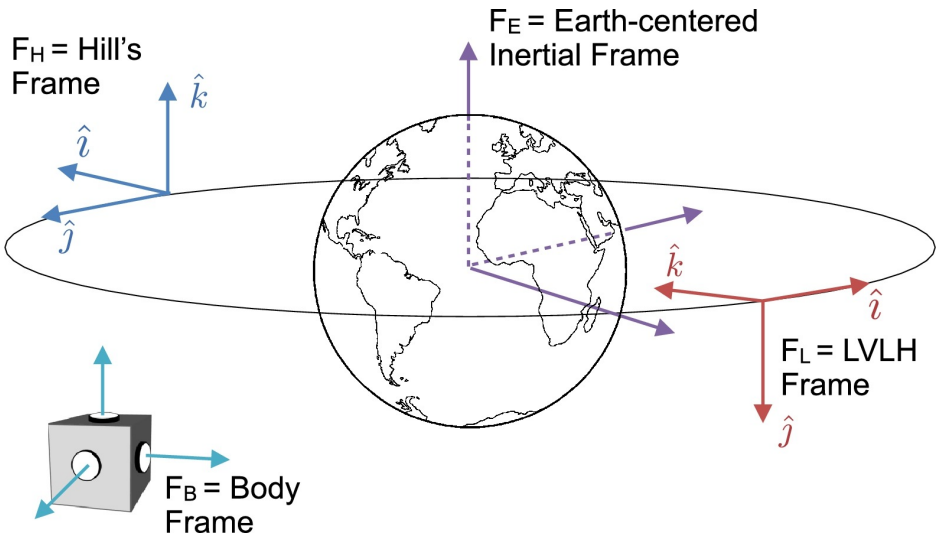


Figure 2.2: Illustration of the inertial, Hill and Body reference frames.[2]

The analysis of satellite motion requires the definition of several reference frames in which quantities such as position, velocity, and attitude are expressed. This section introduces the coordinate systems used throughout this thesis and outlines the conventions adopted for transformations and rotational parameterizations.

Inertial Frame ($\mathcal{I}$)

The Inertial frame $\mathcal{I}$ is defined as the Earth-Centered Inertial (ECI) frame. It is a non-accelerating Cartesian coordinate system with its origin at the Earth's center of mass and a fixed orientation relative to distant stars.

- $\hat{\mathbf{x}}$ points toward the vernal equinox.
- $\hat{\mathbf{z}}$ is aligned with the Earth's mean rotation axis, pointing toward the celestial north pole.
- $\hat{\mathbf{y}}$ completes the right-handed triad.

This frame is treated as inertial for the purposes of orbital mechanics, meaning that it is sufficiently free of acceleration that Newton's second law applies directly, without introducing fictitious forces. Physical models such as the two-body gravitational equations are typically formulated in this frame before being transformed into more application-specific frames for analysis or control.

Body Frame ($\mathcal{B}$)

The body-fixed frame $\mathcal{B}$ is rigidly attached to the satellite and rotates with it. Its origin is located at the satellite's center of mass, and its axes are aligned with the satellite's principal axes of inertia. All rotational dynamics are formulated with respect to $\mathcal{B}$ under the rigid-body assumption, in which the mass distribution and thus inertia tensor remain constant.

Hill Frame ($\mathcal{H}$)

The Hill frame, denoted $\mathcal{H}$, is a rotating reference frame centered on a target satellite in orbit. It is commonly used to describe the motion of a chaser satellite during close-proximity operations such as rendezvous or formation flying.

Multiple definitions of the Local-Vertical Local-Horizontal (LVLH) frame exist in the literature, depending on whether the axes are aligned with orbital velocity, angular

momentum, or other geometric references. In this work, the Hill and LVLH frames are used interchangeably and are defined as follows:

- $\hat{\mathbf{x}}$ points radially outward from the center of the Earth to the target.
- $\hat{\mathbf{z}}$ is aligned with the orbital angular momentum vector, normal to the orbital plane and pointing in the out-of-plane direction.
- $\hat{\mathbf{y}}$ completes the right-handed triad and lies in the orbital plane, pointing in the direction of motion.

By expressing the chaser's position and velocity relative to the target in $\mathcal{H}$, the bulk inertial motion of the system is removed, simplifying the relative dynamics. Unlike the inertial frame, the Hill frame is non-inertial and introduces fictitious forces such as Coriolis and centrifugal accelerations, which must be considered in the equations of motion. It is important to distinguish the Hill frame from the target's body frame, which may not be aligned with the orbit depending on the target's attitude.

Rotation Conventions

All rotations are defined using the right-hand rule. Rotations between frames are described using passive transformations, which re-express fixed vectors in different coordinate systems. For example, the direction cosine matrix (DCM) $\mathbf{C}_{\mathcal{B}\leftarrow\mathcal{I}}$ maps a vector $\mathbf{v}_{\mathcal{I}}$ expressed in the inertial frame to $\mathbf{v}_{\mathcal{B}}$ in the body frame. Subscripts always follow the convention that the right-hand side is the original frame and the left-hand side is the destination frame.

2.2.2 Attitude Modeling and Dynamics

Attitude Representations

The orientation of a rigid body in three-dimensional space is typically described by a rotation from a reference frame, such as the inertial frame $\mathcal{I}$, to the body-fixed frame $\mathcal{B}$. The attitude-kinematic and dynamic formulations used in this thesis follow the derivations presented in [50, 51]. Several mathematical representations exist for parameterizing such rotations, each with its own advantages and limitations. The three representations used in this thesis are Euler angles, rotation matrices, and quaternions.

Euler angles define orientation through a sequence of three elemental rotations about specified coordinate axes. In this work, a 3-2-1 (yaw–pitch–roll) convention is

used, in which the rotation from the inertial frame to the body frame is parameterized by the following angles:

- ψ: rotation about the inertial z-axis (yaw),
- θ: rotation about the intermediate y-axis (pitch),
- ϕ: rotation about the final x-axis (roll).

The resulting direction cosine matrix (DCM) from $\mathcal{I}$ to $\mathcal{B}$ is given by:

$$\mathbf{C}_{\mathcal{B}\leftarrow\mathcal{I}} = \mathbf{R}_x(\phi)\,\mathbf{R}_y(\theta)\,\mathbf{R}_z(\psi) \tag{2.21}$$

where $\mathbf{R}_x(\cdot)$, $\mathbf{R}_y(\cdot)$, and $\mathbf{R}_z(\cdot)$ are the elementary rotation matrices about the respective axes.

Euler angles offer intuitive physical meaning and are often used for visualization, however, they suffer from singularities (gimbal lock) that occur when the pitch angle θ approaches $\pm 90°$. To avoid these issues in computation and control, quaternions are often used instead.

A quaternion is a four-dimensional vector that provides a non-singular, smooth parameterization of rotation. A unit quaternion is defined as:

$$q = [q_0,\ \mathbf{q}_v]^T = [q_0, q_1, q_2, q_3]^T \tag{2.22}$$

where $q_0 \in \mathbb{R}$ is the scalar part and $\mathbf{q}_v \in \mathbb{R}^3$ is the vector part. A unit quaternion satisfies:

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1 \tag{2.23}$$

Quaternions offer several computational benefits: they avoid singularities, enable smooth interpolation, and allow efficient composition of rotations through quaternion multiplication. Given two rotations represented by quaternions q_1 and q_2, their composition is:

$$q_3 = q_1 \otimes q_2 \tag{2.24}$$

where $\otimes$ denotes quaternion multiplication, defined as:

$$p \otimes q = \begin{bmatrix} p_0 q_0 - \mathbf{p}_v^T \mathbf{q}_v \\ p_0 \mathbf{q}_v + q_0 \mathbf{p}_v + \mathbf{p}_v \times \mathbf{q}_v \end{bmatrix} \tag{2.25}$$

Note that q and $-q$ represent the same physical rotation.

In the context of control, it is often necessary to define the rotation required to align the current orientation with a desired one. Given a current quaternion q and a desired quaternion q_d, the quaternion error q_e is defined as:

$$q_e = q_d \otimes q^{-1} \tag{2.26}$$

where $q^{-1} = [q_0,\ -\mathbf{q}_v]^T$ is the quaternion conjugate. This formulation is widely used for attitude control, as it provides a continuous and globally non-singular representation of rotational error.

It is occasionally useful to represent the attitude error as a single rotation angle, given a unit quaternion q, the corresponding rotation angle is:

$$\phi = 2\arccos(q_0) \tag{2.27}$$

Transformations between different representations are well established. For instance, a unit quaternion q can be converted into a rotation matrix $\mathbf{C} \in SO(3)$ using:

$$\mathbf{C} = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{2.28}$$

The quaternion corresponding to a set of Euler angles (ϕ, θ, ψ) under the 3-2-1 convention is:

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) \end{bmatrix} \tag{2.29}$$

Kinematics

The attitude kinematics of a rigid body describe the evolution of its orientation over time as a function of its angular velocity. This section presents the differential relationships between the attitude representations introduced earlier and the angular velocity vector.

Let the orientation of the body be represented by the Euler angles (ϕ, θ, ψ) according to the 3-2-1 sequence. The angular velocity vector expressed in the body frame,

$\boldsymbol{\omega}_{\mathcal{B}} = [\,\omega_x,\, \omega_y,\, \omega_z\,]^T$ can be computed from the Euler angles by:

$$\boldsymbol{\omega}_{\mathcal{B}} = \mathbf{T}(\phi, \theta) \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.30}$$

where the transformation matrix $\mathbf{T}(\phi, \theta)$ for the 3-2-1 sequence is given by

$$\mathbf{T}(\phi, \theta) = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{2.31}$$

This mapping becomes singular at $\theta = \pm 90°$, reflecting the gimbal lock condition inherent to Euler angles.

Quaternions provide a singularity-free means of propagating attitude over time. Let q be the unit quaternion representing the current attitude, then the quaternion kinematic differential equation is given by:

$$\dot{q} = \frac{1}{2}\, \boldsymbol{\Omega}(\boldsymbol{\omega}_{\mathcal{B}})\, q \tag{2.32}$$

where $\boldsymbol{\Omega}(\boldsymbol{\omega}_{\mathcal{B}})$ is the 4×4 matrix defined as

$$\boldsymbol{\Omega}(\boldsymbol{\omega}_{\mathcal{B}}) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \tag{2.33}$$

In numerical implementations, the quaternion must be periodically renormalized to enforce the unit norm constraint $\|q\| = 1$ due to numerical integration errors.

An equivalent vector form of the quaternion kinematics may also be written. Let $\boldsymbol{\omega}_q = [0, \boldsymbol{\omega}_{\mathcal{B}}]^T$ be the pure quaternion formed from the angular velocity vector. Then,

$$\dot{q} = \frac{1}{2} q \otimes \boldsymbol{\omega}_q \tag{2.34}$$

When the attitude is represented by a rotation matrix $\mathbf{C}_{\mathcal{B}\leftarrow\mathcal{I}}$, the kinematic equation is expressed as:

$$\dot{\mathbf{C}} = \mathbf{C}\, [\boldsymbol{\omega}_{\mathcal{B}}]_\times \tag{2.35}$$

where $[\boldsymbol{\omega}_{\mathcal{B}}]_\times$ is the skew-symmetric matrix defined by

$$[\boldsymbol{\omega}_{\mathcal{B}}]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{2.36}$$

Dynamics

The rotational dynamics of a rigid body is governed by the evolution of its angular momentum. The angular momentum vector $\mathbf{H}_{\mathcal{B}}$ of the body, expressed in $\mathcal{B}$, is defined as

$$\mathbf{H}_{\mathcal{B}} = \mathbf{I}\boldsymbol{\omega}_{\mathcal{B}} \tag{2.37}$$

where $\mathbf{I} \in \mathbb{R}^{3\times3}$ is the moment of inertia matrix relative to the body frame.

In general, $\mathbf{I}$ is a symmetric positive-definite matrix that encodes the distribution of mass about the rotation axes. However, for most satellite, it is customary to define the body-fixed frame to coincide with the principal axes of the inertia tensor. This implies that $\mathbf{I}$ is diagonal and of the form:

$$\mathbf{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.38}$$

The time derivative of the angular momentum vector in an inertial frame $\mathcal{I}$ is equal to the applied torque:

$$\left.\frac{d\mathbf{H}_{\mathcal{B}}}{dt}\right|_{\mathcal{I}} = \boldsymbol{\tau}_{\mathcal{I}} \tag{2.39}$$

To express this equation in the body frame, the transport theorem is applied:

$$\left.\frac{d\mathbf{H}_{\mathcal{B}}}{dt}\right|_{\mathcal{I}} = \left.\frac{d\mathbf{H}_{\mathcal{B}}}{dt}\right|_{\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{B}} \times \mathbf{H}_{\mathcal{B}} \tag{2.40}$$

where the second term accounts for the rotation of the body frame relative to the inertial frame.

Substituting equation 2.37 into equation 2.40 yields:

$$\boldsymbol{\tau}_{\mathcal{B}} = \mathbf{I}\dot{\boldsymbol{\omega}}_{\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{B}} \times (\mathbf{I}\boldsymbol{\omega}_{\mathcal{B}}) \tag{2.41}$$

This is the vector form of Euler's equations of motion. It describes how the angular

velocity of the satellite evolves in response to applied torques.

In component form, using the diagonal inertia matrix and denoting $\boldsymbol{\tau}_{\mathcal{B}} = [\tau_x, \tau_y, \tau_z]^T$, Eq. (2.41) expands to:

$$\begin{aligned} I_x\dot{\omega}_x + (I_z - I_y)\omega_y\omega_z &= \tau_x \\ I_y\dot{\omega}_y + (I_x - I_z)\omega_z\omega_x &= \tau_y \\ I_z\dot{\omega}_z + (I_y - I_x)\omega_x\omega_y &= \tau_z \end{aligned} \tag{2.42}$$

Together with the kinematic equations described previously, these dynamics define the full nonlinear rigid body rotational motion of a satellite and form the basis of the dynamic model used throughout this thesis.

2.2.3 Orbital and Relative Translational Dynamics

Modeling the relative motion between two satellite in orbit is fundamental to the analysis and design of rendezvous and proximity operations. Various models exist to describe this relative motion, each with differing levels of fidelity and computational complexity. For the purposes of this thesis, the Clohessy–Wiltshire (CW) equations are adopted to describe the relative motion of a chaser satellite with respect to a target [52, 53].

The CW equations form a linear time-invariant (LTI) model derived under the assumption of a circular reference orbit. They are expressed in the Hill frame, which is both an accelerating and rotating frame of reference. Although this would appear to make the equations of motion more complex, the CW model simplifies the problem by linearizing the chaser's equations of motion around the target. This linearization is valid under the assumptions that the chaser remains close to the target, that gravitational perturbations and atmospheric drag are negligible, and that the target's orbit is circular. The result is a tractable model that is widely used for mission planning and control design in close-proximity operations.

Under these assumptions, the homogeneous CW equations governing the relative motion of the chaser are:

$$\ddot{x} = 3n^2x - 2n\dot{y} \tag{2.43}$$

$$\ddot{y} = 2n\dot{x} \tag{2.44}$$

$$\ddot{z} = -n^2z \tag{2.45}$$

Here, (x, y, z) represent the chaser's position in the radial, along-track, and cross-

track directions, respectively, and $n = \sqrt{\mu/a_t^3}$ is the mean orbital motion of the target, which orbits at a fixed radius a_t. These equations describe the unforced relative motion of the chaser when no control inputs are applied.

To include the effect of actuation, external forces $\mathbf{F} = [F_x, F_y, F_z]^T$ acting on the chaser (e.g. from thrusters) can be incorporated as:

$$\ddot{x} = 3n^2 x - 2n\dot{y} + \frac{F_x}{m} \tag{2.46}$$

$$\ddot{y} = 2n\dot{x} + \frac{F_y}{m} \tag{2.47}$$

$$\ddot{z} = -n^2 z + \frac{F_z}{m} \tag{2.48}$$

where m is the chaser's mass, and $\mathbf{F} = [F_x, F_y, F_z]^T$ is the applied force vector. These forced CW equations form the basis for modeling translational dynamics and designing control laws for close-range maneuvers.

Due to their linear time-invariant nature, the CW equations admit a closed-form analytical solution. Defining the state vector as

$$\mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} \tag{2.49}$$

the time evolution of the system can be written compactly as

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{\Phi}_{rr}(t) & \mathbf{\Phi}_{rv}(t) \\ \mathbf{\Phi}_{vr}(t) & \mathbf{\Phi}_{vv}(t) \end{bmatrix} \mathbf{x}(0) \tag{2.50}$$

where the state transition matrix is composed of four submatrices $\mathbf{\Phi}_{ij}(t)$ that map initial position and velocity to $\mathbf{x}(t)$. These are given by:

$$\mathbf{\Phi}_{rr}(t) = \begin{bmatrix} 4 - 3\cos nt & 0 & 0 \\ 6(\sin nt - nt) & 1 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{2.51}$$

$$\mathbf{\Phi}_{rv}(t) = \frac{1}{n} \begin{bmatrix} \sin nt & 2(1 - \cos nt) & 0 \\ 2(\cos nt - 1) & 4\sin nt - 3nt & 0 \\ 0 & 0 & \sin nt \end{bmatrix} \tag{2.52}$$

$$\mathbf{\Phi}_{vr}(t) = \begin{bmatrix} 3n\sin nt & 0 & 0 \\ 6n(\cos nt - 1) & 0 & 0 \\ 0 & 0 & -n\sin nt \end{bmatrix} \tag{2.53}$$

$$\mathbf{\Phi}_{vv}(t) = \begin{bmatrix} \cos nt & 2\sin nt & 0 \\ -2\sin nt & 4\cos nt - 3 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{2.54}$$

These expressions allow the full relative state to be computed analytically at any time t from a given initial condition. This property is particularly advantageous in applications such as trajectory design, onboard guidance, and verification of rendezvous scenarios.

While the CW model provides an efficient and tractable framework for close-range operations, its precision diminishes for larger separations, longer time horizons, or when the target follows a significantly elliptical orbit. In such cases, higher-order models or numerical integration of the full nonlinear dynamics may be required. In this thesis, all mission profiles involve short-duration maneuvers at small separations, well within the regime in which the CW model assumptions remain valid. The CW model therefore serves as the translational dynamics model used throughout this thesis.

2.2.4 Environmental Disturbances

Satellite operating in Earth orbit are subject to a range of environmental disturbances that introduce external forces and torques. These perturbations arise from the interaction between the satellite and its surrounding environment, including gravitational asymmetries, atmospheric effects, magnetic fields, and solar radiation. Although such effects can influence both translational and rotational dynamics, they are not explicitly modeled in the simulation framework used in this thesis. Instead, their aggregate influence is represented by stochastic disturbances. The primary sources of environmental

disturbances relevant to the analysis are summarized below.

Gravity-Related Disturbances

The dominant gravity-related disturbance is the gravity gradient torque, which arises due to the variation in gravitational force across the length of a satellite. This differential force creates a torque that tends to align the satellite's axis of maximum moment of inertia with the local vertical direction. The torque depends on the satellite's geometry, attitude, and altitude, and typically lies in the range of 10^{-5} to 10^{-7} N m for satellites in low Earth orbit (LEO) [54, 55].

Other gravity-related perturbations include third-body gravitational forces (e.g. from the Moon or Sun) and J2 perturbations. These can introduce small secular variations in orbital elements over long durations but are negligible for the time horizons and spatial scales considered in this thesis. [56, 57, 58]

Aerodynamic Disturbances

The primary aerodynamic disturbance is aerodynamic drag, which acts opposite to the satellite's velocity vector and induces a decelerating force. The magnitude of this force depends on atmospheric density, satellite velocity, exposed surface area, and the drag coefficient. At LEO altitudes of 400–500 km, drag forces typically range from 10^{-7} to 10^{-12} N for small satellites [59, 55].

In addition to translational effects, drag can induce a disturbance torque if the center of pressure is not aligned with the center of mass. This results in a moment that can affect the satellite's attitude. Typical aerodynamic torques are on the order of 10^{-8} to 10^{-6} N m. These effects are negligible in higher orbits, such as geostationary Earth orbit (GEO), where atmospheric density is effectively zero.

Magnetic Disturbances

Magnetic disturbance torques arise from the interaction between the satellite's residual magnetic dipole and the Earth's magnetic field. Even passively magnetized materials within the satellite can generate a dipole moment, which, when crossed with the geomagnetic field, results in a torque. The strength of this torque depends on orbital location and the orientation of the dipole relative to the field lines. In LEO, typical values range from 10^{-5} to 10^{-7} N m [60, 55].

Solar Radiation Pressure

Solar radiation pressure results from the momentum transfer of incident solar photons impacting the satellite's surfaces. It generates both a force and a torque, depending on the orientation of the satellite and the distribution of reflective and absorptive surfaces. The disturbances caused by radiation pressure effects are generally small and accumulate over longer time scales. For the short-duration scenarios considered in this work, radiation pressure is not a significant contributor to system dynamics.

Modeling Strategy

Rather than simulate each environmental disturbance using detailed physical models, this thesis approximates their combined effects by injecting zero-mean Gaussian noise into the simulation. Disturbances are applied independently to the translational and rotational states, with standard deviations selected to exceed the nominal magnitudes of the physical effects discussed previously. This ensures that the control system is evaluated against a sufficiently challenging disturbance environment. The modeling approach is appropriate for the class of missions considered in this thesis, which involve short-duration operations over small relative distances, typically on the order of minutes and tens to hundreds of meters. Within these constraints, the cumulative effect of environmental disturbances remains limited and can be effectively approximated as stochastic excitation without loss of fidelity.

In addition to environmental perturbations, coupling between translational and rotational dynamics may also introduce indirect disturbances. This coupling typically occurs in two ways: through off-axis thrusting, which can induce torque, and through attitude-dependent force models, such as drag or solar pressure varying with satellite orientation. However, it is common practice in proximity operations to model translational and rotational dynamics as decoupled subsystems. The justification for this simplification is both physical and methodological:

- Actuators are typically arranged such that forces and torques can be applied with minimal cross-coupling. For instance, thrusters used for attitude control are fired in counteracting pairs to cancel net translational impulse, while translational thrust is generally aligned with the center of mass to avoid torque generation [61, 62].

- Attitude dynamics evolve on faster time scales than orbital motion. Satellite

orientation can typically be adjusted and stabilized within seconds, whereas translational maneuvers unfold over several minutes.

- The decoupled modeling assumption is well established in both academic literature and operational practice, particularly for autonomous rendezvous, formation flying, and other short-range relative motion scenarios [63, 64, 65, 66].

In light of these considerations, this thesis models translational and rotational dynamics as concurrent but decoupled systems, allowing for modular controller design, simplified analysis, and efficient simulation without compromising accuracy for the intended mission class.

Chapter 3

Deep Reinforcement Learning Architectures for Satellite Proximity Operations

3.1 Motivation

This chapter explores the application of deep reinforcement learning to the problem of integrated satellite guidance and control during close-range proximity operations. Specifically, it investigates the use of DRL for the autonomous control of a chaser satellite's position and attitude as it approaches a non-cooperative target. The ability to perform this task reliably and without ground intervention is essential for many mission types, including on-orbit servicing, inspection, and active debris removal.

Close-range operations, defined here as distances under 100m, are among the most demanding phases of rendezvous requiring fine-grained control under tight performance constraints. Classical controllers like PID, LQR, and MPC are commonly used in satellite guidance and control, but face limitations when system models are uncertain or when the mission profile varies significantly [67, 68, 69]. This motivates the use of model-free control methods that can learn effective policies through interaction and adapt to system dynamics directly from data.

A growing body of literature has demonstrated the viability of DRL for both satellite attitude control [70, 71, 72] and position control during rendezvous [73, 74]. However, much of this work addresses either position or attitude control in isolation, with limited discussion of how architectural decisions impact policy performance, stability,

or interpretability. This leaves open questions about how to structure DRL agents for practical satellite control: should translation and attitude control be treated as independent learning problems, or combined into a single policy?

To address a gap, this chapter introduces and evaluates two novel DRL-based control architectures designed for 6-DoF satellite guidance:

- Decentralised TD3 (D-TD3): Two agents are used, one learns to control translational motion by producing force commands, while the other controls rotational dynamics by generating torque commands. This modular design mirrors conventional control architectures and allows each agent to operate on a lower-dimensional observation space tailored to its specific subtask.

- Centralised TD3 (C-TD3): A single agent receives the full relative state and jointly outputs both force and torque commands allowing a single policy to manage the entire 6-DoF control task. This approach is attractive when tight coordination is required, as it removes the need for a hand-designed interface between translation and attitude subsystems and can exploit shared context in the full state to produce coupled, mission-level behaviour.

The purpose of this study is to examine the trade-offs between these two architectural approaches. On one hand, the decentralised design reduces the dimensionality of each learning problem and reflects established control practice. On the other hand, a centralised design offers a unified control policy that may enable more efficient learning by jointly optimising across both subsystems and making use of shared context from the full state. These hypotheses are evaluated through controlled experiments that train and test both agents using a common reward structure and simulation framework.

A CubeSat is a standardised class of small satellite built from modular units, where one unit (1U) is a cube of approximately 10 cm on each side. CubeSats typically use commercial off-the-shelf components and a common mechanical interface, enabling low-cost and rapid development compared with conventional spacecraft. In this thesis, the CubeSat platform is used as a representative small satellite for evaluating learning-based guidance and control methods.

The simulation environment used in this work models a 1U CubeSat positioned up to 100m from a passive target with an arbitrary initial orientation. The agent must bring the satellite within 5m of the target while aligning its body frame to within 2.5° of the desired orientation. These thresholds are selected to represent practical handover

conditions for downstream docking or inspection operations, where relative pose errors must be small enough for short-range sensors and local controllers to complete the final approach safely. If either criterion is not met by the end of an episode, the manoeuvre is treated as unsuccessful, and the controller is considered not to have achieved the required terminal accuracy for handover.

This chapter makes the following contributions:

1. It presents a structured evaluation of centralised and decentralised deep reinforcement learning architectures for satellite guidance and control. Using a consistent simulation framework, shared reward functions, and Monte Carlo simulations, the work isolates how architectural choices affect agent and controller performance.

2. It details the design, development, and evaluation of a DRL-based control framework capable of 6-DoF position and attitude regulation in a close proximity rendezvous scenario. Unlike prior studies that address these domains in isolation, this chapter presents and compares modular and end-to-end architectures, offering new insights into policy structure, coordination, and performance under operational constraints.

These findings establish a validated baseline for DRL-based satellite guidance and control under realistic conditions. More broadly, they highlight the practical benefits of modular control architectures in achieving stable and generalisable policies in integrated 6-DoF tasks. This chapter lays the groundwork for the rest of the thesis, which builds on these results to tackle increasingly complex and uncertain mission conditions using alternative RL frameworks.

3.2 Literature Review

3.2.1 Deep Reinforcement Learning for Satellite Attitude Control

The Attitude Determination and Control System (ADCS) is a critical part of the broader Attitude and Orbit Control System (AOCS), responsible specifically for estimating and regulating the satellite's orientation. A high-level overview of the AOCS is shown in Figure 3.1.

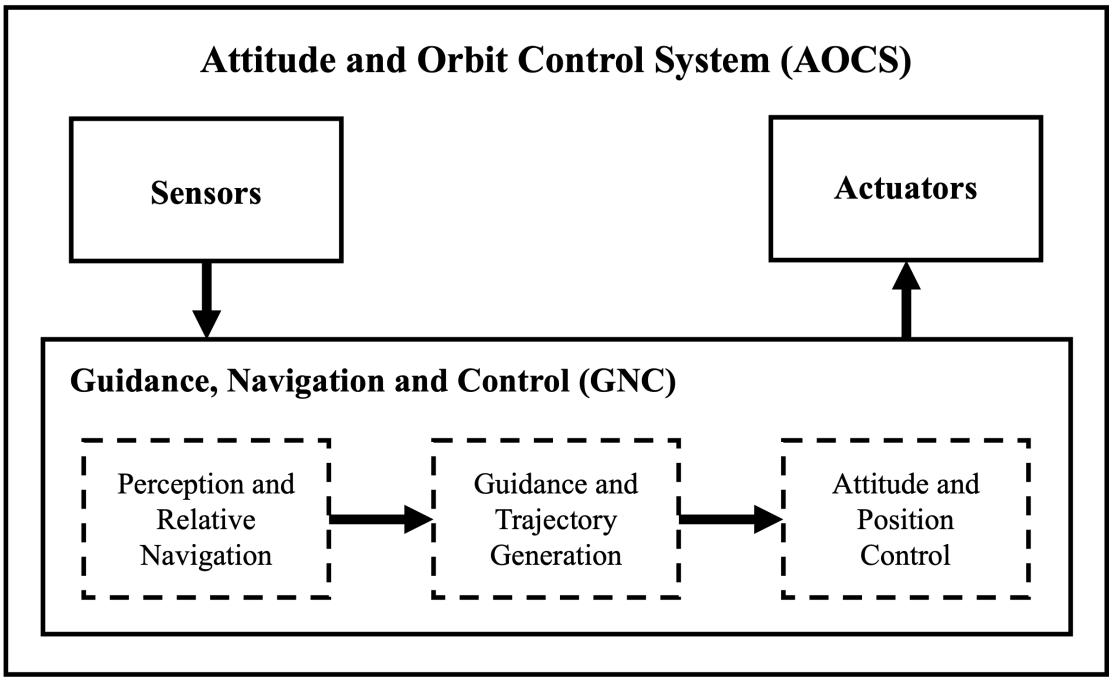


Figure 3.1: Block diagram of the Attitude and Orbit Control System (AOCS), comprising sensors, actuators, and the Guidance, Navigation, and Control (GNC) subsystem. The system maintains satellite orientation and trajectory through closed-loop feedback.

Accurate attitude control is essential for mission objectives such as Earth observation, communication, scientific measurements, and deep-space navigation. The ADCS is composed of two main functions: attitude determination, which estimates the satellite's current orientation, and attitude control, which adjusts it as needed.

Since there is no direct means to measure a satellite's attitude, it must be estimated using sensor data. These sensors can be divided into two broad categories: relative sensors and absolute sensors. Relative sensors, such as gyroscopes, measure angular velocity and provide information on the rate of change of attitude. Absolute sensors, including magnetometers, sun sensors, star trackers, and Earth horizon sensors, measure orientation with respect to known external references like the Earth's magnetic field or celestial bodies. The raw data from these sensors are then combined using estimation algorithms, with Kalman filtering being a common choice.

Attitude control systems can be categorised as either passive or active. Passive attitude control leverages environmental torques to stabilize or control the satellite's attitude. Examples include gravity-gradient stabilisation, where the Earth's gravity field exerts a torque on an extended satellite body; magnetic control, which uses

magnetic torquers interacting with Earth's magnetic field; and aerodynamic control, which exploits atmospheric drag in low-Earth orbit. Passive systems are simple and energy-efficient but have limited control authority and are sensitive to changes in environmental conditions. They are typically unsuitable for missions requiring rapid or consistent manoeuvring.

Active attitude control systems, on the other hand, use actuators such as reaction wheels, control moment gyroscopes, and thrusters to produce controlled torques. These systems provide greater accuracy and responsiveness than passive methods and are essential for missions involving frequent reorientations, precise pointing, or autonomous operations. Although active systems consume more power they are the standard in most modern satellite.

Conventional control methods have been widely used for satellite attitude control. One of the most common is the PID (Proportional-Integral-Derivative) controller [75, 76, 77]. The proportional term provides a control output proportional to the error, the integral term compensates for steady-state error, and the derivative term improves system damping. Variants like the PD controller, which omits the integral term, are also commonly used. While PID control is well understood, it faces limitations when applied to satellite. The system dynamics are non-linear and time-varying, making it difficult to tune PID parameters effectively. PID controllers are also sensitive to sensor noise, which can degrade performance or cause instability, particularly in high-precision applications [67, 68].

More advanced control strategies such as H-infinity ($\mathcal{H}_\infty$) control have been developed to address some of these limitations [78, 79]. $\mathcal{H}_\infty$ control is a robust control technique that optimises the system response under worst-case disturbance scenarios. This makes it particularly suitable for applications with high uncertainty, including those involving environmental disturbances or internal system variations. For instance, changes in mass distribution due to fuel usage or payload deployment can degrade the performance of classical controllers [69]. $\mathcal{H}_\infty$ methods maintain stability and performance in such conditions, although they may result in slower response times [80].

Sliding Mode Control (SMC) is another robust non-linear control strategy well suited to systems with model uncertainties and external disturbances. It operates based on the concept of a sliding surface, a condition set in the system's state space. Once the system state reaches this surface, it is forced to "slide" along it towards the equilibrium point, ensuring stability and robustness despite system non-linearities or disturbances [81]. Its robustness and simplicity make it attractive for space applica-

tions, especially when precise modelling is not feasible. However, SMC is prone to chattering, a phenomenon where the control signal oscillates rapidly due to discontinuous switching. This can lead to excessive actuator wear and energy consumption [82, 83, 84].

These classical and robust controllers require careful selection and tuning of parameters, which can be time-consuming and mission-specific. Even small changes in the satellite's mass or inertia can destabilise a predesigned controller. This has motivated a shift towards adaptive and intelligent control techniques which aim to overcome the limitations of traditional model-based approaches. These methods include adaptive control [85, 86, 87], fuzzy logic control [88, 89], and intelligent control.

Among intelligent control strategies, reinforcement learning has emerged as a strong candidate due to its flexibility, model-free learning capability, and capacity to optimize long-term objectives [90, 91, 92, 93]. In GNC systems, especially for autonomous or partially autonomous satellite, decisions must be made in dynamic and uncertain environments where models may be incomplete or time-varying. DRL provides a framework that can learn control policies directly from experience, allowing it to handle non-linearities, unmodelled dynamics, and disturbances without requiring precise prior models. This is particularly relevant for systems operating in regimes that are difficult to capture analytically, such as planetary descent, proximity operations, or formation flying, where traditional control design becomes increasingly fragile or limited.

A growing body of work has demonstrated the promise of deep reinforcement learning for satellite attitude control, particularly in scenarios involving large-angle slewing, pointing, and trajectory tracking. Early studies focused on nominal conditions, where the satellite is fully actuated and the environment is well understood. In these settings, DRL agents have consistently delivered performance comparable to, or exceeding, traditional controllers. For instance, reliable large-angle slews have been achieved on simulated LM50-class satellite using both discrete-action and continuous-action agents, showing that DRL can manage both quantised torque levels and continuous control effectively [70, 94]. Flexible-body dynamics have been addressed, successfully controlling a flexible satellite during rest-to-rest manoeuvres [71]. Other studies extended this further to time-varying reference tracking, demonstrating that recurrent architectures can improve policy performance by incorporating temporal context [72]. In another example, a two-phase attitude control strategy was designed for agile satellite equipped with CMGs. In this approach, DRL performs fast, coarse pointing manoeu-

vres, after which conventional controllers take over for fine stabilization. This hybrid design allows the system to exploit the adaptability of DRL in complex slewing while retaining the precision of classical control in steady-state conditions [95].

Building on this, other work has investigated how DRL performs with incomplete system information. DRL policies trained with randomised inertial parameters and sensor noise have shown improved robustness, maintaining performance in both pointing and slewing tasks despite significant modelling errors [96]. This robustness appears to scale across platforms as well, with a single policy able to operate effectively across a wide range of satellite configurations, from CubeSats to larger systems, by incorporating noise and dynamic variability during training [97]. In underactuated control scenarios, where actuator faults reduce the available control authority, DRL controllers have also shown success. In such cases, the agents have demonstrated the ability to maintain a stable attitude despite faults [98]. Other methods have shown fault tolerance by incorporating techniques such as hindsight experience replay and dimension-wise clipping [99].

Other work has shown similar results with agents trained on telemetry-derived actuator models being able to generalise across satellite platforms without mission-specific controller retraining [100]. In the context of active debris removal, where satellite properties change rapidly following target capture, agents were able to adapt to large changes in mass and inertia without any explicit identification, instead relying on stacked observations to infer system state from context [101]. These examples demonstrate one of DRL's core strengths: its ability to adapt to evolving internal and external conditions without requiring manual retuning or model updates.

3.2.2 Deep Reinforcement Learning for Satellite Position Control

Satellite rendezvous refers to the process in which a chaser satellite matches, and maintains a desired relative position and velocity with a target satellite. This complex operation is fundamental to a wide variety of space missions, from satellite servicing and debris removal to crewed docking procedures. To achieve a successful rendezvous, precise control strategies are required to cope with the unique challenges of the space environment. Several methods have been developed to solve the rendezvous problem, ranging from classical methods to modern techniques, each with its own strengths and challenges.

A typical satellite rendezvous can be described in three stages. The first stage is known as phasing; in this stage, the chaser satellite is directed by ground control to perform several manoeuvres in order to bring the chaser satellite to within one kilometre of the target satellite. When successful, the close-range rendezvous stage begins. This stage also marks the start of the automated control phase as control is usually transferred from ground control stations to onboard computers. The close-range rendezvous stage ends when the two satellite are approximately ten metres apart. Finally, the docking phase brings both satellite into contact and marks the end of rendezvous operations. This research will be focused on the close-range rendezvous stage of operations.

One of the most widely adopted classical methods for position control has been the Clohessy-Wiltshire targeting method [102, 103]. The CW targeting method, while providing a practical solution, is reliant on impulsive manoeuvres which may not be optimal or even feasible in all mission scenarios. In response to some of the limitations of the CW targeting method, alternative methodologies have been proposed, one of which is the glideslope targeting method as used in the space shuttle [104]. This method represents a conceptual shift from CW targeting, extending the traditional method to include constant relative accelerations, a change well-articulated in [105]. As such, it adopts a constant direction guidance law, thereby facilitating more continuous control. This offers a potential solution to the challenges posed by impulsive manoeuvres in the CW targeting method and provides a framework for more gradual and potentially fuel-efficient approach procedures.

More recent literature reveals a shift towards modern control methods, with authors exploring techniques such as linear quadratic control [106, 107], fuzzy control [108, 109], and model predictive control [110, 111]. These methodologies present a host of intriguing possibilities. Linear quadratic control offers an optimal control approach under the assumption of linear system dynamics and quadratic cost criteria. Fuzzy control provides a unique approach to handling the complexities of imprecision and system uncertainties, while model predictive control incorporates future predictions into the control actions. These modern methods collectively offer new perspectives on the rendezvous problem, aiming to enhance control performance and robustness against uncertainties.

Close-range rendezvous, where the chaser approaches from within several hundred metres of the target, poses a number of challenges: non-linear dynamics, limited sensing, uncertain target motion, and constraints on thrust and fuel. DRL offers a way to

handle these conditions in a unified, data-driven framework, bypassing the need for precise system modelling or manually designed control laws.

To improve performance under partial observability and external disturbances, some work has focused on asymmetric learning architectures. In one example, the actor network was trained using noisy, limited onboard observations while the critic had access to the full state, allowing the policy to generalise to real-world sensing limitations while still benefiting from informative feedback during training [112]. Other studies have applied continuous-action DRL to guide satellite using low-thrust propulsion, achieving reduced fuel usage and faster transfers compared to traditional control schemes like PD [73]. These approaches demonstrate how DRL can be tailored to real-time, fine-grained control tasks that are difficult to handle with classical impulse-based strategies.

Beyond basic approach manoeuvres, DRL has also been used in more complex scenarios that involve interaction with uncertain or poorly characterised targets. One study used PPO to guide a small satellite performing fly-around manoeuvres for autonomous shape reconstruction of an uncooperative object. The agent had to manage trajectory accuracy, fuel constraints, and limited onboard state estimation. The learned policies demonstrated robustness to sensor noise and partial actuator failure, reinforcing the utility of DRL for multi-objective guidance problems [113].

Comparative studies have explored how algorithm design choices impact DRL performance for position control. One such investigation examined the effects of discrete vs. continuous action spaces and varying thrust magnitudes during satellite inspection and docking mission scenarios. Results showed that control granularity significantly affects fuel efficiency and trajectory smoothness, providing insight into practical DRL policy tuning [74]. In a broader benchmarking effort, reinforcement learning was compared to behavioural cloning for low-thrust guidance problems across interplanetary rendezvous and small-body landing tasks. The RL-based agents not only matched expert demonstrations but, in many cases, discovered more robust or fuel-optimal behaviours, particularly when reward shaping and control frequency decoupling were applied [114].

Taken together, these studies demonstrate that DRL is not only capable of solving translation control problems but also provides flexibility and generalisation capabilities that traditional guidance approaches often lack. Review work surveying DRL across GNC applications has consistently identified rendezvous as a key area where the ability to operate without detailed models, optimise over long horizons, and adapt

to changing mission dynamics offers a clear advantage [9]. While challenges remain, DRL represents a promising path forward for autonomous satellite position control in dynamic and uncertain environments.

3.3 Mission Description

This work focuses on the development and evaluation of an intelligent, learning-based guidance and control system capable of autonomously managing a chaser satellite's relative position and attitude with respect to a target. The system operates autonomously by processing relative pose data and generating control commands to manoeuvre the chaser satellite.

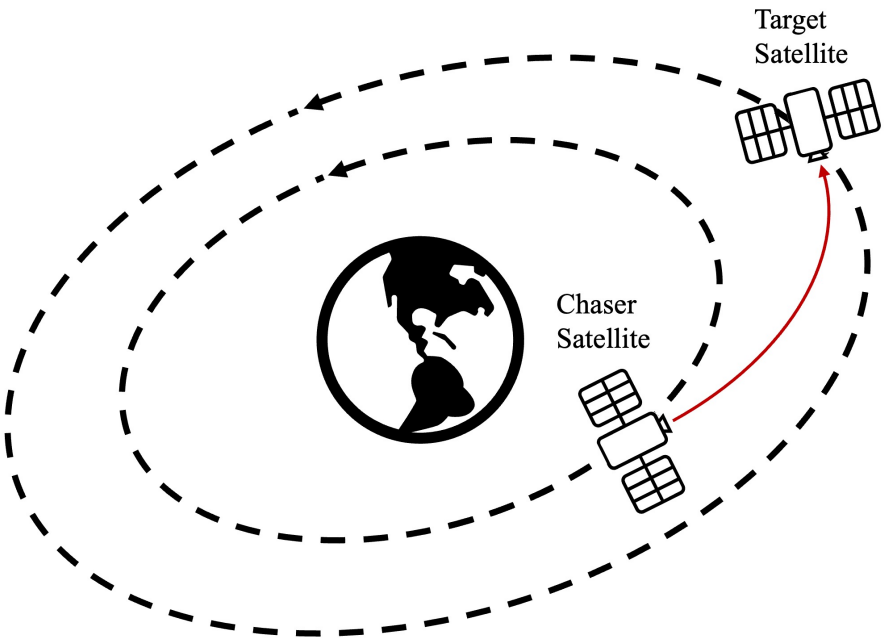


Figure 3.2: Illustration of a satellite rendezvous scenario. The chaser satellite must autonomously approach the target while aligning its orientation to face it, a typical requirement for docking, inspection, or capture operations.

The mission scenario considered in this research is a short-range proximity operations maneuver, focusing specifically on the terminal phase of an autonomous rendezvous and attitude alignment. The chaser satellite is required to approach a passive, non-cooperative target and align its body frame to face the target. Final docking or physical interaction is excluded from the scope. Instead, the focus is placed on approach and alignment, which are critical components of a wide range of mission types,

including inspection, rendezvous, on-orbit servicing, and active debris removal. Orbital transfer and long-range phasing manoeuvres are not considered; the scenario assumes the chaser is already established within the close-range regime at the start of each episode.

This mission profile is selected for its generality and operational relevance. A control system capable of achieving and maintaining accurate relative pose alignment under a variety of initial conditions is inherently mission-agnostic. Whether used for cooperative docking or uncooperative inspection, the ability to autonomously regulate both position and attitude at close range represents a transferable capability applicable to numerous downstream mission phases. This scenario isolates a challenging portion of proximity operations where both precision and robustness are crucial.

The chaser satellite modelled in this study is a 1U CubeSat, representative of low-mass, compact satellite operating in low Earth orbit. The satellite is assumed to have a total mass of 0.5 kg, with each principal moment of inertia taken as 0.001 $\mathrm{kg\,m^2}$. The target satellite is modelled as a passive, inertial body representing an uncooperative or inactive object.

In terms of actuation, the CubeSat is equipped with a translational propulsion system, such as a cold gas or electric microthruster, capable of producing up to 0.1 N of force along each principal axis. For attitude control, the satellite is outfitted with a reaction wheel system capable of delivering up to 0.001 $\mathrm{N\,m}$ of torque and storing a maximum angular momentum of 0.015 $\mathrm{N\,m\,s}$. These parameters are chosen to reflect realistic actuation limits for CubeSats operating in close-range scenarios [115, 116]. Magnetorquers and momentum dumping mechanisms are not included in the simulation, as the short time horizons and limited angular momentum accumulation during each episode do not necessitate their use.

Each episode begins with the chaser approximately 100 m from the target, with an arbitrary orientation and no initial relative velocity. The starting position includes non-zero displacement along all three translational axes, rather than being restricted to a straight-line trajectory along the radial (R-bar) or velocity (V-bar) vectors. This setup ensures that the controller is trained to handle general, curvilinear approach trajectories rather than relying on simplified, axis-aligned intercepts.

The objective of the agent is to regulate the chaser's motion and achieve a final state that satisfies the following mission success criteria:

- Minimize attitude error to within 2.5°

- Achieve a final separation of less than 5 m

This final proximity range is intended to represent a handover point to a downstream control system responsible for completing mission-specific tasks such as docking, capture, or inspection. As such, successful completion of this manoeuvre serves as a proxy for evaluating the controller's autonomous guidance and control capabilities under mission-relevant conditions.

3.4 Control Architecture and Methodology

3.4.1 Agent Structure and Design

This section describes the design and implementation of the deep reinforcement learning agents within the autonomous CubeSat guidance and control architecture. Two control architectures are considered: a decentralised controller (D-TD3) and a centralised controller (C-TD3). Both use TD3 agents trained to produce continuous force and torque commands to regulate the chaser satellite's 6-DoF motion based on relative state information.

The D-TD3 controller is composed of two agents trained independently. The first agent, D-TD3-R, governs translational dynamics by outputting force commands, while the second, D-TD3-A, governs rotational dynamics by outputting torque commands. Each agent has its own observation space, reward function, and policy, reflecting a modular design consistent with conventional guidance and control architectures. The overall structure of this controller is illustrated in Figure 3.3, showing the separation of translational and rotational control to distinct agents, each with its own input state, reward signal, and output commands.

The C-TD3 controller employs a single agent that jointly regulates translation and rotation. This agent receives a combined observation space and outputs an action vector of command forces and torques. This design allows the policy to learn coordinated behaviours directly from experience. This structure is shown in Figure 3.4, where a single agent processes the full state observation and learns to map it directly to combined control actions.

At each timestep, the agents observe the chaser's relative state with respect to the target, composed of its relative position $\mathbf{r} \in \mathbb{R}^3$, velocity $\mathbf{v} \in \mathbb{R}^3$, attitude error quaternion $q_e \in \mathbb{R}^4$, and angular velocity $\boldsymbol{\omega}_{\mathcal{B}} \in \mathbb{R}^3$. The quaternion error represents the rotation from the desired to the current orientation and is given by

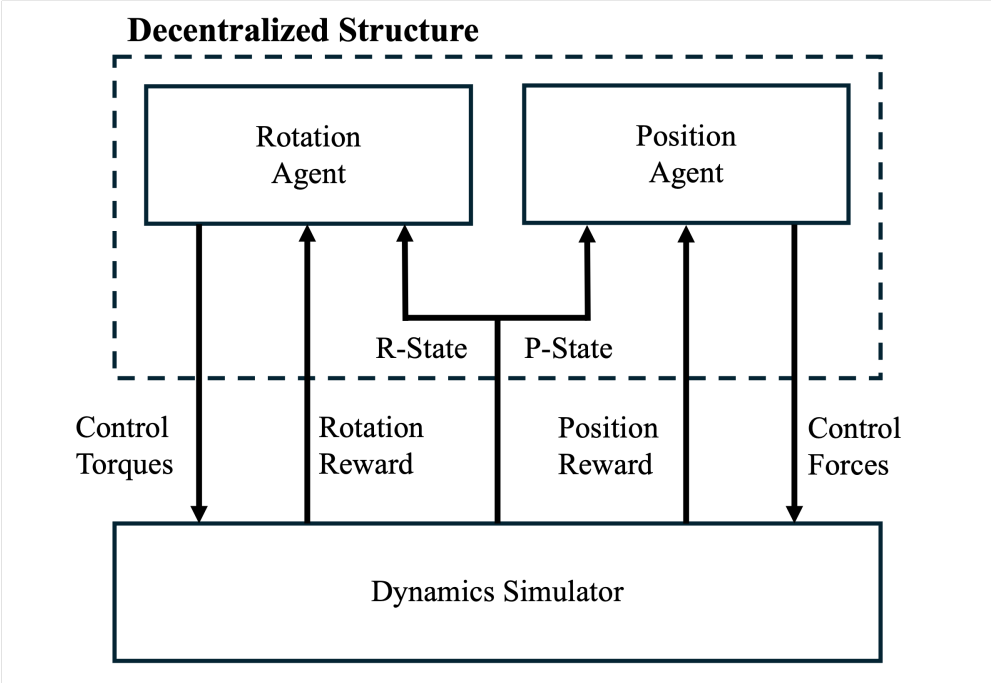


Figure 3.3: Block diagram of the decentralised controller architecture (D-TD3). Translational and rotational dynamics are handled by two separate agents, each operating on a subset of the state and producing independent control outputs.

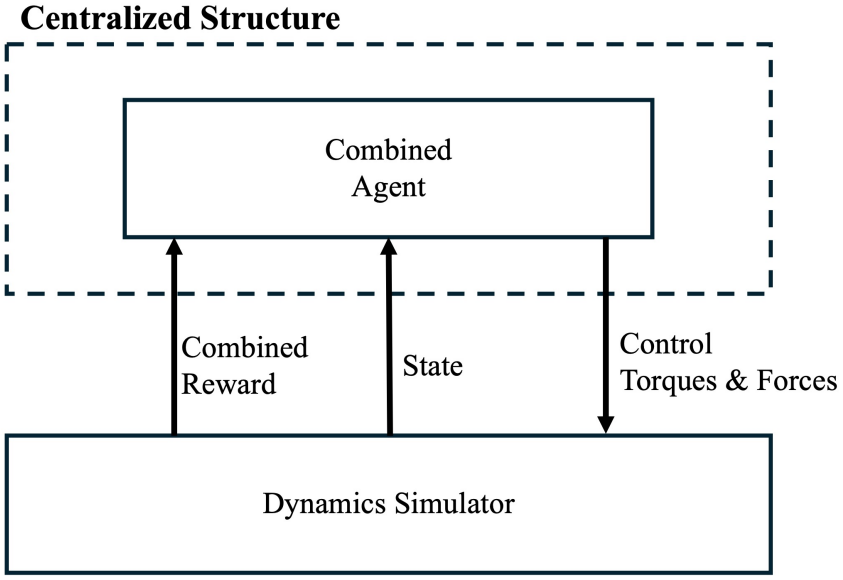


Figure 3.4: Block diagram of the centralised controller architecture (C-TD3). A single agent receives the full state observation and outputs a unified action vector containing both force and torque commands.

$$q_e = q_d \otimes q_c^{-1}, \tag{3.1}$$

where q_d and q_c denote the desired and current attitude quaternions, respectively.

In the decentralised configuration, the translational agent (D-TD3-R) observes $[\mathbf{r}, \mathbf{v}]$ and outputs a control force $\mathbf{F} \in \mathbb{R}^3$, while the rotational agent (D-TD3-A) observes $[q_e, \boldsymbol{\omega}_{\mathcal{B}}]$ and outputs a control torque $\boldsymbol{\tau}_{\mathcal{B}} \in \mathbb{R}^3$. In the centralised configuration, the C-TD3 agent receives the full observation $[\mathbf{r}, \mathbf{v}, q_e, \boldsymbol{\omega}_{\mathcal{B}}]$ and outputs the combined action $[\mathbf{F}, \boldsymbol{\tau}_{\mathcal{B}}]$.

The decentralised approach reflects established practice in satellite control, where translational guidance and attitude control are treated as distinct subsystems. This reduces the dimensionality of each learning problem, improving stability during training and interpretability. In contrast, the centralised approach has access to the full state observation and can leverage interactions between translational and rotational dynamics, particularly if thrust vector alignment and line-of-sight pointing is required. A unified policy enables joint optimisation of both behaviors but requires larger networks and more training data to achieve stable performance. Together, these two designs provide a means of evaluating the trade-off between modularity and coordination in DRL-based satellite control.

All agents were implemented using the TensorFlow [117] library based on the original paper [49] and open-source best practices. Each actor and critic network consists of two fully connected hidden layers, with the D-TD3-R and D-TD3-A agents using smaller networks with 64 and 128 units, and the C-TD3 agent employing larger networks of 512 and 1024 units to accommodate the higher-dimensional state and action spaces. Preliminary experiments showed that smaller networks for the centralised agent led to unstable training and poor convergence. A summary of the agent structures is provided in Table 3.1.

Table 3.1: Summary of D-TD3 and C-TD3 agent architectures

Property	**D-TD3-R**	**D-TD3-A**	**C-TD3**
Agent Role	Translational Control	Rotational Control	Combined Control
Input Dimensions	6	7	13
Output Dimensions	3	3	6
Hidden Layer 1	64	64	512
Hidden Layer 2	128	128	1024

These two controllers are compared under identical simulation conditions to assess

relative performance and highlight trade-offs between the decentralised and centralised approaches for DRL-based satellite guidance and control.

3.4.2 Reward Function and Training Setup

This section describes the reward structure and training methodology used for both the D-TD3-R, D-TD3-A and C-TD3 agents. The reward function is designed to incentivise accurate and robust 6-DoF satellite control, with both translational and rotational objectives. The training framework, including episode configuration, simulation details, and hyperparameter selection, is also presented.

Reward Function Design

The reward function is composed of two main components: a position reward r_p encouraging translational error correction, and an attitude reward r_a encouraging correct orientation. These components are supplemented by milestone bonuses and terminal penalties to guide exploration and penalise unsafe behaviour.

During the early stages of training, a binary reward is used as the position reward to provide frequent feedback based on changes in relative position. At each timestep Δt, a small positive reward was given if the chaser moved closer to the target, and a small negative reward was given otherwise:

$$r_p(t) = \begin{cases} +0.1 & \text{if } \|\mathbf{r}(t)\| < \|\mathbf{r}(t-\Delta t)\| \\ -0.1 & \text{otherwise} \end{cases} \tag{3.2}$$

Once the chaser approached within 25 m of the target, the reward transitioned to a continuous, distance-based signal to encourage finer tuned control:

$$r_p(t) = \exp\left(-\frac{\|\mathbf{r}(t)\|}{45}\right) \tag{3.3}$$

A binary reward at long ranges and a continuous reward near the target served to provide dense feedback early while prioritising precision in the terminal phase.

A similar structure was used for attitude alignment. Early in training, the scalar component of the quaternion error $q_{e,w}$ was used to determine a binary reward, encouraging reductions in angular misalignment:

$$r_a(t) = \begin{cases} +0.1 & \text{if } |q_{e,w}(t)| > |q_{e,w}(t - \Delta t)| \\ -0.1 & \text{otherwise} \end{cases} \tag{3.4}$$

When the angular error fell below 10°, the reward switched to a continuous one based on a quaternion alignment score, constructed to maximise alignment and penalise axis deviation:

$$r_a(t) = q_{e,w}^2 - q_{e,x}^2 - q_{e,y}^2 - q_{e,z}^2 \tag{3.5}$$

To accelerate convergence and reinforce mission objectives, small bonus rewards were issued when task goals were achieved, such as reducing the attitude error below 2.5° or reaching within 5 m of the target. Conversely, large negative terminal rewards were applied in cases of critical failure, such as exceeding a maximum relative distance of 250 m or angular velocity exceeding 90 $^\circ\,\mathrm{s}^{-1}$:

- Success Bonuses: +1.0 for coming within 5 m of the target and an additional +1.0 for attitude alignment within 2.5°.
- Failure Penalty: −10 and early termination if relative distance exceeds 250 m or angular velocity exceeds $90^\circ\,\mathrm{s}^{-1}$

For the decentralised controller, each agent received a task-specific reward: D-TD3-R received the translational component r_p, while D-TD3-A received r_a. The centralised C-TD3 agent received a combined reward $r = r_p + r_a$, and was responsible for balancing the two objectives during learning. This ensures that both architectures receive consistent shaping signals, though the centralised controller must implicitly learn to balance the two control objectives.

Training Framework

All agents were trained in a disturbance-free environment with ideal actuators to simplify the training environment. Each training episode lasted for a maximum of 2000 steps corresponding to 200 seconds of simulation time with the controllers operating at 10 Hz. The underlying dynamics were integrated at a higher resolution (100 Hz) to ensure numerical stability. Episodes terminate early if the success criteria are met or if unsafe termination conditions are triggered.

Training was terminated once the agent's average episode reward plateaued over successive runs, or when the performance reliably satisfied the mission criteria. At this point, the learned network weights were frozen for testing and evaluation.

Hyperparameters were selected using an iterative, experiment-driven process. Initial values were drawn from commonly used DRL benchmarks [118] and refined through experimental tuning. To maintain a fair comparison, a set of hyperparameters that achieve good performance for both the D-TD3 and C-TD3 controllers were selected. Final values are summarised in Table 3.2.

Table 3.2: Training hyperparameters for D-TD3 and C-TD3 controllers

Parameter	**D-TD3**	**C-TD3**
Learning rate	0.001	0.001
Replay buffer size	10^6	10^6
Mini-batch size	512	512
Optimiser	Adam	Adam
Discount factor (γ)	0.995	0.995

3.4.3 Simulation Results

The simulation environment was configured to reflect the operational constraints and disturbances expected during CubeSat proximity operations. Each episode was initialized with the chaser satellite positioned at a random location within a 100 m radius of the stationary target, with initial relative velocity and angular velocity set to zero. The chaser's orientation was randomly sampled within $\pm 45^\circ$ about each principal axis.

To improve simulation fidelity beyond the idealised training setup, actuator outputs were subject to rate limits of 10% of maximum actuation per second and perturbed by zero-mean noise with a magnitude of 1% of the commanded output to emulate real-world imperfections. Additionally, zero-mean sensor noise capped at 2.5% of the measured observation was added to the state estimates to reflect expected navigation errors and estimation inaccuracies.

Mission success was evaluated against the terminal conditions established in Section 3.3, requiring a final relative separation of less than 5 m and an attitude alignment error within 2.5°. These criteria represent practical handover conditions for downstream mission phases such as visual inspection or capture operations.

To evaluate both controllers, 100 episodes were simulated with randomised ini-

tial conditions. The controller outputs and chaser states were tracked throughout each episode to assess performance, robustness, and consistency across the operational envelope. This Monte Carlo approach enabled quantitative comparison of mean performance, variability, and convergence behaviour across architectures. In addition, a single representative trial was selected for each controller using identical initial conditions to support qualitative comparison.

Monte Carlo Evaluation Results

The Monte Carlo analysis over 100 trials indicates that the D-TD3 controller consistently outperforms the C-TD3 controller across all evaluated performance metrics. Summary statistics are presented in Table 3.3, and the corresponding aggregate trajectories are visualised in Figures 3.5 – 3.8.

Table 3.3: Summary statistics for D-TD3 and C-TD3 across 100 trials.

Metric	**D-TD3**	**C-TD3**
Final Position Error (m)	2.22 ± 0.87	4.17 ± 1.05
Max Linear Velocity ($\mathrm{m\,s^{-1}}$)	1.24 ± 0.66	2.57 ± 1.41
Time to Convergence (Position, s)	58.27 ± 30.28	49.43 ± 32.81
Final Orientation Error ($^\circ$)	0.80 ± 0.31	12.82 ± 20.12
Max Angular Velocity ($^\circ\,\mathrm{s^{-1}}$)	2.81 ± 1.12	19.90 ± 2.78
Time to Convergence (Orientation, s)	19.74 ± 12.28	94.85 ± 17.47

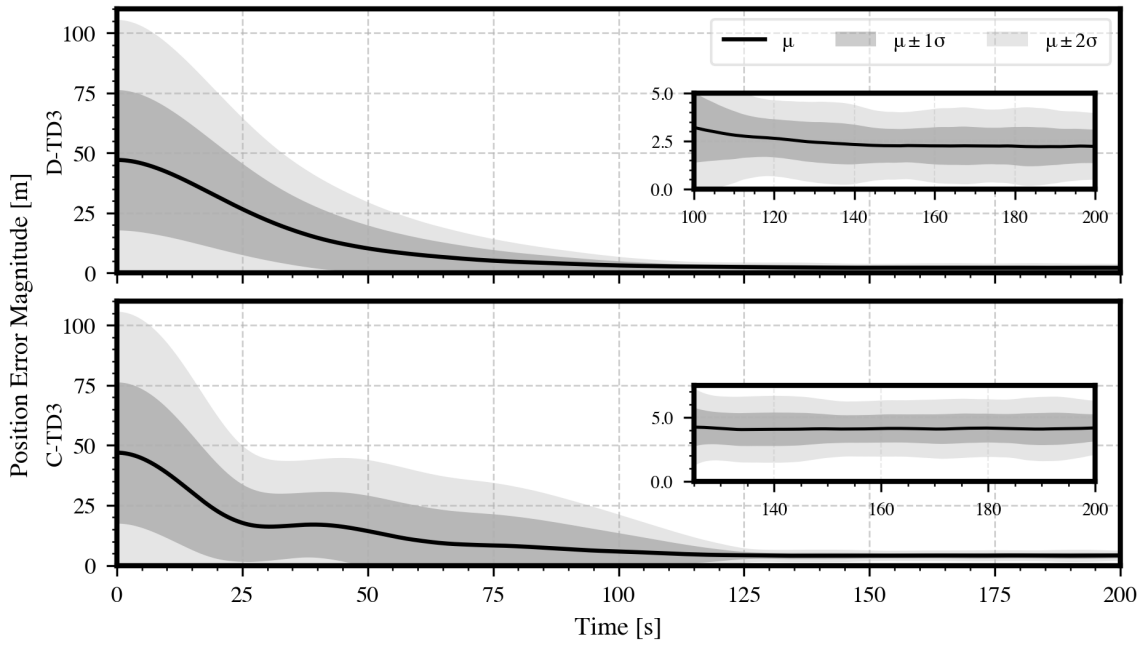


Figure 3.5: Position error magnitudes for the D-TD3 and C-TD3 controllers across 100 trials.

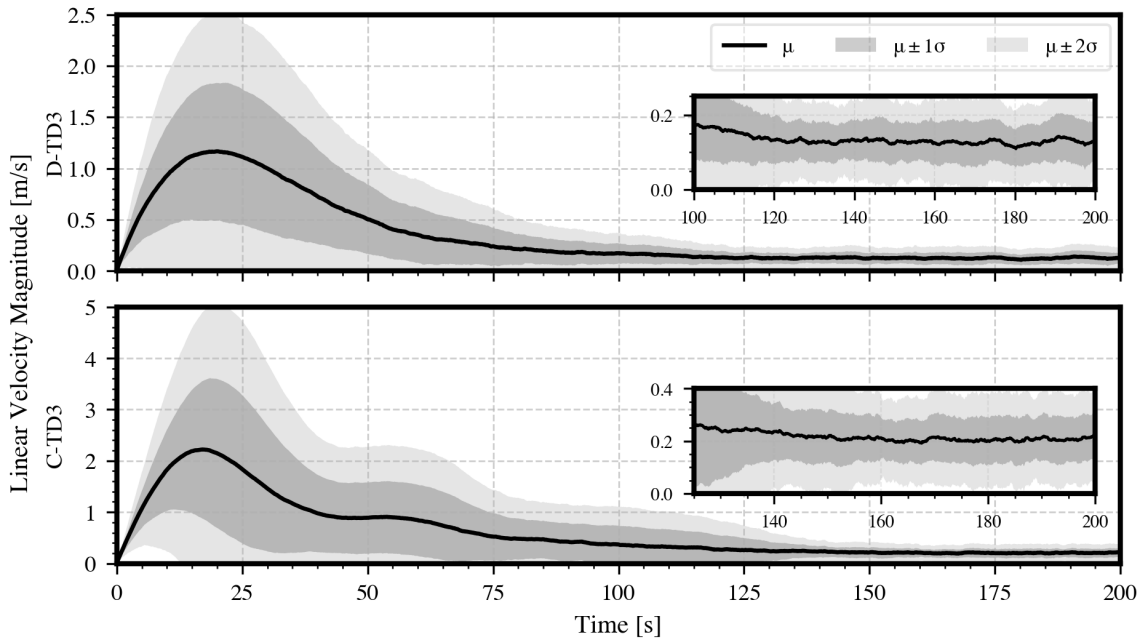


Figure 3.6: Linear velocity magnitudes for the D-TD3 and C-TD3 controllers across 100 trials.

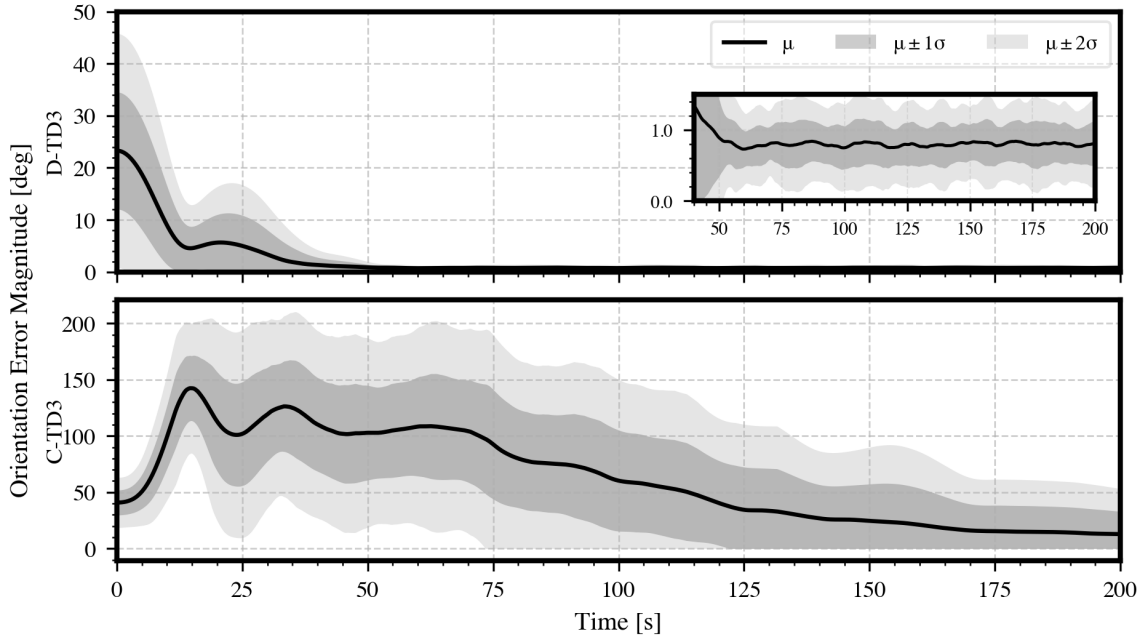


Figure 3.7: Orientation error magnitudes for the D-TD3 and C-TD3 controllers across 100 trials.

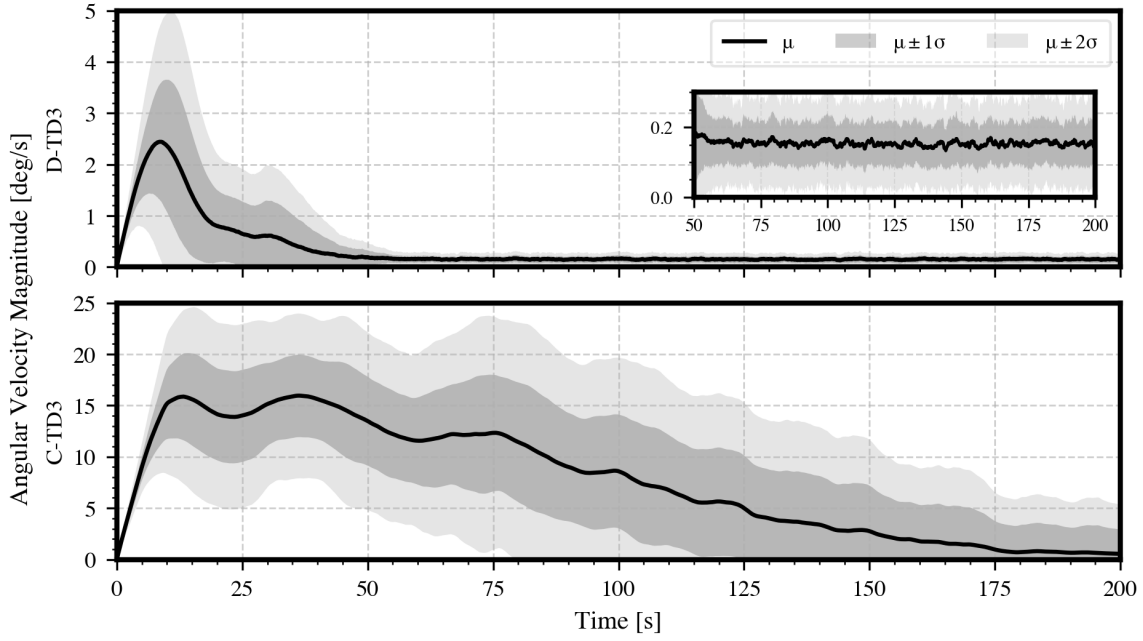


Figure 3.8: Angular velocity magnitudes for the D-TD3 and C-TD3 controllers across 100 trials.

The D-TD3 controller demonstrates stable and consistent behaviour throughout

the evaluation space. Final position errors averaged 2.22 ± 0.87 m, comfortably within the 5 m mission threshold. Peak linear velocities remained moderate at $1.24 \pm 0.66\ \mathrm{m\,s^{-1}}$, and the controller typically reached a translational steady-state in under 60 s. Orientation control exhibited similar consistency, with a final angular error of $0.80 \pm 0.31\ ^\circ$ and maximum angular velocities constrained to $2.81 \pm 1.12\ ^\circ\,\mathrm{s^{-1}}$. These results show that the decentralised controller is robust to a wide range of initial conditions.

In contrast, the C-TD3 controller exhibited significantly more variability. The mean final position error was 4.17 ± 1.05 m, which, while often within the mission requirement, approached the upper limit and showed greater episode-to-episode variance. Peak linear velocities reached $2.57 \pm 1.41\ \mathrm{m\,s^{-1}}$, often requiring longer periods to settle. Most notably, the final orientation error averaged $12.82 \pm 20.12\ ^\circ$, failing to reach the 2.5° requirement, with several trials failing to converge within acceptable time limits. The high standard deviation and angular rates exceeding $19\ ^\circ\,\mathrm{s^{-1}}$ indicate a lack of rotational stability.

These differences can be attributed to the architectural properties of each controller. In the C-TD3 agent, translational and rotational objectives are handled by a single policy operating in the full 13-dimensional state observation space. This coupling introduces interdependencies between subsystems, leading to control interference. By contrast, D-TD3 decomposes the task into two independently trained agents (6D and 7D subspaces), each specialising in a specific control domain. This separation prevents control objectives from interfering, leading to more stable and interpretable trajectories.

Moreover, the D-TD3 agent exhibits architectural fault isolation: performance degradation in one domain (e.g., rotation) does not propagate into the other (e.g., translation). This contrasts with C-TD3, where instabilities in one domain can cascade through shared control pathways, amplifying deviations and reducing overall reliability.

Overall, the Monte Carlo results support the selection of the decentralised architecture. The D-TD3 controller not only meets mission performance thresholds with greater consistency, but also exhibits the robustness and modularity desirable for deployment in real-world autonomous operations.

Direct comparison under shared initial conditions

To illustrate controller behaviour, a representative sample episode was selected with a fixed set of initial conditions for both controllers:

$[\mathbf{r}, \mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\omega}_{\mathcal{B}}] = [10, 45, 100, 0, 0, 0, 5, 30, 10, 0, 0, 0]$

where $\mathbf{r}$ and $\mathbf{v}$ denote relative position and velocity (in m and $\mathrm{m\,s^{-1}}$), $\boldsymbol{\theta}$ represents the initial orientation offset (in degrees), and $\boldsymbol{\omega}_{\mathcal{B}}$ is the initial angular velocity (in $^{\circ}\,\mathrm{s^{-1}}$).

The results for both controllers under these initial conditions are presented in Figures 3.9 – 3.14. These plots include the position and attitude errors, velocity profiles, and control command outputs for both controllers.

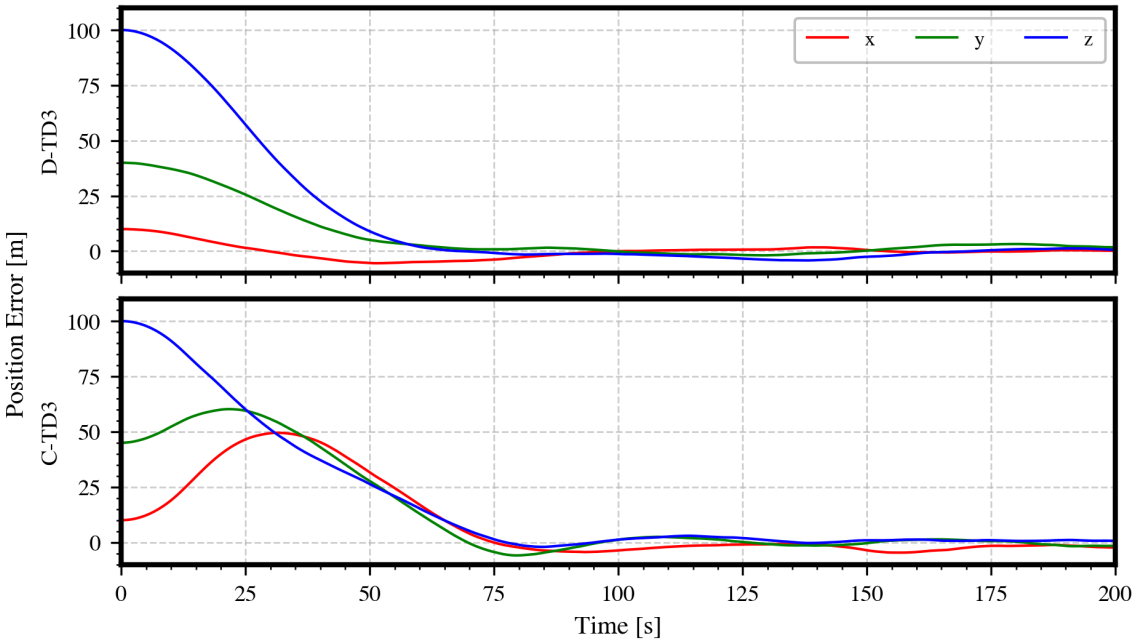


Figure 3.9: Position errors for the D-TD3 and C-TD3 controllers during a representative episode.

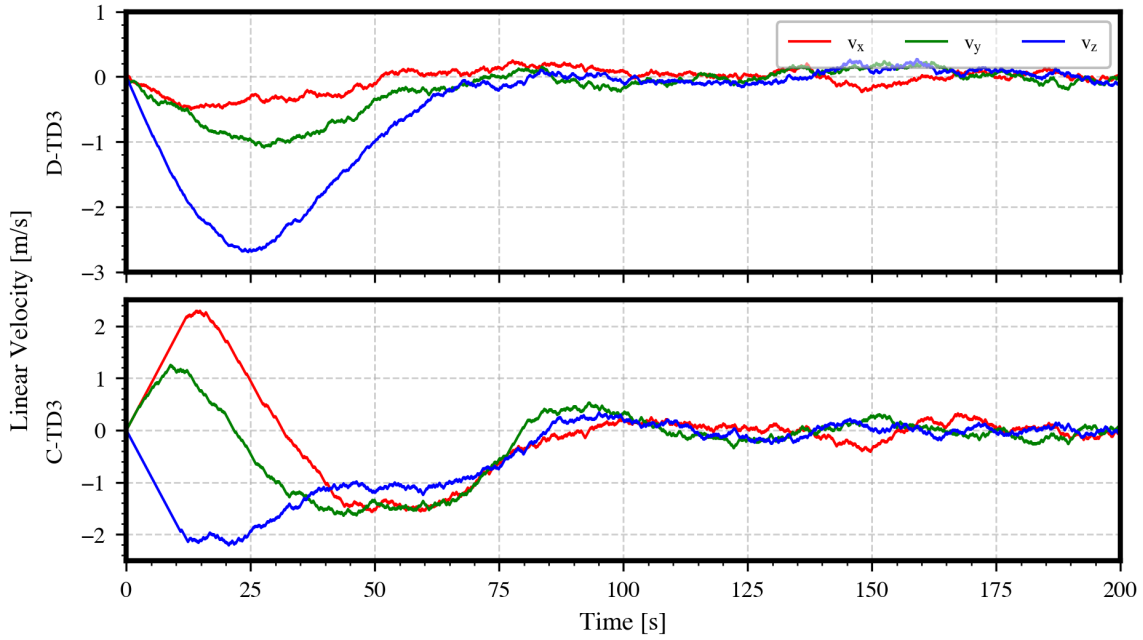


Figure 3.10: Linear velocities for the D-TD3 and C-TD3 controllers during a representative episode.

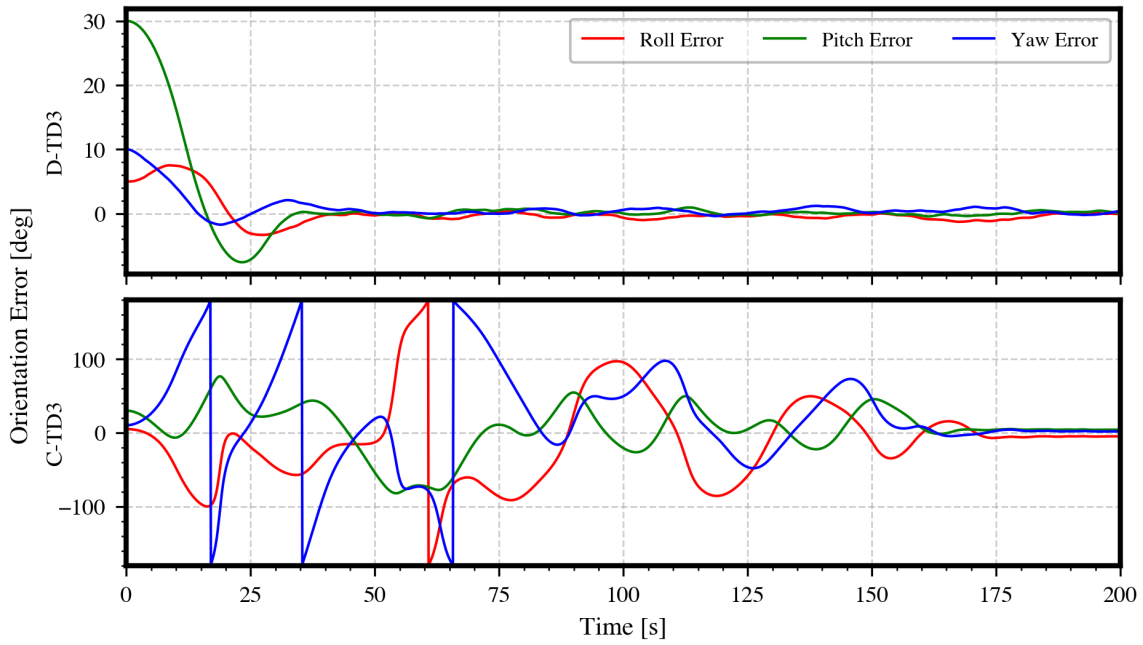


Figure 3.11: Orientation errors for the D-TD3 and C-TD3 controllers during a representative episode.

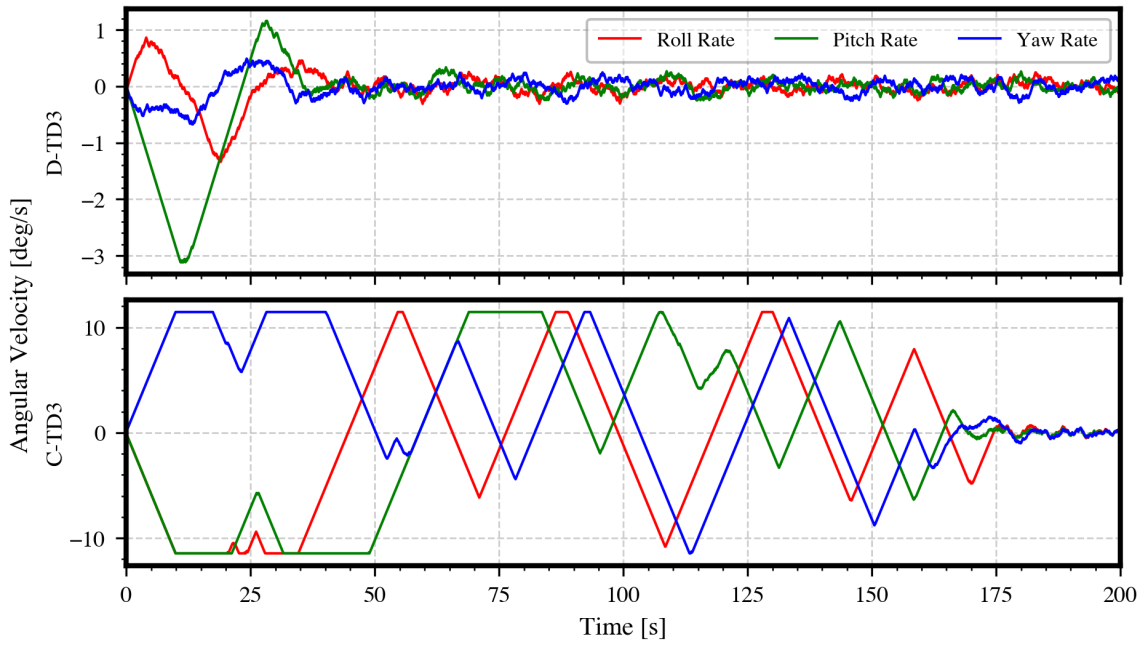


Figure 3.12: Angular velocities for the D-TD3 and C-TD3 controllers during a representative episode.

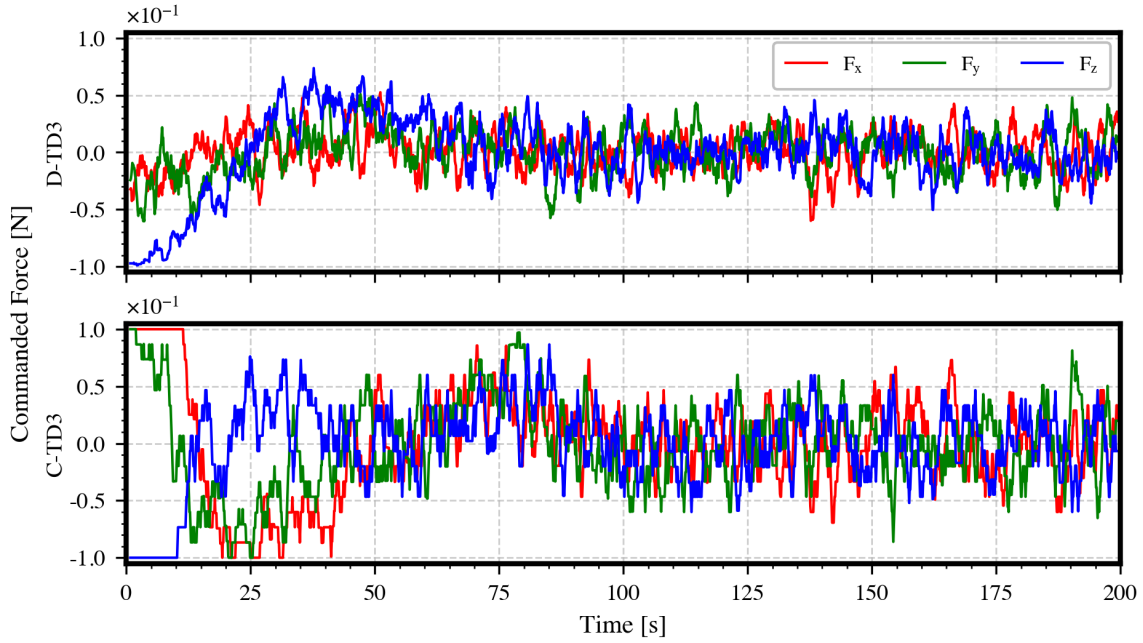


Figure 3.13: Commanded forces from the D-TD3 and C-TD3 controllers during a representative episode.

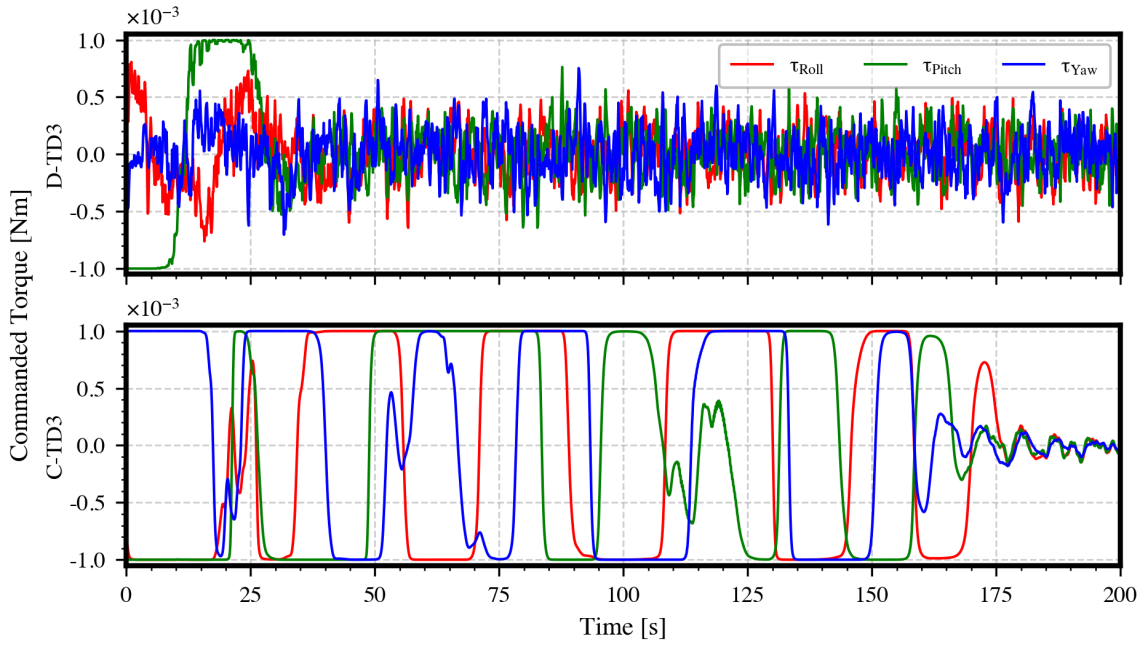


Figure 3.14: Commanded torques from the D-TD3 and C-TD3 controllers during a representative episode.

The D-TD3 controller demonstrated smooth and well-coordinated control across all channels. Position errors along all axes converged steadily with minimal overshoot, and linear velocities peaked at approximately 3 $\mathrm{m\,s^{-1}}$ before executing a controlled deceleration. Rotational convergence was similarly well-managed: angular velocities remained below 3 $^\circ\,\mathrm{s^{-1}}$, and attitude alignment was achieved within 45 s. The control inputs from both agents showed brief periods of saturation during initial corrections, followed by modulated, noise-responsive control as the system approached convergence and the signal-to-noise ratio decreased, making environmental disturbances more prominent relative to the diminishing errors.

In comparison, the C-TD3 controller exhibited more erratic behaviour, consistent with the trends identified in the Monte Carlo analysis. Translational convergence was delayed and oscillatory, with velocity peaking slightly above 3 $\mathrm{m\,s^{-1}}$, taking longer to decelerate and stabilise. Attitude control was more aggressive, with angular velocity exceeding 20 $^\circ\,\mathrm{s^{-1}}$ during the initial phase, leading to large rotational excursions before convergence. The commanded control forces and torques were broader in range and more irregular, reflecting a less confident policy and the challenges of resolving multiple control objectives simultaneously.

These results highlight the advantages of the decentralised architecture. The D-

TD3 controller's modular design enables clearer control allocation and reduces coupling between subsystems, resulting in more predictable and interpretable dynamics. C-TD3, constrained by the need to simultaneously satisfy competing objectives within a single policy, tends to produce control actions that overcompensate or interfere across domains, degrading both stability and efficiency.

This representative trial reinforces the conclusion that D-TD3 offers operational benefits not only in statistical performance but also in control behaviour, interpretability, and modularity, characteristics that are required for autonomous satellite guidance and control.

3.5 Summary

This chapter explored the use of deep reinforcement learning for satellite position and attitude control during close-range proximity operations. The focus was on developing autonomous control strategies for a 1U CubeSat performing a 6-DoF approach to a passive, non-cooperative target. The task represents a critical capability for missions involving on-orbit servicing, inspection, and debris removal, where precise autonomous control is essential. This work addressed the limitations of traditional control methods in dynamic or uncertain scenarios by leveraging DRL to learn control policies directly from data.

To evaluate architectural design choices, two control strategies were developed and compared: a decentralised controller (D-TD3), with separate agents for translation and rotation, and a centralised controller (C-TD3), which jointly optimises both using a single agent. Both agents were trained under symmetrical reward structures and simulation environments, with performance assessed across a range of initial conditions.

Simulation results showed that the decentralised controller consistently outperformed the centralised one in terms of stability, precision, and robustness. The D-TD3 agent achieved lower position and orientation errors, reduced control effort, and more consistent convergence behaviour across 100 randomised trials. These improvements were attributed to the architectural separation of control objectives, which reduced interference and enabled clearer allocation of control effort.

The centralised controller's poorer performance is consistent with learning interference between translational and rotational objectives within a single policy and critic, operating over a higher-dimensional state-action space. In this setting, the agent must

implicitly balance competing signals from the combined reward and resolve coupled credit assignment across force and torque outputs, which can lead to aggressive or inconsistent attitude actions even when translation is improving. The larger function approximation burden also increases sensitivity to hyperparameters and data requirements, making stable convergence harder to achieve under the same training budget.

Several avenues could improve the centralised approach while preserving an end-to-end structure. These include scheduling or rebalancing the reward terms over time to avoid one objective dominating the gradient signal, adopting a multi-objective formulation that explicitly balances translation and attitude objectives rather than combining them into a single summed reward, and using architectures that share early representations while separating later control heads for force and torque.

These results establish a baseline for the use of DRL in satellite guidance and control and form the foundation for the subsequent chapters of this thesis, where more advanced learning strategies are introduced to address degraded sensing, sparse rewards, and real-time safety constraints.

Chapter 4

Learning Fault-Tolerant Satellite Control with Embedded Safety Constraints

4.1 Motivation

Reinforcement learning offers a flexible framework for satellite guidance and control, with the ability to learn complex behaviours directly from interaction with a simulation environment. As shown in Chapter 3, deep reinforcement learning agents can learn effective control policies for small satellite. However, two limitations remain: the absence of formal safety guarantees and the difficulty of transferring learned policies from simulation to real-world systems. These problems are critical in the space domain, where safety violations carry high risk and on-orbit testing is infeasible.

DRL agents trained in unconstrained environments can produce unsafe or unstable actions when operating outside their training distribution. Without mechanisms to enforce safety these agents may violate operational constraints, especially in previously unseen scenarios. In addition, policies trained in simulation are unlikely to encounter exactly the same dynamics or conditions once deployed. These mismatches between simulated and real environments, often referred to as the sim-to-real gap, reduce trust in learned policies and make reliable deployment difficult.

This chapter addresses both concerns by introducing two components into the DRL control pipeline: adaptive domain randomisation (ADR) for improving robustness to variation and uncertainty, and relaxed control barrier functions (CBFs) for enforcing

formal safety constraints at runtime.

ADR is included to increase the generalisation ability of the trained policy across variations in system parameters. In contrast to standard domain randomisation, which samples from a fixed distribution, ADR gradually expands the range of environment parameters during training, effectively shaping a curriculum that exposes the agent to more difficulty as it improves.

To address the lack of safety guarantees, a relaxed control barrier function layer is added to the control system. CBFs operate as a post-processing step, modifying the agent's output in real-time to keep the system within a predefined safe set. Traditional CBFs strictly enforce these safety constraints, which can make them overly conservative or infeasible under limited actuation or model uncertainty. By introducing a slack variable, relaxed CBFs allow for constraint violations, which are penalised but not strictly prohibited. This makes them more practical in settings where occasional deviations are unavoidable. In this chapter, the relaxed CBF is applied during both training and testing. The safety layer filters the actions generated by the DRL agents and penalises deviations from the original proposed action in the reward function. This guides the agent to avoid unsafe regions of the action space during learning, improving policy alignment with the safety layer.

To assess the impact of these additions, three controller configurations are trained and evaluated:

- A Baseline controller trained with fixed domain randomisation and no safety layer.
- An Adaptive controller trained with adaptive randomisation but no safety layer.
- A Safe controller trained with both ADR and a relaxed CBF safety layer.

These controllers are trained on a close range rendezvous task using a CubeSat-scale satellite model. Performance is tested under both nominal conditions and a failure scenario, in which actuator and sensor systems degrade progressively over time. This scenario pushes each controller beyond its training distribution and evaluates its ability to maintain stability and satisfy safety constraints.

This chapter makes the following contributions:

1. The development of a modular DRL control architecture that integrates adaptive domain randomisation (ADR) with relaxed control barrier functions (CBFs) to produce policies that are both robust and safety-aware. The approach combines

curriculum-based training with real-time safety filtering and uses reward shaping to align learning with formal constraints, improving policy generalisation under uncertain dynamics.

2. A structured experimental framework for evaluating learning-based satellite controllers under both nominal and degraded conditions. The framework introduces a progressive degradation scenario that simulates compounding actuator and sensor failures, enabling systematic stress-testing beyond training distributions. By comparing the Safe controller against ablated variants (Baseline and Adaptive), the research isolates the specific contributions of ADR and CBF components, providing a clear, data-driven assessment of their impact on fault tolerance.

3. A reward shaping strategy that integrates relaxed CBF corrections to promote inherently safe policy learning. By penalising deviations between the agent's actions and the safety-corrected outputs, the training process encourages the development of policies that align with formal safety constraints. This reduces the frequency of runtime interventions and results in more compliant control, demonstrating a practical method for fusing learning and safety in reinforcement learning pipelines.

Together, these contributions advance the development of DRL-based satellite controllers that are both robust and safety-aware. By combining adaptive training with runtime safety enforcement, the proposed approach improves generalisation to unseen dynamics, enhances resilience under partial system failures, and further reduces dependence on exact model fidelity. This represents a step toward more trustworthy learning-based control systems for high-uncertainty autonomous space operations.

4.2 Literature review

4.2.1 Adaptive domain randomisation

Domain randomisation is a widely used strategy in reinforcement learning for robotics to address the sim-to-real transfer problem. The central idea is to train an agent across a distribution of simulated environments with randomized parameters so that the resulting policy becomes robust to discrepancies between simulation and the real world. By learning to operate across a wide range of conditions, the agent becomes less sensitive to modelling errors or unmodelled dynamics during deployment.

In its standard form, domain randomisation involves sampling parameters from a fixed distribution defined prior to training. These samples are typically drawn independently for each episode, without regard to the agent's performance. This approach has been successfully used in a variety of domains. For example, early work on visual domain randomisation demonstrated that randomising textures, lighting conditions, and camera parameters could enable deep visual policies trained entirely in simulation to generalise to real-world perception tasks without additional adaptation [119]. Similar strategies have been used in dynamics randomisation, where properties such as link masses, actuator gains, and joint friction are varied across training episodes to enable sim-to-real transfer of control policies [120]. More complex applications, such as dexterous manipulation, have shown that combining visual, physical, and sensor-level randomisation can support the successful transfer of high-dimensional control policies [121].

Despite its effectiveness, standard domain randomisation has several limitations. First, the use of a fixed randomisation range throughout training may lead to suboptimal coverage of the state space. If the initial parameter ranges are too narrow, the policy may not generalise beyond a small subset of scenarios. Conversely, overly wide ranges can introduce excessive variability, making it difficult for the agent to learn stable behaviours, especially early in training [122]. Moreover, random sampling does not guarantee that difficult or high-risk scenarios will be encountered frequently enough to guide policy development. Empirical analyses have also shown that overly broad domain randomisation windows can lead to inefficient training, increased sample complexity, and internal representations that are sensitive to network initialisation and perturbations [123].

Adaptive Domain Randomisation addresses these limitations by introducing a feedback-driven approach to environment parameter sampling. Instead of uniformly sampling from a fixed distribution, ADR adjusts the environment parameters dynamically based on feedback from the learning agent. The training distribution is adapted to prioritise more informative or challenging parameter sets, often using criteria such as performance gradients or success rates. This creates a form of curriculum learning where the agent initially trains on easier scenarios and gradually encounters more difficult ones as it improves, facilitating faster convergence and more robust generalisation.

Different strategies for ADR have been proposed. One approach monitors the agent's performance in randomised environments and compares it against a reference

environment. Parameter settings that cause the agent's behaviour to diverge from the reference are prioritised during training, ensuring that the policy is exposed to the most informative scenarios [124]. Others use optimisation-based objectives to guide the sampling process. For instance, Bayesian domain randomisation techniques use a limited set of real-world rollouts to update the simulator's parameter distribution via Bayesian optimisation, thereby improving sim-to-real transfer with fewer real-world interactions [125]. Another recent approach selects domain randomisation distributions using limited real world data by fitting simulation parameters that best match observed transitions, allowing policies to train entirely in simulation while still capturing real-world dynamics [126].

Compared to uniform domain randomisation, ADR offers several advantages. It mitigates the risk of underexposing the agent to more difficult sections of the state space and prevents over-randomisation that can destabilise learning. By aligning training difficulty with policy improvement, ADR improves sample efficiency and encourages broader generalisation. In robotics, it has been shown to reduce performance variance and enhance sim-to-real transfer even in high-dimensional or multimodal environments [124, 125, 126]. However, this adaptivity introduces additional complexity, requiring mechanisms to monitor performance and adjust distributions without destabilising training. Poorly tuned ADR schedules can result in stagnation or overfitting to specific curriculum patterns.

Overall, domain randomisation remains an effective tool for sim-to-real transfer in reinforcement learning, but its standard implementation has well-documented limitations. Adaptive domain randomisation enhances this technique by making the randomisation process adaptive, prioritising learning from the most challenging and informative scenarios. This improves the robustness and generalisation of learned policies and is particularly relevant in safety-critical domains such as autonomous satellite guidance.

4.2.2 Satellite Control Under Faults

Maintaining control despite faults is a critical requirement for the autonomous operation of satellite, where hardware degradation or subsystem failure can compromise performance or lead to mission loss. In the space environment, faults in actuators, sensors, or other subsystems can arise from a range of causes, including mechanical degradation, thermal cycling, radiation exposure, and manufacturing defects [127]. These risks highlight the need for onboard control architectures that can maintain

safe operation even under degraded conditions.

Failures within the Attitude and Orbit Control System (AOCS) are particularly impactful, as they directly affect a satellite's ability to maintain orientation and trajectory control. Among AOCS components, reaction wheels are particularly vulnerable to mechanical wear over extended mission duration. As reaction wheels accumulate excess momentum over time they must be periodically desaturated using torque-exchange devices such as magnetorquers or thrusters. If these momentum management systems fail or prove inadequate, the wheels may saturate and become incapable of applying further control torques.

Mechanical degradation presents another common failure mode. The bearings and lubricants within reaction wheels degrade over time due to wear, temperature fluctuations, and vacuum exposure, resulting in increased friction, power consumption, and reduced control authority. In more severe cases, complete actuator failure may occur [128]. Environmental factors such as differential surface charging have also been identified as contributing to friction events and long-term performance degradation [129].

Thrusters, used for both attitude and orbital manoeuvres, are susceptible to different but equally critical vulnerabilities. Thermal cycling, particulate contamination, and mechanical fatigue can lead to inconsistent valve behaviour, resulting in delayed or partial firings. These irregularities introduce uncertainty in force production, directly affecting stability and trajectory tracking. In addition, propellant leakage due to seal degradation or material fatigue may cause uncommanded thrusts, leading to trajectory drift and loss of precise orbital control. [127].

Sensors within the AOCS are also susceptible to degradation and malfunction, often due to radiation exposure, thermal cycling, or aging electronics [130]. Common issues include gyroscope drift, star tracker blinding or failure, and increased magnetometer noise [131]. Faulty or intermittent sensor measurements can corrupt the system's state estimates and propagate errors through the control loop, increasing the risk of instability or unsafe actions. Robust control strategies must therefore be resilient not only to actuator degradation but also to unreliable feedback from navigation sensors.

Fault Tolerant Control (FTC) refers to a class of control strategies designed to maintain system stability and acceptable performance in the presence of faults or failures. FTC is typically divided into two main categories: passive and active. Passive FTC designs a robust controller that tolerates expected faults without needing to reconfigure. In contrast, active FTC involves real-time fault detection and controller

adaptation to accommodate the detected fault. Common options include observer-based methods, adaptive control, and model predictive control with constraints that reflect degraded capabilities [132, 133, 134, 135, 136, 137]. FTC is widely applied in areas where reliability and safety are critical, and its effectiveness generally relies on accurate fault modelling and real-time diagnostics.

The range and frequency of faults encountered in satellite systems underscore the need for control architectures that remain effective under partial failures and degraded conditions. While traditional fault-tolerant control approaches provide important mechanisms for preserving stability and performance, they often depend on explicit fault models, predefined recovery strategies, or tightly integrated with fault identification. These constraints can limit adaptability, particularly in unforeseen failure scenarios. Reinforcement learning offers a pathway to address these limitations by enabling flexible, data-driven control. When combined with real-time safety enforcement, it can extend the flexibility of FTC frameworks and improve resilience to both modelled and unmodelled faults.

4.2.3 Control barrier functions

Safety is a critical concern in autonomous control systems, particularly in aerospace applications where unmodelled dynamics, actuator limitations, and sensor degradation can result in unsafe behaviour or mission failure. To address this, several techniques have been developed to enforce safety constraints during training or deployment. These include model predictive control (MPC), Lyapunov-based methods, and control barrier functions, each offering different trade-offs between safety, flexibility, and computational cost.

MPC-based safety layers use a model of the system to solve an optimization problem at each timestep. The agent's action is either replaced or adjusted to ensure that predicted future states remain within defined constraints. This offers strong safety guarantees when using an accurate model [138]. Extensions using Gaussian Processes or probabilistic models account for uncertainty and allow risk-aware control in systems with limited knowledge of dynamics [139, 140, 141]. However, MPC-based methods rely heavily on the quality of the system model and are often computationally expensive, particularly for real-time embedded systems like satellite.

Lyapunov-based shields ensure safety by enforcing global stability through scalar Lyapunov functions that decrease along system trajectories. In reinforcement learning, they are used to translate global safety objectives into local linear constraints that can

be embedded into policy updates [142]. This allows policies to remain within predefined safe regions during both learning and deployment. Earlier work has shown that switching among pre-designed Lyapunov-stable controllers maintains safety without restricting learning [143], and more recent approaches combine Lyapunov functions with other safety mechanisms like CBFs to handle model uncertainty [144]. While effective, these methods are typically conservative, often limiting exploration near safety boundaries and reducing flexibility in dynamic environments.

Control barrier functions provide a lower-overhead alternative. CBFs enforce forward invariance of a safe set by restricting the system's control input to maintain a barrier function $h(x) \geq 0$ over time (see Figure 4.1). The resulting constraint can be enforced through a Quadratic Program (QP) at each timestep, making CBFs well-suited for online correction of RL agent actions. A typical implementation is shown in Figure 4.2, where the control signal from the primary system is modified by a CBF layer to ensure safety.

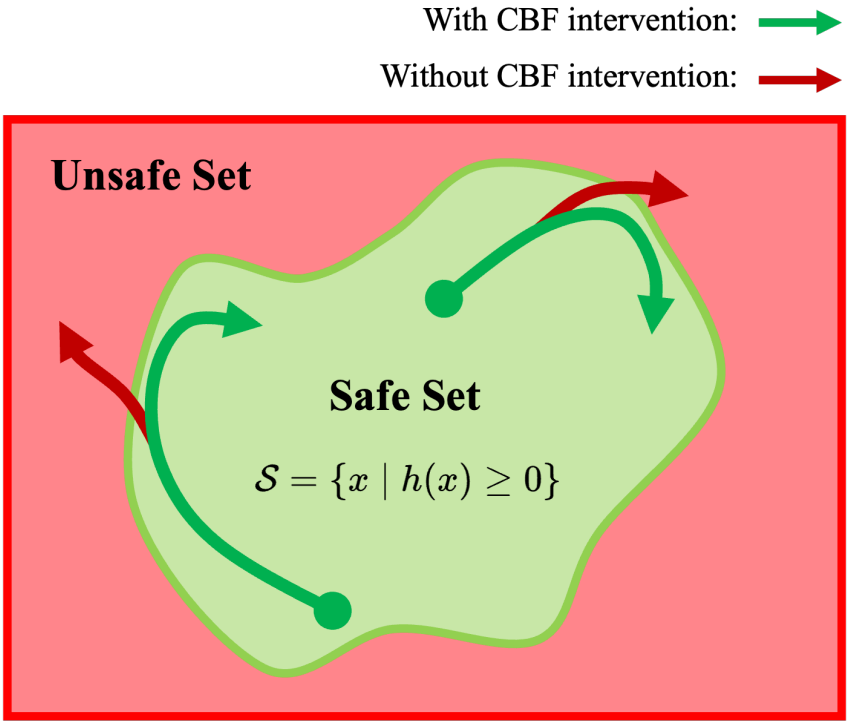


Figure 4.1: Illustration of a safe set defined by a Control Barrier Function, where the system state x must remain within the set $S = x \mid h(x) \geq 0$. Without CBF intervention (red), the trajectory exits the safe region; with CBF intervention (green), the trajectory is modified to remain within the safe set.

CBF-based safety layers have been successfully applied in RL settings, where they

act as a post-processing step to modify unsafe actions while preserving learning performance [145]. Some methods incorporate CBF constraints directly into the training process by modifying the reward function or using hybrid controllers [146, 147]. Robust extensions also account for input limits and model uncertainty [148].

The barrier function $h(x)$ defines a set of admissible states:

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid h(x) \geq 0\} \tag{4.1}$$

Given a control-affine system of the form:

$$\dot{x} = f(x) + g(x)u \tag{4.2}$$

where $x \in \mathbb{R}^n$ is the system state, $u \in \mathbb{R}^m$ is the control input, and f, g are locally Lipschitz continuous, safety can be enforced by ensuring that:

$$\frac{d}{dt}h(x) = L_f h(x) + L_g h(x)u \geq -\alpha(h(x)) \tag{4.3}$$

where $L_f h(x)$ and $L_g h(x)$ denote the Lie derivatives of h along f and g, and $\alpha(\cdot)$ is an extended class-$\mathcal{K}$ function (commonly $\alpha(h) = \kappa h$ for some $\kappa > 0$). This inequality ensures that as the system evolves, the state remains inside the safe set $\mathcal{S}$.

In practice, CBFs are often implemented by solving a quadratic program that minimally modifies a desired control input u generated by a primary controller (such as a DRL policy) to ensure safety. This optimization finds the closest safe control u^* to the original command.

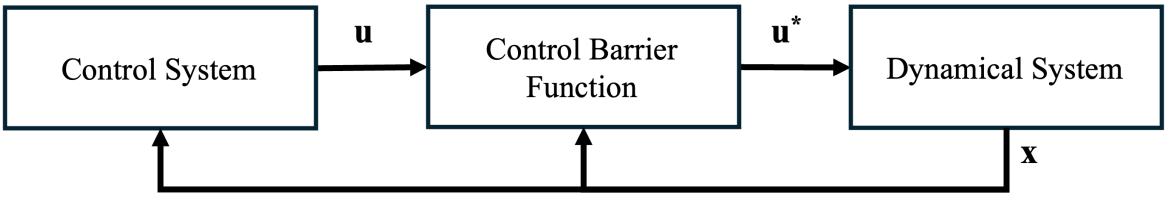


Figure 4.2: Block diagram of a control loop augmented with a Control Barrier Function. The nominal control input $\mathbf{u}$ is adjusted by the CBF to produce a modified control signal $\mathbf{u}^*$, ensuring safety constraints are satisfied by the dynamical system.

While classical CBFs strictly enforce the condition $h(x) \geq 0$ at all times, this can be overly restrictive, particularly in the presence of unmodelled disturbances, noise, or temporary actuator limitations. Relaxed Control Barrier Functions introduce a slack variable $\delta \geq 0$ into the safety constraint, enabling the controller to temporarily violate the barrier condition in a controlled manner:

$$L_f h(x) + L_g h(x) u + \alpha(h(x)) \geq -\delta \tag{4.4}$$

This leads to a relaxed quadratic program of the form:

$$\begin{aligned} u^* = \underset{u,\,\delta}{\operatorname{argmin}} \; & \frac{1}{2} \, ||u - u_{RL}||^2 + \rho \, \delta^2 \\ \text{subject to} \; & \begin{cases} L_f h_i(x) + L_g h_i(x) u + \alpha\big(h_i(x)\big) \geq -\delta \\ \delta \geq 0 \end{cases} \end{aligned} \tag{4.5}$$

where the relaxation is introduced by replacing the strict barrier inequality with a softened constraint that permits bounded violation through the slack variable $\delta \geq 0$. In effect, when the original constraint would be infeasible (for example due to actuation limits or model mismatch), the QP remains solvable by allowing the constraint to be violated by an amount δ, while penalising this violation via the quadratic term $\rho\delta^2$. The solution u^* is therefore the closest admissible control to the policy output u_{RL}, with feasibility preserved by the slack term and large violations discouraged through the penalty weight ρ.

In the context of this thesis, relaxed CBFs are integrated as a post-processing safety layer on top of a DRL policy. At each control step, the proposed action u_{RL} is passed through a QP-based correction mechanism that enforces the relaxed CBF condition. This approach preserves the exploration and learning dynamics of DRL while providing real-time safety enforcement during training and deployment.

This formulation provides a practical way to enforce safety in DRL-based systems. By using efficient quadratic programs, It enables real-time filtering of actions without significantly increasing computational overhead. Unlike strict safety mechanisms, the use of a relaxation term allows the system to remain feasible even in edge cases, such as when actuator limits or model inaccuracies would otherwise make the constraints unsatisfiable. Rather than failing outright, the controller can continue operating while penalising any safety violations, enabling graceful degradation under uncertainty. Its lightweight and modular structure also makes it well suited to integrate into existing reinforcement learning pipelines without altering the core training loop.

Together, these properties make relaxed control barrier functions a compelling choice for safe reinforcement learning in aerospace contexts. When combined with robust training strategies, they support adaptive and resilient control of autonomous satellite operating under uncertainty. By integrating reinforcement learning with CBF-based safety layers, this work contributes a fault-tolerant control framework that cou-

ples the adaptability of learning with the formal guarantees of safety filtering, enabling reliable operation even under actuator or sensor degradation.

4.3 Methodology

4.3.1 Environment setup

This section outlines the simulation environment used to develop and evaluate the proposed guidance and control system. The simulation models a 1U CubeSat satellite equipped with representative actuation and sensing systems, and captures both nominal and degraded operating conditions. The environment incorporates various actuator and sensor non-idealities to more closely reflect realistic performance. This environment serves as the foundation for training and testing the controllers presented later in this chapter.

The satellite is modelled as a rigid body with a total mass of 0.5 kg and a principal moment of inertia of $0.001\text{kg}\,\text{m}^2$ on each axis, a simplification appropriate for a symmetric 1U form factor. Full 6-DoF motion is simulated providing the necessary fidelity for assessing controller performance across diverse operational scenarios.

The control task targets a close range rendezvous manoeuvre with an initial relative distance of up to 25m from the target and a pointing error of up to 60°, without assuming any specific alignment along principal axes. This scenario approximates the initial phase of several proximity operation missions where the satellite must autonomously approach a passive object and establish a stable relative pose. The goal is to reduce the separation distance to under 1m and achieve an angular alignment within 2.5°. These thresholds define a terminal condition suitable for handover to a fine-pointing, docking, or capture controller. The coarse controller is therefore responsible for navigating through the more uncertain and dynamic regime handling a wide range of initial conditions while remaining within safe operational boundaries. Final contact, docking dynamics, and station-keeping are not modelled. Instead, the agent's success is evaluated by its ability to consistently reduce both position and orientation errors to within the specified thresholds under a wide range of conditions.

Attitude Control Subsystem

The satellite's attitude is regulated using a set of three orthogonally mounted reaction wheels. Each wheel is capable of producing a maximum torque $\tau_{\max} = 1$ mN m, a

maximum momentum storage capacity $h_{\max} = 5$ mN m s, and a maximum rotational speed $\omega_{\max} = 5000$ rpm. These limits are chosen based on specifications from commercial off-the-shelf CubeSat-grade reaction wheels [149, 150]. To improve simulation fidelity, the reaction wheel model includes several key non-idealities.

The reaction wheel motor dynamics such as electrical time constants, back-EMF effects, and current control bandwidth were approximated by imposing a torque rate limit to constrain how rapidly the wheel can change its torque output. The torque rate was bounded by

$$\left| \frac{d\tau_{\mathrm{rw},i}}{dt} \right| \leq \gamma_{\mathrm{rw}} \tag{4.6}$$

where $\gamma_{\mathrm{rw}} = 0.05$ N m s^{-1} is based on the conservative assumption that a wheel capable of producing a maximum torque of 1mN m would require at least 20ms to ramp from zero to full torque. This estimate reflects practical limitations imposed by small-scale motor drivers and current control loops found in CubeSat-class systems. The constraint is enforced by clipping any changes in commanded torque that exceed this rate limit.

To reflect actuator response limitations, a first-order lag was introduced between the commanded and actual torque outputs of the reaction wheels. This lag approximates the effects of motor inductance, finite response time of the motor drive electronics, and signal processing delays present in reaction wheel systems. The lag dynamics are modelled as:

$$\boldsymbol{\tau}_{\mathrm{applied}}(t) = \alpha\, \boldsymbol{\tau}_{\mathrm{command}}(t) + (1 - \alpha)\, \boldsymbol{\tau}_{\mathrm{applied}}(t - \Delta t) \tag{4.7}$$

where $\alpha = \frac{\Delta t}{\tau_{\mathrm{rw}} + \Delta t}$, with Δt denoting the control time step and τ_{rw} representing the actuator time constant. For the simulations in this study, $\alpha = 0.9$ was used, corresponding to a response time constant of $\tau_{\mathrm{rw}} = 10$ ms.

To represent small fluctuations in torque due to quantisation errors, current ripple, and mechanical disturbances, a zero-mean Gaussian noise term was added to the applied torque. This approach is used as the aggregate effect of multiple independent noise sources can be approximated by a Gaussian process. The noisy torque is modelled as:

$$\boldsymbol{\tau}_{\mathrm{noisy}} = \boldsymbol{\tau}_{\mathrm{applied}} + \boldsymbol{\eta}_{\tau}, \quad \boldsymbol{\eta}_{\tau} \sim \mathcal{N}(\mathbf{0},\, \sigma_{\tau}^2) \tag{4.8}$$

where $\sigma_\tau = 0.01$ mN m defines the noise standard deviation. This value was selected to introduce visible but bounded stochasticity in the actuator response.

The wheels' stored angular momentum is tracked throughout the simulation, and any torque commands that would exceed the maximum stored momentum are clipped. This prevents the wheels from accumulating momentum beyond their physical limits and emulates actuator saturation behaviour.

To approximate the effect of a momentum dumping system, an exponential decay model was applied to the stored angular momentum vector. This approach represents the gradual reduction in wheel momentum typically achieved via magnetic torque rods or other actuators over time. The momentum is updated at each time step according to:

$$\mathbf{h}(t) = \mathbf{h}(t - \Delta t)\, \exp(-\lambda \Delta t) \tag{4.9}$$

where $\lambda = 0.01$ governs the rate of decay.

This formulation captures the effect of continuous momentum unloading while avoiding the need to explicitly model external momentum dumping actuators, whose detailed dynamics are not relevant within the scope and time horizon of this mission.

Position Control Subsystem

Position control is handled via a compact thruster system capable of producing thrust along each axis. Each axis is actuated by a pair of opposing thrusters, each capable of producing a maximum continuous thrust command of $f_{max} = 0.05$ N. To reflect the behaviour of a high throughput CubeSat-class thrusters, several modifications are applied.

Thrust quantisation was applied to the commanded force vector to reflect the effect of a minimum impulse bit (MIB). At each control step, commanded thrust values are rounded to the nearest quantised level:

$$\mathbf{f}_{\text{quantised}} = f_{\text{MIB}} \cdot \left\lfloor \frac{\mathbf{f}_{\text{command}}}{f_{\text{MIB}}} + 0.5 \right\rfloor \tag{4.10}$$

The granularity $f_{\text{MIB}} = 1$ mN s is selected based on the minimum impulse bit of cold gas and electric propulsion systems rated for use on 1U cubesats [151, 152].

To account for the response characteristics of the thruster system, a first-order lag was applied to the commanded force vector, emulating valve response time and delays in flow regulation. The lag was implemented in the same form as for the reaction

wheels, using a time constant of $\tau_{\text{th}} = 10$ ms.

Additionally, zero-mean Gaussian noise was added to model random variations in thrust output due to valve jitter, flow instability, or partial blockages. The noisy thrust is given by:

$$\mathbf{f}_{\text{noisy}} = \mathbf{f}_{\text{applied}} + \boldsymbol{\eta}_f, \quad \boldsymbol{\eta}_f \sim \mathcal{N}(\mathbf{0}, \sigma_f^2) \tag{4.11}$$

where $\sigma_f = 5$ mN defines the standard deviation of thrust fluctuations and was chosen to reflect small but persistent disturbances within the thruster system.

Sensing and Navigation Subsystem

The agent's observation space comprises measurements of relative position $\mathbf{r}$, relative velocity $\mathbf{v}$, angular displacement $\boldsymbol{\theta}$, and angular velocity $\boldsymbol{\omega}$. During training, the agent is given ideal, noise-free observations. However, to evaluate performance under more realistic sensing conditions each observation is corrupted with zero-mean Gaussian noise. The noisy measurement model is defined as:

$$\hat{\mathbf{x}} = \mathbf{x} + \boldsymbol{\eta}, \quad \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}^2) \tag{4.12}$$

where the noise standard deviation σ varies by signal type:

- $\sigma_{\text{pos}} = 0.05$ m
- $\sigma_{\text{vel}} = 0.01\ \text{m}\,\text{s}^{-1}$
- $\sigma_{\text{att}} = 0.1^\circ$
- $\sigma_{\text{rate}} = 0.05^\circ\text{s}^{-1}$

To emulate the effects of onboard signal conditioning and sensor processing pipelines, all signals are passed through a first-order low-pass filter:

$$x_{\text{filtered}}(t) = \beta \cdot x(t) + (1 - \beta) \cdot x_{\text{filtered}}(t - 1) \tag{4.13}$$

with $\beta = 0.9$ used across all channels. This introduces a small lag representative of digital filtering delays.

4.3.2 Controller architecture

The satellite guidance and control system is implemented as a modular, learning-based architecture composed of two independent deep reinforcement learning agents: one responsible for translational control and the other for rotational control. Each agent is trained to operate in a distinct control domain using task-specific observations and outputs, allowing the system to decouple attitude and position regulation tasks. This architectural decision is motivated by the findings presented in Chapter 3, which demonstrated that separating the control of translational and rotational dynamics results in improved training stability, interpretability, and robustness when compared to a combined control policy.

The attitude agent is responsible for regulating the satellite's orientation and angular velocity, while the position agent governs relative motion with respect to the target frame. Both agents produce incremental control actions that represent desired changes in the respective velocity states along each axis. The attitude agent outputs $\Delta\boldsymbol{\omega}_{\text{cmd}} \in \mathbb{R}^3$, specifying the required adjustment in angular velocity, which is passed through a control barrier function safety enforcement layer and then mapped to torque commands for the reaction wheel system. Similarly, the position agent outputs $\Delta\mathbf{v}_{\text{cmd}} \in \mathbb{R}^3$, indicating the desired change in translational velocity. This command is also filtered through the CBF layer before being converted into thrust commands for the propulsion system.

Each agent operates independently and is trained in isolation from the other, using domain-specific state observations, action spaces, and reward functions tailored to their respective objectives. During inference, however, the agents operate concurrently. At each control timestep, the satellite's observed state is used to update both agents, which then produce their respective control outputs.

Both the attitude and position agents employ the TD3 algorithm, as implemented in the Stable Baselines3 framework [153]. Each agent uses a policy network composed of two hidden layers with 400 and 300 units, respectively. The hidden layers use the ReLU activation function, followed by a final linear output layer that is is scaled to match the action bounds of the relevant actuator:

- For the Attitude Agent: $\Delta\boldsymbol{\omega}_{\text{cmd}} \in [-2.5^\circ/\text{s}, 2.5^\circ/\text{s}]$
- For the Position Agent: $\Delta\mathbf{v}_{\text{cmd}} \in [-0.1, 0.1]\ \text{m}\,\text{s}^{-1}$

The critic architecture follows a similar structure, with twin Q-networks each using two hidden layers of the same size (400 and 300 units). To encourage exploration

during training, action noise is added to the policy output. A zero-mean Gaussian noise process with a standard deviation of 0.1 is used for each action. This helps the agent explore the continuous action space more thoroughly and avoid premature convergence to suboptimal policies.

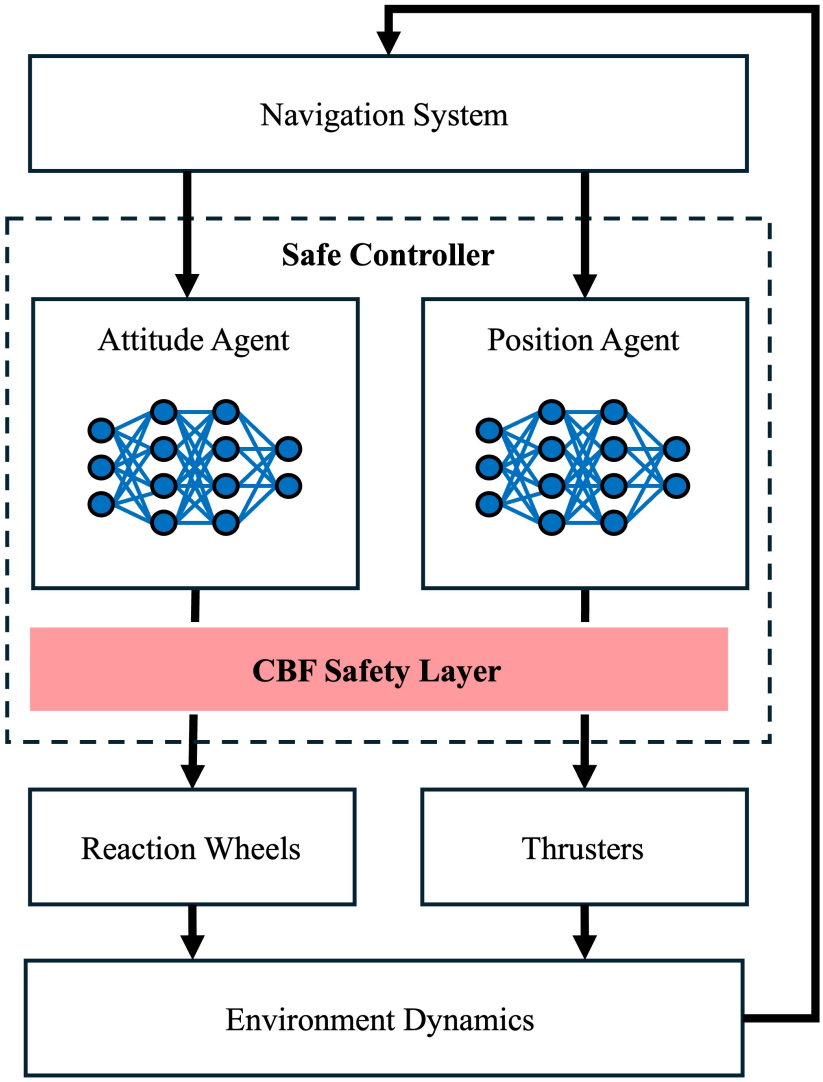


Figure 4.3: Structure of the Safe Controller. Two DRL agents operate simultaneously on their respective control domains, generating nominal actions that are passed through a Control Barrier Function (CBF) safety filter. The filter enforces safety constraints and outputs admissible control commands, which are then applied to the corresponding actuators.

4.3.3 Reward shaping with relaxed control barrier functions

The performance and safety of a reinforcement learning-based control system are critically influenced by the design of its reward function. In the absence of well-defined feedback, agents may converge to unstable or unsafe behaviours. To guide learning towards mission-relevant objectives while maintaining operational constraints, structured reward shaping was combined with feedback from a relaxed CBF safety layer. The reward function incentivises goal-directed behaviour while penalising unsafe or inefficient control. The CBF layer enforces formal safety constraints through real-time action correction and integrates directly with the reward signal to influence agent behaviour during training.

Reward Function Design

Separate reward functions are defined for the attitude and position agents. Both functions are composed of three core components: an exponential proximity term, a dynamic efficiency term, and an actuator penalty term. The overall structure is defined as follows:

$$r_a = \exp(-2\,\|\boldsymbol{\theta}(t)\|) - 0.1 \cdot \frac{\|\boldsymbol{\omega}(t)\|}{\|\boldsymbol{\theta}(t)\| + 0.01} - 0.005 \cdot \|\mathbf{a}(t)\| \tag{4.14}$$

$$r_p = \exp(-0.09\,\|\mathbf{r}_t\|) - 0.75 \cdot \frac{\|\mathbf{v}_t\|}{\|\mathbf{r}_t\| + 0.01} - 0.1 \cdot \|\mathbf{a}_t\| \tag{4.15}$$

The first term in both expressions provides a high-resolution shaping signal that encourages convergence to the target state. The exponential decay penalises large errors while rewarding proximity. The decay constants were selected to balance learning stability and convergence speed.

The second term penalises high linear or angular velocities, particularly when far from the target. The use of a division by $\|\theta\| + 0.01$ (and analogously for position) causes the penalty to decrease as the agent approaches its goal, implicitly rewarding smoother deceleration near the target and reducing overshoot.

The third term penalises actuator effort, measured as the magnitude of the command vector $\mathbf{a}$. This promotes energy-efficient control and discourages oscillatory or aggressive actions, which are often associated with instability and high fuel consumption.

In addition to the continuous reward, bonus rewards are applied when goal conditions are met. These include:

- A bonus of +1 if the relative distance is below 1m
- A bonus of +1 if the angular error is below 2.5°

These thresholds align with the final proximity corridor requirements, ensuring the agent is explicitly rewarded for remaining within the proximity thresholds.

Relaxed Control Barrier Functions

To formally enforce safety constraints and enhance robustness under uncertainty, a Relaxed CBF mechanism is integrated into the control architecture. Unlike hard-constraint methods, relaxed CBFs allow temporary, bounded violations via a slack variable. This flexibility makes them compatible with learning-based controllers, which may occasionally produce suboptimal or exploratory actions.

A penalty weight of $\rho = 1000$ is used in this study, selected via heuristic tuning to strike a balance between safety adherence and policy flexibility.

The CBF layer enforces four constraints on the satellite's motion, formulated to promote both safety and task stability:

1. Maximum velocity constraints: Prevent excessively fast motions that could lead to instability or overshoot.

$$h_{1,\text{pos}}(v) = v_{\max}^2 - ||v||^2 \tag{4.16}$$
$$h_{1,\text{att}}(\omega) = \omega_{\max}^2 - ||\omega||^2 \tag{4.17}$$

   where $v_{\max} = 0.5\ \text{m}\,\text{s}^{-1}$, and $\omega_{\max} = 5^\circ\,\text{s}^{-1}$.

2. Minimum approach constraints: These are activated only when the agent enters its target threshold (within 1 m in position and 2.5° in attitude). Once inside this region, the constraints prevent excessive overshoot or reversing past the target.

$$h_{2,\text{pos}}(r) = ||r||_{\min}^2 - ||r||^2 \tag{4.18}$$
$$h_{2,\text{att}}(\theta) = ||\theta||_{\min}^2 - ||\theta||^2 \tag{4.19}$$

where $||r||^2_{\min}$ and $||\theta||^2_{\min}$ are boundaries set slightly above the best-achieved accuracy (by 0.1 m and 0.25° respectively). This allows limited deviation while ensuring the agent remains close to its goal without oscillating or reversing.

In both cases, the barrier conditions are reformulated as linear inequalities in the control input $\mathbf{u}$, which allows the QP to be solved at each timestep. This formulation ensures real-time feasibility while preserving the safety guarantees of the CBF layer. The resulting set of active constraints are:

$$v^\top u \geq \frac{\alpha}{2}(v^2_{\max} - ||v||^2) + \delta_{1,\text{pos}} \tag{4.20}$$

$$r^\top u \leq \frac{(||r||_{\min} + 0.1)^2 - ||r||^2}{2\Delta t} - r^\top v + \delta_{2,\text{pos}} \tag{4.21}$$

$$\omega^\top u \geq \frac{\alpha}{2}(\omega^2_{\max} - ||\omega||^2) + \delta_{1,\text{att}} \tag{4.22}$$

$$\theta^\top u \leq \frac{(||\theta||_{\min} + 0.25)^2 - ||\theta||^2}{2\Delta t} - \theta^\top \omega + \delta_{2,\text{att}} \tag{4.23}$$

Reward Function Integration

To promote compatibility between the learned policy and the safety mechanism, the CBF correction signal is incorporated directly into the reward function. At each timestep, the agent receives an additional penalty proportional to the deviation between its proposed action $\mathbf{u}_{\text{RL}}$ and the final action $\mathbf{u}^*$ produced by the QP:

$$r_{\text{cbf}} = -\lambda \cdot |\mathbf{u}^* - \mathbf{u}_{\text{RL}}| \tag{4.24}$$

A value of $\lambda = 5$ was selected to ensure the penalty is sufficient to discourage unsafe actions without dominating the task-relevant reward. This feedback discourages the agent from repeatedly relying on the CBF to correct invalid behaviours and encourages it to learn policies that remain within the safe set on their own.

This approach leads to a convergence pattern in which the CBF intervenes frequently during early training but is rarely activated in later stages. The agent learns to avoid unsafe regions of the state-action space not by memorising specific cases, but by adapting its policy to align with the underlying safety constraints.

The combined use of structured reward shaping and relaxed CBF enforcement allows the guidance system to learn task-optimal policies that are inherently safe and robust to disturbances. The reward function ensures that the agents pursue precise

and efficient control strategies, while the CBF layer guarantees that safety boundaries are never persistently violated. The integration of CBF penalties into the reward function further reinforces this alignment, resulting in agents that not only optimize performance but also cooperate effectively with formal safety mechanisms.

4.3.4 Adaptive Training Framework

The training pipeline is designed to produce robust and safety-aware policies using progressively more challenging environments. Agents are initially trained in a simplified environment with ideal actuators and observations. There are three controller configurations considered in this work:

- Baseline: trained using the full range of domain parameters from the start with no modifications or additions.
- Adaptive: trained using adaptive domain randomisation to encourage robustness and generalisation.
- Safe: trained with both adaptive domain randomisation and the CBF safety layer.

The control frequency is set at 10Hz, while the underlying simulation is integrated at 100Hz to preserve dynamic fidelity. Episode durations are set at a maximum of 100 control steps for the attitude agent and 250 steps for the position agent, corresponding to 10s and 25s of simulated time respectively. Each agent is trained independently for a maximum of 1 million timesteps, with learning progress and evaluation metrics logged every 25 training episodes.

Adaptive Domain Randomisation

Agents trained with ADR are exposed to a gradually expanding distribution of environmental parameters. Unlike static domain randomisation, which may over-constrain or overwhelm the agent with variability from the start leading to it learning overly conservative policies, ADR incrementally increases difficulty as performance improves. This forms a curriculum-like structure in which the training distribution shifts in response to policy competence.

The ADR mechanism used in this work monitors the agent's average reward over recent episodes. If the reward exceeds a predefined threshold, the randomisation

bounds for a randomly selected parameter is expanded by a fixed increment, this is repeated until all parameters are at their maximum randomisation range. If the reward drops below a performance floor, domain expansion is paused to allow policy recovery. This adaptive pacing stabilises learning and prevents catastrophic forgetting or overfitting to narrow scenarios.

Table 4.1 summarises the parameters selected for randomisation, along with their initial and final bounds and the incremental step size. The ranges were selected to reflect environmental conditions and actuator variability that the agent could encounter. Parameters were selected to ensure representative coverage of nominal conditions without inducing excessive instability during early training.

Table 4.1: Adaptive Domain Randomisation Parameter Ranges

Parameter	**Initial Range**	**Final Range**	**Step Size**
Initial Position [m]	[10, 10]	[0, 25]	0.5
Initial Velocity [m/s]	[0, 0]	[–0.4, 0.4]	0.01
Maximum Thrust [N]	[0.1, 0.1]	[0.05, 0.25]	0.01
Initial Attitude [deg]	[30, 30]	[0, 60]	1.0
Initial Angular Velocity [°/s]	[0, 0]	[–4, 4]	0.1
Maximum Torque [Nm]	[0.001, 0.001]	[0.0005, 0.005]	0.0005

This curriculum allows the agent to gradually master basic motion planning before encountering edge cases such as underpowered actuators or aggressive initial conditions. In doing so, ADR mitigates the limitations of fixed-range domain randomisation and improves both convergence stability and generalisation.

4.4 Performance Evaluation Under Nominal and Degraded Conditions

Controller performance is evaluated under two testing conditions: nominal operation and a progressive failure cascade. The nominal scenario assesses standard task execution under realistic but fixed system dynamics, enabling consistent comparison across controllers. The failure cascade introduces time-varying disturbances that push the system far outside the training distribution, testing robustness under unexpected and unmodelled degradation.

Under nominal conditions, the simulation environment follows the actuator and

sensor models described in Section 4.3.1. Actuator dynamics, sensor noise, and non-idealities such as latency, quantisation, and rate limits are applied as fixed parameters representative of CubeSat-class systems. Controllers are tested over 100 trials, each with randomised initial position and attitude errors of up to 25 m and 60°, respectively. Testing in this configuration allows comparison of controller performance under realistic but stable conditions.

In contrast, the failure scenario is designed to simulate worst-case degradation across both sensing and actuation subsystems. Each of the 100 test episodes begins with the satellite in a converged state, with zero relative error and no residual velocity. After 25 seconds, the failure cascade is triggered. From that point onward, both actuator and sensor noise levels increase linearly with time. The rate of increase is randomly selected at the start of each episode to introduce variability across trials. Simultaneously, actuator and sensor latency are worsened by increasing their respective lag time to new values, also randomly sampled per episode.

This progressive degradation introduces a dynamic and uncertain disturbance regime that is not encountered during training. The intent is to evaluate controller resilience under adverse, unmodelled conditions. Controllers are judged based on their ability to tolerate failures while limiting velocities and errors despite compounding noise, delayed feedback, and reduced control authority. This setup allows for systematic comparison of controller performance and fault tolerance beyond the nominal envelope.

4.4.1 Nominal Performance

The nominal testing scenario evaluates the Baseline, Adaptive and Safe controller configurations, under stable conditions with fixed actuator and sensor parameters. Over 100 randomised trials, each controller is assessed based on convergence speed, final accuracy, dynamic behaviour, and control efficiency. A summary of quantitative results is presented in Table 4.2, with performance and distribution plots shown in Figures 4.4 – 4.11.

Table 4.2: Summary statistics for each controller across 100 trials in the nominal scenario. Values are reported as mean ± standard deviation over episodes.

Metric	**Baseline**	**Adaptive**	**Safe**
Final Position Error [m]	0.049 ± 0.029	0.172 ± 0.163	0.125 ± 0.076
Max Linear Velocity [m/s]	0.768 ± 0.329	0.917 ± 0.377	0.439 ± 0.079
Time to Convergence (Position) [s]	21.81 ± 12.93	24.11 ± 12.48	36.36 ± 21.95
Final Orientation Error [deg]	0.413 ± 0.226	0.341 ± 0.216	0.682 ± 0.219
Max Angular Velocity [deg/s]	7.249 ± 3.478	7.594 ± 4.146	4.463 ± 1.381
Time to Convergence (Orientation) [s]	9.97 ± 4.36	10.94 ± 7.74	13.56 ± 7.94

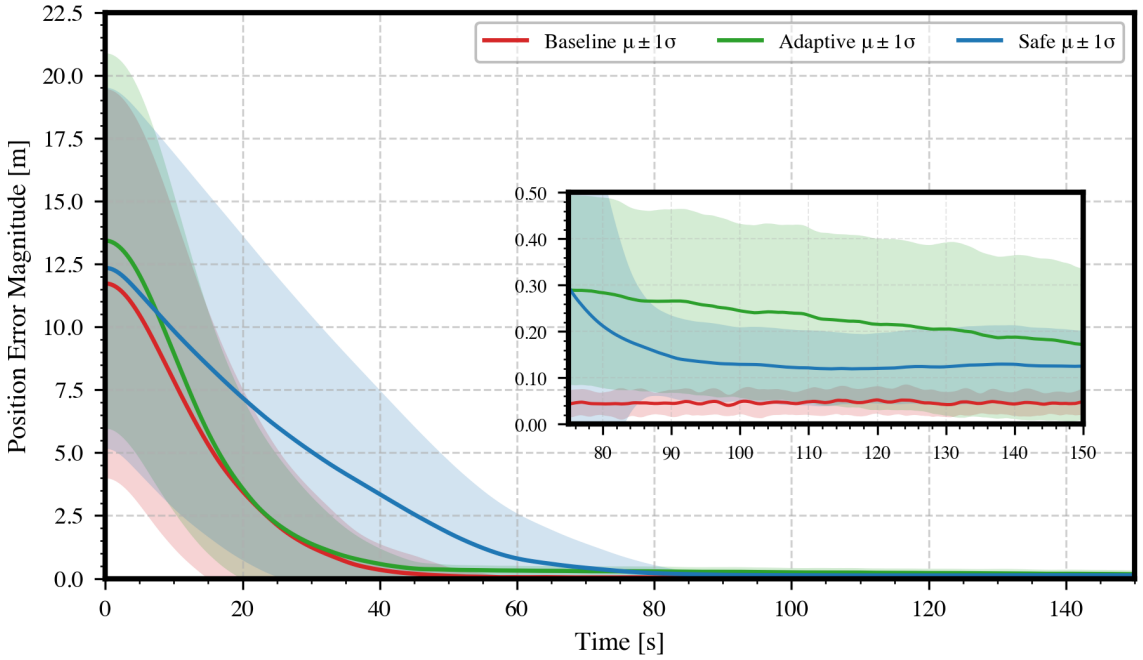


Figure 4.4: Position error magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under nominal conditions.

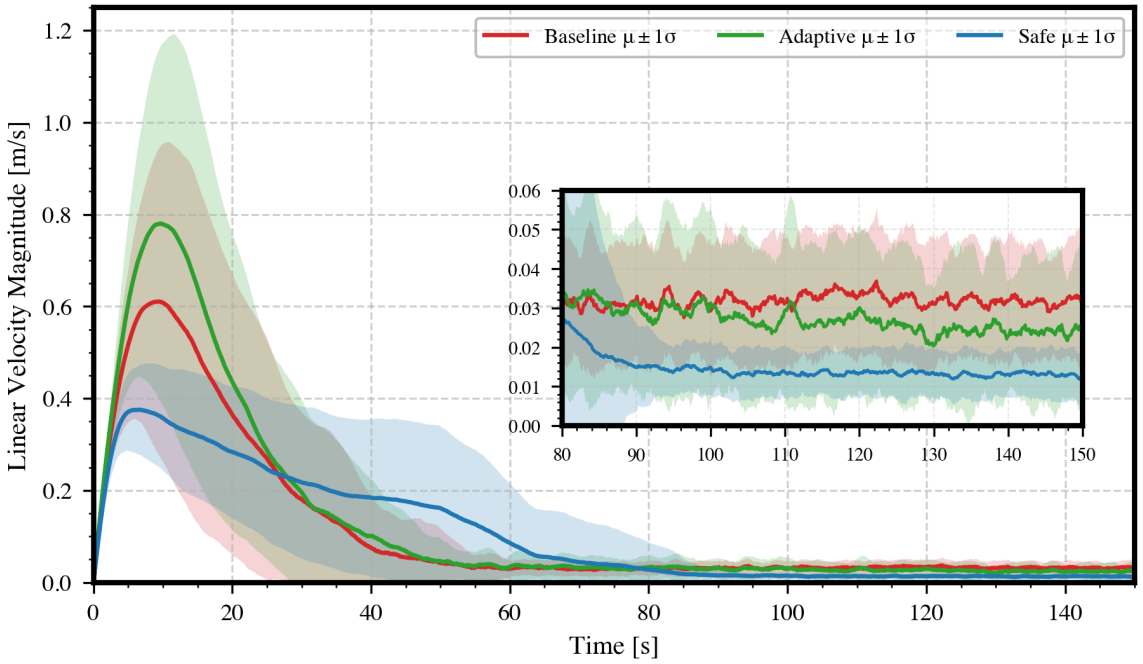


Figure 4.5: Linear velocity magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under nominal conditions.

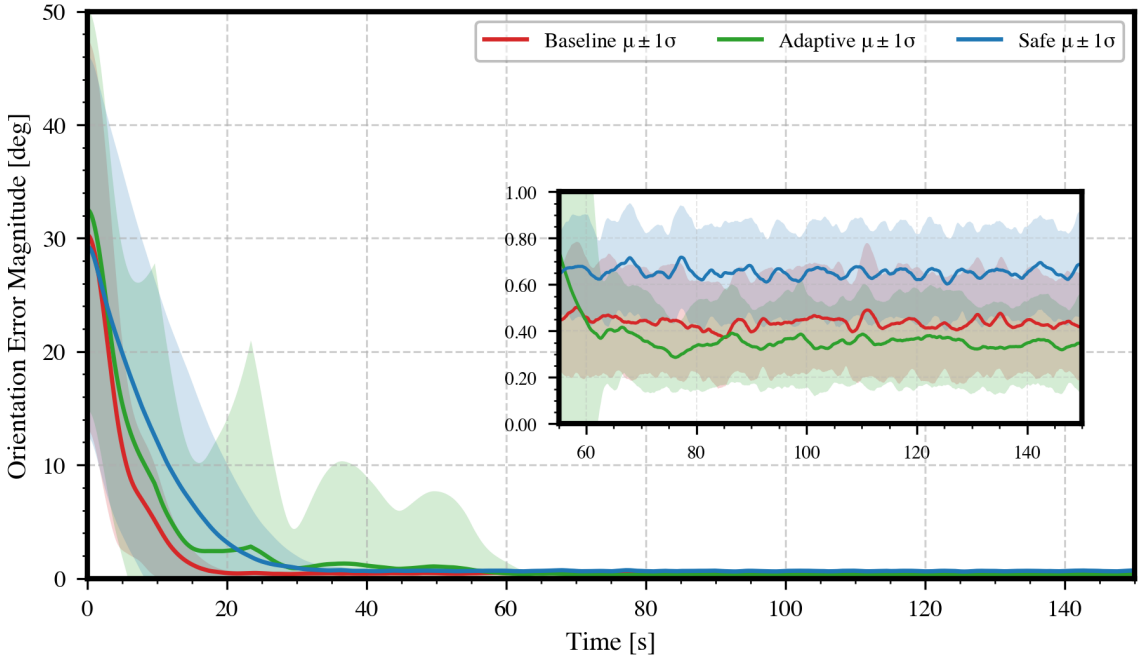


Figure 4.6: Orientation error magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under nominal conditions.

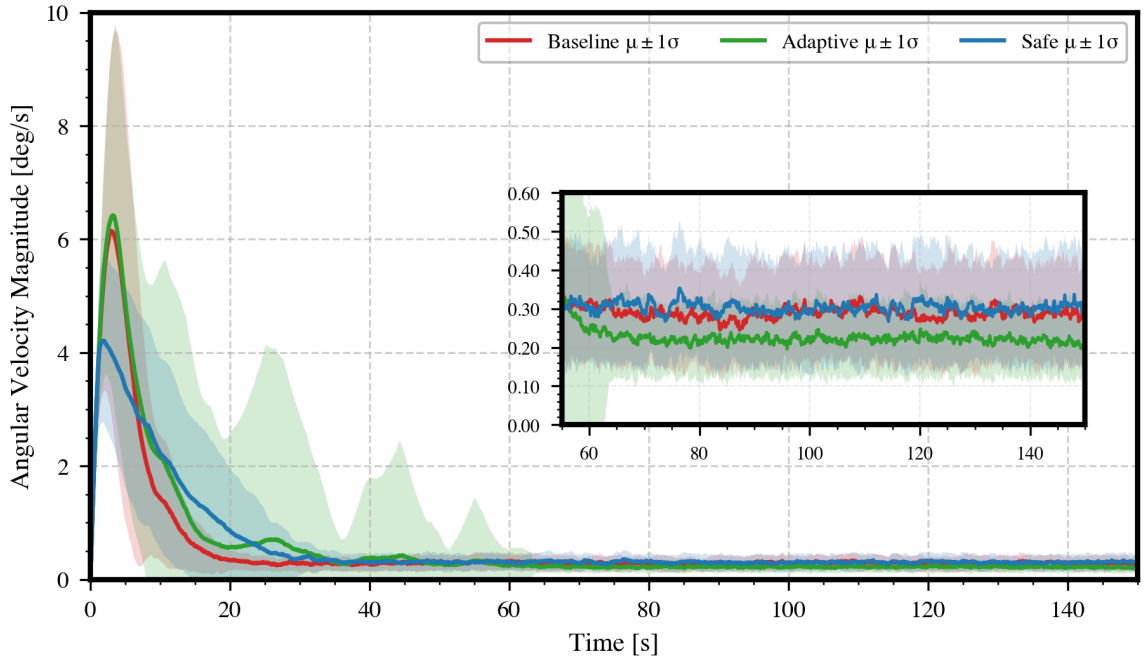


Figure 4.7: Mean angular velocities for the Baseline, Adaptive and Safe controllers over 100 trials under nominal conditions.

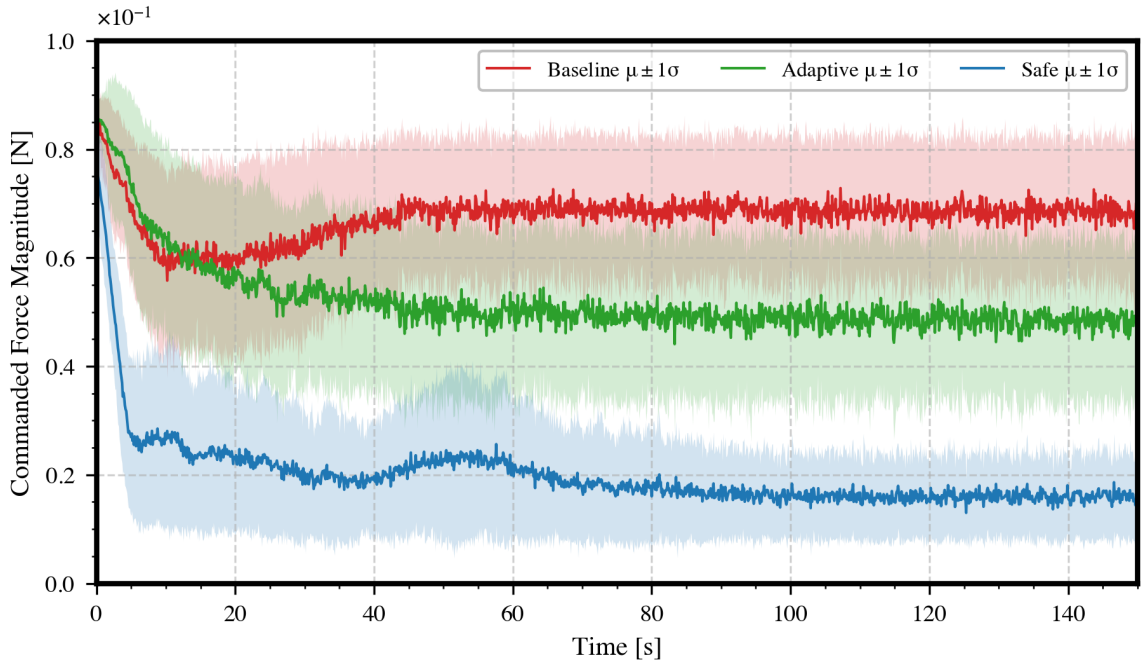


Figure 4.8: Commanded force magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under nominal conditions.

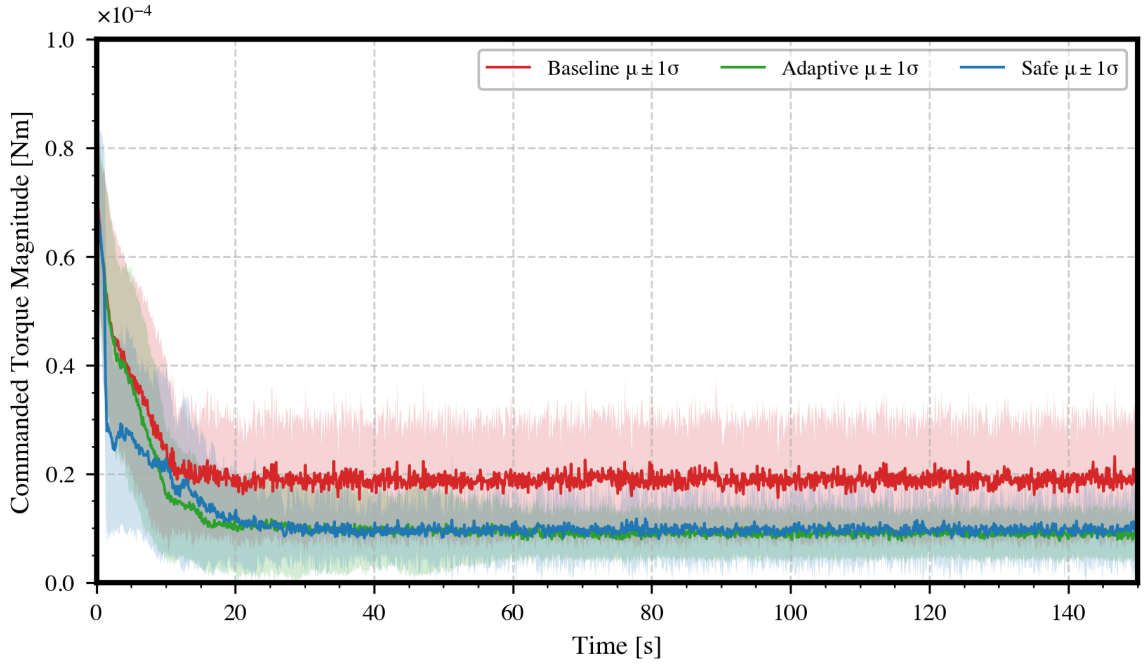


Figure 4.9: Commanded torque magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under nominal conditions.

All three controllers successfully completed the rendezvous task under nominal conditions. As shown in Figure 4.4, each controller reliably reduced position error magnitudes to below 0.25 m, well within the terminal condition threshold. The Baseline controller achieved the highest positional accuracy, with a final mean error of 0.049 ± 0.029 m, followed by the Safe controller (0.125 ± 0.076 m) and the Adaptive controller (0.172 ± 0.163 m). The Safe controller took longer to converge (36.36 ± 21.95 s) compared to the Baseline (21.81 ± 12.93 s) and Adaptive (24.11 ± 12.48 s) configurations, suggesting that the integrated CBF constraints led to a more cautious trajectory, prioritising safety over speed.

Velocity profiles (Figure 4.5) support this interpretation. The Safe controller maintained average maximum linear velocities below the imposed CBF limit of 0.5 $\mathrm{m\,s^{-1}}$, with a peak of 0.439 ± 0.079 $\mathrm{m\,s^{-1}}$, in contrast to the Baseline and Adaptive configurations, which reached 0.768 ± 0.329 $\mathrm{m\,s^{-1}}$ and 0.917 ± 0.377 $\mathrm{m\,s^{-1}}$, respectively.

Orientation regulation was consistently achieved across all controllers. As shown in Figure 4.6, all configurations reduced angular error to below 1°, satisfying the orientation threshold. The Adaptive controller reached the lowest final orientation error at $0.341 \pm 0.216°$, followed by the Baseline ($0.413 \pm 0.226°$) and Safe ($0.682 \pm 0.219°$) controllers. Convergence times followed a similar trend, with the Safe configuration again

being the slowest (13.56 ± 7.94 s). Angular velocity profiles in Figure 4.7 indicate that the Safe controller also imposed smoother rotational corrections, with peak angular velocities constrained to 4.463 ± 1.381 °/s, below those of the Baseline (7.249 ± 3.478) and Adaptive (7.594 ± 4.146) configurations.

Differences in control efficiency are evident in the commanded force and torque magnitudes. As shown in Figure 4.8, the Safe controller stabilised the satellite using notably lower force magnitude during steady state, approximately 0.015 N on average, compared to the Baseline and Adaptive controllers, which continued to issue forces of 0.05–0.07 N after convergence. Torque usage (Figure 4.9) followed a similar pattern, with the Safe controller maintaining lower average torque magnitudes (0.1 mN m) than the Baseline (0.2 mN m), reflecting more conservative and stable control behaviour.

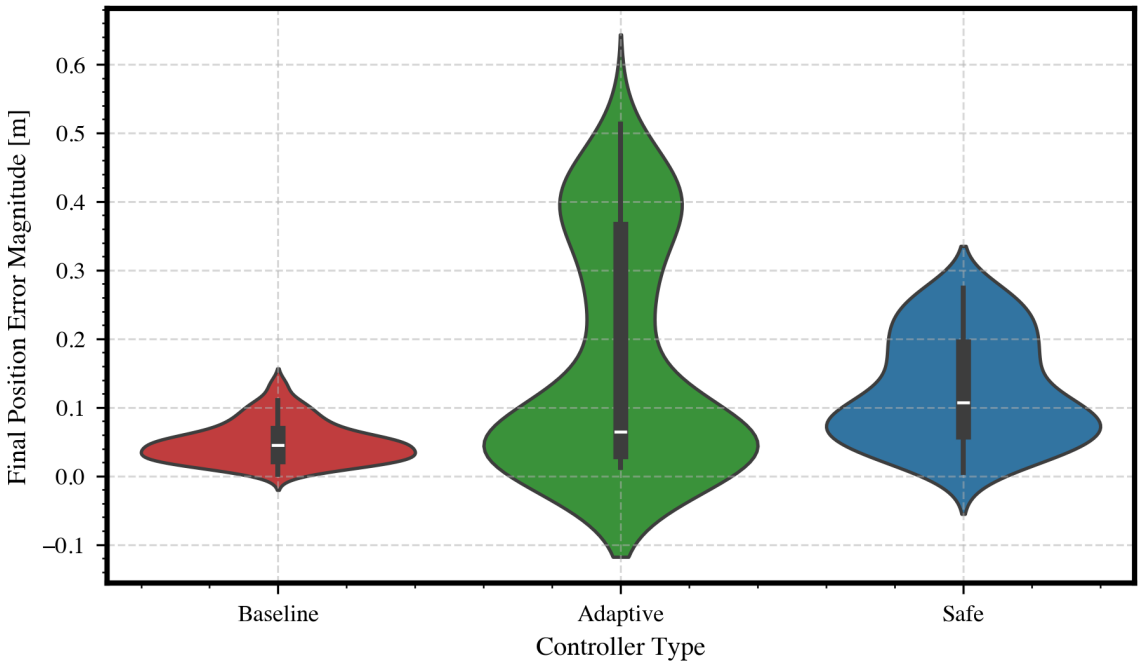


Figure 4.10: Violin plot showing the distribution of final position error magnitudes for the Baseline, Adaptive, and Safe controllers across 100 trials under nominal conditions.

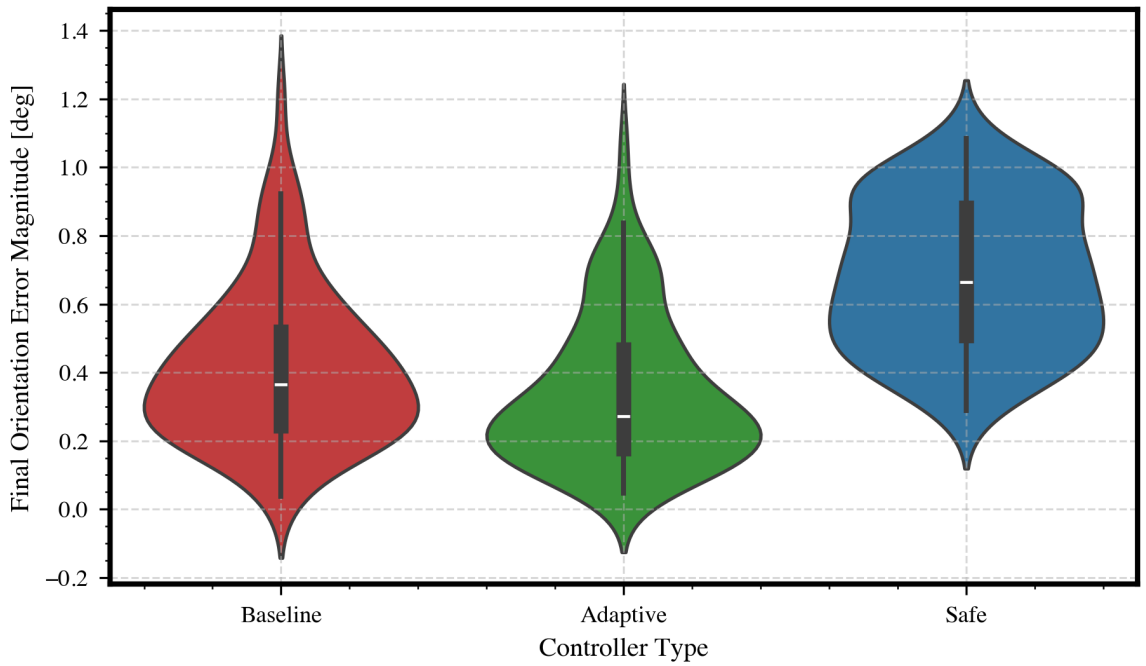


Figure 4.11: Violin plot showing the distribution of final orientation error magnitudes for the Baseline, Adaptive, and Safe controllers across 100 trials under nominal conditions.

Distribution plots further highlight these trends. The violin plot in Figure 4.10 shows that the Baseline controller achieved the lowest distribution of final position errors, while the Adaptive controller displayed greater variance, despite a comparable mean. The Safe controller's distribution lies between the two, demonstrating improved reliability relative to the Adaptive controller while maintaining a safety-compliant profile. For final orientation error (Figure 4.11), the Safe controller's slightly higher mean error is offset by its narrower distribution compared to the other two, suggesting more predictable convergence despite slower response.

Overall, the nominal scenario results confirm that all three controllers can successfully complete the rendezvous task within acceptable thresholds. The Baseline controller prioritised speed and precision but relied on aggressive actuation, while the Adaptive controller traded some stability for generalisation, showing a greater variability across trials. In contrast, the Safe controller delivered consistently cautious, compliant behaviour with less control effort. However, this came with a trade-off in the form of slower convergence and slightly reduced accuracy. Notably, even though the CBF layer was never actively triggered during execution, the Safe controller behaved more cautiously as a result of having been trained with CBF in the loop. This

conservative behaviour highlights the influence of safety-aware training on policy development. These findings establish a solid reference point for evaluating controller performance under more challenging or uncertain conditions.

4.4.2 Failure Scenario

The failure scenario introduces progressive and compounding disturbances to both the actuator and sensor models, pushing the system far outside the training distribution. This section evaluates the robustness of the Baseline, Adaptive, and Safe controllers under these conditions. Results are drawn from 100 randomised trials, each beginning in a stable, zero-error state and transitioning into failure after 25 seconds. Figures 4.12 – 4.21 illustrate the system behaviour throughout the episode, highlighting divergence, stability, and the effect of safety filtering via the CBF layer.

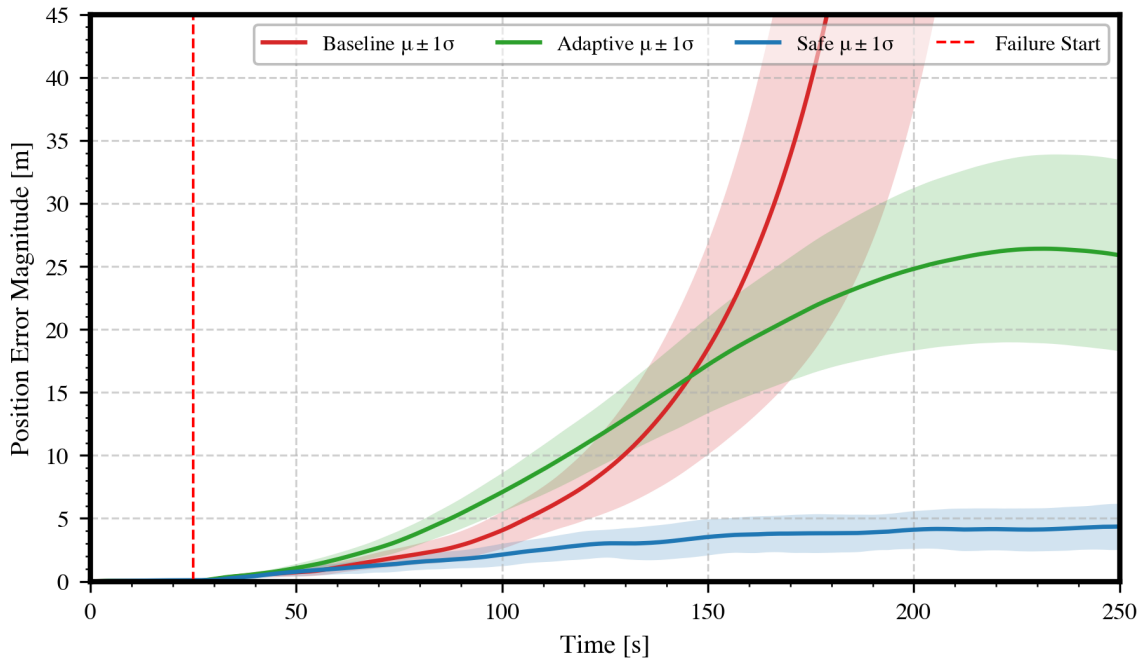


Figure 4.12: Position error magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under failure conditions.

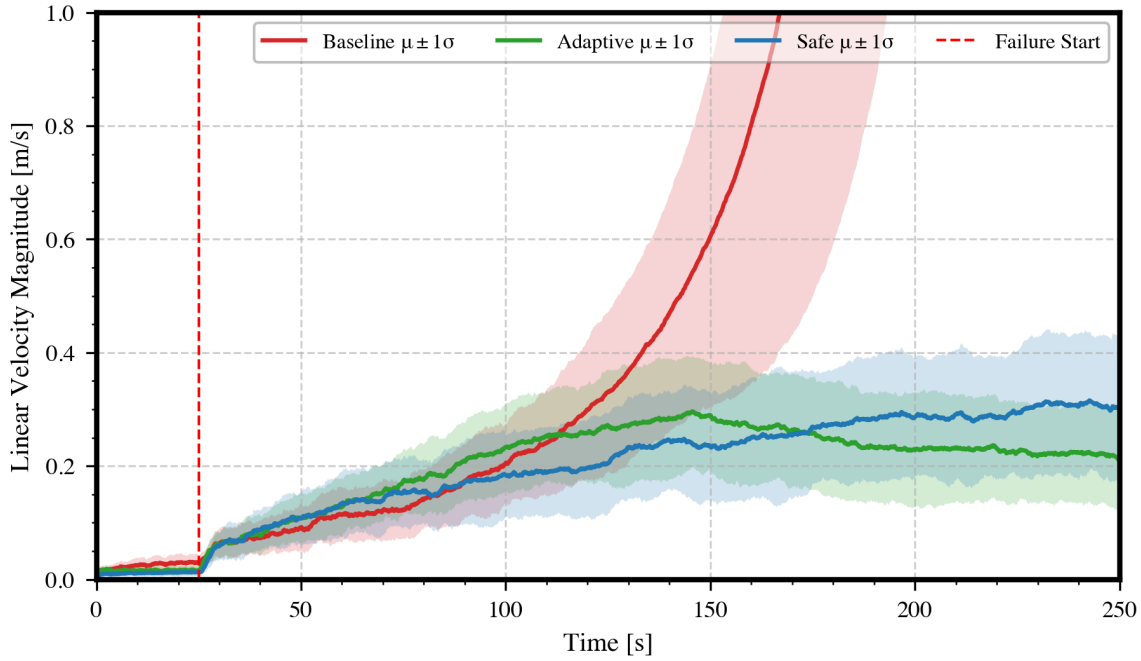


Figure 4.13: Linear velocity magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under failure conditions.

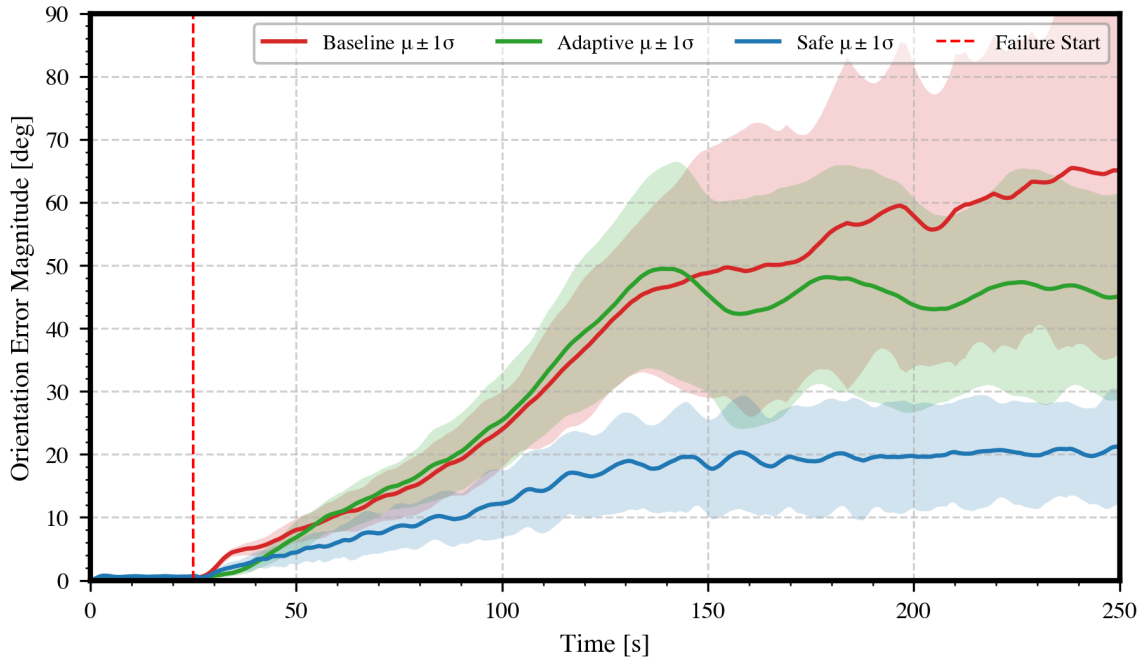


Figure 4.14: Orientation error magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under failure conditions.

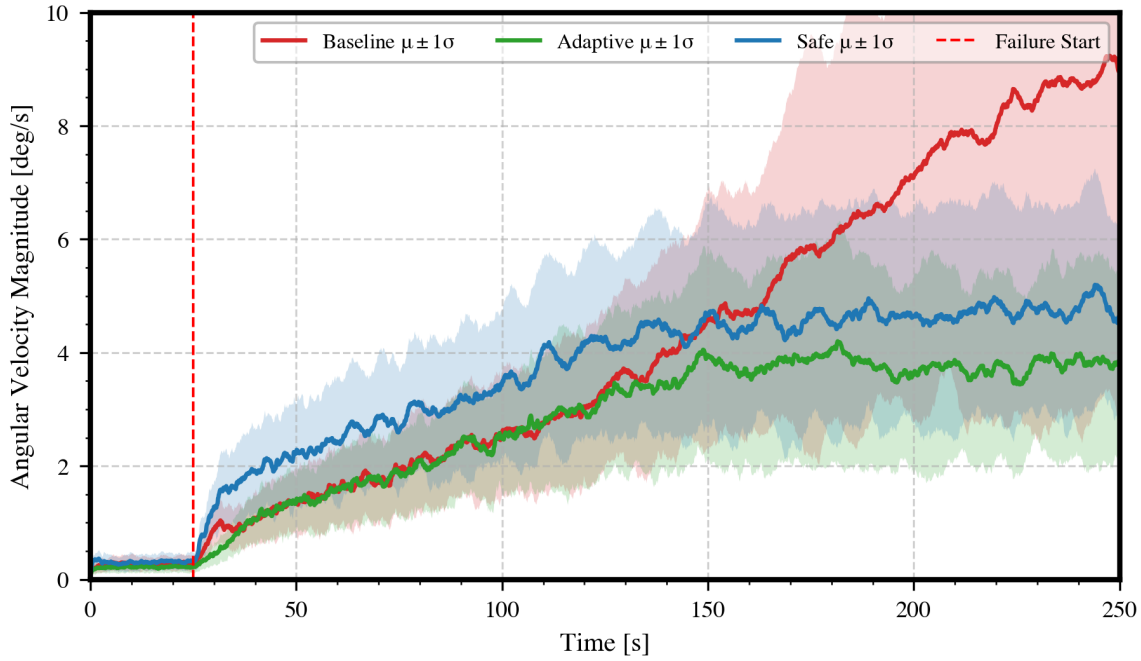


Figure 4.15: Mean angular velocities for the Baseline, Adaptive and Safe controllers over 100 trials under failure conditions.

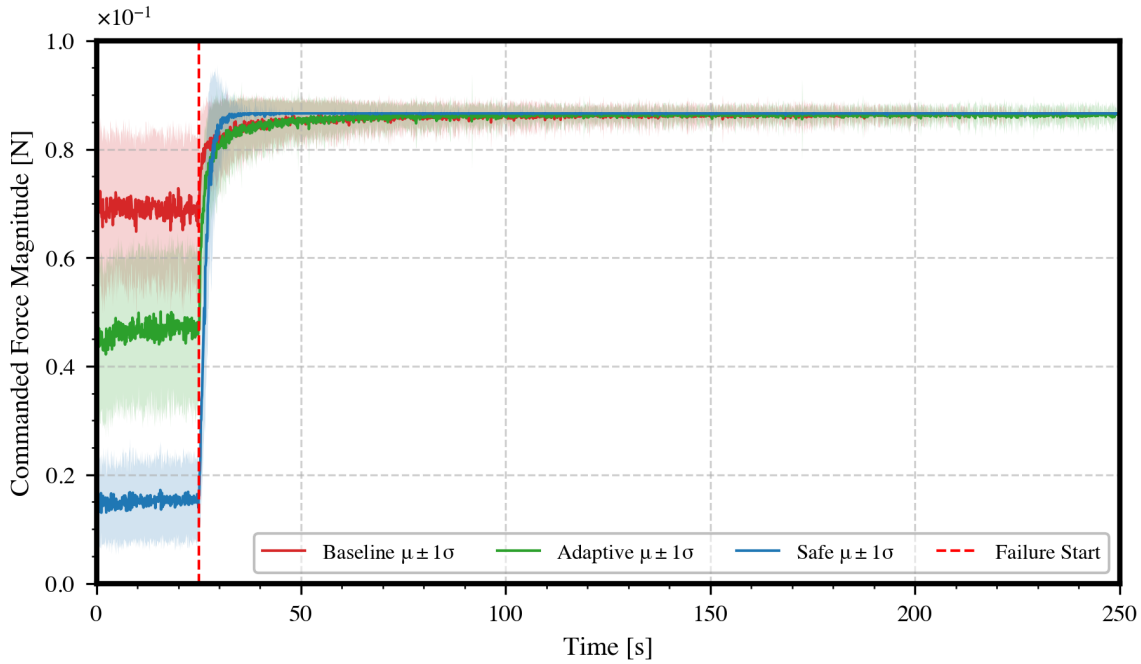


Figure 4.16: Commanded force magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under failure conditions.

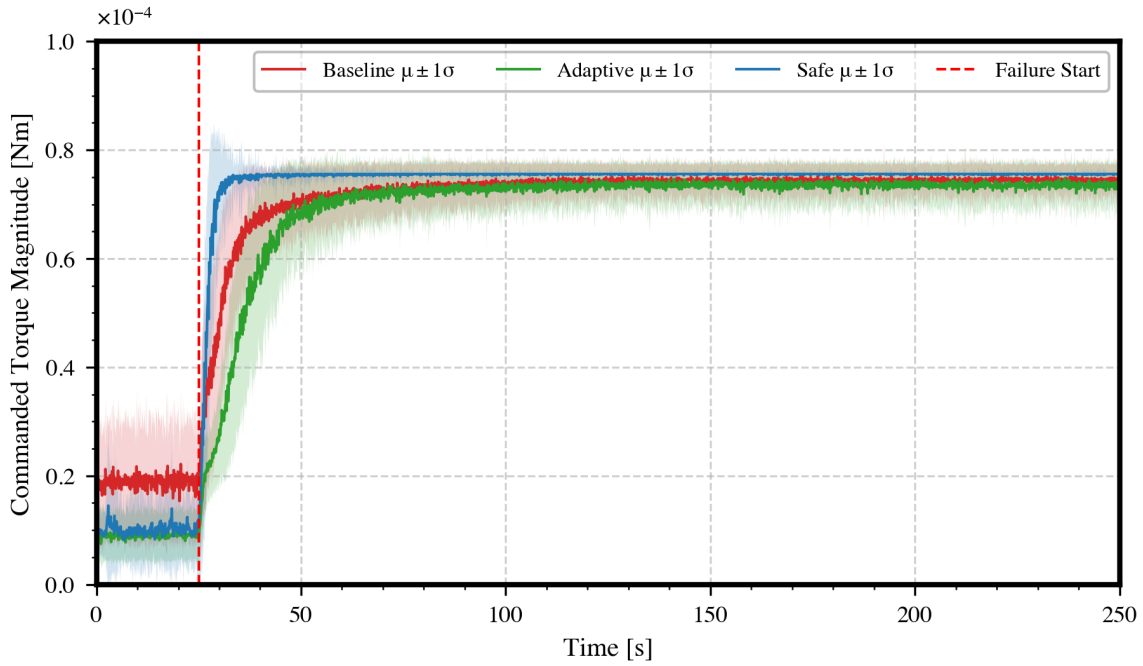


Figure 4.17: Commanded torque magnitudes for the Baseline, Adaptive and Safe controllers across 100 trials under failure conditions.

Position tracking performance degrades across all controllers once the failure cascade begins, but the degree of degradation varies significantly (Figure 4.12). The Baseline controller fails to stabilise the spacecraft, with position error growing uncontrollably beyond 45 m and continuing to diverge throughout the episode. Once the time-varying noise and latency push the observed dynamics outside the training distribution, the policy increasingly outputs near-saturation commands in an attempt to compensate. Under degraded sensing and actuation, these large commands do not produce the intended corrections and instead drive the state further from the region in which the policy was trained, which in turn reinforces further extreme actions. The result is a positive feedback loop between out-of-distribution state visitation and aggressive control outputs, leading to rapid divergence.

The Adaptive controller initially follows a similar trajectory but shows partial resilience after error reaches approximately 25 m, the upper bound of its training distribution, indicating some generalisation from adaptive domain randomisation. However, it fails to correct the error effectively. In contrast, the Safe controller maintains position error within 5 m across all trials, demonstrating its superior capacity to tolerate disturbances under severe degradation.

Velocity behaviour supports this interpretation. As shown in Figure 4.13, the

Baseline controller's linear velocity grows rapidly in response to increasing noise and delay, while both the Adaptive and Safe controllers cap their velocities near the 0.5 $\mathrm{m\,s^{-1}}$ threshold. However, the Safe controller maintains tighter position regulation despite similar velocity profiles, underscoring the role of the control barrier function in modifying actions to improve resilience.

Rotational behaviour shows analogous patterns. Figure 4.14 shows that both the Baseline and Adaptive controllers experience increasing orientation error over time, with the Baseline exceeding 100° and continuing to diverge. The Adaptive controller shows a stabilisation trend around 50°, while the Safe controller levels off around 20°. Notably, the standard deviations of the Safe controller's error profiles are narrower, indicating not only higher fault tolerance but also greater consistency in response to varying failure rates across episodes.

Control commands during failure conditions reveal further insights. Both force (Figure 4.16) and torque (Figure 4.17) magnitudes rise rapidly within 25–50 seconds of the cascade onset, reaching actuator saturation limits. This aggressive response is consistent with the agent attempting to counteract rapidly increasing disturbances. However, in the absence of effective safety guidance, these actions can become destabilising, particularly for the Baseline configuration. The Safe controller still produces large corrective commands but manages to limit errors, highlighting the corrective role of the CBF even when control signals reach their extremes.

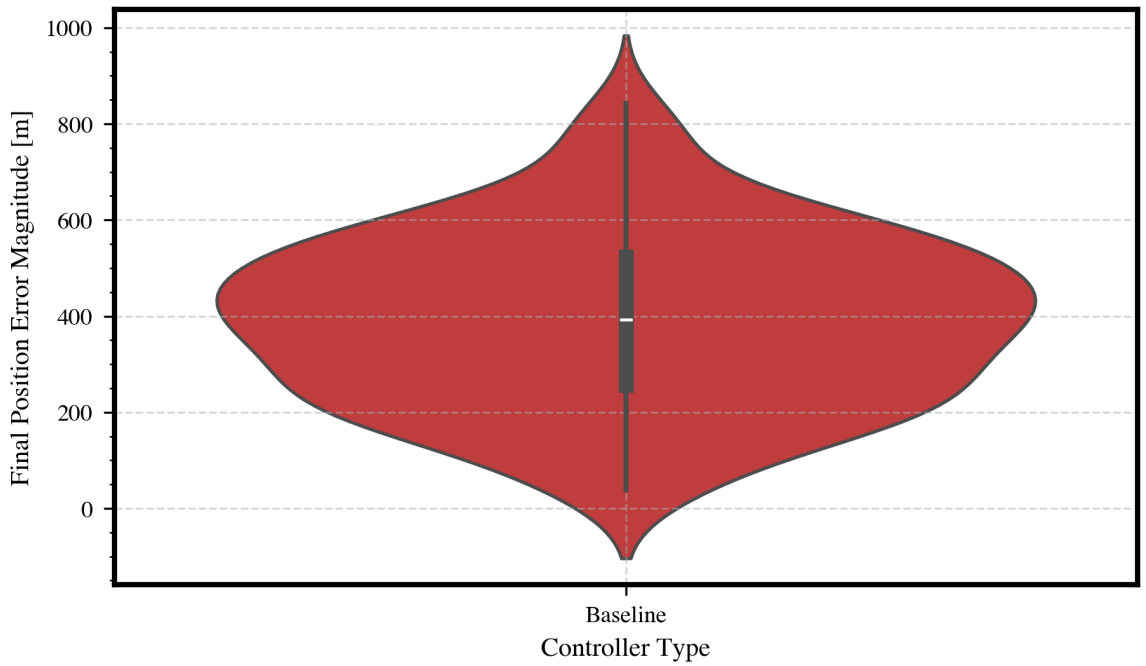


Figure 4.18: Violin plot showing the distribution of final position error magnitudes for the Baseline controller across 100 trials under failure conditions.

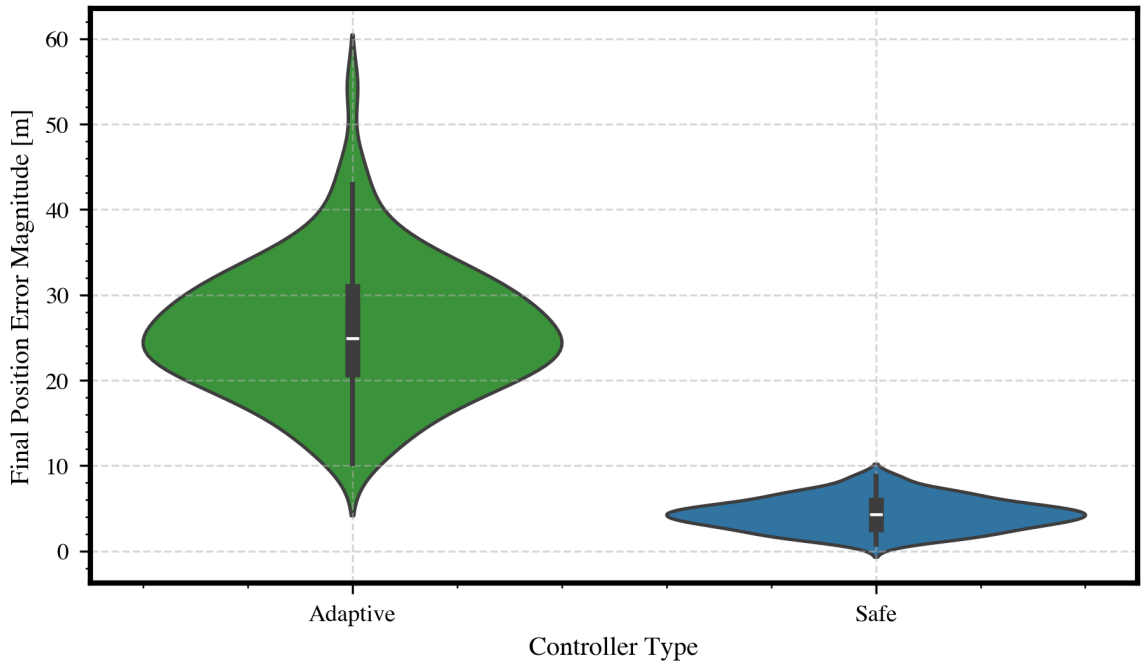


Figure 4.19: Violin plot showing the distribution of final position error magnitudes for the Adaptive, and Safe controllers across 100 trials under failure conditions.

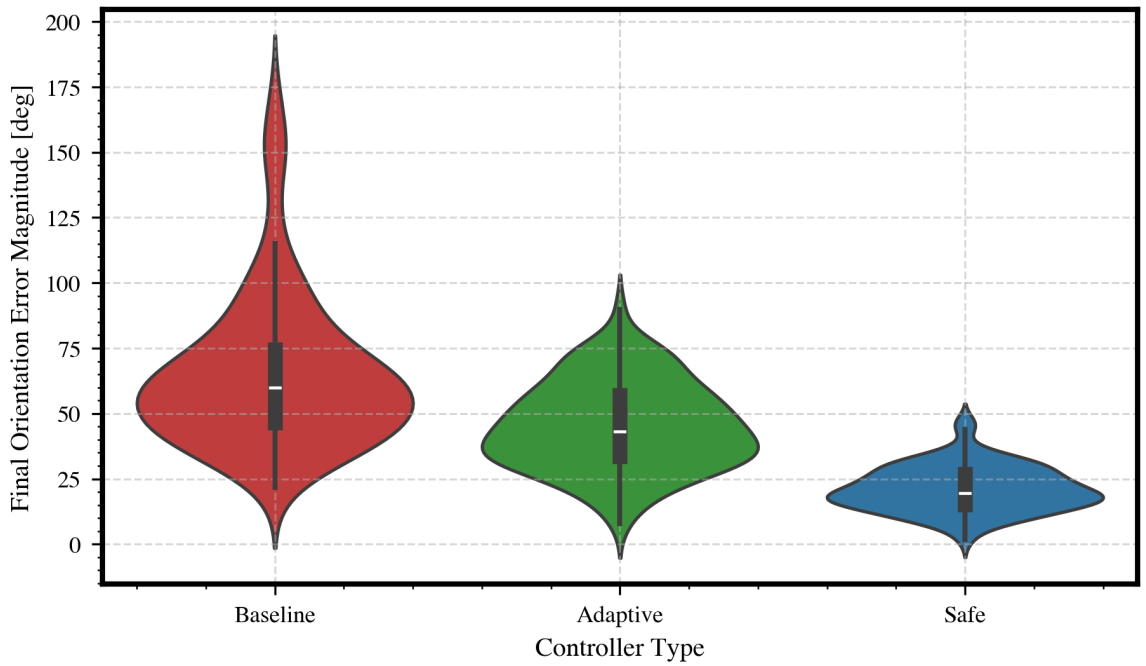


Figure 4.20: Violin plot showing the distribution of final orientation error magnitudes for the Baseline, Adaptive, and Safe controllers across 100 trials under failure conditions.

Distribution plots of final position errors reinforce the observed trends. Due to the magnitude of failure in the Baseline controller, its violin plot is shown separately in Figure 4.18. It displays final errors reaching 1000 m, with a mean error of approximately 400m, indicating total control loss. The Adaptive controller (Figure 4.19) performs significantly better, maintaining final errors below 60 m. The Safe controller demonstrates the best performance, with all final errors contained below 10 m and a mean of around 5m, confirming reliable containment of position errors across all trials.

Final orientation errors follow a similar pattern (Figure 4.20). The Baseline controller exhibits failures with orientation errors exceeding 150°, and the Adaptive controller frequently surpasses 50°. The Safe controller caps at approximately 50° in the worst case, but maintains a mean near 20°, reflecting a higher degree of control authority retention despite fault accumulation.

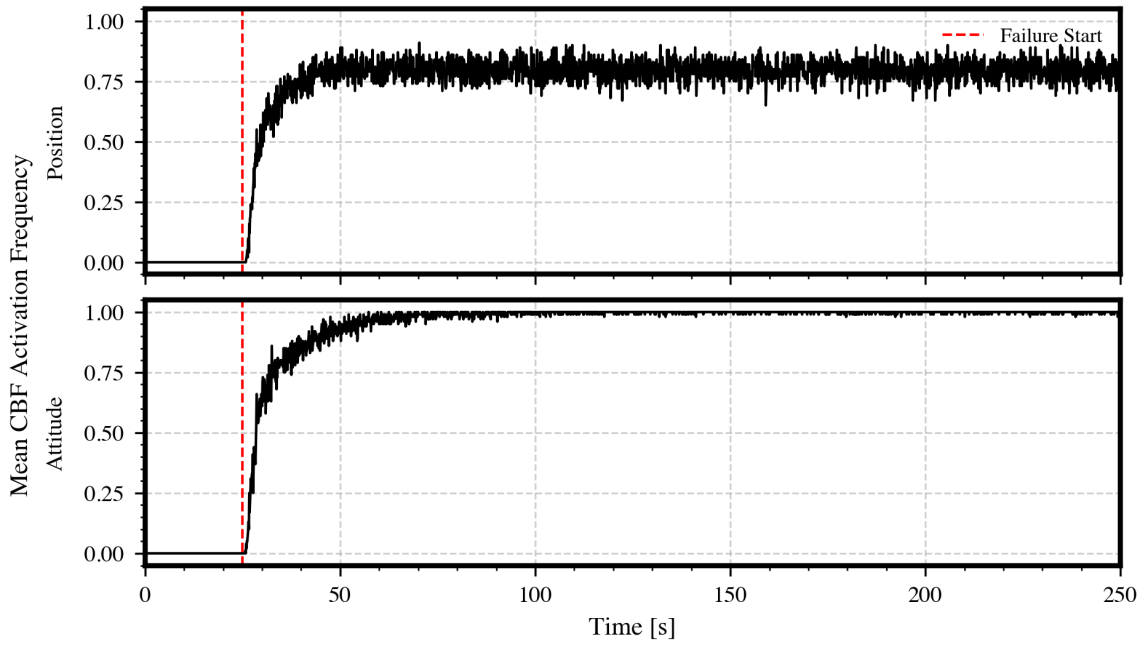


Figure 4.21: Mean CBF activation frequency over 100 trials under failure conditions for the Safe position and attitude controllers. The frequency represents the proportion of time steps in which the CBF actively modified the agent's control input.

The effectiveness of the CBF safety mechanism during failure is quantified in Figure 4.21, which shows the proportion of time steps in which the CBF layer modified the agent's proposed control. For the attitude controller, the CBF activation frequency rapidly rises to nearly 100% within 50 seconds, indicating near-constant intervention once the failure cascade begins. The position controller stabilises around 75% activation, suggesting intermittent corrections where the agent's actions remain marginally safe. This illustrates how the CBF layer dynamically compensates for policy degradation under failure, enforcing operational limits when the agent's own output becomes unreliable.

The failure scenario results clearly differentiate the controllers in terms of robustness and safety. While the Baseline controller collapses under compounding faults, and the Adaptive controller offers some resilience, only the Safe controller consistently shows fault tolerant behaviour across all trials. Its ability to constrain both position and orientation errors, even as actuator and sensor fidelity degrade, demonstrates the benefit of real-time safety filtering. These results validate the use of CBFs not only as a last-resort safety filter but as an active stabilisation mechanism in high-uncertainty environments. When exposed to unmodelled disturbances and dynamic degradation,

safety-aware controllers not only prevent catastrophic failure but can maintain controlled, predictable behaviour, marking a necessary step toward operational viability in autonomous satellite systems.

4.5 Summary

This chapter introduced a new reinforcement learning–based control architecture for autonomous close-range satellite operations, designed to tackle two key limitations in applying current DRL approaches to flight-critical systems: limited generalisation to real-world variability and the lack of reliable mechanisms for ensuring stable behaviour under failure. Rather than relying solely on conventional training strategies, the proposed framework integrates adaptive domain randomisation to systematically build robustness to a range of conditions, and relaxed control barrier functions to encourage safety-aware policies and preserve control authority even as the system degrades. This combination moves beyond performance in ideal conditions, establishing a structured approach to producing controllers that remain resilient, stable, and predictable under off-nominal and fault scenarios.

To assess the contribution of each component, three controller variants were trained and evaluated: a Baseline model using fixed domain randomisation, an Adaptive model using ADR, and a Safe model combining ADR with a relaxed control barrier function layer. All were tested on a 6-DoF spacecraft rendezvous task under both nominal conditions and a progressive fault cascade involving actuator and sensor degradation. Under nominal conditions, all controllers reached the terminal thresholds, but the Safe configuration exhibited more cautious behaviour, remaining within the imposed $0.5\ \mathrm{m\,s^{-1}}$ linear velocity constraint (mean peak $0.439 \pm 0.079\ \mathrm{m\,s^{-1}}$) at the cost of slower positional convergence (36.36 ± 21.95 s versus 21.81 ± 12.93 s for the Baseline controller).

The use of ADR enabled the agent to generalise across variable initial conditions and system parameters, while the CBF layer, active during both training and deployment, acted as a dynamic filter, guiding behaviour through real-time correction and reward shaping. This integration produced controllers that remained robust under uncertainty and demonstrated stable, fault-tolerant performance even as the system underwent progressive degradation.

In the failure cascade, the Baseline controller diverged rapidly (position error exceeding 45 m and reaching up to 1000 m across trials), whereas the Safe controller

consistently bounded position error within 5 m and maintained a controlled attitude response (levelling near 20°). This retained stability was underpinned by sustained safety-layer intervention once faults accumulated, with the attitude CBF modifying the control input on nearly every time step after cascade onset and the position CBF stabilising around 75% activation, indicating that the CBF layer actively compensated as sensing and actuation degraded. This integration offers a practical path toward deployable DRL solutions for autonomous spacecraft.

Chapter 5

Goal-conditioned Hierarchical Reinforcement Learning for CubeSat Proximity Operations

5.1 Motivation

Training reinforcement learning agents to control satellite motion remains a challenging problem, especially when considering that control must be maintained over long time horizons with natural reward signals often sparse or delayed. While Chapters 3 and 4 demonstrated that flat DRL agents can learn 6-DoF CubeSat control policies under shaped reward functions, their training remains sensitive to reward design, typically requiring dense feedback to learn effectively. This dependence on manual shaping limits generalisation and raises questions about scalability to more complex or realistic task formulations.

In this chapter, the focus shifts toward improving learning efficiency and robustness in sparse-reward environments by introducing Hierarchical Deep Reinforcement Learning (HDRL) into the satellite control domain. Rather than assuming access to continuous, externally provided shaping rewards, HDRL architectures are designed to learn layered policies that coordinate goal-directed behaviour over longer timescales. This has the potential to improve credit assignment, accelerate training, and reduce the need for engineered reward signals, key challenges in applying RL to real-world guidance and control tasks.

Although HDRL has shown promise in other robotics and navigation domains, it

remains unexplored within the context of satellite guidance and control. Prior work in using RL for GNC work has focused on flat policies, often relying on dense, hand-engineered reward functions [70, 73, 74]. HDRL remains unexamined in the context of 6-DoF satellite control, based on currently available literature.

This chapter introduces a goal-conditioned hierarchical control framework based on the Hierarchical Actor-Critic (HAC) algorithm and evaluates its performance on a close-range proximity operations task. The framework decomposes the control task into two levels: a high-level policy selects subgoals in the state space, and a low-level policy learns to achieve these subgoals using primitive control actions. This structure is used independently for translational and rotational control, maintaining the decentralised agent design shown in Chapter 3. Each agent is trained using sparse binary rewards that indicate task success only when the chaser's position and attitude fall within mission-defined thresholds. No intermediate shaping signals are used.

To support stable learning under sparse reward conditions, the hierarchical agents incorporate hindsight goal relabelling. This technique enables agents to extract learning signals from failed episodes by reinterpreting encountered states as successful subgoals, improving exploration and accelerating convergence. The resulting controllers are evaluated under the same simulation setup as in Chapter 3, enabling direct comparison with both flat TD3 agents and a classical PID controller.

This chapter makes three main contributions:

1. A hierarchical reinforcement learning framework for 6-DoF satellite control that enables learning under sparse terminal rewards. Unlike prior work that depends on continuous shaping signals, this architecture learns from binary feedback and internal subgoal generation, demonstrating the feasibility of HDRL in long-horizon, low-feedback control scenarios common in satellite GNC.

2. A decentralised HRL controller with modular goal-conditioned policies. Building on the decentralised structure from Chapter 3, this controller integrates temporal abstraction, goal relabelling, and modular actor-critic training across translational and rotational subsystems. The result is an approach to satellite control that supports autonomous subgoal discovery across both control domains.

3. A hardware-in-the-loop (HIL) testbed for deep reinforcement learning in satellite GNC. This chapter presents the design and implementation of a real-time HIL system that enables direct comparison between attitude and position controllers under realistic sensing, timing, and actuation constraints. Experimental results

show that HDRL agents can maintain stable performance when deployed on embedded hardware, demonstrating robustness beyond simulation.

Together, these contributions establish hierarchical reinforcement learning as a practical and effective paradigm for satellite guidance and control. By introducing modular, goal-conditioned policies, this work addresses core challenges in traditional RL, namely inefficient exploration and weak credit assignment in high-dimensional, long-horizon tasks. Building on the decentralised architecture from Chapter 3, this chapter extends the investigation into DRL-based control by showing how temporal abstraction and task decomposition can enhance both learning efficiency and real-world performance.

5.2 Literature Review

5.2.1 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning is a framework that extends traditional reinforcement learning by incorporating temporal and state abstraction with the goal of solving complex, long-horizon tasks more effectively. In standard or flat RL, agents accomplish a task by learning policies that map states directly to actions. On the other hand, HRL breaks down a given task into a hierarchy of smaller tasks, often referred to as subtasks, which operate at different levels of abstraction, mirroring how humans tend to approach complex problems by breaking them down into smaller, more manageable parts [154].

One way HRL can simplify complex problems is by leveraging temporal abstraction through constructs like "options" or "macro-actions", temporally extended sequences of actions that allow the agent to operate at multiple timescales. These options can represent reusable skills or policies that simplify planning and learning, reducing the effective horizon of the decision-making process [37]. Additionally, HRL can utilize state abstraction, where higher level decision-making considers only coarse-grained features of the environment, while the lower-level controllers handle the detail provided by the complete observation.

By learning reusable subpolicies and structuring behavior around intermediate goals, HRL can improve sample efficiency, accelerate learning, and enhance exploration. These advantages become especially critical in environments with sparse rewards, long time horizons, or large state-action spaces; scenarios where flat RL algo-

rithms typically struggle.

Reward Design in Hierarchical RL

Reward functions can generally be split into two types; dense reward functions that provide frequent feedback or sparse reward functions where feedback is received rarely often only when completing a task successfully. Dense reward functions are often preferred as they can accelerate learning by providing more information that the RL agent can use to evaluate its policy. Since these types of reward functions are rarely found in real-world environments they are often manually crafted using heuristics or engineered signals that approximate task progress. The issue with these is that they are frequently suboptimal and may introduce unintended behaviors due to poorly aligned incentives or reward hacking where the agent exploits loopholes in reward design to achieve high rewards without completing the task [155], [156]. In addition, crafting such rewards requires significant domain expertise and trial-and-error tuning, which becomes increasingly infeasible in real-world settings [157]. Conversely, sparse rewards are easier to specify and are often intuitively found as a subset of the state space. The main issue with them is that it is often difficult for a standard RL agent to successfully learn using them often from insufficient exploration, a lack of reward signal to guide the agent means it must resort to random exploration until it happens upon a reward especially when considering environments with large or continuous state spaces [158].

Hierarchical reinforcement learning gets around this problem by introducing intrinsic motivation by way of learning intermediate goals [159], [160]. By decomposing a high level task into subtasks, HRL enables agents to generate intrinsic rewards for achieving subgoals, thereby providing more frequent learning signals, shortening the reward feedback loop without relying on expert design of an external reward function [161]. These pseudo-rewards are often automatically defined based on subgoal completion [162], [163], making learning feasible even when the environment provides sparse or delayed rewards. By solving localized subtasks, lower-level policies receive faster feedback, naturally simplifying the problem of credit assignment between actions and rewards, resulting in quicker convergence and greater robustness to sparse extrinsic rewards [164].

Sample Efficiency and The Credit Assignment Problem

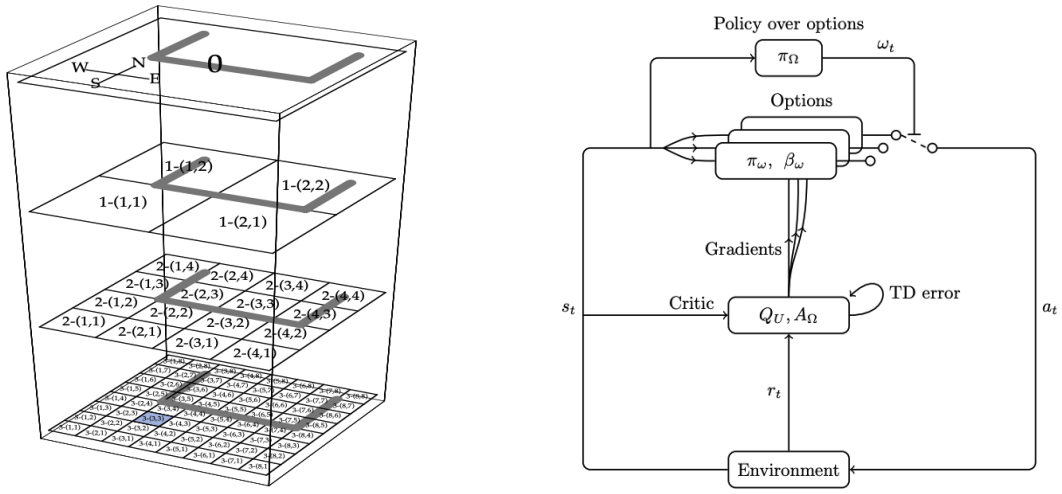
Sample efficiency can be another challenge when considering environments with high-dimensional state spaces and sparse reward signals, often inquiring millions of environmental interactions to learn effective policies. RL agents typically rely on random exploration methods such as ϵ-greedy or entropy-based sampling [29], [165]. These techniques can lead to myopic behaviors, where the agent explores mostly local neighborhoods around its starting state. As a result, many promising regions of the environment remain unreachable within reasonable training timeframes, especially in tasks where reaching the goal requires long sequences of coordinated actions.

Hierarchical reinforcement learning enhances exploration by employing temporally extended actions, or options, which allow agents to execute multi-step behavior sequences that span longer timescales [37], [166]. These high-level skills, once learned, act as abstract actions that can transport the agent across large portions of the state space more efficiently than primitive actions. For example, instead of taking a series of low-level movements to explore a maze, a high-level "navigate-to-door" skill can rapidly move the agent into previously unseen areas. This capability significantly broadens the agent's effective exploration radius and addresses the locality bias inherent in flat RL agents. In effect, HRL transforms sparse, inefficient exploration into goal-driven behavior, where experience is structured and aggregated in a way that accelerates learning. Recent benchmarks, such as DIAYN [167] and HIRO [163], have empirically validated these benefits by showing substantial gains in sample efficiency compared to baseline RL algorithms in both simulated and real-world tasks.

When considering problems with long time horizons it can become difficult to determine which actions, potentially taken hundreds or thousands of steps in the past, were responsible for a delayed reward signal. Known as the credit assignment problem, standard RL algorithms often struggle to propagate meaningful gradients back through long action sequences, which can lead to unstable or slow learning, especially when the reward does not directly correlate with immediately preceding actions [1], [168]. Traditional value-based or policy-gradient methods apply credit at the granularity of individual steps, which becomes both inefficient and unreliable in complex tasks. HRL mitigates this issue by compressing the temporal abstraction space. Since high-level policies operate at a coarse timescale, selecting subgoals or options that persist for many low-level steps, it becomes easier to assign credit to the high-level decision that initiated a skill, rather than to each atomic action individually. This not only reduces variance in the learning signal, but also makes optimization more tractable,

essentially shifting the burden of credit assignment from many fine-grained steps to a few subgoal choices. Such abstraction has been shown to lead to more stable learning, faster convergence, and better performance in sparse and delayed-reward settings [169], [170].

Hierarchical RL Algorithms



(a) Feudal RL hierarchy [169].

(b) Option-Critic architecture [166].

Figure 5.1: Hierarchical reinforcement learning frameworks. (a) Feudal model, where the environment is recursively decomposed into smaller regions across the multiple levels of hierarchy. (b) Option-Critic architecture, where options are selected by a policy over options and executed through intra-option policies.

One of the earliest attempts at hierarchical reinforcement learning is the Feudal Reinforcement Learning model [169], illustrated in Figure 5.1a. Introducing a hierarchical Q-learning framework in which control is distributed through a multi-level "feudal" management structure. High-level managers operated on abstract state spaces and set sub-tasks for lower-level workers, who in turn executed primitive actions to achieve them. Each worker is rewarded intrinsically by its manager solely for sub-task completion, regardless of whether the sub-task advanced the superordinate goal directly. This principle of "reward hiding" simplified learning by localizing credit assignment, though it still required manually specified state decompositions and goal vectors. Despite these limitations, the feudal model established the central HRL intuition, that multi-level structures could make otherwise intractable problems solvable.

Shortly afterwards, the field saw the emergence of value function decomposition approaches. MAXQ [171], provided a framework for recursively decomposing Markov Decision Processes into subroutines (max nodes), allowing hierarchical value functions to be explicitly represented and learned. However, like feudal models, MAXQ relies heavily on hand-designed task hierarchies, limiting scalability to open-ended environments. Subsequent extensions such as HEXQ [172] extended hierarchical Q-learning by automating subtask discovery, though subtask identification remained reliant on the structure of the environment and practical deployment remained constrained without the addition of deep learning.

A major leap came with the development of the Options framework [37], moving HRL toward temporal abstraction by the introduction of parameterized policies that encapsulate complex behaviors over extended time scales known as "options". This framework subsumed earlier models: a feudal subgoal or MAXQ subtask could be interpreted as a particular kind of option. Options allowed agents to reason and act at multiple temporal scales, significantly reducing decision frequency and improving long-term planning. The challenge, however, was that useful options often had to be manually designed or discovered through ad-hoc heuristics. The Option-Critic architecture [166], shown in Figure 5.1b, addressed this limitation by enabling end-to-end learning of option policies and termination conditions directly from interaction with the environment, thus removing the need for explicit option definitions. Option-Critic demonstrated promising results on Atari benchmarks, producing meaningful subpolicies, though training stability and sample efficiency remained problematic.

In parallel with policy abstraction, goal-based hierarchies emerged as an alternative to fixed options. One early example is h-DQN [161], which utilized a two-level hierarchy where a high-level Q-network selected discrete subgoals (e.g., "obtain key," "enter room") and a low-level goal-conditioned Q-network executed primitive actions to achieve them. By providing pseudo-rewards for subgoal completion, h-DQN could explore extremely sparse-reward environments where standard DQN failed. The drawback, however, was that subgoals still had to be manually defined, limiting generality. To overcome this, FeUdal Networks (FuN) [173] revived the feudal idea within a deep learning framework, replacing discrete pre-specified subgoals with continuous goal vectors produced by a manager module. A worker module was intrinsically rewarded for aligning its behavior with these latent goals, making the representation "directional" rather than absolute and avoiding the difficulty of specifying exact subgoal states in high-dimensional spaces. FuN successfully demonstrated emergent subskills such as

navigation in maze-like environments, but stability remained an issue with the indirect coupling between manager and worker made training sensitive to hyperparameters and prone to oscillatory goal-setting.

To address stability and sample efficiency, later works like HIRO [163] introduced a two-level, goal-conditioned agent where the high-level policy set continuous subgoals directly in the environment's state space. A key innovation was off-policy correction: as the low-level policy evolved, HIRO relabeled past experiences with subgoals that matched what was actually achieved, allowing effective reuse of replay buffer data. This greatly improved sample efficiency, making HIRO comparable with off-policy training regimes common in deep RL. HIRO proved effective in continuous control benchmarks, outperforming earlier goal-based and feudal approaches on tasks with sparse rewards.

Building on this, HAC [162] extended goal-conditioned hierarchies to more than two levels and introduced Hindsight Action Transitions (HAT), an adaptation of Hindsight Experience Replay (HER) [158] for hierarchical control. HAC trains each level of its hierarchy as if the levels below are already optimal, insulating higher-level learning from lower-level non-stationarity. This allows for successful training of three-level hierarchies on tasks like robotic manipulation and maze navigation, where prior methods had struggled. HAC thus represented one of the first demonstrations of stable, multi-level deep HRL in continuous domains.

Overall, the field has steadily moved away from expert-defined hierarchies toward self-discovered and transferable subgoals, reflecting the broader trend in RL of reducing human engineering while improving autonomy and generality. The challenge of stabilizing end-to-end training in multi-level hierarchies still remains, despite this, modern methods like HIRO and HAC demonstrate that HRL is a viable option for tackling real-world, long-horizon, and sparse-reward problems.

HRL for Satellite Guidance and Control

Despite the promise of reinforcement learning for autonomous satellite guidance and control, applying a standard flat RL to the problem of close-range rendezvous remains problematic. Such tasks often have long time horizons and sparse, delayed rewards, with success only clearly determined upon arrival at a specified waypoint or final docking, leaving the agent without meaningful feedback during critical phases of approach. Flat RL agents, which treat all decisions at the same granularity, often struggle to learn robust policies under these conditions due to poor sample efficiency and limited

capacity for long-term credit assignment. Without structure, these agents can become trapped in local exploration loops or fail to discover the sequential dependencies between intermediate objectives that make rendezvous feasible.

Hierarchical Deep Reinforcement Learning offers a natural and effective alternative by structuring the learning process around temporal and functional abstraction. This mirrors traditional guidance and control system design, where proximity operations are decomposed into high-level planning (guidance) and low-level actuation (control) subsystems. The task of satellite rendezvous is often broken down into sequential stages such as coarse approach, trajectory alignment, and fine docking maneuvers, each with different temporal and precision requirements. HDRL frameworks, by design, allow such decomposition: high-level policies define strategic subgoals (e.g., reaching a specific waypoint, aligning orientation), while lower-level policies execute primitive actions to fulfill them. This separation supports modular learning, where each layer can be optimized independently but coordinated through a hierarchy. Importantly, HDRL's temporal abstraction reduces the effective decision-making horizon at each level, enhancing both sample efficiency and policy interpretability, and enabling more robust generalization across mission profiles.

Among the HDRL architectures discussed, Hierarchical Actor-Critic was selected for this work due to several features that align well with the requirements of close-range satellite rendezvous. Its support for multi-level hierarchies and concurrent policy learning across different timescales provides a natural fit for separating high-level guidance from low-level control. Additionally, HAC's goal-conditioned hierarchy enables the agent to learn meaningful subgoal selection and execution strategies aligned with the sequential structure of rendezvous maneuvers. Furthermore, the hierarchical modularity of HAC allows for independent tuning or retraining of specific layers; a practical advantage in aerospace contexts where hardware constraints, mission phases, or docking protocols may vary. As orbital servicing and on-orbit assembly evolve, HAC provides a flexible control framework that can adapt without requiring full retraining of the system.

5.2.2 Satellite Attitude Testing Platforms

Attitude testing platforms provide a controlled means of replicating the frictionless dynamics of satellite in orbit, thereby enabling experimental validation of attitude control approaches under physical conditions. Their value lies in bridging the gap between numerical simulations and on-orbit operation, where the absence of friction

and the presence of actuators and noisy sensor input shape system behavior in ways that cannot be fully reproduced in software alone. By allowing for unconstrained rotational motion in one or more axes, these platforms create an environment where GNC algorithms can be exercised in real time, responding to realistic dynamics rather than idealized models. This is particularly true in the context of small satellite, where actuation is more limited and control authority must be carefully managed As demonstrated by experimental studies across a range of nanosatellite-focused testbeds [174], [175], [176].

Several experimental methods have been developed to reproduce microgravity for satellite control system testing. Parabolic flights are one of the most direct means of creating a weightless environment. In this approach, an aircraft conducts a series of parabolic flights, after reaching the top of its parabolic arc engine thrust is reduced to allow the aircraft to free-fall, thus creating a period of simulated weightlessness [177], [178], [179], [180]. During each arc, a period of roughly 20-30 seconds of usable microgravity is available, during which control systems can be operated and evaluated. However, this method has two major drawbacks: the cost of aircraft operation makes repeated campaigns expensive, and the short duration of each microgravity window severely limits the scope of experiments that can be carried out. Drop towers operate on the same principle of free fall, but instead of flying a trajectory, the test article is released inside a vertical shaft, usually within a vacuum chamber to minimize aerodynamic drag [181], [182], Depending on the tower height, these facilities typically provide a few seconds of free fall before the payload is safely decelerated. Drop towers are a less expensive test method than parabolic flights but they still restrict testing to very short time windows and require specialized infrastructure that is limited to only a few locations worldwide.

Neutral buoyancy testbeds integrate the test article within a neutrally buoyant vehicle and operate underwater [183]. This allows for effectively unconstrained six-degree-of-freedom motion over extended periods. The main limitation of this approach arises from the drag produced by moving through the water, which can introduce disturbances that alter vehicle dynamics, especially during high-speed maneuvers. Although mitigation strategies such as active drag compensation have been explored [184], the requirement for a dedicated pool or large water facility remains a drawback, restricting where such experiments can be conducted and adding considerable infrastructure requirements. Industrial robotic systems offer another option for emulating satellite dynamics, particularly in the context of rendezvous and docking stud-

ies. These setups, which will be discussed further in the following section, use one or more robotic arms to reproduce the relative trajectories of a chaser and target satellite, allowing precise, repeatable, and programmable scenarios to be carried out in a controlled environment. However, their motion is inherently constrained by the mechanical limits of the arms themselves. High-rotation maneuvers such as tumbling cannot be reproduced faithfully, as the joints are subject to range limitations and can encounter singularities that restrict smooth multi-axis motion. As such, robotic arm testbeds are well suited to testing controlled approach and docking trajectories, but are less effective for simulating unconstrained attitude dynamics.

Air bearings have become one of the most widely adopted solutions for ground-based satellite attitude and dynamics simulation, owing to their ability to approximate microgravity conditions with minimal friction. Their operation is based on maintaining a thin film of pressurized gas, typically air or carbon dioxide, between a stationary stator and a freely moving rotor. This gas film eliminates direct mechanical contact, which in turn reduces friction to negligible levels and enables nearly unconstrained motion. Air bearings can broadly be categorized into planar and spherical designs. In planar configurations, a flat or disk-shaped pad with a porous or channeled surface is used to continuously expel compressed gas, allowing a platform to levitate above a smooth surface with minimal friction [185]. This arrangement enables two translational degrees of freedom and one in-plane rotational degree of freedom, making them particularly well suited for simulating nanosatellite operations such where planar motion dominates the relative dynamics.

A typical approach, demonstrated in [3] , [186], is to mount free-floating satellite mock-ups on multiple precision air bearings and actuate them with cold-gas thrusters, thereby allowing docking dynamics to be reproduced on a flat granite table (see Figure 5.2). The two platforms differ primarily in scope: while [3], focused on demonstrating closed-loop docking experiments using embedded control hardware, [186], placed greater emphasis on evaluating capture mechanisms, adapting their design so that the mass and inertia properties of the mock-ups could be tuned to represent different servicing scenarios. Other research has utilized planar testbeds for cooperative multi-agent experiments; for example, [187] used distributed cooperative control to coordinate the relative motion of two vehicles during a net-deployment task, extending the applicability of planar platforms beyond pairwise docking to more complex interactions. A comprehensive review in [185] provided an overview of planar air-bearing facilities worldwide, highlighting their wide use in formation flying, tethered

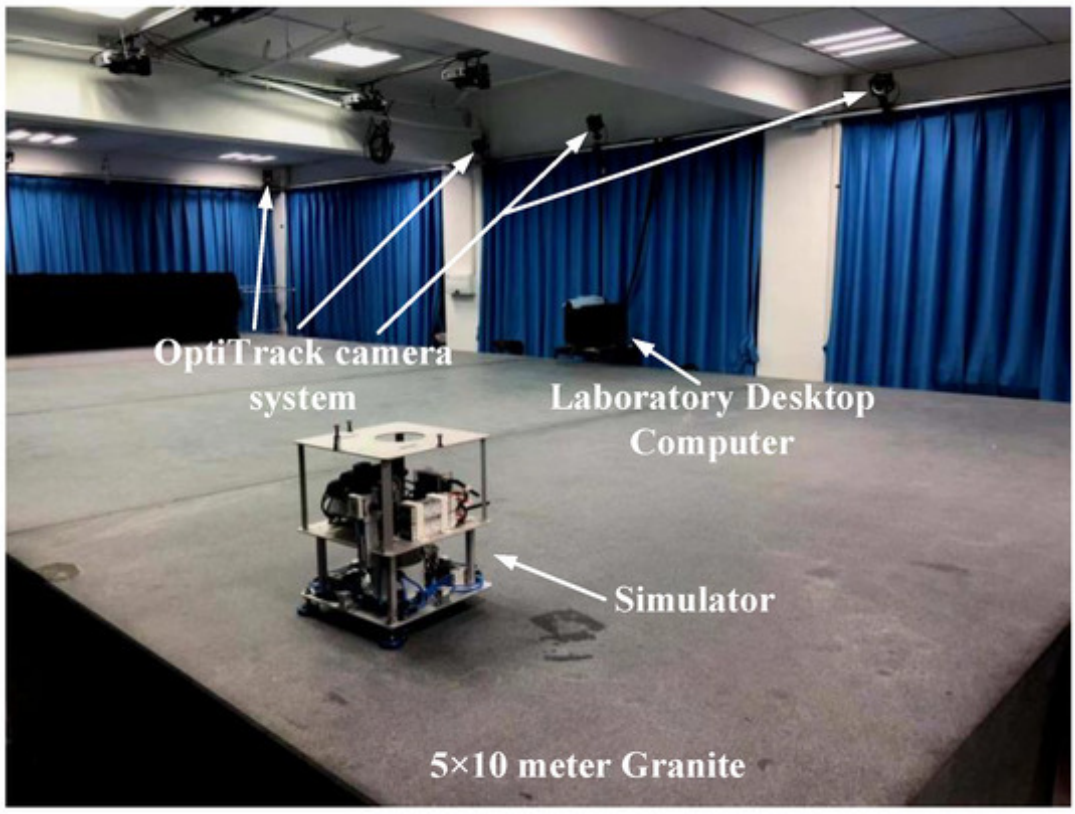


Figure 5.2: Example of a planer air bearing testing platform [3].

satellite studies, and space robotics, while also noting that their fidelity is fundamentally constrained by their restriction to motion in a single plane. Collectively, these works demonstrate the adaptability of planar platforms for multi-agent and cooperative experiments, though they remain dependent on large, flat, and leveled surfaces, typically made from granite, glass, or epoxy, which add cost and space requirements. Furthermore, their dynamics are generally limited to two-dimensional motion, which restricts their ability to test full three-axis attitude control. Many designs also rely on onboard compressed-gas canisters to supply the bearings, limiting experiment duration.

Spherical air bearings replace the flat air bearing interface with a curved socket that allows the rotor, which contains the testing article, to pivot freely in all three rotational degrees of freedom (see Figure 5.3). Compared to planar bearings, spherical bearings permit continuous tumbling, spin stabilization, and more complex attitude maneuvers to be evaluated. The primary trade-offs are that they sacrifice translational capability and impose stricter requirements on balancing the payload so that its center

of mass coincides with the center of rotation of the bearing; with any offsets introducing disturbance torques that degrade fidelity. High-precision commercial bearings are available from suppliers such as New Way Air Bearings [188] and Physik Instrumente [189], but many research groups design and manufacture custom spherical bearings to optimize size, payload capacity, and cost for their specific missions.

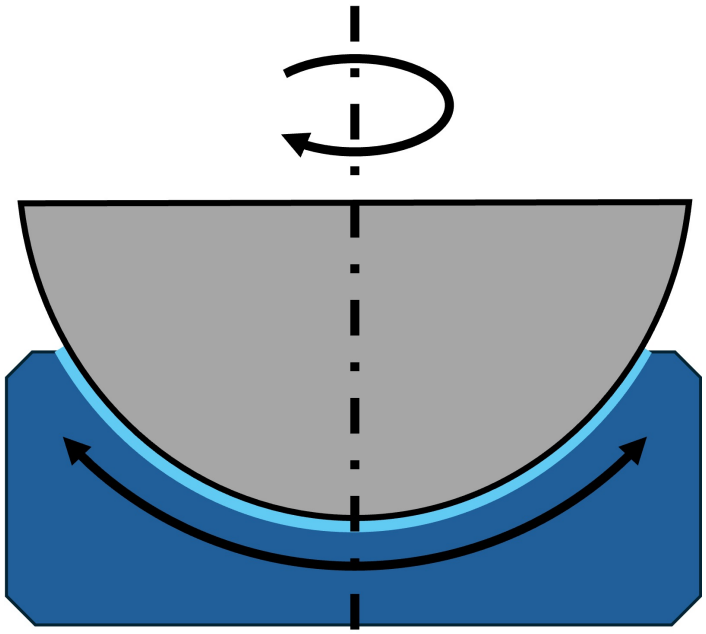


Figure 5.3: Schematic of a spherical air bearing platform, showing rotational degrees of freedom enabled by a layer of pressurized gas separating the rotor from the stator.

Spherical air bearing platforms have been realized in a variety of configurations, distinguished primarily by their choice of actuators and design priorities. An example is shown in Fig. 5.4. A large subset of testbeds rely on reaction wheels as their primary actuators, reflecting their widespread use in CubeSat and nanosatellite ADCS. In [175] a custom spherical air bearing platform was designed and machined for testing the attitude control of 1U CubeSats. This testbed was equipped with reaction wheels that served both to adjust the center of gravity of the platform by adjusting their position, and to provide the control torques necessary for executing attitude maneuvers. A similar compact and low-cost nanosatellite testbed is presented in [190], with the testbed being manually balanced by its design. Its effectiveness was validated by the successful implementation and comparison of three control algorithms. The work in [191] focused more narrowly on the design of the reaction wheel assembly itself, first simulating its performance, then integrating a single wheel onto a spherical air bearing

platform and evaluating its dynamic response under hardware-in-the-loop conditions. Other studies, such as [192], [193], [194], and [195] focused on the construction of low cost and small footprint platforms. However, both [192], and [195] highlighted the difficulties of precise balancing, with [192] reporting that its configuration was limited to single-axis yaw control as a result of these constraints.

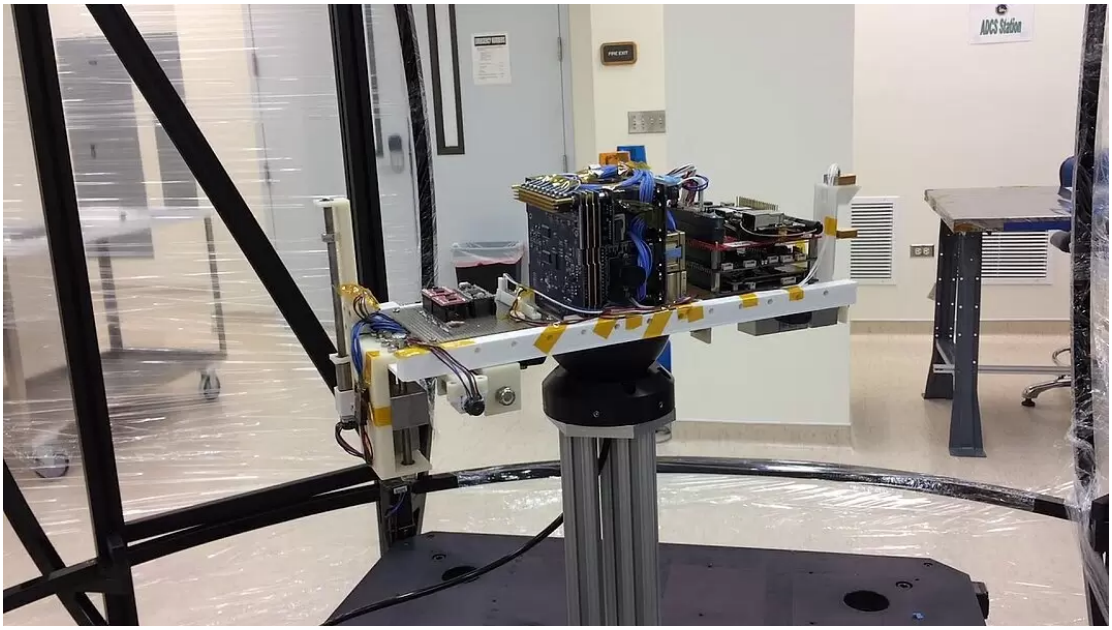


Figure 5.4: Example of a spherical air-bearing platform used for satellite attitude simulation [4].

In contrast to wheel-driven setups, several works investigated magnetically actuated testbeds, either using magnetorquers as the primary actuators or in combination with reaction wheels. An example of a magnetorquer based platform can be seen in Figure 5.5. These systems require a Helmholtz cage to generate a uniform magnetic field that allows magnetorquers to function realistically. In [196], the CubeSat Three-Axis Simulator (CubeTAS) was developed, integrating both reaction wheels and magnetorquers alongside an automatic balancing mechanism, and validated its operation through experiments using a simple PD control law. Building on this, [197] introduced a dedicated magnetic control system to the CubeTAS platform and evaluated its ability to reject disturbance torques simulated by the reaction wheels. A similar platform in [198] emphasized purely magnetic actuation for 1U and 3U CubeSats, also supported by an automatic balancing system. Collectively, these studies demonstrate that magnetic actuation offers a viable alternative to reaction wheels, though it requires more extensive laboratory infrastructure to reproduce geomagnetic

conditions.

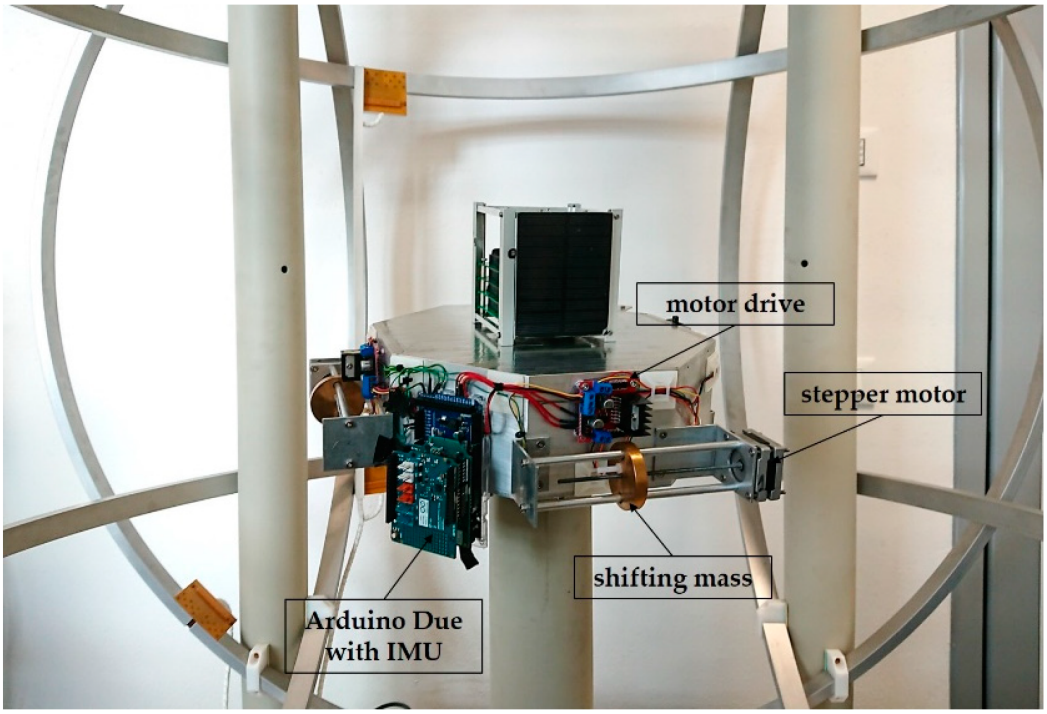


Figure 5.5: Example of a magnetically actuated CubeSat test platform surrounded by Helmholtz cage [5].

Beyond experimental implementations, a number of review papers provide broader context on the development of spherical air-bearing testbeds. The survey in [199] presented a technical overview of spherical satellite simulators, detailing their dynamics, kinematics, and control strategies, while emphasizing recurring engineering challenges such as precise center-of-mass alignment, fabrication tolerances, and the trade-off between simplicity and fidelity. The work in [200] situated spherical air bearings within the wider historical development of satellite simulators, contrasting them with planar, buoyancy, and robotic-arm-based platforms. Their review highlighted the superior fidelity of spherical systems for reproducing unrestricted three-axis motion, while also underscoring the infrastructure and cost burdens that can limit their use.

Building on these insights, the platform proposed in this thesis adopts a spherical air-bearing design tailored for integration with a robotic-arm-based rendezvous and docking simulator discussed later in this chapter. The key design goals include enabling $\pm 30^{\circ}$ unrestricted motion in pitch, roll, and yaw, thereby supporting realistic yet manageable ranges of attitude maneuvering. The experimental payload will consist

of a 1U CubeSat mock-up actuated by reaction wheels, providing a compact but representative configuration for testing control algorithms.

5.2.3 Rendezvous and Docking Testing Platforms

The rendezvous and docking phase of a mission requires precise translational and rotational control of the chaser satellite under stringent safety margins. To mitigate risks, ground-based hardware-in-the-loop testbeds have been developed to emulate these dynamics and validate guidance, navigation, and control strategies prior to flight. Such facilities must balance fidelity, repeatability, and cost while approximating the near-frictionless conditions of orbit.

It is possible to use planar air-bearing platforms for RvD testing, in this setup satellite mock-ups are equipped with air puck "feet" that trap a cushion of air between themselves and a flat surface, allowing nearly frictionless motion. These testbeds have been used for docking and formation flying experiments [201], [202], [203]. However, planar systems are inherently constrained to motion in a single plane and require large, precision-engineered surfaces, making them unsuitable for full 6DoF testing. Despite these drawbacks, planar facilities remain valuable for cooperative multi-agent testing and proof-of-concept demonstrations, particularly when scaled to nanosatellites or CubeSat-class vehicles.

An alternative method is the use of robotic arms to simulate satellite motion. Unlike planar platforms, robotic manipulators can reproduce the full six degrees of freedom, enabling more complete RvD testing. A notable example is the European Proximity Operations Simulator (EPOS 2.0), shown in Figure 5.6, operated at the DLR Space Operations and Astronaut Training institution in Oberpfaffenhofen [204], [205], [206], [207], [208], [209], [210]. EPOS 2.0 employs two KUKA industrial robots, with one mounted on a 25m linear rail, to simulate the relative motion of a chaser approaching a target. The system achieves sub-millimeter positioning accuracy over the full rail length, enabling high-fidelity reproduction of docking dynamics. The facility also includes a solar simulator to reproduce realistic illumination for vision-based navigation experiments.

Similarly, the Space Catapult ISAM test facility employs two KUKA industrial robots, one mounted on a 17m track and capable of handling payloads up to 120kg [211]. These large-scale facilities set the benchmark for robotic RvD testing, though their infrastructure costs and operational complexity limit accessibility.

Beyond large-scale facilities, a number of smaller simulators have been developed to

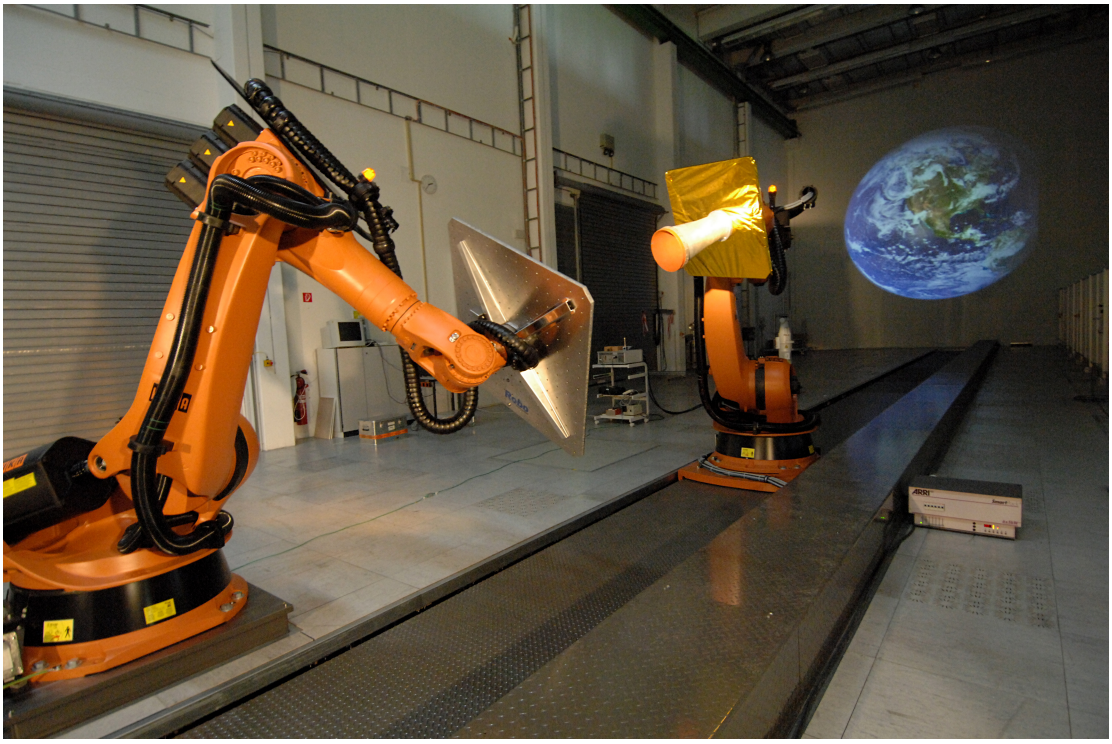


Figure 5.6: The European Proximity Operations Simulator (EPOS 2.0) facility at DLR, featuring dual industrial robot arms for high-fidelity proximity operations testing [6].

enable laboratory-scale rendezvous and docking research. For example, [212] presents a hardware-in-the-loop simulator using a single EFFORT ER210-2700 robotic arm on a 12 m rail, equipped with a hybrid force/velocity control strategy improve the fidelity of contact dynamics during docking experiments. Similarly, [213] simulates a dual-robot setup using ABB manipulators to reproduce chaser dynamics with trajectory planning constrained by joint space and kinematic limits. Work in [214], proposes a force and moment compensation strategy to mitigate errors caused by time lag of the force sensor, motion lag of the motion simulator and deformation of the simulator's mechanical structure. Complementing these, review studies such as [215] have mapped the development of RvD simulators, including robotic-arm-based simulators, emphasizing both their advantages in programmability and repeatability, and their persistent limitations, including joint singularities, workspace restrictions, and dynamic scaling issues. Collectively, these works demonstrate that compact robotic simulators can provide valuable experimental platforms for docking and proximity operations, though their fidelity depends on addressing the dynamic and kinematic limitations of the robotic manipulators.

An alternative approach is represented by hybrid testbeds, which combine robotic manipulators with air-bearing-supported payloads. In such systems, the robotic arm reproduces translational motion, while the air-bearing component provides frictionless rotational freedom. An example is "AirBall" satellite simulator [216], where a CubeSat mock-up is mounted within a spherical air-bearing assembly that is itself carried by an Adept Viper s650 6-DoF robotic manipulator. This design merges the high-fidelity rotational dynamics of air bearings with the programmability of robotic translation, offering a compact and cost-effective compromise between planar tables and large robotic facilities, although its mechanical construction is relatively complex compared to conventional testbeds.

In this thesis, the proposed RvD testbed adopts a hybrid approach by integrating a hemispherical air-bearing platform with a Kinova Gen 3 6-Dof robotic arm [217]. This configuration provides $\pm 30^{\circ}$ unrestricted rotation for attitude simulation while enabling programmable three-degree translational motion through the robotic manipulator. The design offers a compact alternative to large robotic facilities while retaining the key advantage of air bearings: high-fidelity reproduction of rotational dynamics. This combination ensures compatibility with both attitude and rendezvous control testing, positioning the platform as a practical and versatile solution for CubeSat-scale ISAM research.

5.3 Hierarchical Control Methodology

5.3.1 HAC Architecture

The Hierarchical Actor-Critic algorithm is a hierarchical reinforcement learning framework designed to improve performance in sparse-reward, long-horizon tasks by combining temporal abstraction with goal-conditioned learning. The architecture consists of a stack of independently trained actor-critic networks, each operating at a different level of temporal resolution and trained using the DDPG algorithm. Higher-level agents propose subgoals for the level below, which are treated as temporary objectives to be achieved within a fixed subgoal horizon. The structure of a typical HAC agent is shown in Figure 5.7. By organizing control and decision-making across multiple time scales, HAC enables deep exploration, modular learning, and more effective credit assignment in environments where flat policies typically underperform.

Agent Structure and Hierarchy

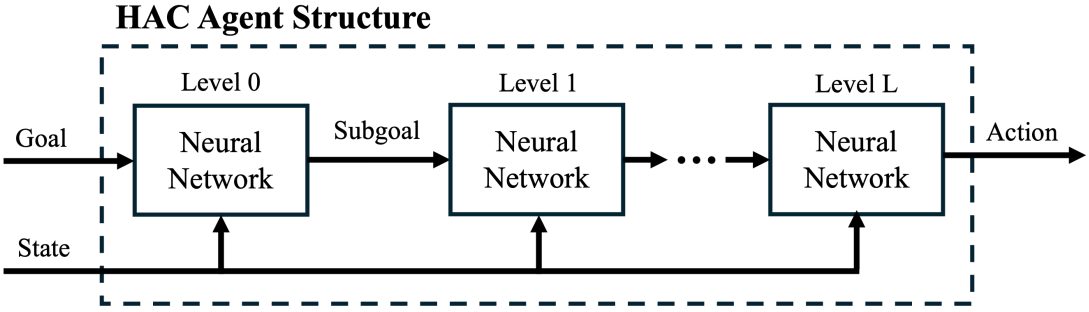


Figure 5.7: Structure of a Hierarchical Actor-Critic (HAC) agent. The agent is composed of L levels of actor-critic networks arranged hierarchically. Each level receives the full system state and outputs a subgoal to the level below, with the final level producing a primitive action.

A HAC agent is structured into L levels, typically 2 or 3, each operating at a different temporal resolution. To enable goal-directed behavior, each level maintains its own policy $\pi^i(s^i, g^i)$, conditioned on both the current state $s^i \in \mathcal{S}^i$ and a goal $g^i \in \mathcal{G}^i$.

The base level $i = 0$ interacts directly with the environment by producing primitive actions $a \in \mathcal{A}$, while each higher level $i \in 1, \ldots, L-1$ outputs subgoals $g^{i-1} \in \mathcal{S}^{i-1}$, which represent target states for the level below to achieve. The top-level policy $\pi^{L-1}(s^{L-1}, g^{L-1})$ is conditioned on the overall task goal g^{L-1} and generates subgoals accordingly. This hierarchical formulation allows each policy to focus on a narrower scope of the overall task, simplifying the learning problem at each level.

Each level operates at a temporal resolution defined by a fixed subgoal horizon H, which specifies the maximum number of time steps the level below is allowed to act in pursuit of a given subgoal. Consequently, level i issues a new subgoal only every H^{L-i-1} environment steps. For instance, in a two-level hierarchy $L = 2$, the top level $i = 1$ selects a new subgoal every H steps, while the base level $i = 0$ executes actions at every time step. This nested temporal structure enables higher levels to perform long-term planning while lower levels handle short-horizon control, reducing the effective planning depth at each level (see Figure 5.8).

After a primitive action is executed, the environment returns the next state s', along with a sparse, goal-dependent binary reward r^0, observed only by the base level. Higher levels do not receive direct environmental rewards but instead compute intrinsic rewards based on subgoal achievement:

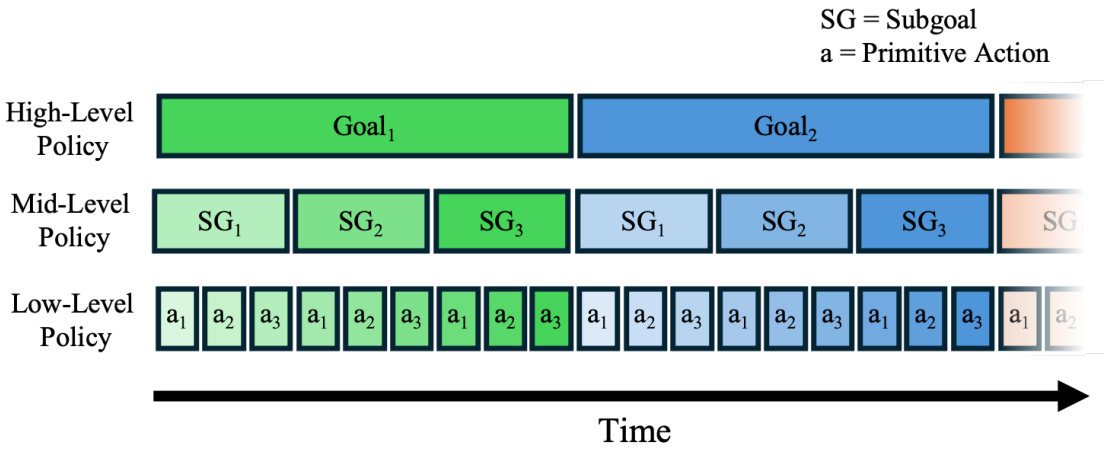


Figure 5.8: Temporal abstraction in a three-level HAC agent. Subgoal horizons define how often each level updates: higher levels act less frequently, issuing goals that persist across multiple lower-level decisions. This structure enables long-horizon planning at the top and fine-grained control at the bottom, while simplifying learning at each level.

$$r^i = \begin{cases} 0 & \text{if } \|s^{i-1} - g^{i-1}\| \leq \epsilon \\ -1 & \text{otherwise} \end{cases} \tag{5.1}$$

Here, s^{i-1} is the state observed by level $i-1$ after subgoal execution, and ϵ is a predefined success threshold. Each level maintains its own experience replay buffer $\mathcal{D}^i$ and stores transitions as tuples of the form: $(s^i, g^i, a^i, r^i, s'^i)$

Hindsight Relabelling

HAC incorporates two hindsight-based techniques to improve sample efficiency: Hindsight Experience Replay [158] and Hindsight Action Transitions. HER is applied at all levels and involves relabelling a transition by replacing the original goal with one that was actually achieved during the episode. For a given transition: $(s^i, g^i, a^i, r^i, s'^i)$ HER generates a new training sample by substituting the goal g^i with an achieved state $g' = s'^i$ and recomputing the reward accordingly:

$$(s^i, g^i, a^i, r^i, s'^i) \rightarrow (s^i, g', a^i, r', s'^i) \tag{5.2}$$

This allows the agent to extract meaningful learning signals from otherwise unsuccessful episodes.

HAT is used at higher levels ($i > 0$) to address the instability caused by the

evolving behavior of lower-level policies. Instead of relabeling the goal, HAT relabels the action (i.e., the subgoal) with the state that the lower level actually reached. Given a transition $(s^i, g^i, g^{i-1}, r^i, s'^i)$

HAT replaces the subgoal g^{i-1} with the final state achieved by level $i-1$, denoted s^{i-1}_{final}:

$$(s^i, g^i, g^{i-1}, r^i, s'^i) \rightarrow (s^i, g^i, s^{i-1}_{\text{final}}, r^i, s'^i) \tag{5.3}$$

This ensures the high-level policy is trained on outcomes that are actually feasible, stabilizing off-policy learning and improving data efficiency.

To prevent overreliance on hindsight relabeling, a fraction of transitions are left unrevised during training. These test transitions ensure that subgoals proposed by higher levels are achievable given the current capabilities of lower-level agents, maintaining alignment between levels and promoting realistic subgoal proposals.

Goal-Conditioned Policies and Critics

To enable goal-directed behavior, HAC extends standard actor-critic methods by introducing goal-conditioned actor-critic pairs at each level. These pairs are trained using DDPG, adapted to operate over both state and goal spaces. Unlike flat policies, which are conditioned only on the current state, each HAC policy also receives a goal input, either an intermediate subgoal or the overall task goal, depending on the level.

Each critic Q^i is implemented as a Universal Value Function Approximator (UVFA) [218], which generalizes the value function across both states and goals:

$$Q^i(s, g, a) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t^i \,\middle|\, s_0^i = s, g, a_0^i = a\right] \tag{5.4}$$

This formulation allows each level of the hierarchy to estimate the utility of an action with respect to a specific goal, and supports effective reuse of both actual and relabelled experiences. By training over a diverse distribution of goals, UVFAs promote generalization and facilitate transfer to new or recomposed goals without the need for retraining.

Each policy π^i is similarly defined over the joint state-goal space:

$$\pi^i(a \mid s, g) : \mathcal{S}^i \times \mathcal{G}^i \rightarrow \mathcal{A}^i \tag{5.5}$$

Actor and critic updates follow the standard DDPG algorithm, with Bellman tar-

gets and gradients computed over state-goal-action triples. These modifications form the basis of HAC's ability to operate effectively across multiple levels of abstraction and temporal resolution.

The overall training loop of the HAC framework, along with its hindsight-based transition logic, is detailed in the Appendix

5.3.2 Agent Structure and Training

Based on the superior performance demonstrated by the D-TD3 architecture in Chapter 3, the hierarchical reinforcement learning implementation employs two independent HAC agents: one for translational control (HAC-R) and one for rotational control (HAC-A). The overall controller architecture is shown in Figure 5.9. This preserves the interpretability and fault isolation benefits of domain separation while introducing temporal abstraction to address challenges of sparse reward signals and long-horizon planning.

The HAC implementation utilizes a two-level hierarchy for both translational and attitude control agents. This hierarchy depth was able to provide sufficient temporal abstraction for the mission duration while maintaining tractable subgoal spaces at each level. The hierarchy operates with a fixed time scale of 15 time steps between levels.

Each actor network within the hierarchy follows a shared neural architecture consisting of two fully connected layers with 64 nodes and ReLU activation functions. However, network inputs and outputs differ between hierarchy levels to accommodate the state and action spaces associated with subgoal planning for the higher levels versus primitive actions for the lowest level.

The translational agent operates in a 6D state space consisting of relative position and velocity, producing continuous force commands within the bounds $f_i \in [-0.1, 0.1]$ N. The rotational agent uses a 6D input of Euler angle error and angular velocity, outputting torque commands within $\tau_i \in [-0.001, 0.001]$N m. In both cases, higher levels generate subgoals within the same respective state spaces.

The terminal objective for each agent is to drive the system to a zero relative state (i.e., zero position and attitude error, and zero velocity), corresponding to successful docking alignment, as such the end-goals are encoded as fixed zero vectors in both hierarchies.

To support efficient exploration in both subgoal and primitive action spaces, stochasticity is introduced at multiple levels. Gaussian noise is added to both subgoal outputs

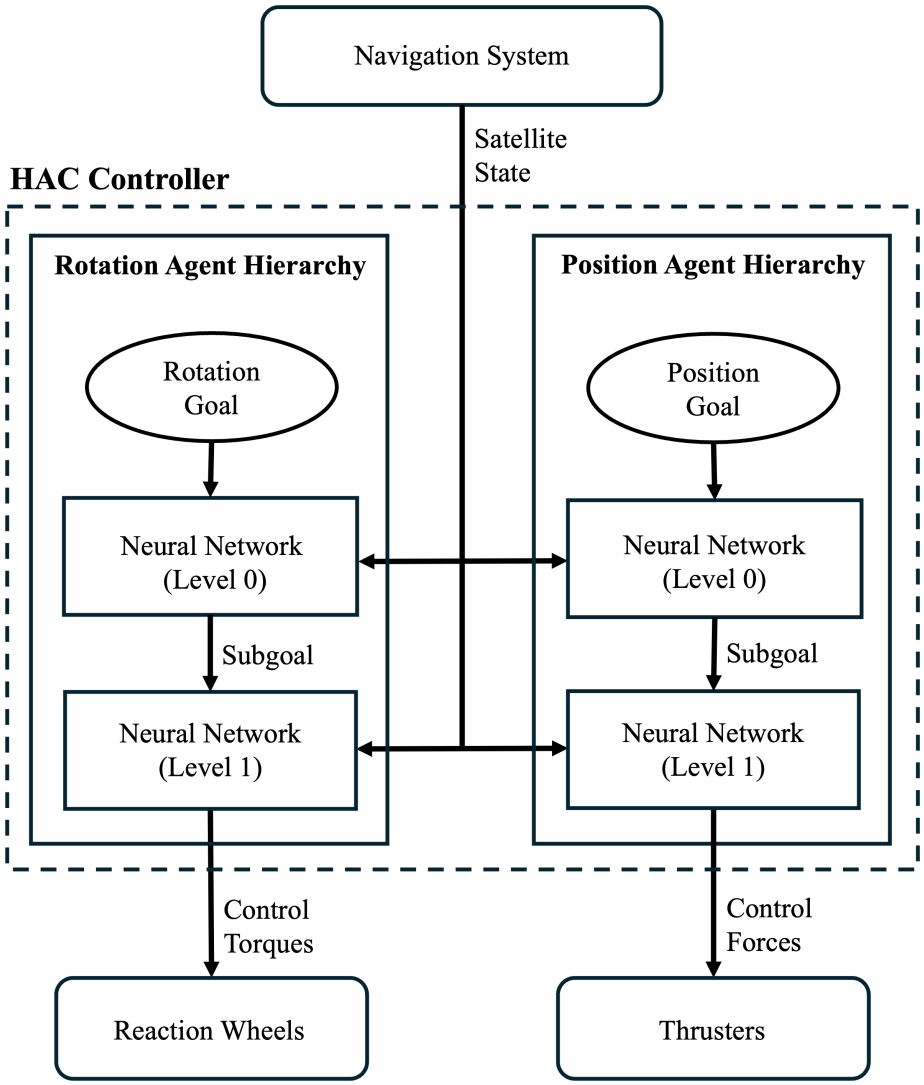


Figure 5.9: Structure of the HAC-based satellite controller. The controller consists of two Hierarchical Actor-Critic (HAC) agents, each with two levels of temporal abstraction. It receives navigation data as input; the translational agent outputs force commands, and the rotational agent outputs torque commands, each sent to its respective actuator.

and low-level actions, and a 30% subgoal testing rate is applied to verify the achievability of higher-level plans by lower-level policies. Additionally, a 10% random action probability is used during training to prevent premature convergence and ensure sufficient coverage of the state-action space.

Training is conducted using separate replay buffers per hierarchy level, with a capacity of 1024 episodes each. Learning commences after an initial exploration phase of 100 episodes to populate the buffers. Training is conducted using the Adam optimiser

with a learning rate of 0.0001, a discount factor of 0.95, and mini-batches of size 1024. A summary of key training parameters is provided in Table 5.1.

Table 5.1: Training parameters for the hierarchical HAC agents.

Parameter	**Value**
Learning rate	0.0001
Discount factor (γ)	0.95
Batch size	10^7 transitions
Mini-batch size	1024
Replay buffer size (per level)	1024 episodes
Optimizer	Adam
Network architecture	[64, 64]
Exploration episodes before training	100
Subgoal testing rate	30%
Random action probability	10%
Hierarchy depth	2 levels
Subgoal horizon	15 steps

Training is conducted in the same idealised environment used in 3, featuring idealised actuators and navigation feedback. Each episode terminates either upon successful convergence to the docking condition or after 2000 simulation steps (equivalent to 200 s). This HDRL framework forms the foundation for subsequent comparison against both classical and deep RL controllers in high-fidelity simulation and hardware-in-the-loop evaluation.

5.3.3 Simulation results

To establish the performance baseline for hierarchical reinforcement learning controller and provide comparison with classical control approaches, the HAC implementation was evaluated against a conventional PID controller. This comparison aims to assess the viability of hierarchical RL policies in realistic simulation settings.

The PID controller was implemented using a decentralized architecture, comprising separate controllers for the translational and rotational subsystems. The translational PID operated on position and linear velocity errors to generate thrust commands, while the rotational PID controlled attitude and angular velocity errors to compute torque commands. The PID gains were hand-tuned through iterative simulation trials to ensure stable and responsive behavior in the nominal operating environment, yielding the following final values:

- Translational PID: $K_p = 0.3$, $K_i = 1e-5$, $K_d = 10$
- Rotational PID: $K_p = 1.55$, $K_i = 1e-5$, $K_d = 7.5$

Monte Carlo Evaluation Results

Both controllers were evaluated using the same Monte Carlo protocol established in Section 3.4.3, consisting of 100 randomized trials per controller. Initial relative positions and orientations were sampled uniformly within mission-defined bounds, with zero initial velocity. Simulation conditions included actuator saturation, quantisation effects, sensor noise, and other disturbances to emulate operational constraints.

Table 5.2: Summary statistics for PID and HAC across 100 trials.

Metric	**PID**	**HAC**
Final Position Error [m]	2.53 ± 0.89	2.76 ± 1.11
Max Linear Velocity [m/s]	1.20 ± 0.60	1.11 ± 0.68
Time to Convergence (Position) [s]	61.96 ± 27.27	87.13 ± 45.10
Final Orientation Error [deg]	0.45 ± 0.18	0.95 ± 0.42
Max Angular Velocity [deg/s]	2.41 ± 0.93	8.91 ± 4.24
Time to Convergence (Orientation) [s]	12.22 ± 4.16	29.28 ± 17.05

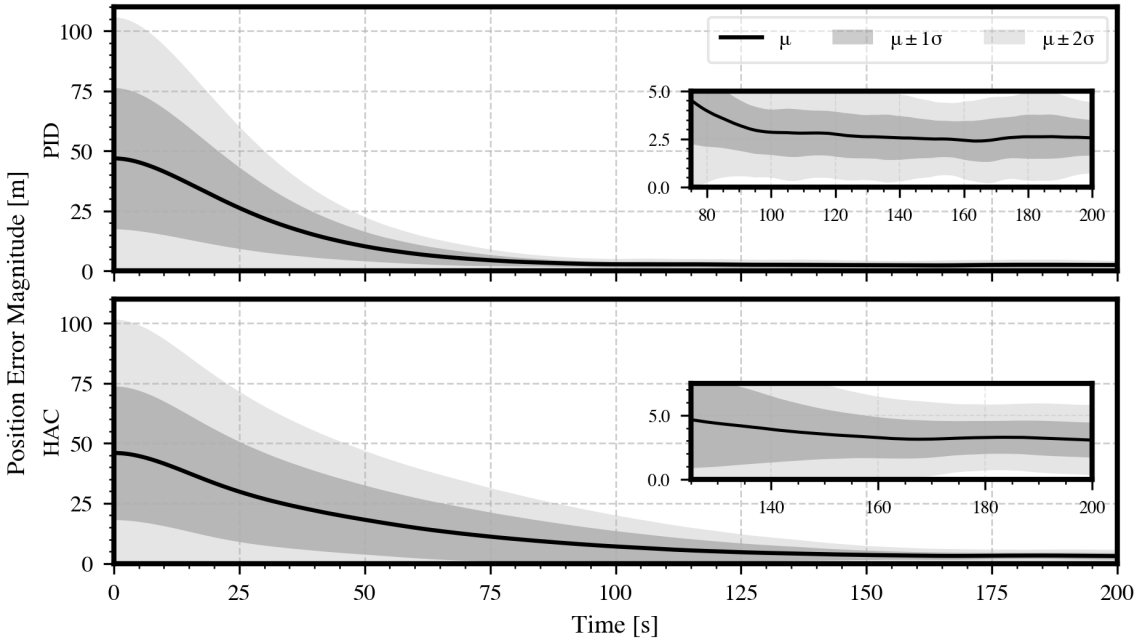


Figure 5.10: Position error magnitudes for the PID and HAC controllers across 100 trials.

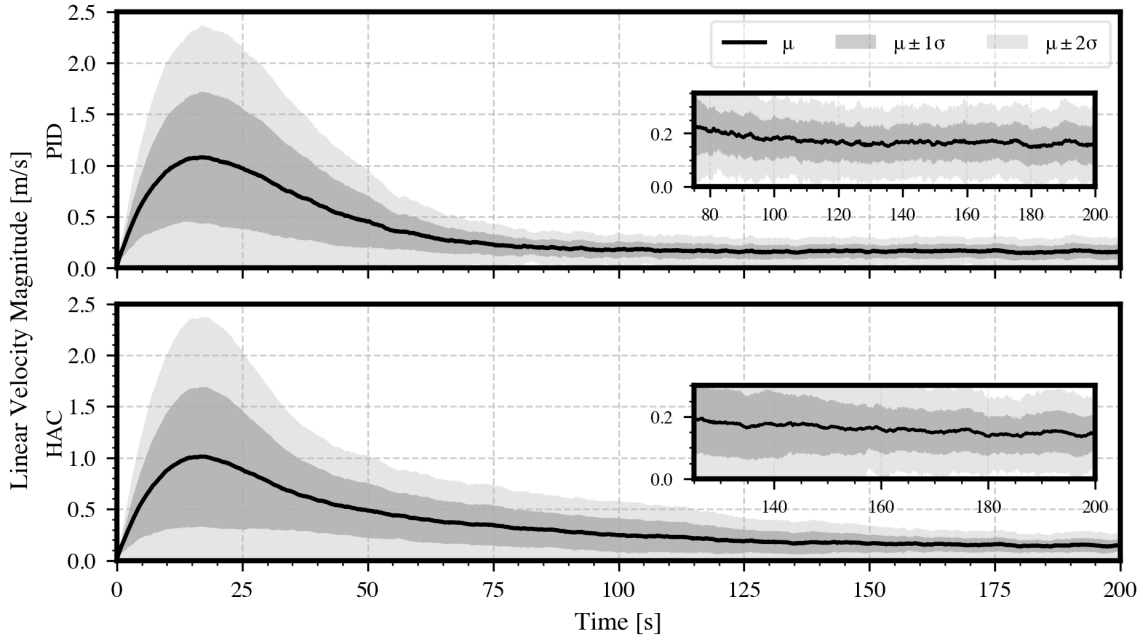


Figure 5.11: Linear velocity magnitudes for the PID and HAC controllers across 100 trials.

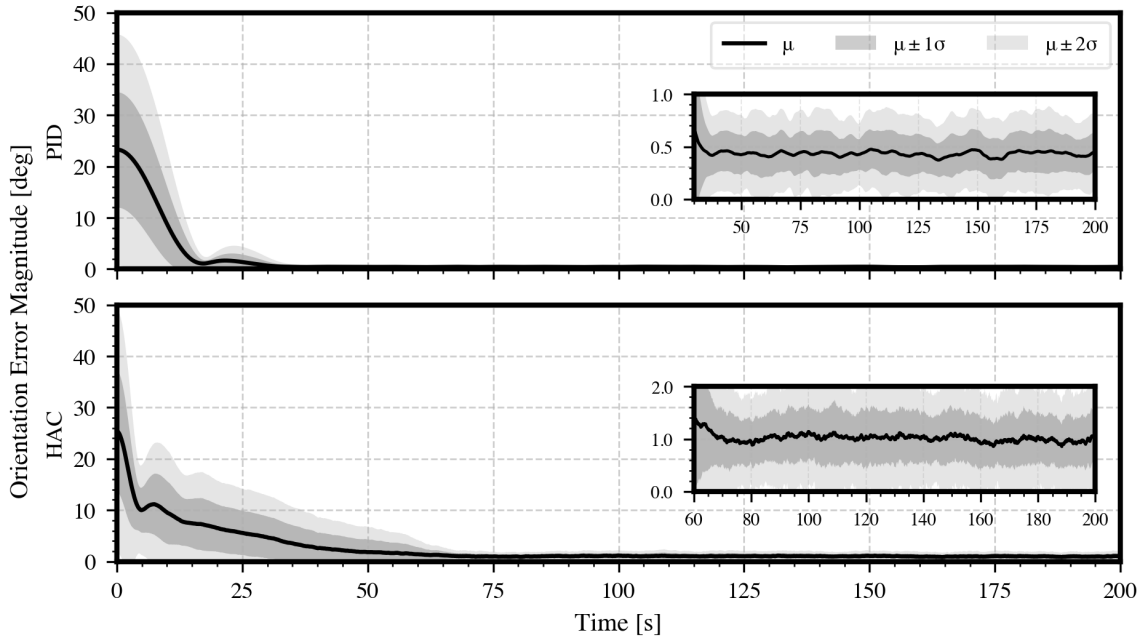


Figure 5.12: Orientation error magnitudes for the PID and HAC controllers across 100 trials.

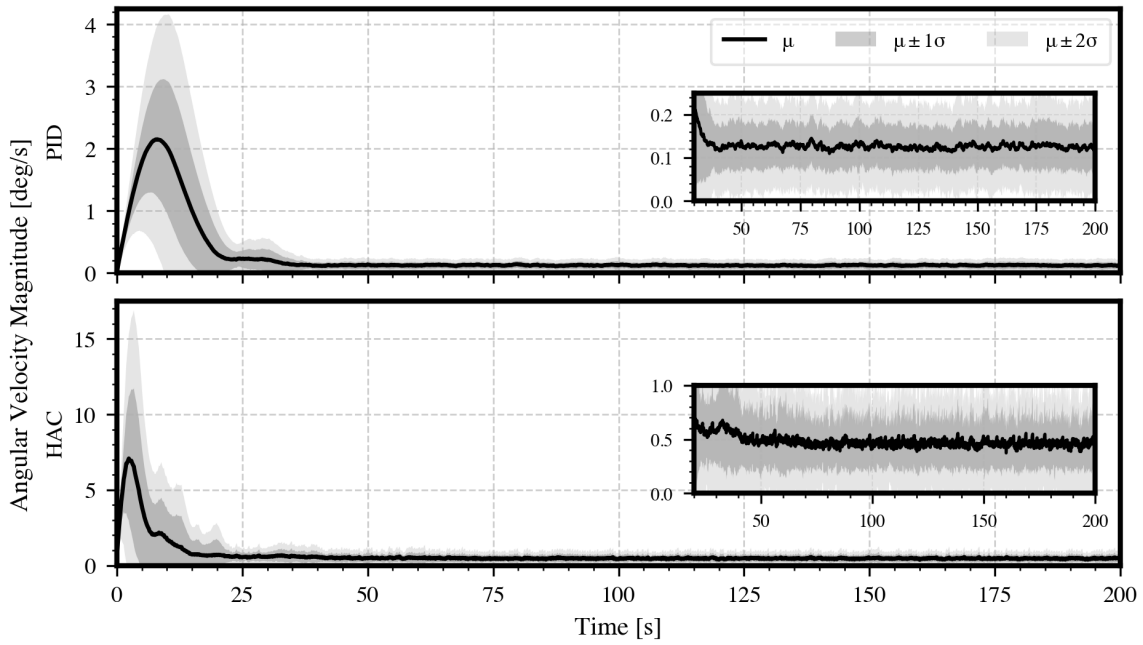


Figure 5.13: Angular velocity magnitudes for the PID and HAC controllers across 100 trials.

The performance profiles shown in Figures 5.10 5.13 and the statistics in Table 5.2 indicate broadly comparable final performance between HAC and PID. Both controllers achieved final position error norms below 3 m and final orientation errors within 1 °, satisfying the defined mission criteria.

In terms of translational behaviour, the PID controller converged slightly faster (61.96 ± 27.27 s) than HAC (87.13 ± 45.10 s). However, the difference in final position error between the two controllers was small (2.53 m vs. 2.76 m). Both controllers maintained peak linear velocities near 1.2 $\mathrm{m\,s^{-1}}$, indicating similarly conservative approach profiles.

More distinct differences appeared in the rotational domain. PID achieved faster orientation convergence (12.2 s vs. 29.3 s) and lower final orientation error (0.45 ± 0.18 ° vs. 0.95 ± 0.42 °). Peak angular velocity was higher for HAC (8.91 ± 4.24 $^\circ\,\mathrm{s^{-1}}$) compared to PID (2.41 ± 0.93 $^\circ\,\mathrm{s^{-1}}$), suggesting that the learned policy relied on more aggressive corrections, particularly in the early stages.

To contextualise HAC's performance within the broader reinforcement learning framework, it is informative to compare it with the D-TD3 architecture previously evaluated in Chapter 3. D-TD3 demonstrated faster convergence in both position (58.27 ± 30.28 s) and orientation (19.74 ± 12.28 s), along with lower final errors ($2.22 \pm$

0.87 m and 0.80 ± 0.31 °, respectively). D-TD3 also maintained lower peak angular velocities (2.81 ± 1.12 $^\circ\,s^{-1}$) than HAC, though similar peak linear velocities were observed. These results suggest that while HAC performs slightly behind D-TD3, its overall performance remains competitive.

Direct Comparison Under Shared Initial Conditions

To further explore the control behaviour of both architectures, a representative trial was selected using the same initial conditions employed in previous simulation sections. The trajectories and control profiles for both PID and HAC under these conditions are shown in Figures 5.14 – 5.19.

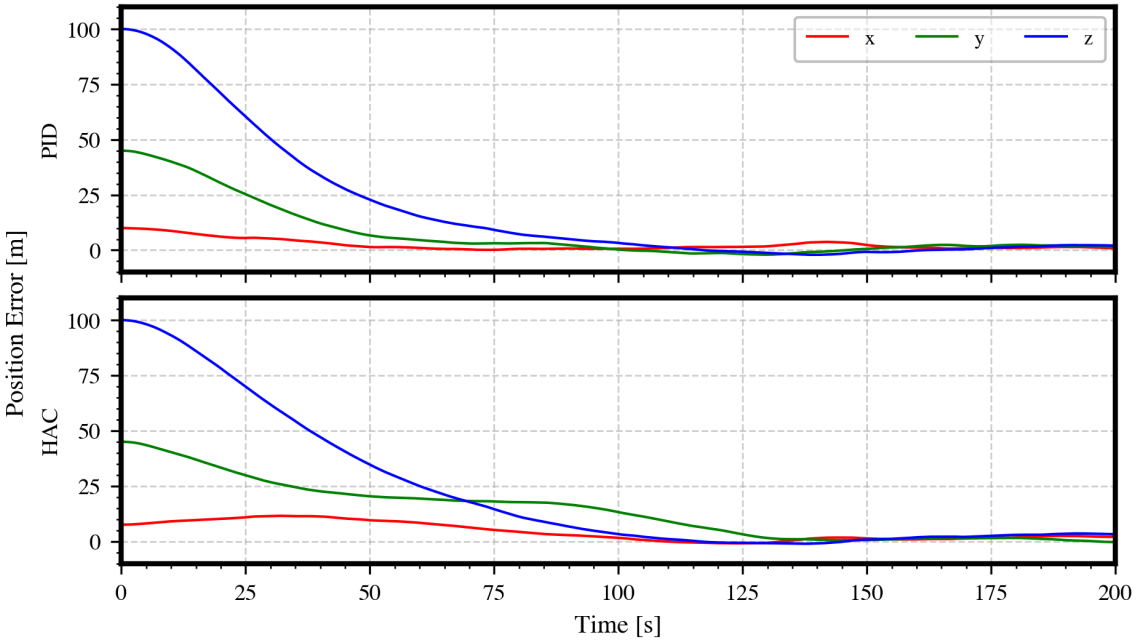


Figure 5.14: Position errors for the PID and HAC controllers during the representative trial.

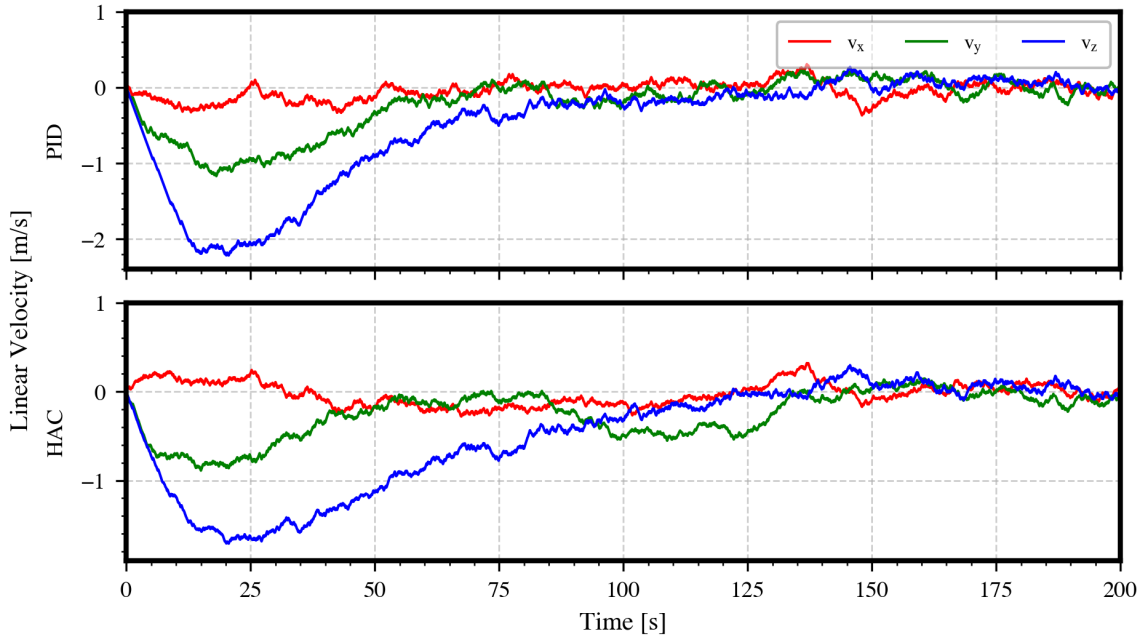


Figure 5.15: Linear velocities for the PID and HAC controllers during the representative trial.

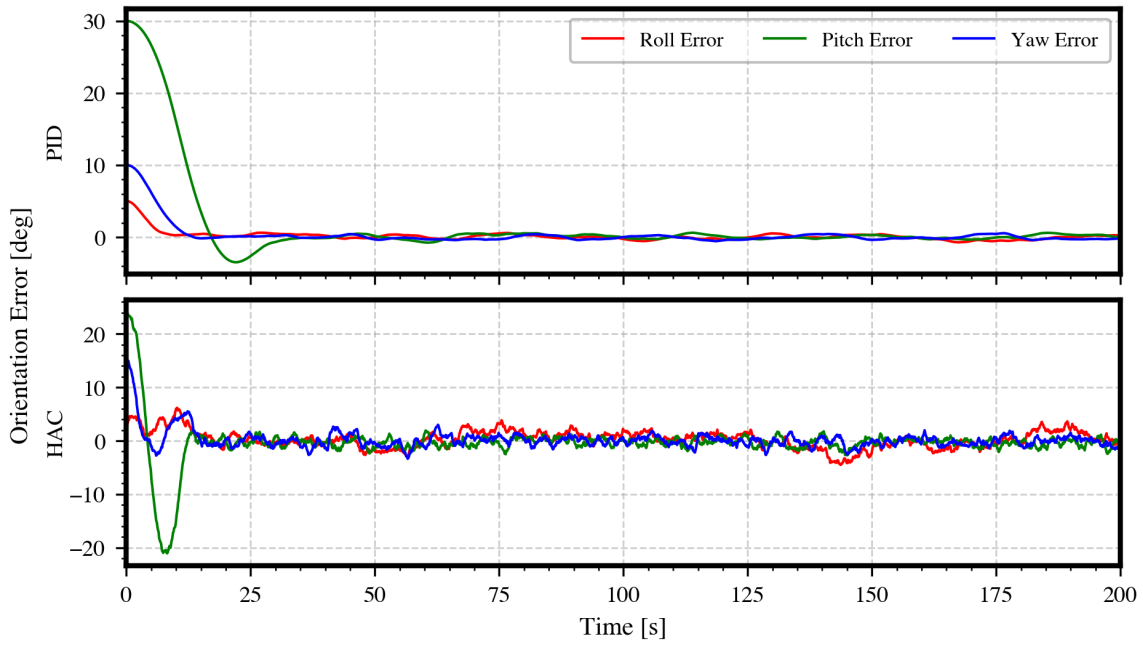


Figure 5.16: Orientation errors for the PID and HAC controllers during the representative trial.

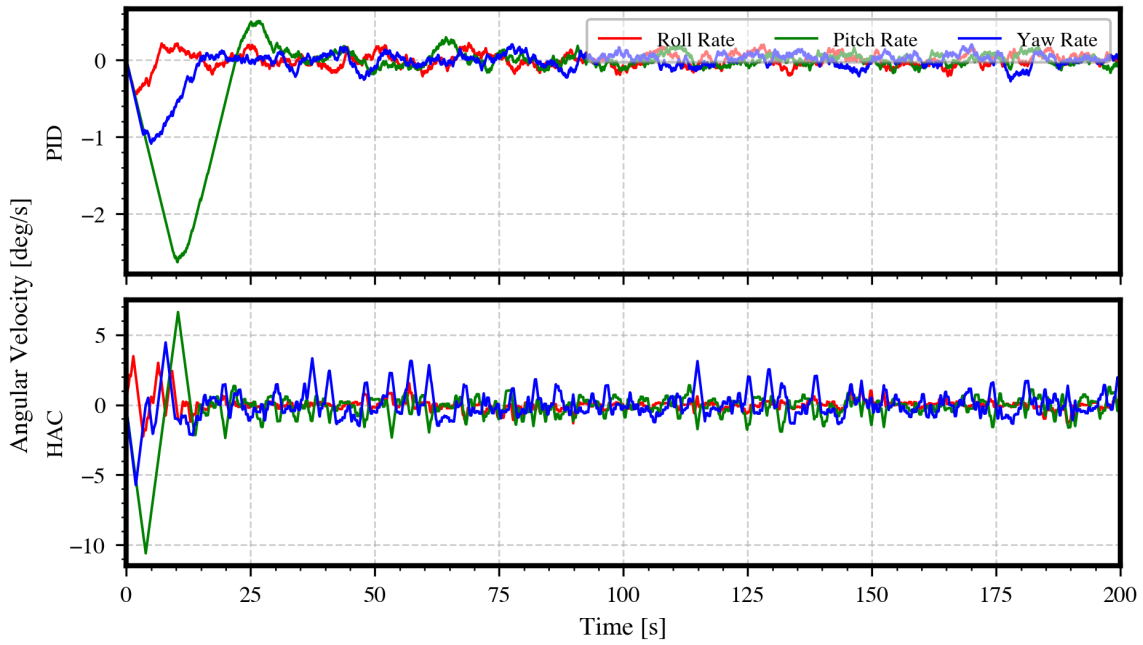


Figure 5.17: Angular velocities for the PID and HAC controllers during the representative trial.

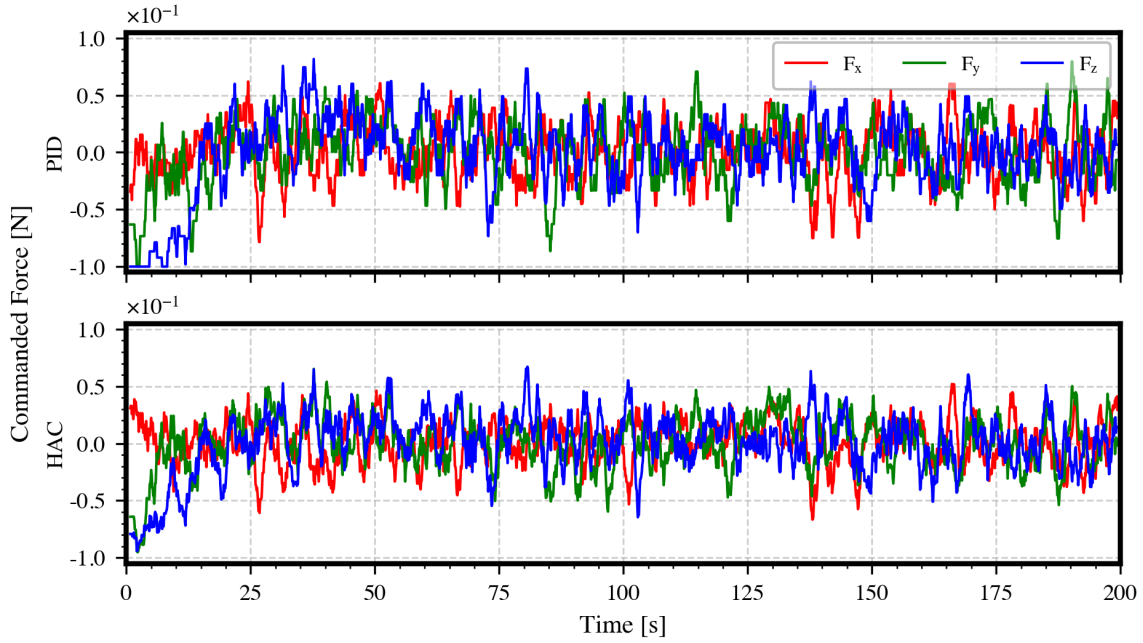


Figure 5.18: Commanded forces from the PID and HAC controllers during the representative trial.

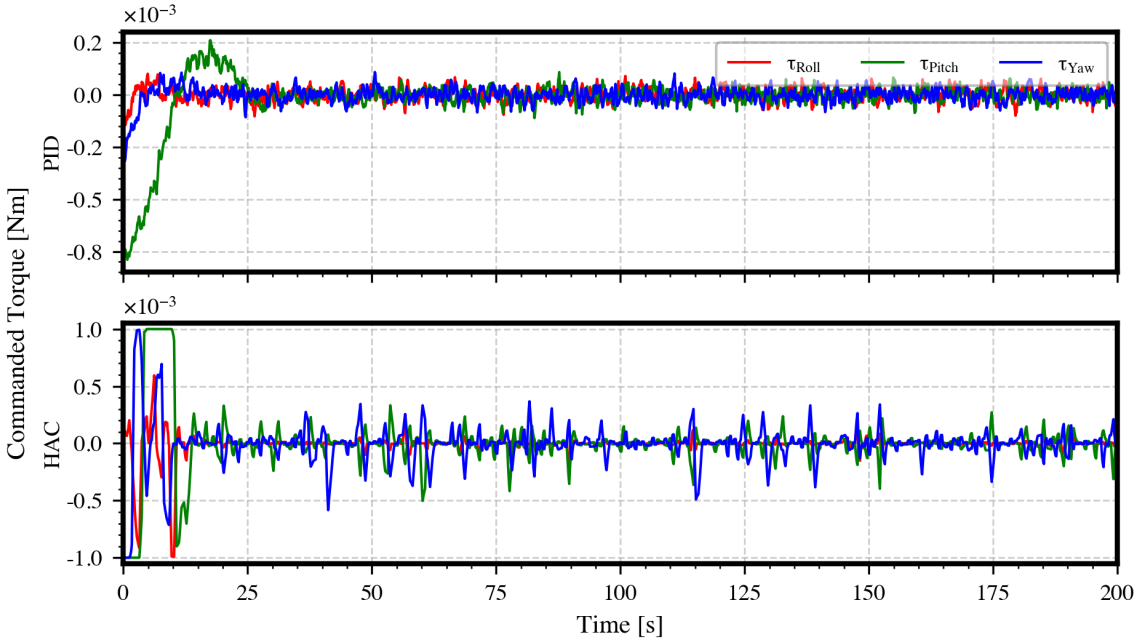


Figure 5.19: Commanded torques from the PID and HAC controllers during the representative trial.

In this example, the PID controller maintained a smooth approach trajectory, with peak velocity around 1.2 $\mathrm{m\,s^{-1}}$ and convergence within 60 s. Orientation error reduced rapidly and remained below 1 ° after 15 s. Control commands were initially saturated, but quickly settled to smooth, bounded responses.

HAC followed a similar translational trajectory, with slightly longer convergence time and more fluctuation in velocity. Rotational behaviour was more dynamic, with angular velocities exceeding 8 $^\circ\,\mathrm{s^{-1}}$ during early corrections. Alignment was achieved in approximately 30 s, consistent with the average observed across Monte Carlo trials.

In comparison, the previously evaluated D-TD3 controller also completed this maneuver, exhibiting a convergence time of approximately 55 s and lower angular velocity peaks under 3 $^\circ\,\mathrm{s^{-1}}$, reflecting smoother control. While both PID and D-TD3 offered slightly faster or more stable convergence, HAC achieved successful task completion with similar final performance.

These results demonstrate that the HAC controller offers a viable alternative to both classical and learned control architectures. While it does not match the precise convergence or control smoothness of the PID or D-TD3 controllers, it remains within mission-compliant performance bounds and exhibits consistent task completion across a broad set of initial conditions.

5.4 Hardware-in-the-Loop Testing

This section describes the hardware-in-the-loop testing platform developed to validate the guidance and attitude control systems presented in this work. The goal of the platform is to enable physical testing of trained control policies using embedded hardware and representative actuation mechanisms. The system is designed to replicate key physical and computational characteristics of a CubeSat while remaining compatible with laboratory constraints. It supports both individual subsystem testing and integrated 6-DoF evaluation through modular components. The following subsections outline the design and implementation of the mock CubeSat, the attitude dynamics testing platform, and the rendezvous mission simulator.

5.4.1 Mock CubeSat setup

This section describes the design and implementation of a mock 1U CubeSat used for hardware-in-the-loop validation of the autonomous guidance and attitude control strategies developed in this work. The goal of this platform is to replicate the mechanical and control characteristics of a real CubeSat, while enabling controlled experimentation with onboard embedded guidance and control algorithms. The mock satellite is designed to be representative of a 1U CubeSat, while remaining functional within the constraints of a laboratory testing environment.

System Overview and Design Requirements

The platform serves as a physical validation testbed for the guidance and control outputs generated by the trained reinforcement learning agents. The CubeSat is required to sense its own pose, process control logic onboard in real-time, the control policy in realtime, and execute commanded torques using internal actuators. To meet this functionality, the design was constrained by the size and mass limits of the CubeSat standard [219]: a 10 cm cube with a maximum mass of 2 kg.

The CubeSat consists of a two part external housing 3D-printed using a Markforged Onyx Pro printer [220] employing Fused Filament Fabrication (FFF). This additive manufacturing method deposits layers of carbon-fiber reinforced nylon, producing a part with high structural strength with minimal mass. It is printed in two interlocking halves, allowing internal access for component integration and reconfiguration.

The internal hardware was selected to meet the functional and physical requirements of the platform while minimising weight. The MPU-6050 IMU provides 6D

gyroscope and accelerometer data allowing for onboard angular velocity and orientation estimates in real time. The ESP32 microcontroller acts as the central processing unit, hosting the trained RL agent, communicating with sensors and actuators, and handling wireless telemetry. Its dual-core processor, low power consumption, and integrated Bluetooth/Wi-Fi capabilities make it suitable for embedded applications in constrained environments. Power is supplied by a 3S (11.1 V) lithium-polymer battery, selected for its high energy density and low mass. Actuation is provided by three compact Faulhaber brushless DC motors [221] with integrated speed controllers, allowing for direct PWM control. These motors are mounted orthogonally to drive the reaction wheels described in the following section.

Reaction Wheel Design and Characterisation

Three orthogonally mounted reaction wheels provide full 3-axis attitude control. The reaction wheels were designed and manufactured in-house using stainless steel to achieve high inertia in a compact form. To maximise angular momentum while keeping the system compact, the reaction wheel geometry was designed as a disc with a raised outer ring, this geometry is shown in Figure 5.20. Concentrating mass near the perimeter increases the moment of inertia without significantly increasing volume. The inner disc provides structural support, while the raised outer rim contributes the majority of the rotational inertia.

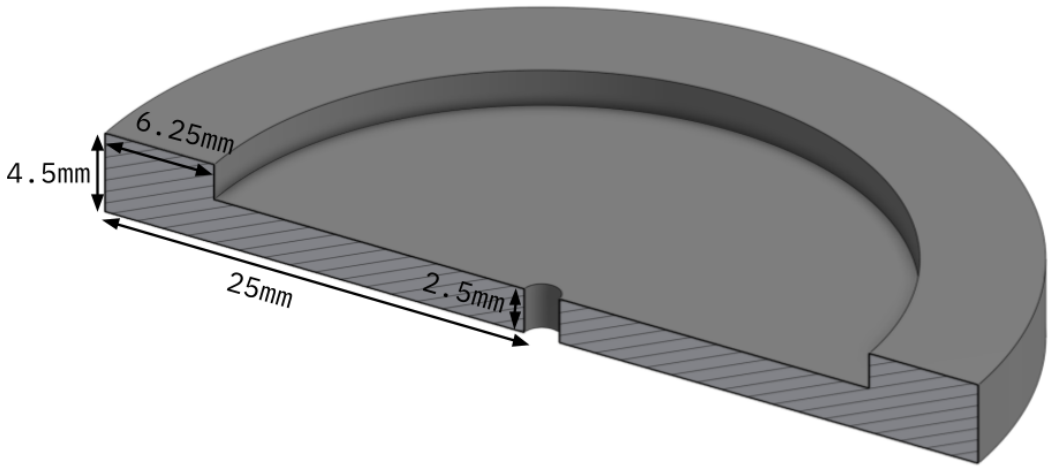


Figure 5.20: CAD model of the reaction wheel with mass-concentrated disc-ring geometry.

Table 5.3: Reaction wheel design parameters

Parameter	**Value**
r_{disk} (inner radius)	18.75 mm
r_{ring} (outer radius)	25.00 mm
h_{disk}	2.5 mm
h_{ring}	4.5 mm
m_{disk}	0.022 kg
m_{ring}	0.031 kg

Design parameters of the reaction wheels are provided in Table 5.3. The total moment of inertia I_{rw} of each wheel is the sum of the contributions from the inner disc and the ring:

$$I_{\text{disk}} = \frac{1}{2} m_{\text{disk}} r_{\text{disk}}^2 = 3.87 \times 10^{-6} \text{ kg m}^2 \tag{5.6}$$

$$I_{\text{ring}} = \frac{1}{2} m_{\text{ring}} (r_{\text{ring}}^2 + r_{\text{disk}}^2) = 6.30 \times 10^{-4} \text{ kg m}^2 \tag{5.7}$$

$$I_{\text{rw}} = I_{\text{disk}} + I_{\text{ring}} = 6.34 \times 10^{-4} \text{ kg m}^2 \tag{5.8}$$

At a motor speed of 7000 rpm, the maximum angular momentum deliverable by a single wheel $L_{\text{rw, max}}$ is:

$$L_{\text{rw,max}} = I_{\text{rw}} \cdot \omega_{\text{max}} = 6.34 \times 10^{-4} \cdot \left(7000 \cdot \frac{2\pi}{60} \right) \approx 0.077 \text{N m s} \tag{5.9}$$

To estimate responsiveness, the minimum time t it will take for the reaction wheel to rotate the CubeSat 90° degrees can be approximated using the CubeSat's maximum moment of inertia I_{max} [222]:

$$t = \frac{2 I_{\text{sat}} \theta}{L_{\text{rw,max}}} \tag{5.10}$$

For $I_{\text{sat}} = 0.00167 \text{ kg m}^2$ and $\theta = \pi/2$, this yields:

$$t = \frac{2 \cdot 0.00167 \cdot \pi/2}{0.077} \approx 3.90 \text{ s} \tag{5.11}$$

Assembly, Integration, and Balancing

The output of the trained guidance controller is a desired body torque vector $\mathbf{T}_c$. To generate this torque using internal reaction wheels, the required change in angular momentum must be computed and translated into new motor speeds. The relationship between commanded torque and wheel speed is [223]:

$$\mathbf{T}_c = -\frac{d\mathbf{L}_{rw}}{dt} = -I_{rw}\frac{d\boldsymbol{\omega}_{rw}}{dt} \tag{5.12}$$

$$\Delta\boldsymbol{\omega}_{rw} = -\frac{\mathbf{T}_c \cdot \Delta t}{I_{rw}} \tag{5.13}$$

The current wheel speeds are measured using onboard Hall-effect sensors. At each control step, the desired change in angular velocity is calculated, and a new target is set. A simple PID controller, running on the ESP32, then adjusts PWM signal to reach this velocity target. This closed-loop control ensures smooth tracking of torque commands.

To facilitate stable motion on the frictionless attitude testing platform the CubeSat's center of mass was aligned with its geometric center. This was achieved through iterative balancing using small adhesive masses placed on selected faces of the frame. A full CAD model was created to pre-visualise internal layout and aid in manual balancing.

For accurate physical validation, the CubeSat must rotate cleanly about its geometric center on the testbed. To achieve this, its center of mass was aligned with the geometric center of the 3D-printed frame. A full CAD model was created to support both component layout and mass balancing. Small adhesive weights were added to correct residual offsets. Figure 5.21 shows the final assembly of the platform.

The resulting CubeSat hardware platform provides a compact, modular, and dynamically relevant testbed for evaluating onboard attitude control systems. Its integration of sensing, computation, and actuation within a 1U form factor enables full closed-loop testing of intelligent control policies in a physical setting, bridging the gap between simulation and deployment.

5.4.2 The Attitude Dynamics Testing Platform

To validate the dynamic performance of the attitude control system in a physical environment, a custom-built hemispherical air bearing platform was designed and

Figure 5.21: Final assembly of the 1U CubeSat hardware platform.

constructed. This platform allows the mock CubeSat to rotate with minimal friction, enabling near-torque-free motion about three rotational degrees of freedom. Its primary function is to enable hardware-in-the-loop testing where the closed-loop behaviour where the closed-loop behaviour of the attitude control system can be observed under realistic operating conditions.

The platform consists of three core subsystems: the rotor, the stator, and the compressed air delivery system. The rotor is a commercially sourced hollow acrylic hemisphere with an outer diameter of 200 mm and a mass of approximately 100 g. It holds the mock CubeSat during testing and serves as the rotating component of the system. Acrylic was selected due to its smooth surface finish, rigidity, and low mass, all of which contribute to low-friction levitation and stable rotation about each axis.

Beneath the rotor sits the stator, a custom-designed structure made up of two parts; a bottom manifold section which receives the compressed air and distributes it through internal chambers and a top bowl-shaped section that directs compressed air through a ring of angled vents on its upper surface. The vent design generates both vertical lift and radial centering forces. The airflow effectively suspends the rotor,

while the geometry of the vents encourages it to remain centred within the bowl, thereby preserving rotational stability. Finally, the compressed air delivery system provides regulated airflow to the stator, delivering a constant pressure via pressure regulator.

The stator was manufactured using additive manufacturing, chosen for its ability to produce the geometrically complex internal features, such as integrated air channels and angled outlet vents. Unlike traditional subtractive manufacturing, 3D printing allows rapid iteration of custom geometries and integration of multiple functionalities into a single structure, reducing both fabrication time and cost.

Two types of printing technologies were used based on the performance requirements of each section. The bottom manifold was printed using a Markforged Onyx Pro printer, while the top bowl section, which interfaces directly with the rotor, was printed using Selective Laser Sintering (SLS) on a Formlabs Fuse 1+ system [224]. SLS was selected for its higher resolution and improved surface finish, which were necessary to achieve a uniform airflow and reliable rotor levitation. An exploded view of the stator is shown in Figure 5.22, and the air outlet geometry is shown in Figure 5.23.

To ensure an airtight seal between the stator's two sections, a press-fit joint was designed with minimal tolerances. Prior to assembly a conformable sealing compound was applied along the mating surfaces to compensate for any imperfections in the 3D-printed surfaces. This method was chosen over adhesives to allow disassembly and reconfiguration without permanent bonding. The fit was designed to be tight enough to resist separation under internal pressure while maintaining the integrity of the airflow chamber.

The lower section of the stator contains the compressed air inlet port. A brass threaded insert was heat-set into the 3D print to create a strong, reusable threaded connection point for the pneumatic interface. A push-to-connect fitting was screwed into the brass insert, allowing direct attachment of 6mm pneumatic tubing. The airtightness of the inlet assembly was improved using thread-seal tape and verified via leak testing under 8 bar of pressure.

The air outlets on the stator were arranged in a ring around the bowl surface, designed to be radially inward-facing and tilted at a shallow angle, based on recommendations in prior literature [194]. This orientation was selected so that the compressed air provides not only vertical lift to counteract the weight of the rotor, but also horizontal restoring forces that center the rotor in the bowl. As the air escapes through the vents, any lateral displacement of the rotor leads to an imbalance in pres-

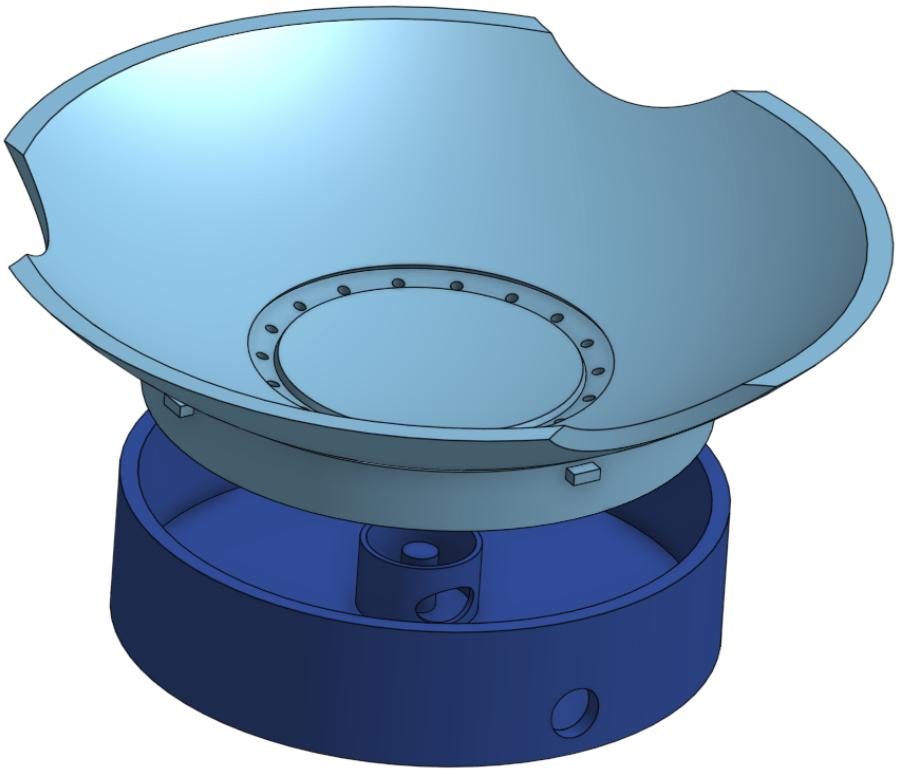


Figure 5.22: Exploded view of the two stator parts showing the internal air chamber.

sure forces, automatically nudging it back toward the center. This passive centering mechanism improves the stability of the platform and prevents drift during operation.

Upstream of the platform, a pneumatic pressure regulator reduces the pressure from the laboratory's central air system to a steady 6 bar. From there, flexible pneumatic tubing connects the regulator output to the stator inlet via the previously described fitting. An image of the completed platform with the CubeSat mounted is shown in Figure 5.24.

5.4.3 The Rendezvous Mission Simulator

To enable hardware-in-the-loop validation of translational guidance strategies, a rendezvous mission simulator was developed. This platform uses a robotic manipulator to emulate the motion of a CubeSat during short-range proximity operations. The system allows for 3-DoF translational testing using the mock CubeSat alone, or full 6-DoF testing when integrated with the air-bearing-based attitude simulator described

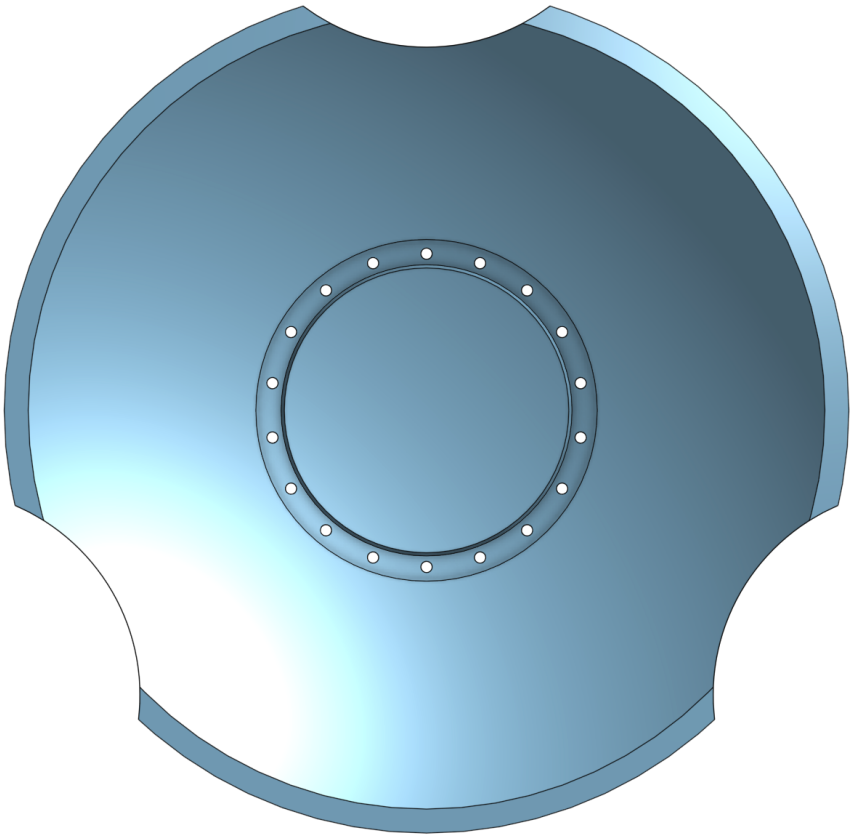


Figure 5.23: Top view of the stator, showing the ring of air vents.

in the previous section.

The Kinova Gen3 robotic arm was selected as the actuation platform. This 6-DoF serial manipulator supports both Cartesian position and velocity control via the KINOVA KORTEX API. It has a maximum payload capacity of 2 kg and a reach of 902mm [217], providing sufficient workspace and lifting capacity for the mock CubeSat and associated hardware. All translational motion is scaled to a 1:100 ratio, such that a 1m displacement in the lab corresponds to 100m in orbit. The arm's internal controller uses joint encoders, torque sensors, and trajectory planning to execute smooth, real-time motion. Control commands are issued from a central PC hosting the trained guidance and control algorithms.

The translational guidance controller outputs a force command $\mathbf{F}_c$ at each timestep. Since the Kinova arm does not directly accept force inputs, these must be mapped to a Cartesian velocity $\mathbf{v}_c$ for execution. Assuming a control interval Δt and known satellite mass m, the mapping is derived from Newton's second law as follows:

Figure 5.24: Assembled air bearing platform with CubeSat mounted.

$$\mathbf{v}_c = \frac{\mathbf{F}_c \cdot \Delta t}{m} \tag{5.14}$$

In this setup, $m = 0.5$ kg and $\Delta t = 0.01$ s, providing adequate resolution for emulating low-thrust satellite maneuvers while remaining within the actuation limits of the robotic arm.

Internally, the Kinova controller maps the Cartesian velocity command to joint velocities, this transformation is handled automatically by the arm's firmware, allowing the external controller to operate entirely at the Cartesian level. The result is a physical emulation of the desired translational acceleration of the CubeSat in response to the output of the learned policy.

To facilitate testing across different configurations, two custom mounting brackets were fabricated and attached to the robotic end-effector. The first allows for the direct attachment of the CubeSat mockup, while the second supports the full integration of the mock satellite with the attitude simulation platform. This modular approach enables both isolated translational testing and full 6-DoF dynamic validation within the same hardware framework. Photographs of both setups are shown in Figures 5.25 and 5.26.

Figure 5.25: Rendezvous testing platform with mock CubeSat mounted.

The rendezvous mission simulator provides a high-fidelity testbed for evaluating guidance policies under physical constraints. By reproducing the translational dynamics of a low-thrust CubeSat within a scaled environment, the platform enables direct observation of closed-loop tracking performance and dynamic response, bridging the gap between simulation and real-world operation.

5.4.4 Hardware-in-the-Loop Results

The hardware-in-the-loop testbed was developed to evaluate the real-time performance and robustness of the HDRL and D-TD3 controllers under physically grounded conditions. The primary objectives of this validation phase were:

- To bridge the gap between idealised simulation and real-world embedded implementation
- To assess the ability of each controller to cope with realistic disturbances, com-

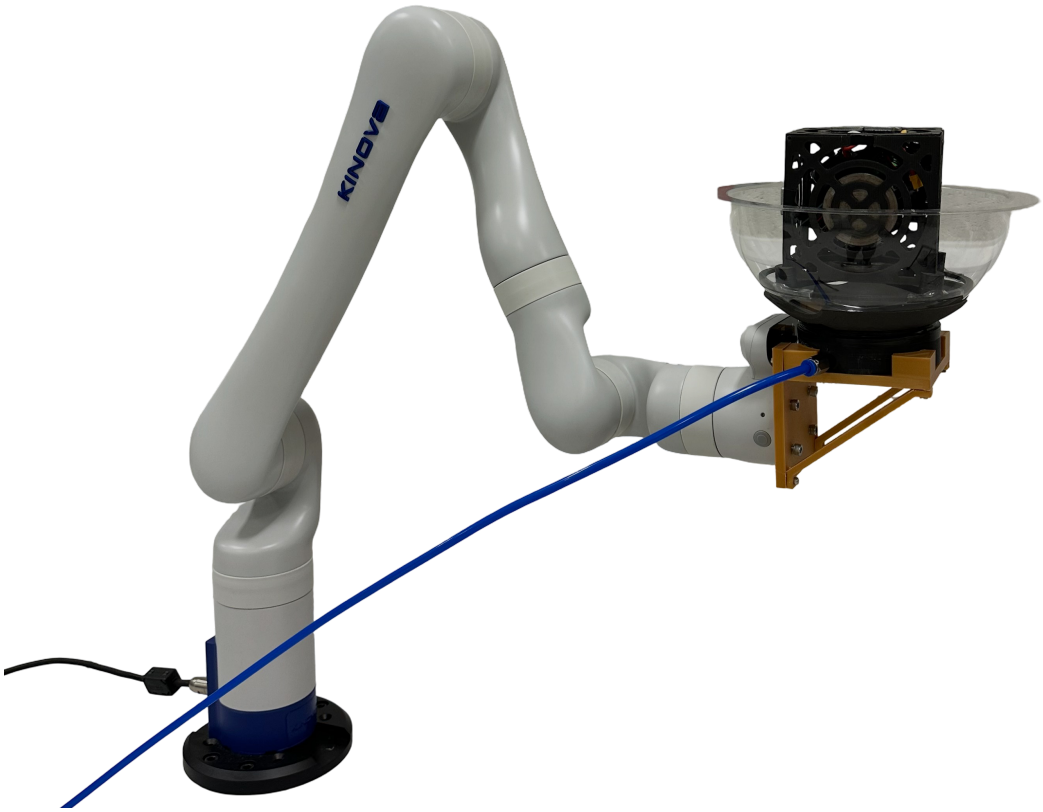


Figure 5.26: Combined rendezvous and attitude platform with CubeSat on air bearing.

munication latency, sensor noise, and actuation imperfections that are not fully captured in simulation.

The experimental platform employs a modular control architecture, where translational and rotational control agents operate in distinct but concurrent execution environments. The rotational agent executes onboard an ESP32 microcontroller embedded within a mock CubeSat. The translational agent, which handles Cartesian positioning, runs externally on a host machine and interfaces with a Kinova robotic arm that emulates satellite translation through velocity control commands.

Both controllers receive navigation data at 10 Hz from the IMU, either directly via a wired connection in the case of the rotational controller or over Wi-Fi in the case of the translational agent. This configuration permits closed-loop 6-DoF control, enabling evaluation of both D-TD3 and HAC controllers on a full rendezvous and alignment task. Each trial was initialised from a fixed relative position, while attitude misalignment was manually induced to simulate tumbling primarily about the yaw axis, with disturbances also applied to pitch and roll.

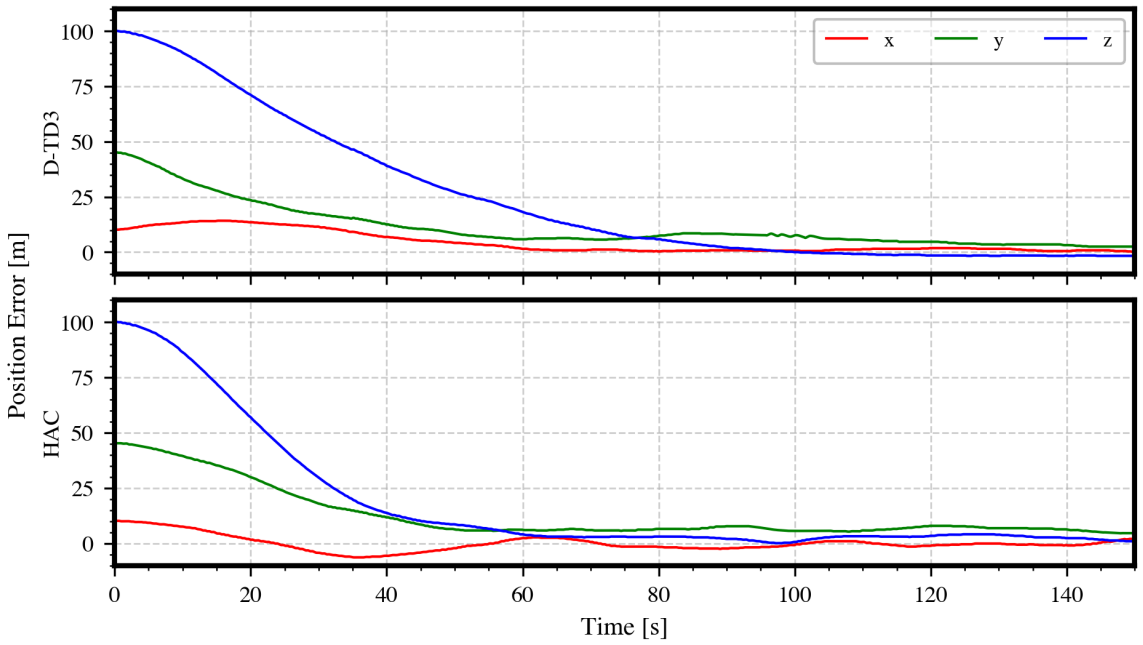


Figure 5.27: Position errors for the D-TD3 and HAC controllers during HIL testing.

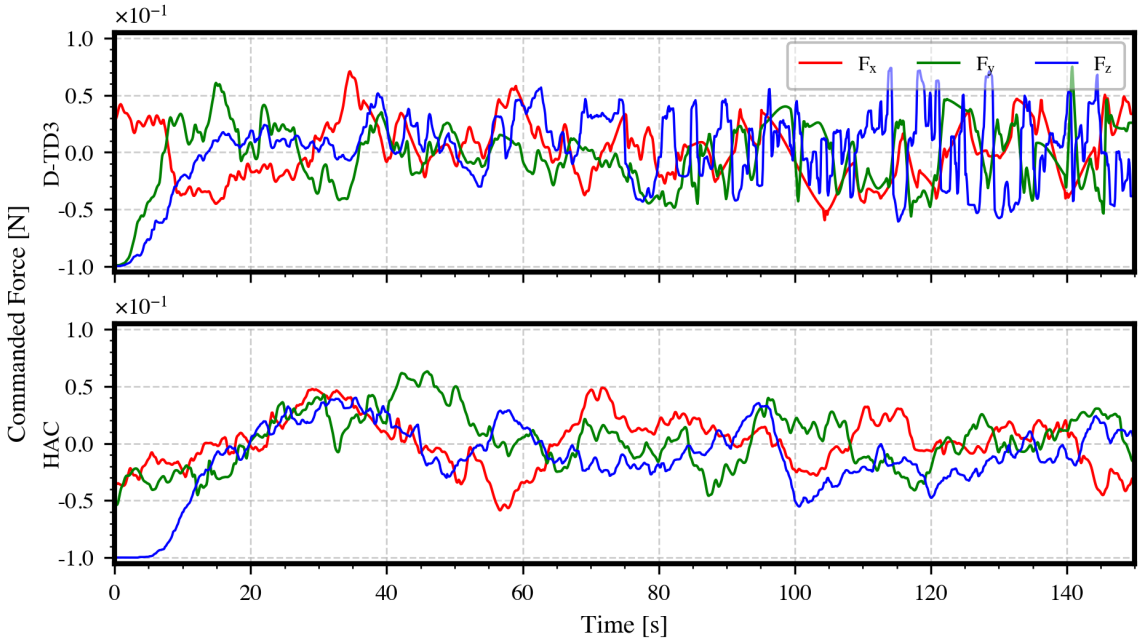


Figure 5.28: Translational control commands for the D-TD3 and HAC controllers during HIL testing.

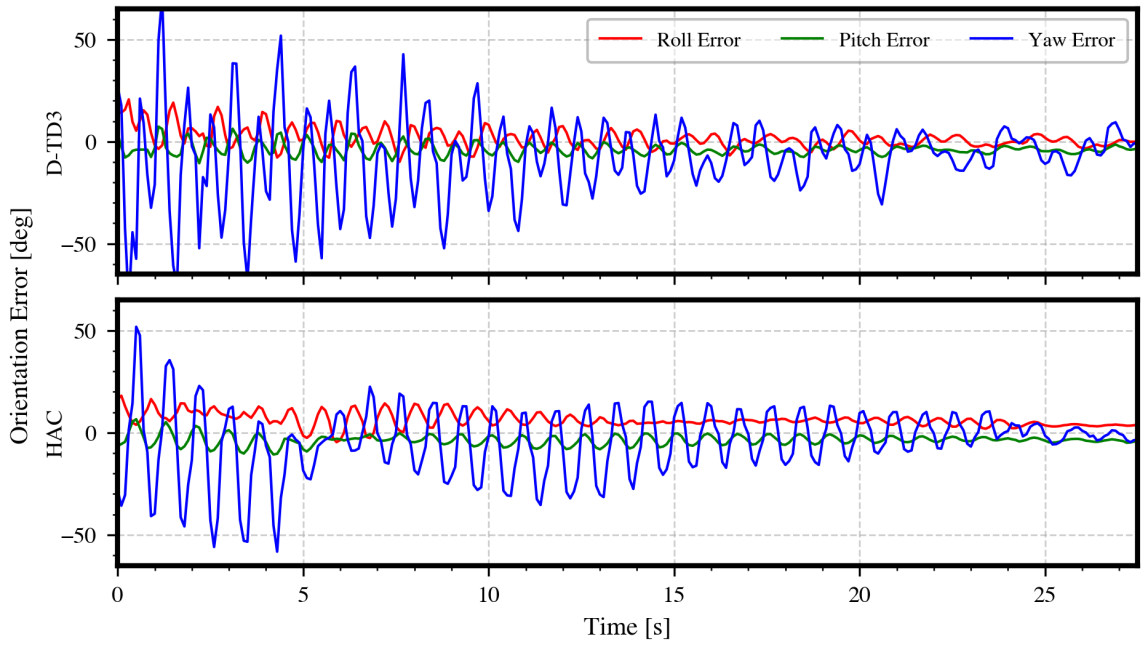


Figure 5.29: Orientation errors for the D-TD3 and HAC controllers during HIL testing.

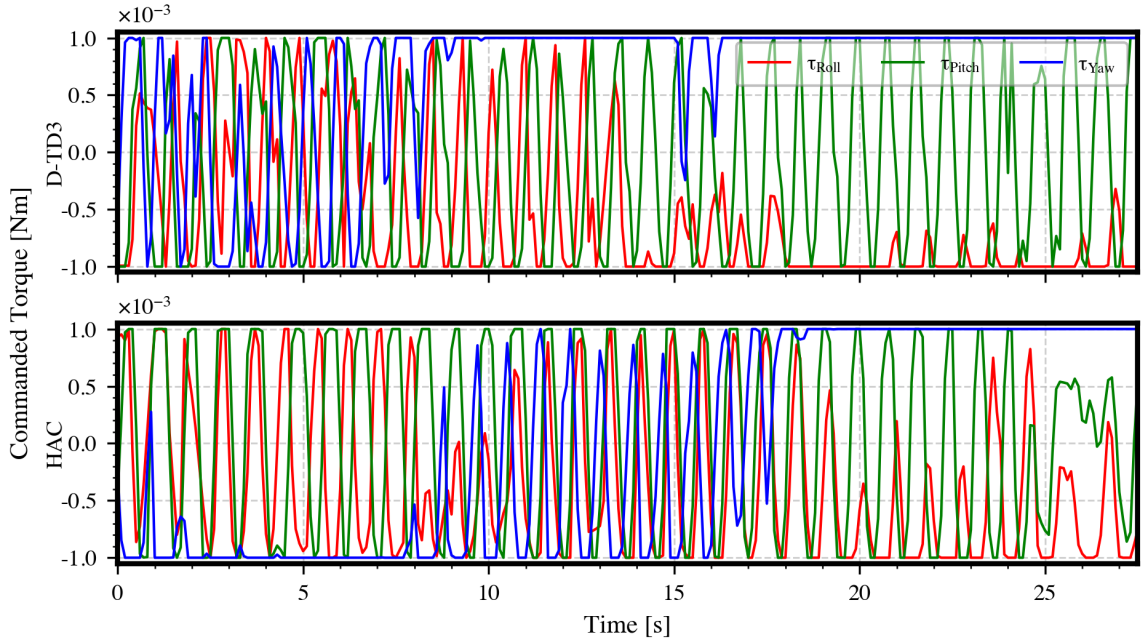


Figure 5.30: Rotational control commands for the D-TD3 and HAC controllers during HIL testing.

Figure 5.29 shows the attitude error profiles for the D-TD3 and HAC controllers, respectively. The D-TD3 agent achieved rotational convergence within approximately

25 seconds. The yaw error peaked at nearly 70°, gradually reducing to within 7.5° by the end of the trial. The pitch and roll errors were reduced to within 5° after an initial correction phase. However, significant early-phase oscillations were observed, particularly in the yaw axis where the most substantial disturbance was introduced. On the other hand, the HAC controller demonstrated a more gradual convergence profile. The yaw error decreased from an initial maximum of 60° to below 5° within 25 seconds. Roll and pitch errors, which began with magnitudes up to 20°, also stabilised to within 5°.

Figure 5.30 presents the corresponding control torque profiles. Both controllers produced saturated torque commands with rapid fluctuations across all axes during the initial response. This behaviour is attributed to the high angular velocities and attitude errors introduced at the start of each trial, compounded by noisy navigation data and actuation. While both agents operated near actuator limits , the HAC agent exhibited less persistent fluctuation as the angular errors were reduced, indicating improved control regulation despite the presence of sensor noise

Figure 5.27 depicts the translational convergence profiles. Both controllers successfully reduced the overall position error from approximately 110 m to below 5 m. The D-TD3 controller reached this threshold by 120 s, while the HAC agent achieved similar convergence slightly earlier, by approximately 90 s. The D-TD3 agent exhibited a more linear descent in position error but showed some overshoot during the final phase. The HAC controller initially exhibited smooth motion but displayed minor irregularities when nearing reaching the 5 m target, likely due to sensor noise affecting the navigation estimates.

Figure 5.28 shows the translational control efforts. Initially, both agents produced comparable force outputs. However, as the D-TD3 agent approached within 10 m of the target, it began exhibiting high-frequency control fluctuations across all axes. This was likely a result of the decreasing signal-to-noise ratio in the state estimate, which the policy could not fully compensate for. The HAC agent maintained more consistent and lower-frequency control behaviour throughout the episode, particularly during the final convergence phase, indicating greater robustness to sensor disturbances and improved control smoothness.

Both the D-TD3 and HAC controllers completed the 6-DoF rendezvous task under hardware-in-the-loop conditions, achieving a 5 m final positioning error and successful course attitude alignment within the allotted timeframe. The D-TD3 agent demonstrated faster convergence, particularly in the early stages of the task, but required

more aggressive control actions, especially in the attitude domain. In contrast, the HAC agent exhibited smoother control authority, less oscillation, and improved noise resilience in both attitude and position regulation.

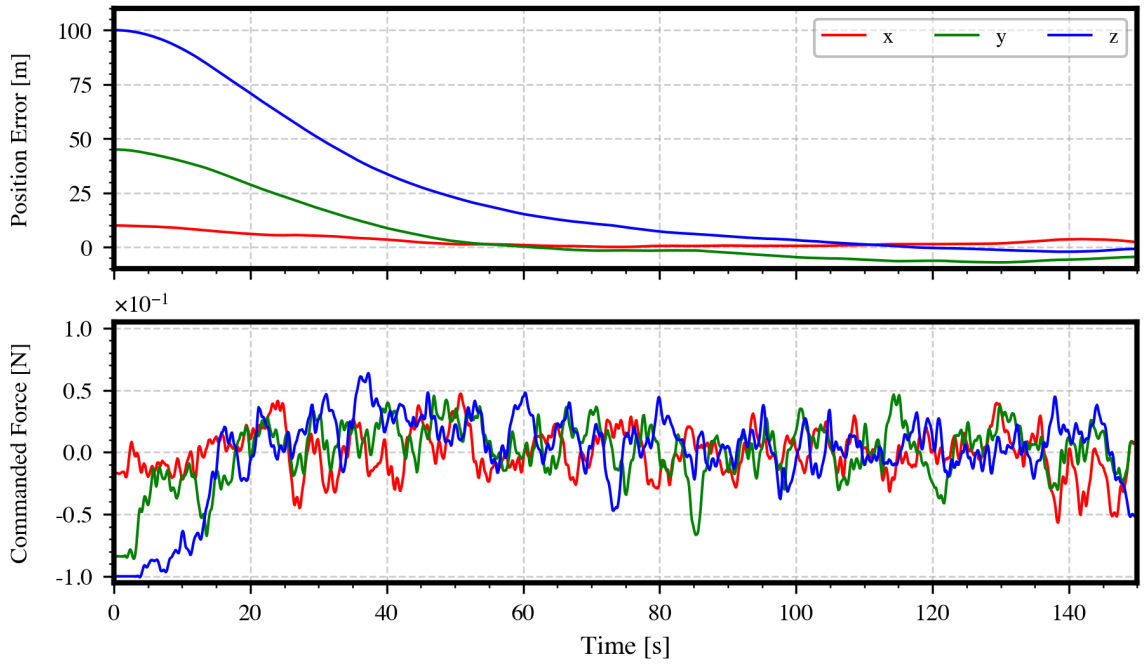


Figure 5.31: Position errors and Translational control commands for the PID controller during HIL testing.

The baseline PID controller, which had previously shown reliable performance in simulation, failed to achieve adequate performance under HIL conditions. As shown in Figure 5.31, the PID agent achieved translational convergence comparable to that of the learning-based controllers, reducing position error to within the mission threshold. However, it was unable to stabilise the CubeSat's orientation and exhibited persistent tumbling throughout the trial. The PID controller's poor performance in the attitude domain highlights a key limitation of classical approaches: their sensitivity to discrepancies between simulation and real-world conditions, particularly in the presence of unmodelled or time-varying dynamics and degraded sensor inputs.

These findings highlight the viability of learning-based controllers for satellite operating in uncertain, noisy environments. The successful real-time deployment of both DRL and HDRL agents on embedded hardware demonstrates that deep reinforcement learning can meet the practical demands of space guidance and control. The HAC controller, in particular, showed improved regulation under sensor noise and system

disturbances, reinforcing the value of hierarchical decomposition for complex, dynamic control tasks subject to actuation limits, and onboard processing constraints.

5.5 Summary

This chapter presented the development and evaluation of a hierarchical deep reinforcement learning (HDRL) framework for autonomous 6-DoF guidance and control of a CubeSat-class satellite during close-range proximity operations. Whereas Chapters 3 and 4 demonstrated the potential of flat DRL architectures under dense, manually shaped rewards, this work addressed the challenge of sparse-reward learning by introducing a hierarchical policy to decompose the control task into layered decision-making processes, improving temporal credit assignment and removing the need for handcrafted shaping signals.

A decentralised HDRL controller was implemented and evaluated in simulation alongside a classical PID controller under identical task conditions. Across 100 Monte Carlo trials, the hierarchical controller achieved mission-compliant terminal accuracy (final position error 2.76 ± 1.11 m and final orientation error 0.95 ± 0.42 °), comparable to the PID baseline (2.53 ± 0.89 m and 0.45 ± 0.18 °), while maintaining similar peak translational speeds (max linear velocity 1.11 ± 0.68 m s^{-1} versus 1.20 ± 0.60 m s^{-1}). Although convergence was slower and initial corrections more dynamic than those of flat DRL, the HDRL controller consistently maintained stability and demonstrated effective subgoal-driven behaviour.

To validate performance under realistic operational constraints, a hardware-in-the-loop testbed was designed and built. The platform combines a reaction-wheel-actuated CubeSat mock-up on a spherical air-bearing with a 6-DoF robotic arm for translational emulation. Both HDRL and D-TD3 controllers were successfully deployed on embedded hardware, maintaining closed-loop stability under real-time sensing, actuation, and communication conditions. In HIL testing, both controllers reduced position error from approximately 110 m to below 5 m, with HAC reaching this threshold by approximately 90 s compared with around 120 s for D-TD3, while both achieved attitude regulation within 5° by roughly 25 s. In contrast, the PID controller was unable to stabilise the vehicle, underscoring the robustness of the learning-based methods.

Overall, this work establishes hierarchical reinforcement learning as a practical and scalable control strategy for autonomous satellite operations in sparse-feedback and uncertain environments. By leveraging temporal abstraction and goal-directed

structure, HDRL offers a promising path toward scalable, deployable learning-based GNC architectures.

Chapter 6

Uncertainty-Aware Distributional Reinforcement Learning for Satellite Final-Approach and Rendezvous

6.1 Motivation

Reinforcement learning offers a powerful framework for learning satellite control policies through interaction with the environment, as demonstrated in Chapters 3-5. However, in real-world space missions, where dynamics are uncertain, sensors are noisy, and hardware is imperfect, ignoring uncertainty can lead to overconfident actions and unsafe behaviour.

This chapter focuses on improving the uncertainty awareness of DRL-based satellite controllers using distributional reinforcement learning. Unlike standard RL methods that estimate only the expected return of an action, distributional RL models the entire distribution of possible returns. This richer representation allows agents to capture both the expected value and the variability of outcomes, enabling risk-sensitive decision-making under uncertainty.

While distributional RL has shown success in terrestrial robotics and arcade benchmarks, it remains largely unexplored in the domain of satellite guidance and control. Most existing work in space applications uses standard value-based or actor-critic methods that offer no measure of return variability, limiting their robustness in uncer-

tain or degraded conditions. Additionally, although algorithms such as Quantile Regression DQN (QR-DQN) and Implicit Quantile Networks (IQN) provide efficient ways to learn return distributions, they do not, in their base form, incorporate uncertainty-aware action selection or modify training based on confidence estimates.

To address this gap, this chapter introduces a novel Uncertainty-Aware IQN (UA-IQN) controller for a 6-DoF autonomous final-approach rendezvous task. A UA-IQN agent learns the full return distribution and uses its variance to guide risk-aware action selection during both training and inference. The controller is evaluated in high-fidelity simulation and deployed on a real-time hardware-in-the-loop testbed to assess performance under realistic sensing and actuation constraints. Comparative studies are performed against a baseline D-TD3 controller to quantify the benefits of explicit uncertainty modeling.

This chapter makes the following contributions:

1. Introduces the first application of distributional reinforcement learning to full 6-DoF satellite guidance and control, enabling risk-sensitive decision-making through explicit modeling of return distributions.

2. Proposes the Uncertainty-Aware Implicit Quantile Network (UA-IQN) architecture, which integrates distributional variance into action selection during both training and inference to promote uncertainty-aware behavior under noisy and uncertain dynamics.

3. Establishes a structured evaluation combining high-fidelity simulation, Monte Carlo analysis, and real-time hardware-in-the-loop (HIL) testing, demonstrating that UA-IQN achieves consistent performance compared to a baseline D-TD3 controller.

By integrating distributional value estimation into the RL control pipeline, this chapter advances the design of satellite controllers that are not only performant, but also aware of their own uncertainty. This work extends the state of the art in DRL for satellite GNC and lays the foundation for more robust, risk-sensitive, and trustworthy autonomous control systems.

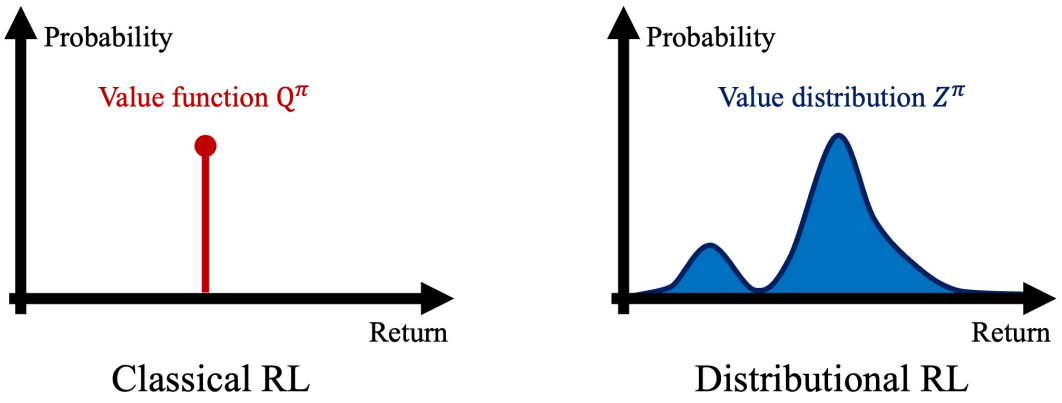


Figure 6.1: Diagram showing the difference between classical and distributional RL, Classical RL predicts a single expected return; distributional RL provides a full return distribution.

6.2 Literature Review

6.2.1 Foundations of Distributional RL

Conventional reinforcement learning algorithms focus on learning a value function $Q(s, a)$ that estimates the expected return (cumulative discounted reward) that an agent can expect for taking action a in state s. This expectation-centric formulation compresses the range of possible outcomes into a single scalar value. While computationally convenient, it inherently disregards the variability and shape of the return distribution. In environments where return distributions are broad, skewed, or multimodal, reliance on the mean can lead to misleading value estimates. Consequently, two actions with identical expected returns may be treated as equivalent, even if their associated risks differ substantially.

To address these limitations, distributional RL was proposed as an alternative framework that models the entire probability distribution of returns, rather than only the expected value. Formally, we can define a random return $Z^\pi(x, a)$ for policy π as the random variable representing the discounted sum of future rewards from state x and action a. Whereas traditional RL learns the expected value $Q(s, a) = \mathbb{E}[Z^\pi(s, a)]$, distributional RL aims to approximate the distribution of $Z^\pi(s, a)$ itself.

This framework is governed by the distributional Bellman equation, an analogue of the classical Bellman equation but in a distributional sense. For a policy π, the return distribution satisfies:

$$Z^\pi(s,a) \overset{D}{=} R(s,a) + \gamma Z^\pi(S', A') \tag{6.1}$$

where $R(s,a)$ is the immediate reward, $S' \sim P(\cdot \mid s,a)$ is the next state, $A' \sim \pi(\cdot \mid S')$ is the next action, $\gamma \in [0,1)$ is the discount factor, and $\overset{D}{=}$ denotes equality in distribution.

Despite learning the full return distribution, most distributional RL methods remain risk-neutral in their policy selection. That is, actions are still selected based on the maximization of the expected return $Q(s,a) = \mathbb{E}[Z(s,a)]$. However, access to the full distribution provides richer supervisory signals during training and facilitates several practical benefits, as described below.

- Stabilizing Learning: Learning a distribution rather than a point estimate introduces robustness against non-stationarity in target values. As observed in [8], explicitly modeling the distribution of returns enhances the stability of training by providing a more consistent and informative learning target. This improved signal encourages more reliable convergence of value estimates across states and actions.

- Improved Performance: Empirically, distributional RL algorithms have achieved state-of-the-art performance on challenging benchmarks. In particular, the introduction of distributional approaches led to significant improvements over DQN in environments such as the Atari 2600 suite, yielding better sample efficiency and higher performance [225, 7].

- Flexibility for Risk-Sensitive Policies: Although the present work considers the risk-neutral case, it's worth noting that access to the full return distribution opens the door to risk-sensitive decision-making. An agent could be adjusted to behave in a risk-averse or risk-seeking manner by using a different statistic of the distribution (e.g. a lower percentile of $Z(s,a)$ for risk-aversion) instead of the mean. In other words, distributional RL provides the agent with a rich understanding of outcomes that can be utilized according to the needs of the task.

6.2.2 Notable Algorithms in Distributional RL

The development of distributional reinforcement learning algorithms has progressed through a series of increasingly flexible models, each building on the limitations of

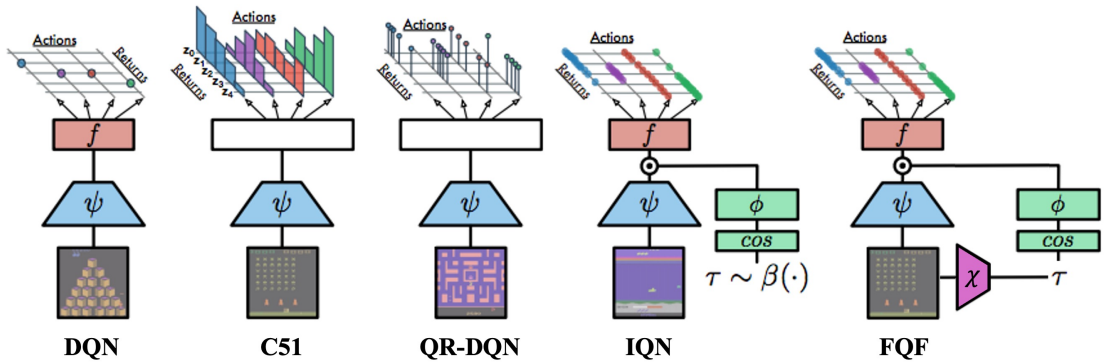


Figure 6.2: Architectural overview of DQN and its distributional variants, including C51, QR-DQN, IQN, and FQF. Adapted from [7].

its predecessors. Figure 6.2 illustrates this progression, from the fixed-support C51 algorithm to the fully parameterized FQF.

Fixed Support Approach: C51

The Categorical DQN (C51) algorithm represents the first successful application of distributional reinforcement learning [8]. Instead of outputting a single Q-value per state-action, the algorithm approximates the return distribution as a categorical distribution over a fixed set of support values, commonly referred to as "atoms". Specifically, the return distribution $Z_\theta(s, a)$ is modeled as a discrete distribution over $N = 51$ support atoms z_i, spaced uniformly over a predefined range $[V_{\min}, V_{\max}]$, with corresponding probabilities $p_i(s, a)$.

The network outputs a probability distribution across these 51 fixed return values for each action. During learning, for a given transition (s, a, r, s'), the target distribution is computed as the distribution of $r+\gamma Z_\theta(s', a^*)$, where $a^* = \arg\max_{a'} \mathbb{E}[Z_\theta(s', a')]$ is the greedy action in the next state. Since the resulting target values may not align with the fixed support, a projection step is employed to map the Bellman target back onto the predefined atom support. This is achieved by redistributing the probability mass to the nearest atomic support points.

Training then proceeds by minimizing the Kullback-Leibler (KL) divergence between the predicted distribution $Z_\theta(s, a)$ and the projected target distribution. The KL divergence is a measure of how one probability distribution diverges from a second, reference distribution. In this context, it quantifies how much the predicted distribution deviates from the projected Bellman target. For two discrete probability

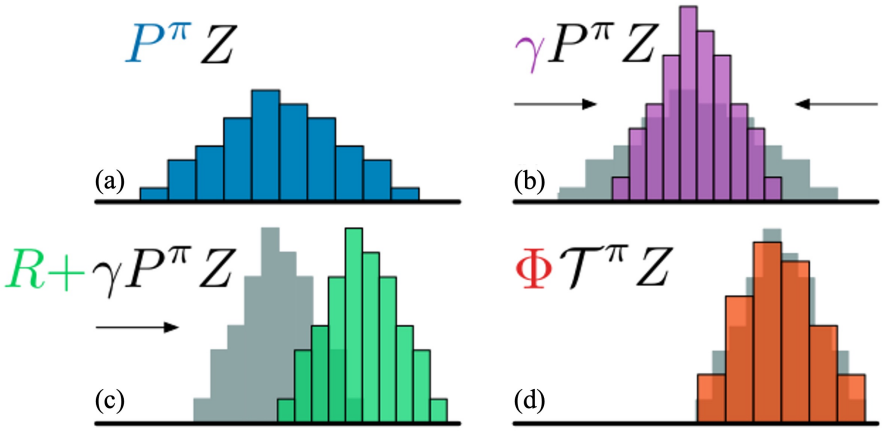


Figure 6.3: Illustration of the distributional Bellman operator. (a) The return distribution under policy π. (b) Scaling by the discount factor γ. (c) Shifting by the current reward. (d) Projection step introduced in [8].

distributions $P = p_i$ and $Q = q_i$ over the same support z_i, the KL divergence is given by:

$$D_{\mathrm{KL}}(Q||P) = \sum_{i=1}^{N} q_i \log\left(\frac{q_i}{p_i}\right) \tag{6.2}$$

Minimizing this quantity penalizes discrepancies between the predicted and target probabilities across the support atoms, encouraging the network to align its output distribution closely with the projected Bellman target. Despite the approximation introduced by the projection step, C51 achieved state-of-the-art results on the Atari 2600 benchmark at the time of its publication, significantly outperforming the baseline DQN [8].

However, the algorithm has notable limitations. The fixed support $[V_{\min}, V_{\max}]$ must be selected a priori, which requires domain knowledge or tuning. Furthermore, the projection step introduces bias, as the value distribution is forced to conform to a limited set of atoms regardless of the actual shape or support of the true return distribution. These constraints motivated the development of more flexible approaches.

Fixed Quantile Levels: QR-DQN

To address the rigidity of fixed-support models like C51, later approaches adopted quantile-based representations that removed the need for pre-defined support bounds

and projection steps.

Quantile Regression DQN (QR-DQN) reformulated the problem by employing a different approach to parameterizing the return distribution [225]. Instead of fixing return values and learning probabilities, QR-DQN fixes the cumulative probabilities (quantile levels) and learns the corresponding return values. This effectively models the quantile function (inverse cumulative distribution function) of returns.

The algorithm represents the return distribution $Z(s, a)$ as a uniform mixture of N Dirac delta distributions located at quantile values $\theta_i(s, a)_{i=1}^N$, where each quantile corresponds to a fixed cumulative probability $\tau_i = \frac{i}{N}$. For example, with $N = 10$, the network outputs ten return estimates per action, corresponding to the $10\%, 20\%, \ldots, 100\%$ quantiles of the value distribution.

QR-DQN improves upon C51 in several key ways:

1. No Fixed Support Bounds: The quantile values are not constrained to lie within a predefined interval. The network is free to assign any values to the quantile estimates, so QR-DQN can naturally handle return distributions that have wide or shifting support without needing domain knowledge of $V_{\min}$ or $V_{\max}$. This makes the method more flexible in approximating return distributions with unknown or variable support.

2. No Projection Step: The target distribution, constructed as $r + \gamma Z(s', a^*)$, can be directly compared to the current quantile estimates without requiring a projection onto fixed atoms, removing a significant source of approximation error and algorithmic complexity present in C51.

3. Wasserstein-Metric-Based Loss: QR-DQN optimizes the 1-Wasserstein distance W_1 between the predicted and target return distributions by applying quantile regression. The 1-Wasserstein distance between distributions P and Q, with respective quantile functions F_P^{-1} and F_Q^{-1}, is given by:

$$W_1(P, Q) = \int_0^1 \left| F_P^{-1}(\tau) - F_Q^{-1}(\tau) \right| d\tau. \tag{6.3}$$

   This metric, unlike KL divergence, provides a meaningful notion of distance between distributions even when their supports do not overlap, making it well-suited for distributional RL. In QR-DQN, this distance is approximated by minimizing a quantile Huber loss between predicted quantiles $\theta_i(s, a)$ and target

quantiles $r + \gamma\theta_j(s', a^*)$. The Huber loss smooths the absolute error, improving robustness to outliers and stabilizing training. This loss penalizes over- and underestimation asymmetrically based on the quantile level τ, encouraging each predicted quantile $\theta_i(s, a)$ to move toward the corresponding target value.

For a transition (s, a, r, s'), the network computes predicted quantiles $\theta_i(s, a)$ and target quantiles $r + \gamma\theta_j(s', a^*)$, where $j = 1, \ldots, N$. The quantile regression loss penalizes discrepancies between the predicted and target quantiles using an asymmetric weighting that depends on whether the prediction under- or overestimates the target. The overall loss is computed across all pairs (i, j), effectively minimizing the distributional discrepancy in quantile space.

Empirical results showed that QR-DQN achieved state-of-the-art performance across the Atari benchmark, outperforming both DQN and C51 in many games [225]. Beyond performance, QR-DQN formalized the modeling of return distributions through quantile regression, laying the foundation for later advances in distributional RL.

Continuous Quantile Modeling: IQN

While QR-DQN used a fixed set of quantile fractions for the distribution approximation, the next advance was to make this approximation even more flexible. Implicit Quantile Networks (IQN) generalized the quantile approach by learning the entire quantile function as a continuous mapping [7]. Instead of having a fixed set of quantile outputs, IQN's network can generate a quantile estimate for any quantile fraction input, essentially providing a continuous model of the return distribution.

The key innovation in IQN is that the network takes an additional input $\tau \in [0, 1]$ (a quantile fraction) and outputs $Z_\tau(s, a)$, an estimate of the return at the τ-quantile of the distribution for state s and action a. In other words, IQN parameterizes a function $f_\theta(s, a, \tau)$ such that for each τ, $f_\theta(s, a, \tau) \approx F^{-1}Z(s, a)(\tau)$, which represents the τ-quantile of the return distribution. By sampling different values of τ, one can obtain as many points on the distribution as desired.

During training, a set of quantile fractions τ_i is sampled from the uniform distribution $\mathcal{U}[0, 1]$. The network then predicts the corresponding quantiles $Z_{\tau_i}(s, a)$, which are compared to Bellman targets of the form $r + \gamma Z_{\tau_j}(s', a^*)$, where $a^* = \arg\max_{a'} \mathbb{E}[Z(s', a')]$. The comparison is performed using the quantile Huber loss, consistent with the regression strategy used in QR-DQN. By integrating over a diverse set of quantile samples, IQN provides a stochastic approximation of the 1-Wasserstein

distance between the predicted and target distributions.

This formulation introduces several important benefits. Since τ is treated as a continuous input, the model is not limited by a fixed grid resolution. The approximation accuracy is instead governed by network capacity and sample count, enabling finer distribution modeling without additional structural changes. In principle, an IQN with a sufficiently expressive network and enough sample budget can approximate the return distribution to arbitrary precision. The method is "implicit" because the distribution is represented implicitly through the function f_θ; one can draw as many samples from the implied distribution as needed by varying τ. This also means that during inference, one could sample many τ values to obtain a detailed picture of the distribution, or simply use the mean of a large number of quantile samples to estimate $Q(s, a)$ more accurately.

In terms of performance, IQN surpassed both C51 and QR-DQN on the Atari benchmark and approached the performance of the more complex Rainbow DQN agent, despite IQN being focused solely on distributional learning, it managed to set new high scores on several Atari games and substantially closed the remaining gap between distributional DQN variants and Rainbow [7, 226].

Furthermore, IQN was shown to be fairly robust to hyperparameters, suggesting that it does not require intensive tuning to generate a successful policy, which is a positive trait when deploying RL algorithms. This robustness, coupled with its architectural flexibility, positions IQN as a strong baseline for modern distributional reinforcement learning.

Adaptive Quantile Modeling: FQF

The Fully Parameterized Quantile Function (FQF) algorithm extends IQN by learning not only the quantile values but also the positions of the quantile fractions themselves [227]. The core idea is to adaptively allocate quantile fractions τ_i to regions of the value distribution that are most informative or variable, thereby improving the approximation of complex return distributions.

FQF consists of two jointly trained neural networks: a quantile value network $f_\theta(s, a, \tau_i)$ that predicts return estimates for each quantile, and a fraction proposal network that learns the optimal placement of the quantile fractions $\tau_1, \tau_2, \ldots, \tau_{N-1} \in (0, 1)$, where $\tau_0 = 0$ and $\tau_N = 1$ are fixed. The quantile value network is trained similarly to IQN using the quantile Huber loss. In parallel, the fraction proposal network is updated via the gradient of the 1-Wasserstein loss with respect to the

quantile positions. This joint optimization allows the model to place more quantile fractions in regions with high curvature or multimodal structure.

By dynamically learning the quantile positions, FQF can achieve a more expressive distributional approximation with the same number of quantile outputs N as IQN or QR-DQN. This results in improved sample efficiency and final performance, as demonstrated by state-of-the-art results on the Atari benchmark at the time of its introduction [227].

However, these performance gains come with certain trade-offs. FQF introduces increased architectural complexity by employing two separate networks increasing the number of parameters and introducing additional hyperparameters. This added complexity raises the tuning burden and can lead to stability issues during training. Furthermore, although FQF adaptively adjusts the placement of quantile fractions, it still represents the return distribution using a finite set of N Dirac points, making it a parametric approximation with limited support. In contrast, IQN offers a more flexible approximation by enabling continuous sampling over the quantile space. While FQF may provide a more accurate distributional estimate for a fixed N, it does not fundamentally extend the class of representable distributions beyond what IQN can approximate given sufficient capacity and sampling.

In summary, FQF advances distributional RL by improving the precision of quantile approximation through adaptive fraction placement. It generally delivers stronger performance than prior methods, but at the cost of additional architectural and computational complexity. This trade-off is a central consideration when selecting an algorithm for deployment in time-sensitive or resource-constrained settings.

The progression of distributional reinforcement learning has demonstrated clear advantages over traditional expectation-based methods by modeling the full return distribution rather than a scalar mean. In this work, we adopt IQN as the foundation for our approach. While FQF offers marginal gains in distributional accuracy, IQN provides a favorable balance between model expressiveness, computational efficiency, and ease of integration. Its implicit quantile formulation is well-suited to real-time decision-making tasks, where algorithmic simplicity and training stability are key. IQN's strong empirical performance, minimal hyperparameter sensitivity, and streamlined design make it a pragmatic and effective choice for developing robust distributional reinforcement learning agents under operational constraints.

6.2.3 Uncertainty in Distributional RL

Uncertainty estimation plays a central role in deploying reinforcement learning agents in high-stakes or safety-critical environments such as autonomous satellite docking. In such regimes, it is not sufficient for an agent to merely maximize expected return; it must also reason about the confidence or reliability of its decisions under partial observability, model mismatch, and dynamic perturbations. This need has spurred increasing interest in uncertainty-aware deep reinforcement learning (UADRL), where agents explicitly quantify and act upon different sources of uncertainty.

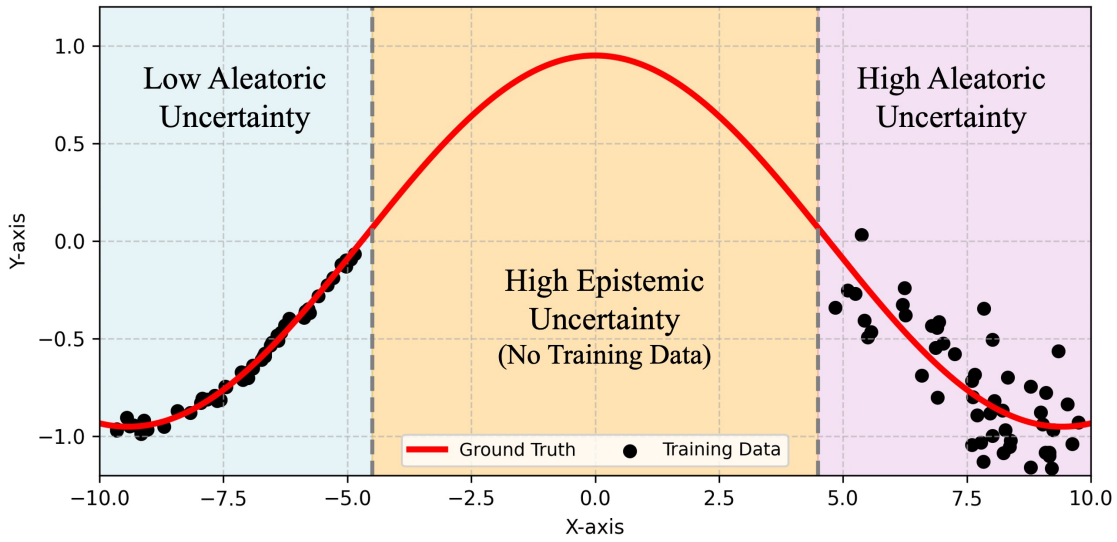


Figure 6.4: Illustration showing epistemic (model) uncertainty and aleatoric (inherent) uncertainty.

Broadly, uncertainty in DRL can be categorized into two types: aleatoric and epistemic.

- Aleatoric uncertainty arises from inherent stochasticity in the environment, such as sensor noise, actuator variability, or unmodeled dynamics. It is irreducible, and persists no matter the amount of training data.
- Epistemic uncertainty, by contrast, stems from the agent's lack of knowledge, such as limited training data or unfamiliar state-action pairs. It is reducible with further experience, and typically higher in regions of the state space that the agent has not explored sufficiently.

Traditional value-based RL algorithms like DQN or TD3 collapse all return variability into a single expected value, discarding potentially informative structure in the

return distribution. In contrast, distributional reinforcement learning retains a richer representation by learning the full distribution over possible returns for each state-action pair. This makes distributional RL a natural fit for studying uncertainty: the spread or shape of the learned return distribution implicitly reflects both epistemic and aleatoric uncertainties affecting the agent's decision-making process.

A growing body of research has leveraged the distributional perspective in RL to quantify or exploit uncertainty in various ways. Some methods attempt to disentangle epistemic and aleatoric components within the return distribution, using auxiliary models or ensemble-based techniques to infer uncertainty type and source [228, 229, 230, 231, 232]. However, methods aimed at disentangling these uncertainties have been criticized for their limited reliability, as they often fail to produce consistent or interpretable estimates across different tasks and environments. Another line of work uses summary statistics of the return distribution, such as the interquantile range or entropy, as heuristics for identifying out-of-distribution states or areas of high uncertainty with varying degrees of success [233, 234].

Beyond pure uncertainty estimation, a number of approaches have focused on using uncertainty during action selection. Some introduce risk-sensitive objectives during training, shaping the policy to be risk-averse or risk-seeking based on domain-specific reward structures or risk constraints [229, 235, 236]. Others adapt the action selection strategy during inference, often favoring actions with lower return variance or applying penalties to high-uncertainty choices in an effort to produce more stable behavior [237, 238, 239]. These methods, while effective in some cases, can add significant implementation and tuning complexity, limiting their ease of deployment in real-world systems.

While these directions are promising, the use of distributional uncertainty for real-time risk-aware decision-making remains an open and active area of research. This work extends the Implicit Quantile Network framework to develop a practical, uncertainty-aware control architecture (UA-IQN) tailored for real-time satellite docking. The proposed method uses the variance of the learned return distribution as a proxy for the combined aleatoric and epistemic uncertainty, integrating this information into the action selection process. By penalizing high-variance actions, the controller favors more stable, predictable behaviors, particularly in regions of the state space where uncertainty and risk are elevated. This enables the controller to exhibit uncertainty-aware behavior during inference, improving reliability without added architectural or training complexity.

6.3 Methodology

6.3.1 Environment Setup

The environment used in this work simulates the final approach phase of a satellite docking mission. Specifically, it models a constrained v-bar approach, a common choice for autonomous docking operations where the chaser satellite approaches the target along the relative velocity vector. This axis points along the target's orbital motion, and a v-bar approach enables a passive abort trajectory that naturally increases separation, enhancing safety.

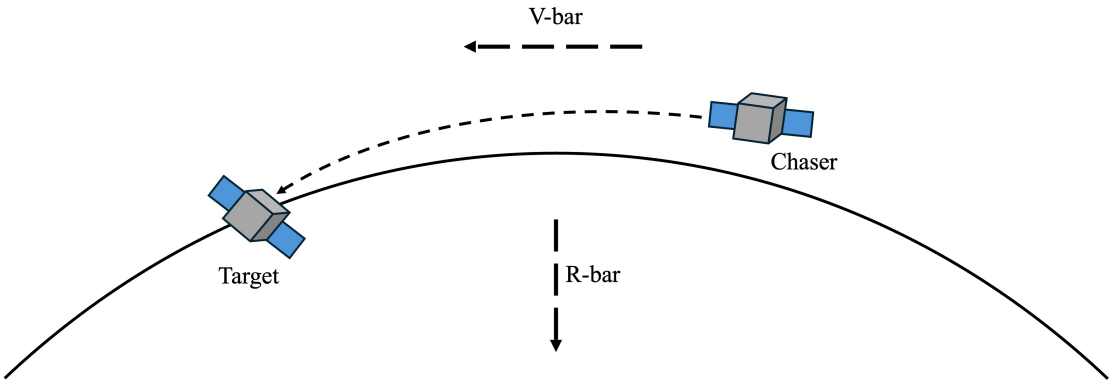


Figure 6.5: Illustration of a short-range satellite rendezvous mission employing a V-bar approach, where the chaser vehicle approaches the target along the velocity vector direction in the target's local orbital frame.

The chaser satellite is modelled after a small refuelling satellite with a mass of 250 kg and a compact structure of approximately 1m × 1m × 1m. It is treated as a rigid body equipped with translational and rotational actuators with thrust limited to 1.0N per axis and torque capped at 1.0N m in each rotational axis. These limits are representative of small satellite-class propulsion systems and reflect realistic constraints under which the controller must operate. Although the chaser is modeled as a small satellite rather than a CubeSat, the controller formulation is spacecraft-agnostic and can be adapted to other platforms by updating the corresponding mass/inertia and actuator limits.

The docking sequence is formulated as a 6-DoF control problem with full translational and rotational dynamics. The reference frame used is centered on the target, with the $+z$ axis corresponding to the v-bar (approach) direction, $+x$ pointing radially outward from the Earth (off-track), and $+y$ completing the right-handed frame

(cross-track). The chaser must maintain strict positional and angular constraints with respect to this frame as it approaches.

The chaser starts at rest, positioned up to 5m away along the v-bar (z-axis), with lateral offsets limited to 10% of that distance, forming a cone-shaped region around the z-axis. Initial orientation is randomized up to $\pm 10^\circ$ in roll, pitch, and yaw, under the assumption that prior control phases have reduced pointing errors to within this range. The use of a dynamic initialization region helps ensure robustness and promotes policy generalization across a diverse set of start states.

Each training episode ends under one of three conditions:

- **Successful docking:** v-bar separation < 0.1m, off-track and cross-track error < 0.01m, and attitude error $< 1^\circ$ in all axes.
- **Failure:** exceeding angular error of 15° or separation beyond 10m.
- **Timeout:** exceeding the maximum allowed episode length of 600s.

Upon successful docking, it is assumed that control transitions to a low-level servoing system for final mechanical capture and hard dock.

To enhance realism and prevent overfitting to deterministic dynamics, stochastic perturbations are applied to both sensing and actuation subsystems. These are modeled as zero-mean Gaussian noise processes with standard deviations, summarized in Table 6.1. Rather than attempting to precisely model each source of disturbance (e.g., thruster misalignment, sensor delay, or thermal drift), this approach injects time-varying uncertainty that collectively simulates the imperfect conditions under which real systems operate. This noise is regenerated at each timestep to simulate time-varying disturbances and avoid introducing systematic bias. The magnitudes are conservative but non-negligible, ensuring that trained policies cannot overfit on deterministic system responses.

Table 6.1: Noise parameters used during testing in nominal conditions

Signal	**Standard Deviation**
Position	0.01 m
Velocity	0.001 m/s
Angular Error	0.1°
Angular Velocity	0.05 rad/s
Force Command	0.1 N
Torque Command	0.01 Nm

Dynamics are simulated using a fixed-step numerical integrator with a timestep of 50 ms (20Hz). Translational motion is modeled using the linearized Clohessy-Wiltshire equations for relative orbital motion, while rotational motion is governed by the Euler rotational dynamics equations for a rigid body.

Contact dynamics and physical docking mechanisms are outside the scope of this work and are not modeled explicitly. Instead, once the chaser reaches its goal separation of 10 cm while remaining within both translational and angular tolerance bounds, the episode is considered successfully complete. This abstraction is appropriate given the focus on guidance and approach control rather than capture mechanisms or final latching dynamics.

6.3.2 Agent Architecture

The agent architecture developed in this work, termed UA-IQN (Uncertainty-Aware Implicit Quantile Network), extends the IQN framework by explicitly incorporating the variance of the return distribution into the decision-making process. By integrating a variance-penalized criterion into the action selection mechanism, the proposed approach enhances the agent's sensitivity to return uncertainty and enables the policy to systematically prefer actions with more reliable outcomes, thereby improving robustness in high-stakes or safety-critical settings such as constrained satellite docking.

Consistent with the control decomposition described in Chapter 3, the system is implemented as a modular architecture composed of two independent agents: one responsible for translational control and the other for rotational control. Each agent operates within its own observation and action space and is trained separately using an identical structure.

The network architecture comprises three primary components: a state encoder, a quantile embedding module, and a quantile-conditioned value estimator. Together, these modules enable the network to produce quantile-specific value estimates conditioned on both the environment state and the sampled quantile index.

The state vector $s \in \mathbb{R}^n$ is first processed by a fully connected encoder $\psi : \mathbb{R}^n \to \mathbb{R}^d$, which produces a latent representation $\psi(s) \in \mathbb{R}^d$. This embedding captures the task-relevant features of the environment and is shared across all quantile computations for a given state.

To incorporate the quantile index, each sampled τ is projected into a d-dimensional embedding via a cosine basis followed by a linear transformation. This is defined as:

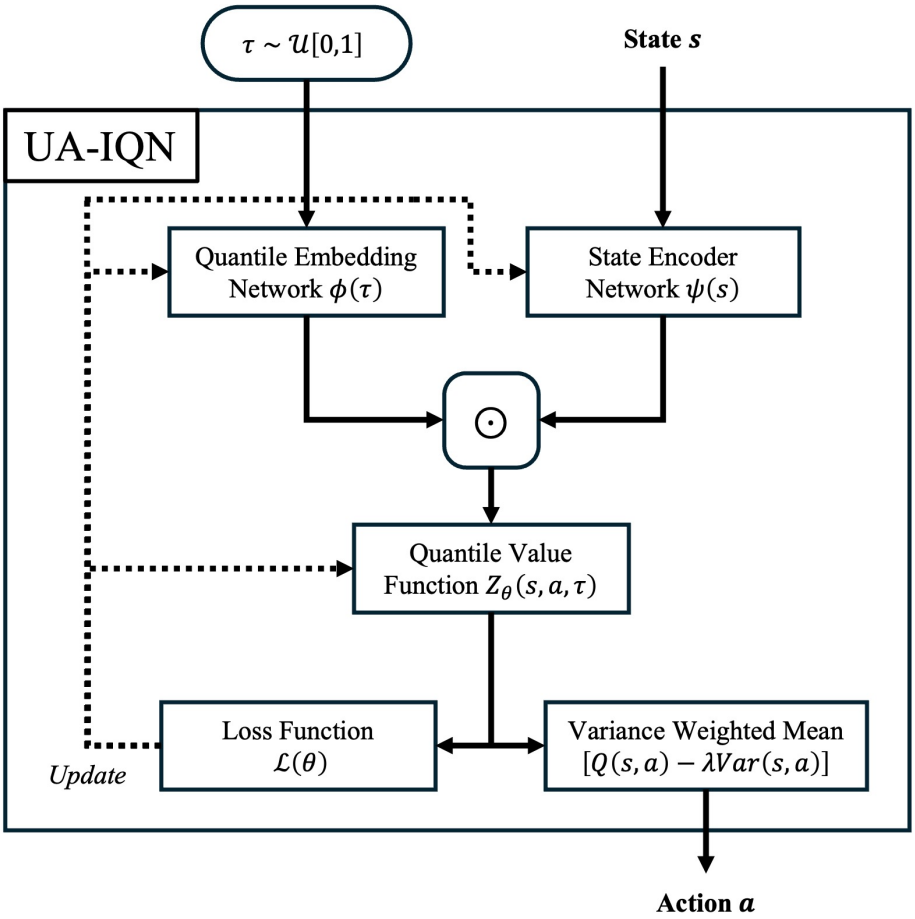


Figure 6.6: Illustration of the UA-IQN architecture, depicting the state encoder, quantile embedding, and quantile-value networks. Dashed lines represent gradient update signals applied during training.

$$\phi(\tau) = \mathrm{ReLU}\left(W_\phi \cos\left(\pi\tau\mathbf{f}\right) + b_\phi\right), \quad \mathbf{f} = [1, 2, \ldots, N_{\cos}]^\top, \tag{6.4}$$

where $W_\phi \in \mathbb{R}^{d\times N_{\cos}}$ and $b_\phi \in \mathbb{R}^d$ are learnable parameters, and $N_{\cos}$ is the number of cosine features. This embedding allows the network to flexibly condition its predictions on arbitrary quantile indices within $[0, 1]$.

The state and quantile embeddings are then combined via an element-wise (Hadamard) product:

$$h(s, \tau) = \psi(s) \odot \phi(\tau), \tag{6.5}$$

which captures the interaction between the environment state and the quantile conditioning. This intermediate representation is passed through a fully connected

output layer $f : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{A}|}$ to produce the quantile-specific action-value estimates:

$$Z_\theta(s, a, \tau) = f(h(s, \tau))_a. \tag{6.6}$$

This formulation enables the network to represent the quantile function $F^{-1}_{Z(s,a)}(\tau)$ for any given quantile index τ, thereby approximating the full return distribution in continuous quantile space.

The agent samples a set of quantile fractions $\{\tau_i\}_{i=1}^N$ from the uniform distribution $\mathcal{U}[0, 1]$. For each action a, the corresponding quantile values $Z * \theta(s, a, \tau_i)$ are used to compute both the expected return and its empirical variance:

$$Q(s, a) \approx \frac{1}{N} \sum_{i=1}^{N} Z_\theta(s, a, \tau_i) \tag{6.7}$$

$$\mathrm{Var}(s, a) \approx \frac{1}{N} \sum_{i=1}^{N} \left(Z_\theta(s, a, \tau_i) - Q(s, a)\right)^2. \tag{6.8}$$

To enforce uncertainty awareness action selection is performed using a variance-aware criterion:

$$a^* = \arg\max_{a \in \mathcal{A}} \left[Q(s, a) - \lambda \cdot \mathrm{Var}(s, a)\right], \tag{6.9}$$

where $\lambda \geq 0$ is a variance sensitivity parameter that controls the influence of variance on the selection policy. This formulation encourages the agent to prefer actions with high expected return while discouraging those associated with excessive return variability.

Given a transition tuple (s, a, r, s'), the target quantile values are constructed using the target network θ':

$$T(\tau'_j) = r + \gamma\, Z_{\theta'}(s', a^*, \tau'_j), \tag{6.10}$$

where $\{\tau'_j\}_{j=1}^{N'}$ are samples from $\mathcal{U}[0, 1]$, and $a^* = \arg\max_{a'} Q(s', a')$ denotes the greedy action in the next state. For each pair of predicted and target quantiles (τ_i, τ'_j), the temporal difference error is defined as:

$$\delta_{ij} = T(\tau'_j) - Z_\theta(s, a, \tau_i). \tag{6.11}$$

The loss function employed is the quantile Huber loss, which combines the robust-

ness of the Huber loss with the asymmetric weighting of quantile regression:

$$\rho_{\tau_i}^{\kappa}(\delta_{ij}) = \begin{cases} |\tau_i - \mathbb{I}_{\{\delta_{ij}<0\}}| \, \frac{1}{2}\delta_{ij}^2, & \text{if } |\delta_{ij}| \leq \kappa, \\ |\tau_i - \mathbb{I}_{\{\delta_{ij}<0\}}| \, \kappa \left(|\delta_{ij}| - \frac{1}{2}\kappa\right), & \text{otherwise.} \end{cases} \tag{6.12}$$

The overall loss is computed as the average over all combinations of predicted and target quantile samples:

$$\mathcal{L}(\theta) = \frac{1}{NN'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_{\tau_i}^{\kappa}(\delta_{ij}), \tag{6.13}$$

which provides a stochastic estimate of the 1-Wasserstein distance between the predicted and target distributions.

At inference time, a fixed number of quantile indices $\{\tau_i\}_{i=1}^{K}$ is sampled, and the mean of the corresponding value estimates is used to select the greedy risk-neutral action:

$$a^* = \arg\max_{a \in \mathcal{A}} \frac{1}{K} \sum_{i=1}^{K} Z_\theta(s, a, \tau_i). \tag{6.14}$$

The full UA-IQN training and inference procedure is summarized in in the appendix, capturing the variance-aware extension to the standard IQN framework and its integration into both value estimation and action selection.

Implementation Notes

The UA-IQN architecture was developed using PyTorch and adapted to the specific requirements of the constraint-aware orbital guidance and control problem considered in this work. A modular implementation was adopted, with separate networks for translational and rotational control. Each network was designed to support real-time inference during hardware-in-the-loop testing, which informed the architectural trade-offs between representational capacity and computational efficiency.

Architectural decisions were made through iterative experimentation. A hidden layer size of 512 was selected to balance model capacity and real-time inference constraints. The number of quantile outputs ($K = 32$) and training samples per update ($N = 64$) were tuned to provide sufficient resolution over the return distribution without incurring excessive computational overhead. Similarly, the quantile embedding dimension ($n_{\cos} = 64$) was found to provide stable and expressive conditioning of

quantile indices. Larger embeddings and higher quantile sample counts were evaluated but led to minimal performance improvements while increasing memory and training time.

Table 6.2 summarizes the key architecture and training hyperparameters used in the final implementation.

Table 6.2: IQN architecture and training parameters

Parameter	**Value**
Hidden Layer Width	512
Quantile Outputs (K)	32
Quantile Samples per Update (N)	64
Cosine Embedding Size ($N_{\cos}$)	64
Variance Sensitivity Parameter λ	0.75
Batch Size	256
Replay Buffer Size	500,000
Learning Rate	1×10^{-4}
Discount Factor (γ)	0.99
Target Update Rate (τ)	0.005
Exploration Decay	250,000 steps (pos.), 150,000 steps (att.)
Training Steps	500,000 steps (pos.), 250,000 steps (att.)
Max Episode Time	600 s (position), 150 s (attitude)
Control Timestep	0.5 s

Both agents operated over discrete action spaces covering three degrees of freedom in translation and rotation, respectively. For each axis, control inputs were discretized into five levels: $\{-1, -0.1, 0, 0.1, 1\}$, resulting in $5^3 = 125$ unique actions per agent. This discretization simplified the action selection problem while preserving sufficient control authority and resolution for precise docking manoeuvres. Exploration during training was governed by an ϵ-greedy strategy, with ϵ linearly annealed from 1.0 to 0.01 over 250,000 environment steps for the translation agent and 150,000 steps for the attitude agent. Each agent was trained for a total of 500,000 and 250,000 environment steps, respectively, with periodic policy evaluation conducted every 25 training episodes.

6.3.3 Training Regime

The agent training process was conducted using a simplified simulation environment with idealised sensor and actuator models. This abstraction allowed for faster conver-

gence and isolated the learning process from external disturbances such as actuator and sensor non-linearities, which were later reintroduced during hardware-in-the-loop validation.

Training was guided by a dense, shaped reward function designed to promote gradual, stable convergence toward the target pose. A modular reward structure was adopted to separately incentivise translational and rotational control performance, aligned with the modular agent design.

Translational control was incentivised using a reward composed of three components:

$$r_{\mathrm{a}} = 10 \cdot (\|\mathbf{p}_{t-1}\| - \|\mathbf{p}_t\|) \tag{6.15}$$

$$r_{\mathrm{b}} = -\frac{\|\mathbf{v}_t\|}{\|\mathbf{p}_t\| + \epsilon}, \quad \epsilon = 10^{-4} \tag{6.16}$$

$$r_{\mathrm{c}} = -0.025 \cdot \|\mathbf{f}_t\| \tag{6.17}$$

Here:

- $r_{\mathrm{p,a}}$ rewards progress toward the target by computing the reduction in position magnitude relative to the previous timestep.
- $r_{\mathrm{p,b}}$ penalises high velocities, especially as the chaser nears the docking interface where overshoot is most dangerous.
- $r_{\mathrm{p,c}}$ discourages excessive actuation.

The total translation reward is given by:

$$r_{\mathrm{pos}} = r_{\mathrm{p,a}} + r_{\mathrm{p,b}} + r_{\mathrm{p,c}} \tag{6.18}$$

Attitude control was shaped using a parallel structure:

$$r_{\mathrm{a,a}} = -\|\boldsymbol{\theta}_t\| \tag{6.19}$$

$$w_\omega = 0.1 + \frac{2.5}{1 + \exp(\|\boldsymbol{\theta}_t\| - 1)} \tag{6.20}$$

$$r_{\mathrm{a,b}} = -w_\omega \cdot \|\boldsymbol{\omega}_t\| \tag{6.21}$$

$$r_{\mathrm{a,c}} = -0.15 \cdot \|\boldsymbol{\tau}_t\| \tag{6.22}$$

This yields the total attitude reward:

$$r_{\text{att}} = r_{\text{a,a}} + r_{\text{a,b}} + r_{\text{a,c}} \tag{6.23}$$

Each term in this structure was designed to encourage rapid gross alignment followed by finer adjustments:

- $r_{\text{a,a}}$ penalises angular error directly, providing a clear objective.
- $r_{\text{a,b}}$ penalises angular velocity in proportion to current misalignment. The weight w_ω is a sigmoid-shaped function that allows faster rotation when far from target but increasingly punishes high rates as alignment improves.
- $r_{\text{a,c}}$ discourages excessive actuation.

This formulation ensures that agents are rewarded not only for reaching the terminal docking state but also for the quality of the approach itself. Progress toward the target is continuously incentivised, while excessive velocity or control effort is penalised. The inclusion of dynamic scaling terms such as w_ω introduces state-dependent shaping that adapts to the agent's current alignment, enabling rapid convergence early in the trajectory and finer, more controlled adjustments near the docking interface. By providing dense, context-aware feedback throughout the approach, the reward scheme supports sample-efficient learning and yields policies that generalise across varying initial conditions.

6.4 Monte Carlo Evaluation

To evaluate the effectiveness of the UA-IQN controller developed in this chapter, a Monte Carlo evaluation was performed across 100 randomized docking scenarios.

Seven metrics were used to characterize final approach performance: off-track and cross-track errors at the docking threshold, Euler angle errors at docking, and the time taken to meet the translational and rotational terminal thresholds (0.1m position error and 1° angular error, respectively). These metrics are summarized in Table 6.3.

Table 6.3: Monte Carlo performance metrics for D-TD3 and UA-IQN controllers across 100 randomized docking trials.

Metric	**D-TD3**			**UA-IQN**		
	Min	Mean	Max	Min	Mean	Max
Off-Track Error at Docking (m)	0.000	0.005	0.011	0.000	0.004	0.017
Cross-Track Error at Docking (m)	0.000	0.004	0.012	0.000	0.006	0.017
Roll at Docking (deg)	0.136	0.400	0.607	0.022	0.263	0.666
Pitch at Docking (deg)	0.001	0.142	0.354	0.001	0.175	0.544
Yaw at Docking (deg)	0.015	0.297	0.549	0.003	0.277	0.761
Time to Position Threshold (s)	158	228	319	221	379	521
Time to Angular Threshold (s)	32.0	76.0	119	6.00	12.4	22.0

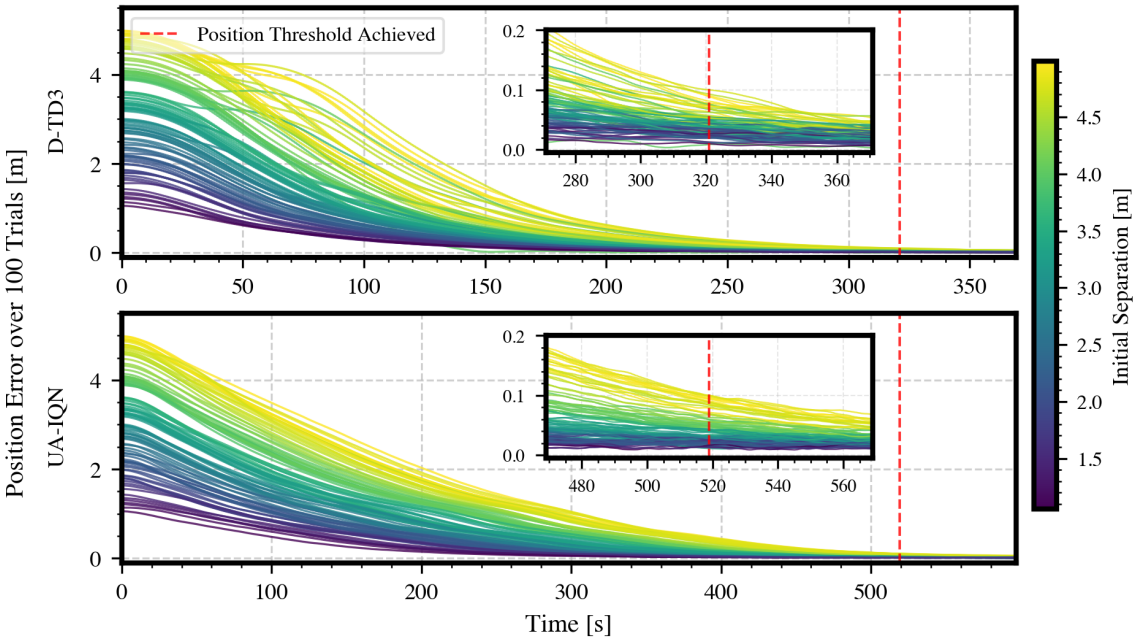


Figure 6.7: Position errors across 100 randomized trials for the D-TD3 and UA-IQN controllers. The red dashed line marks when all trials fall below the linear docking threshold (0.1m).

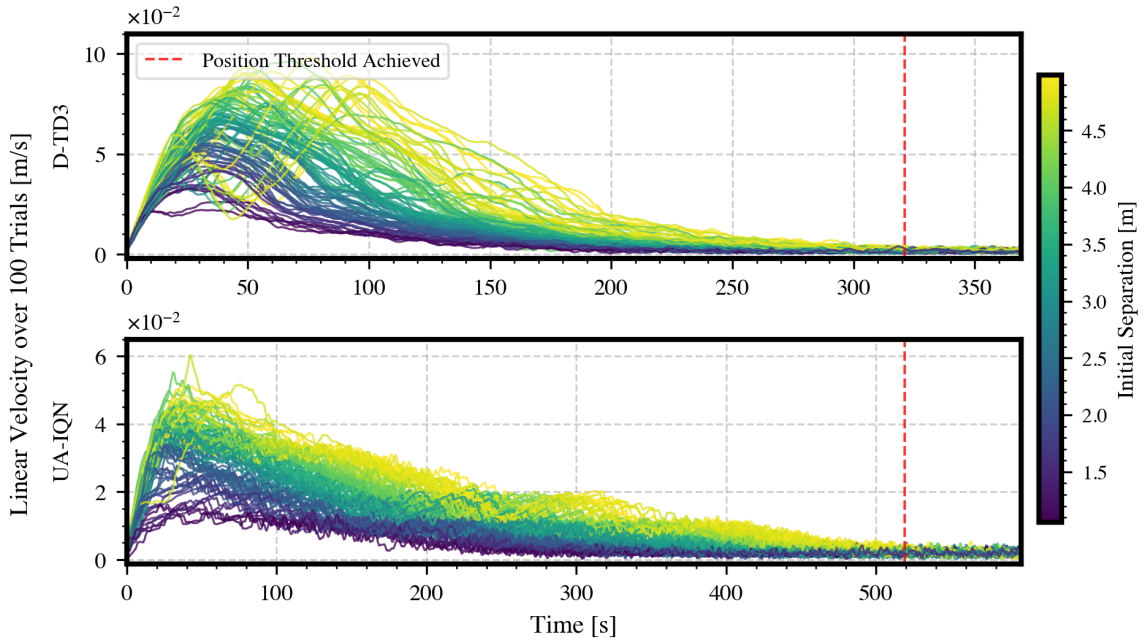


Figure 6.8: Linear velocities across 100 randomized trials for the D-TD3 and UA-IQN controllers. The red dashed line marks when all trials fall below the linear docking threshold (0.1m).

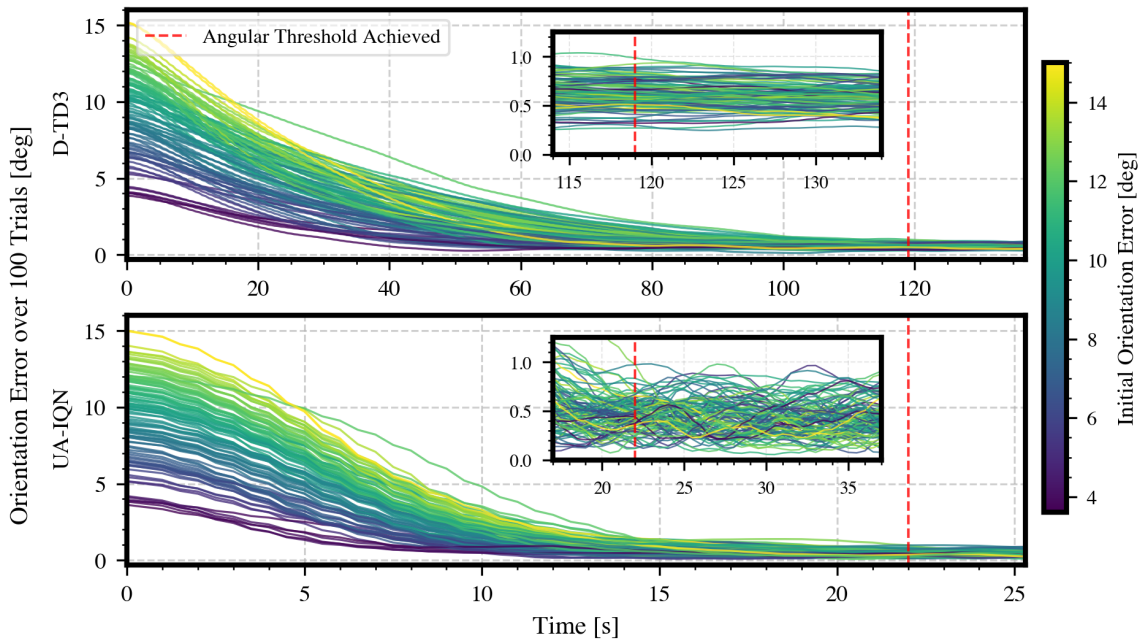


Figure 6.9: Orientation errors across 100 randomized trials for the D-TD3 and UA-IQN controllers. The red dashed line marks when all trials fall below the angular docking threshold (1°).

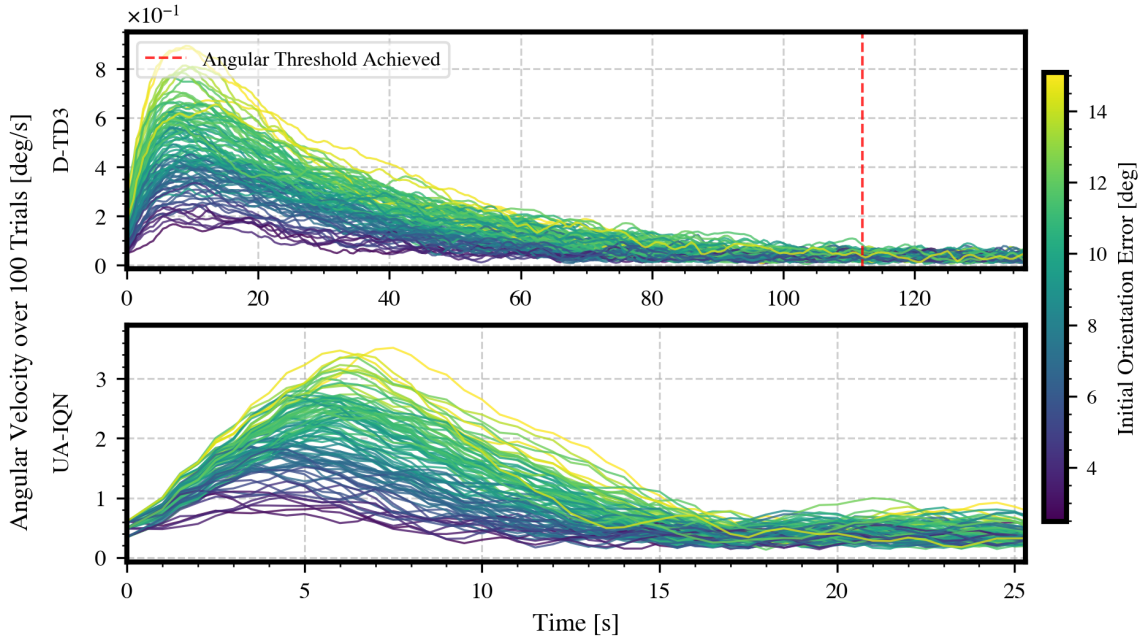


Figure 6.10: Angular velocities across 100 randomized trials for the D-TD3 and UA-IQN controllers. The red dashed line marks when all trials fall below the angular docking threshold (1°).

Figures 6.7-6.10 present the full trajectory profiles for position error, linear velocity, orientation error, and angular velocity. Each line corresponds to one of the 100 Monte Carlo trials, color-coded by initial displacement or angular offset. Red vertical dashed lines indicate the time at which all episodes have crossed the required threshold.

Table 6.3 shows that both controllers achieved successful docking across all trials, with final position and orientation errors consistently within the defined bounds. However, their approaches to achieving this result differ in ways that reveal meaningful trade-offs in strategy and behavior.

Both controllers maintain sub-centimeter alignment accuracy at the docking threshold. The UA-IQN controller records a slightly lower mean off-track error (0.43cm vs. 0.45cm), but a higher mean and maximum cross-track error (0.59cm and 1.69cm vs. 0.43cm and 1.20cm), indicating somewhat greater lateral spread in final positioning. As seen in Figure 6.7, the distributional controller's approach profiles are smoother and more consistent, with nearly all episodes exhibiting monotonic descent toward the target. By contrast, the D-TD3 controller's trajectories are more varied, with some trials showing temporary increases in separation before converging.

This difference is also reflected in convergence speed. The baseline D-TD3 con-

troller reaches the 0.1m threshold faster on average (mean 228s vs. 379s, max 319s vs. 521s). While slower, the UA-IQN controller exhibits tighter grouping in its approach curves, suggesting a more repeatable and reliable convergence pattern. While this comes at the cost of speed, it suggests a strategy shaped by risk aversion; an intended outcome of training with uncertainty awareness.

Velocity trends shown in Figure 6.8 further reinforce this interpretation. The baseline agent reaches up to $0.1\mathrm{m\,s^{-1}}$ with broader variation across trials, while the UA-IQN controller caps at $0.06\mathrm{m\,s^{-1}}$ and follows a more linear acceleration/deceleration pattern. The result is a gentler, more uniform approach. This behavior appears to emerge naturally from the training process, where the agent learns to avoid rare but costly overshoot events. The clustering of the UA-IQN velocity profiles suggests that this strategy generalizes well across varied starting conditions.

Interestingly, this conservative translational behavior is balanced by a more aggressive attitude control strategy. The UA-IQN reaches the 1° orientation error threshold in a mean of just 12.4s, compared to 76.0s, while maintaining comparable final angular errors. Figure 6.10 reveals how this is achieved: the UA-IQN controller executes fast, high-torque maneuvers early in each episode, peaking at up to $3.5°\mathrm{s^{-1}}$ within the first 6s. By comparison, the baseline controller peaks below $1.0°\mathrm{s^{-1}}$ and reduces angular velocity more gradually. While the distributional controller's motion is slightly more oscillatory after reaching the threshold, the rapid alignment offers a substantial gain in settling time, particularly important when attitude convergence is a prerequisite for executing docking maneuvers or when using a vision based navigation system.

Taken together, these results reflect a consistent control pattern learned by the UA-IQN agent: cautious, low-risk behavior in translation, where positional overshoot could result in docking failure, paired with aggressive, high-agility maneuvers in rotation, where overshoots are more forgiving and easily corrected. This asymmetric strategy appears to emerge from the controller's awareness of the spread of the distribution, enabling it to distinguish between high-variance and low-cost actions depending on the task context.

While the UA-IQN controller does not outperform the D-TD3 controller in every metric, it exhibits a clear, interpretable trade-off: deliberate and safe when the cost of error is high; fast and greedy when the risk is lower. This behavior is consistent with the controller's uncertainty-aware design and reflects the intended benefits of distributional reinforcement learning in safety-critical domains like satellite docking.

6.5 Hardware-in-the-loop Testing

6.5.1 Testbed Setup: ISAM Facility

The hardware-in-the-loop evaluation of the proposed distributional reinforcement learning controller was conducted at City, St George's In-Orbit Servicing, Assembly and Manufacturing (ISAM) testbed. This facility provides a high-fidelity testing environment for validating guidance, navigation, and control algorithms under realistic physical conditions. Unlike software-in-the-loop testing, the ISAM platform incorporates real robotic actuation, motion capture sensing, and physical feedback pathways, enabling the closed-loop evaluation of flight-like control behaviors. By bridging the gap between simulation and deployment, the ISAM facility plays a critical role in assessing controller robustness in the presence of partial observability, signal noise, system latency, and actuation delays.

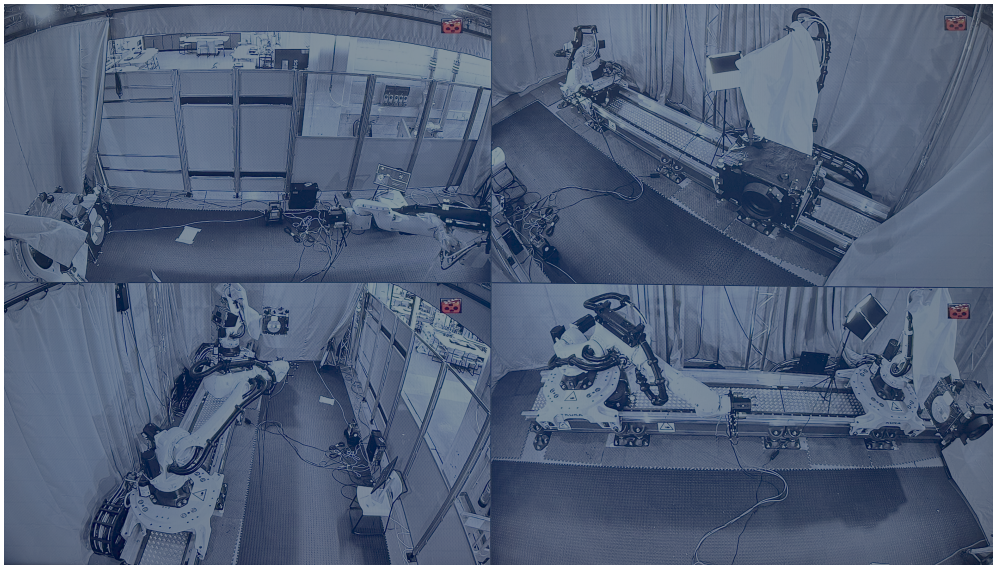


Figure 6.11: Live footage of the ISAM facility during operation.

At its core, the ISAM facility simulates satellite proximity operations by reproducing the relative motion between a chaser and a target vehicle using industrial robotic manipulators. The key components of the testbed include: (1) a high-precision motion tracking system; (2) two industrial robotic arms mounted on a linear track; (3) a ground station responsible for simulating dynamics and translating control inputs into robot joint commands; and (4) a guidance and control module running the learned RL agents that make up the controller.

A. Motion Capture Sensing

The facility is instrumented with an OptiTrack motion capture system [240], which serves as the primary sensor input to the control loop. Reflective markers affixed to the chaser test article allow for six-degree-of-freedom pose tracking with sub-millimeter accuracy. The motion capture system provides updates at a rate of 120 Hz, delivering timestamped position and orientation data relative to a global reference frame. However, since only position and orientation are directly observed; linear and angular velocities are estimated via numerical differentiation across time steps, introducing an additional source of sensor noise and temporal lag. To simulate more realistic sensor performance, additional zero-mean Gaussian noise is injected into the pose data stream before it is passed to the controller, mirroring the noise profile used during testing.

The combination of inferred velocities and noisy pose data creates a partially observable environment for the agent. Unlike in simulation, the agent has no access to ground truth state vectors and must make control decisions based solely on this corrupted observation stream. This setup reflects typical aspects of satellite control, where imperfect state estimation and measurement uncertainty represent common operational constraints.

B. Robotic Actuation System

The testbed employs two KUKA KR 50 R2100 industrial robotic arms, each mounted on a KL 4000 linear track with independent carriages [241]. Each arm provides six rotational degrees of freedom and an additional linear degree of freedom along the rail, offering full spatial control of the mounted test article. The robots have a rated payload of 50 kg, a maximum reach of 2.1 m, and a pose repeatability of ± 0.05 mm, making them suitable for emulating precise, smooth orbital motion trajectories. The end effector of one arm serves as the chaser satellite, while the other holds the target in a fixed pose. Figure 6.12 shows the physical layout of the dual-arm configuration.

To represent the target satellite, the end-effector of one KUKA arm carries a half-scale model of ESA's Proba-3 Coronagraph Spacecraft (CSC) equipped with a commercial refueling mechanism [242]. While docking dynamics and mechanical coupling are outside the scope of this study, this physical setup offers a representative mechanical configuration for assessing guidance and control behavior during final approach.

Robotic actuation is coordinated through the Robotic Operating System (ROS)

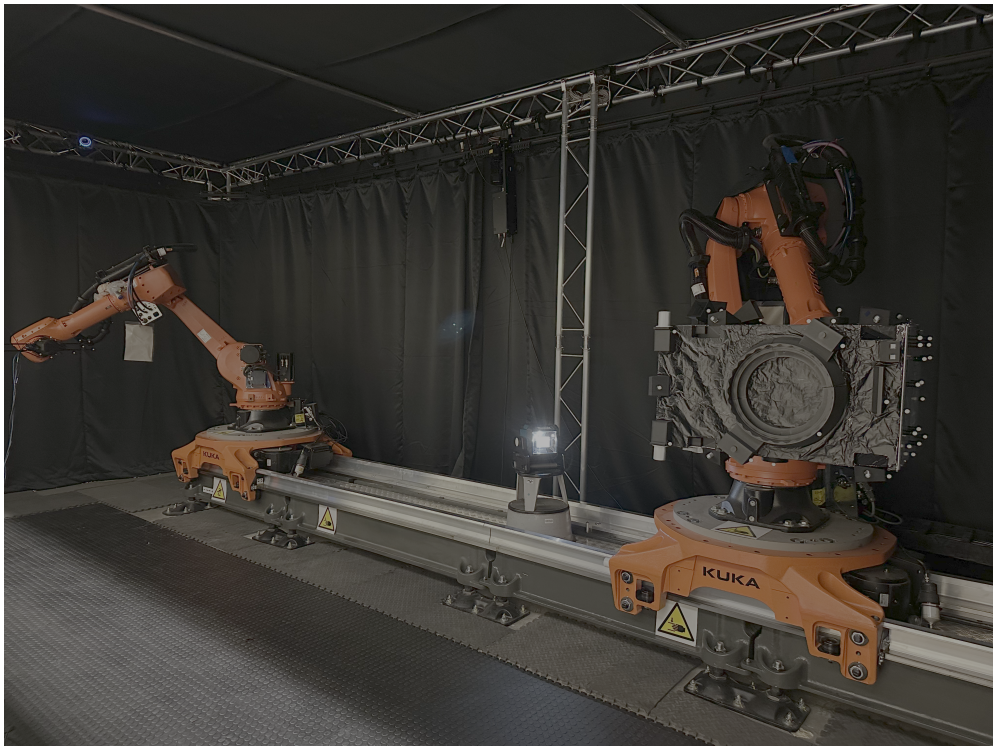


Figure 6.12: ISAM facility setup showing two KUKA robots on a linear track with the Proba-3 model mounted as the target satellite.

based control stack described below. The KUKA arms do not execute control commands directly; instead, they are driven to pose waypoints derived from a simulated dynamics model, ensuring that motion reflects the chaser's virtual momentum, inertia, and external disturbances.

C. Ground Station and Dynamics Simulator

The ground station software serves as the interface between the RL controller and the physical actuation system. It performs two primary functions: (1) simulating the translational and rotational dynamics of the chaser satellite; and (2) converting the resulting pose into joint commands for the robotic arms.

The ground station maintains a virtual state of the chaser satellite by integrating commanded forces and torques through a 6-DoF rigid body dynamics simulator. The internal chaser state is propagated within the dynamics simulator at a fixed rate of 100Hz to maintain high numerical accuracy. However, updates are forwarded to the ground station at a lower frequency matching the control timestep of 0.5 s.

Each updated pose from the simulator is passed to RViz, a 3D visualization and

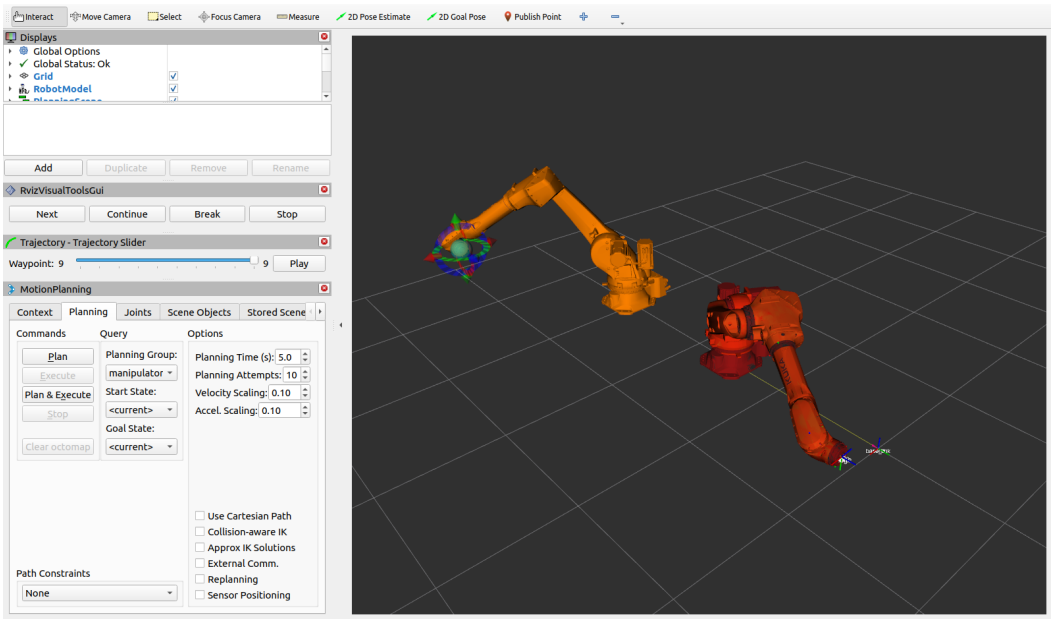


Figure 6.13: Visualization from RViz showing the robot arm during a simulated proximity operation.

simulation tool within the ROS framework. RViz uses a Unified Robot Description Format (URDF) model of the robotic arm to compute the required joint trajectories via inverse kinematics. Before executing the motion, RViz performs a virtual simulation of the movement from the current arm pose to the commanded one, checking for potential issues such as collisions, singularities, or joint limit violations. Only after passing these safety checks is the motion command transmitted to the physical robot via KUKA's Robot Sensor Interface (RSI). This process enables safe and reliable testing of guidance and control algorithms, ensuring that satellite-relative motions are executed in real time with both precision and safety.

D. Guidance and Control Loop

The guidance and control module runs the RL-based controllers in real time on a secondary control computer. Each agent receives a noisy, estimated chaser pose as input, processes it through its neural network policy, and outputs a vector of control forces or torques. These actions are passed to the ground station dynamics simulator, which integrates them to generate the next chaser pose. That pose is then enacted in physical space by the robotic arms, and the process repeats. This closed-loop architecture is shown schematically in Fig. 6.14.

By introducing closed-loop delays, partial observability, sensor noise, and motion

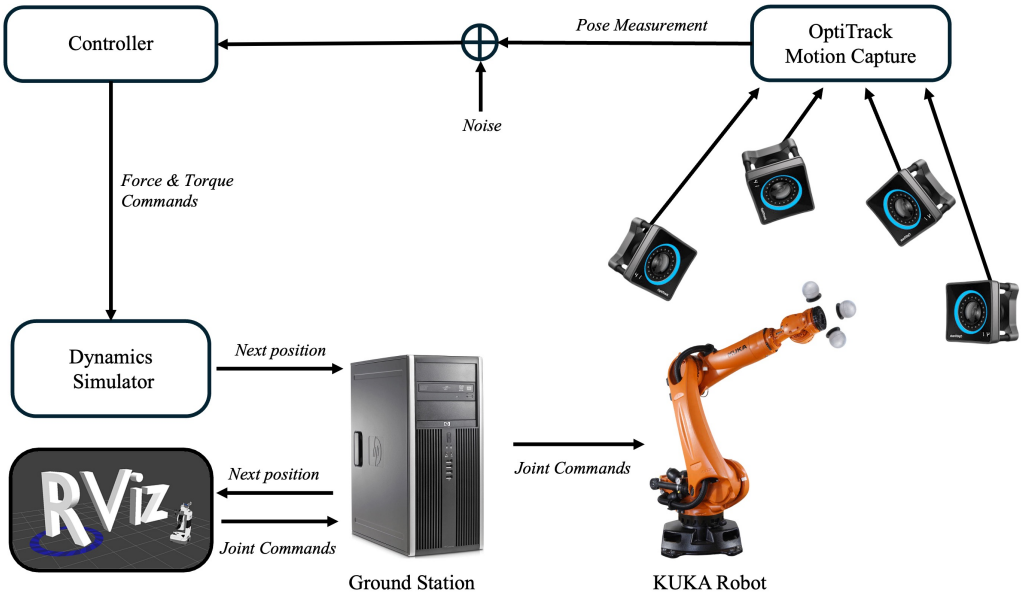


Figure 6.14: Diagram of the full closed-loop ISAM hardware-in-the-loop testbed, showing the interaction between the controller, dynamics simulator, visualizer, KUKA arm, and OptiTrack motion capture feedback.

latency, the testbed exposes the agent to a realistic operational context. Notably, all state estimation, decision-making, and actuation occur under real-time constraints. This ensures that the agent must not only learn an effective policy in simulation but also execute it under hardware limitations.

E. Spatial Scaling and Fidelity

To accommodate the physical size constraints of the laboratory environment, the docking scenario was executed at half scale. All translational distances were scaled by a factor of 0.5, while orientation dynamics remained unchanged. This preserves the dynamic similarity of the trajectory while allowing the complete final approach sequence to be executed within the workspace of the testbed.

The use of scaled test articles and realistic motion profiles ensures that the behavior observed during HIL testing reflects the same relative dynamics encountered in orbital scenarios. Moreover, since both the controller and dynamics simulator operate in scaled coordinates, the policy remains agnostic to the absolute scale, focusing instead on achieving precise relative motion.

6.5.2 HIL Testing Results

Two representative test cases were selected to evaluate the performance of the D-TD3 and UA-IQN controllers in hardware-in-the-loop conditions. Case 1 simulates the full final approach scenario, with the chaser initialized at a 5.0m separation along the v-bar, and moderate lateral and angular offsets: an initial position error of [0.5, -0.25, 5.0]m and orientation error of $[-8°, 5°, -2°]$ in roll, pitch, and yaw respectively. Case 2 represents a shorter approach starting from 2.5 m separation with a position error of $[-0.1, -0.3, 2.5]$ m and orientation error of $[3°, -7.5°, 2°]$. In both cases, the D-TD3 and UA-IQN agents were initialized with identical seeds and environment configurations to ensure a controlled, repeatable comparison. The docking sequence in each test terminates automatically once the chaser reaches a v-bar separation of 0.1 m, under the assumption that low-level servoing and mechanical capture would be triggered at this point.

Case 1: 5.0 m Initial Separation

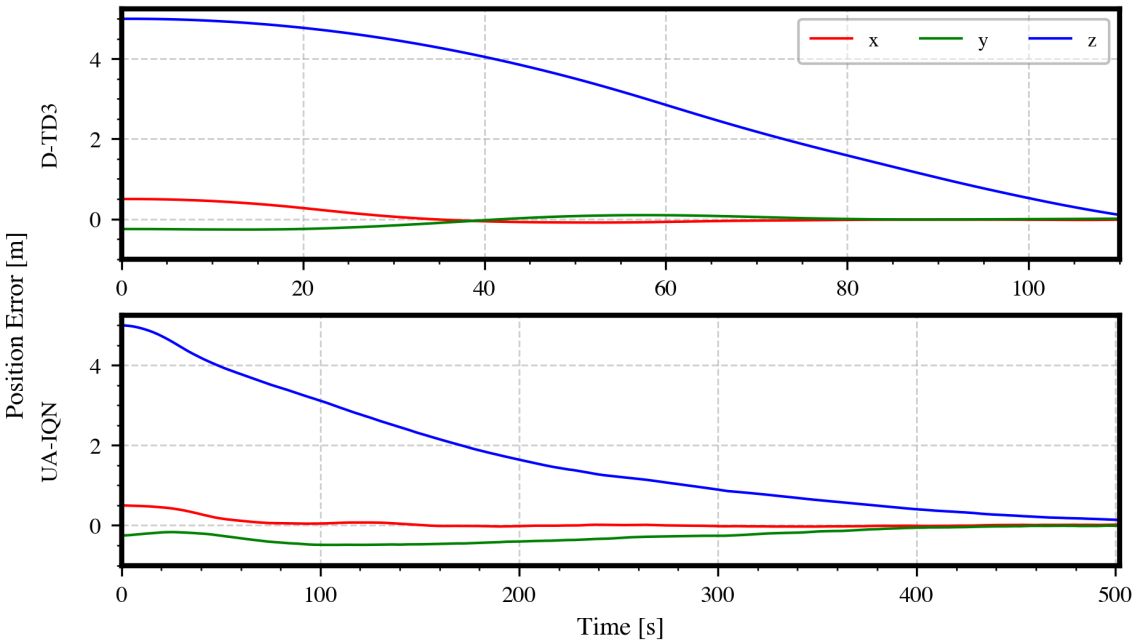


Figure 6.15: Position errors for the D-TD3 and UA-IQN controllers during test case 1

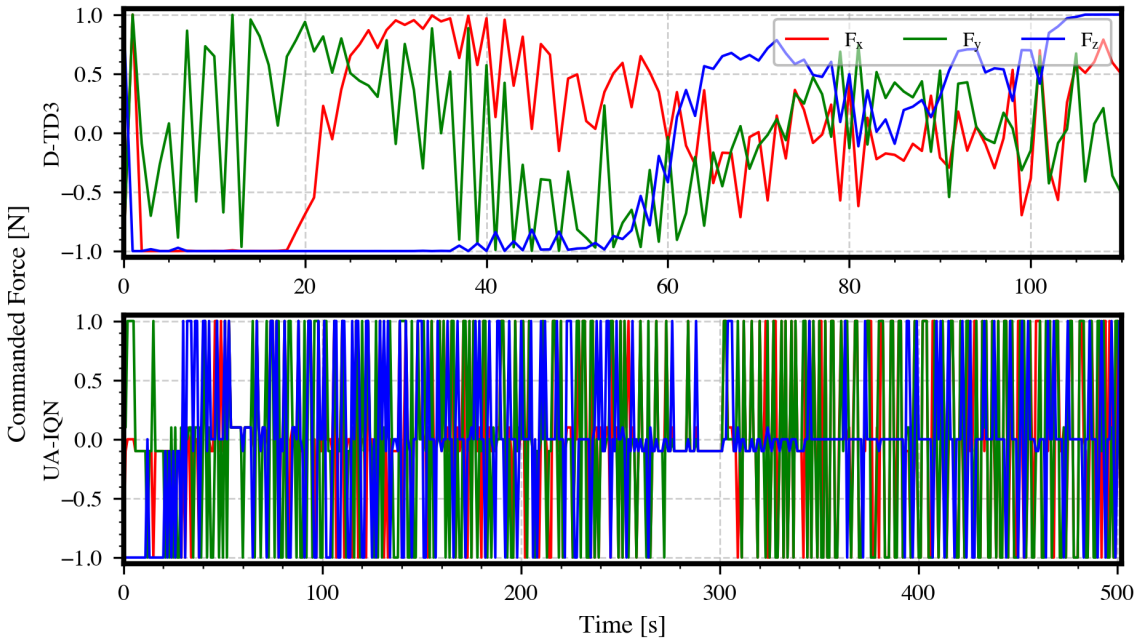


Figure 6.16: Force Commands from the D-TD3 and UA-IQN controllers during test case 1

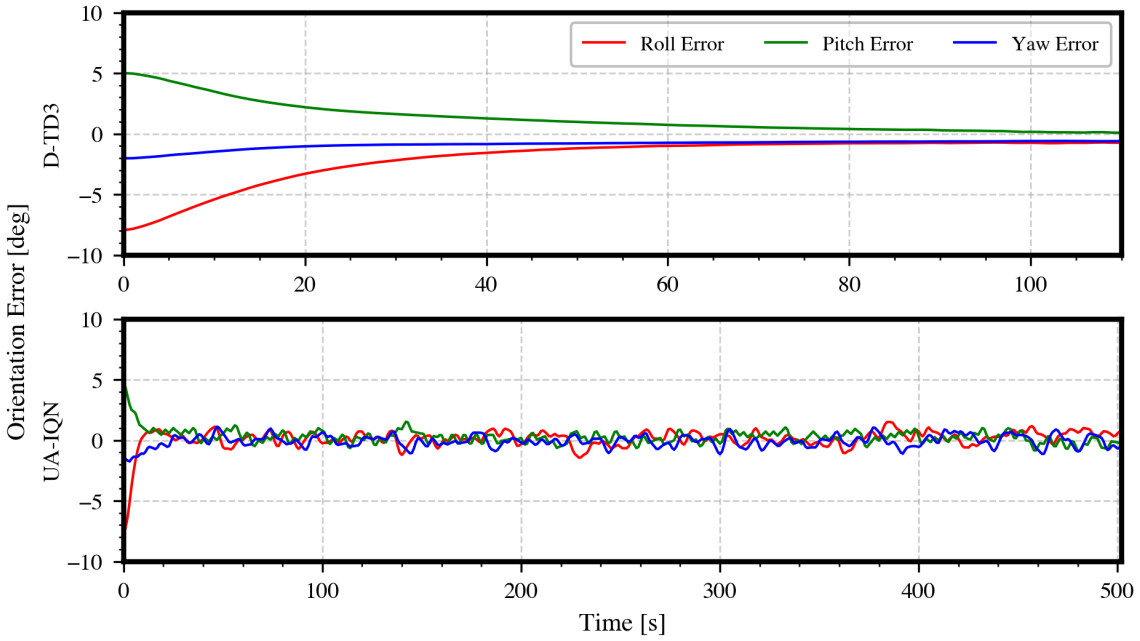


Figure 6.17: Orientation errors for the D-TD3 and UA-IQN controllers during test case 1

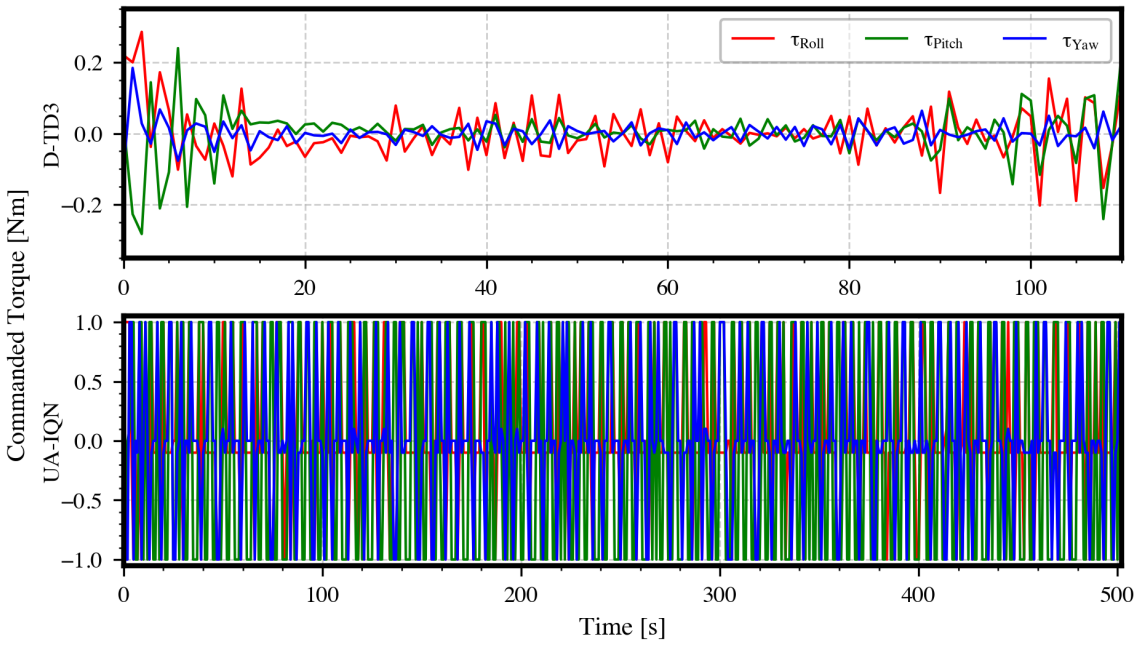


Figure 6.18: Torque Commands from the D-TD3 and UA-IQN controllers during test case 1

In Case 1, both controllers are initialized with a 5.0 m separation and moderate angular offsets. The D-TD3 agent completes the docking maneuver in approximately 110s, reducing the v-bar error quickly, while the UA-IQN agent takes roughly 500s to reach the 0.1m threshold, as shown in Figure 6.15. Lateral errors remain small for both controllers, with D-TD3 finishing with x and y errors of -1.73cm and 0.68cm, and UA-IQN with 1.58cm and -0.69cm, respectively. Although slower, UA-IQN maintains smoother position error trajectories, avoiding overshoot, and demonstrates consistent correction via frequent toggling across discrete force levels, shown in Figure 6.16. In contrast, D-TD3 exhibits more irregular and impulsive force commands, especially in the lateral axes, with sharper transitions during mid-course adjustments.

Attitude control further highlights the divergence in control behavior. As shown in Figure 6.17, D-TD3 gradually reduces angular errors, converging to final roll, pitch, and yaw errors of -0.74°, 0.09°, and -0.60°. UA-IQN, by contrast, eliminates initial orientation error within the first 40s and holds it near zero thereafter, achieving final errors of 0.83°, 0.10°, and -0.23°. This rapid correction is supported by the torque commands in Figure 6.18, where UA-IQN exhibits high-frequency switching without sustained saturation. D-TD3 generates smoother, low-amplitude torques (within ± 0.25 N m) but lacks responsiveness during dynamic disturbances.

Case 2: 2.5 m Initial Separation

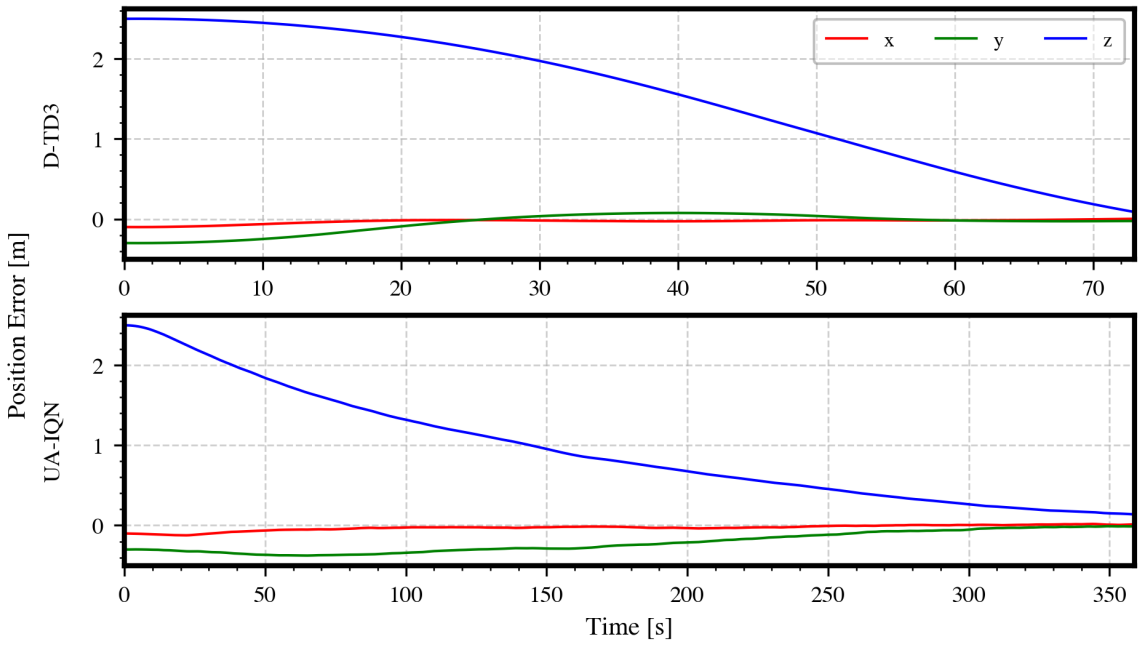


Figure 6.19: Position errors for the D-TD3 and UA-IQN controllers during test case 2

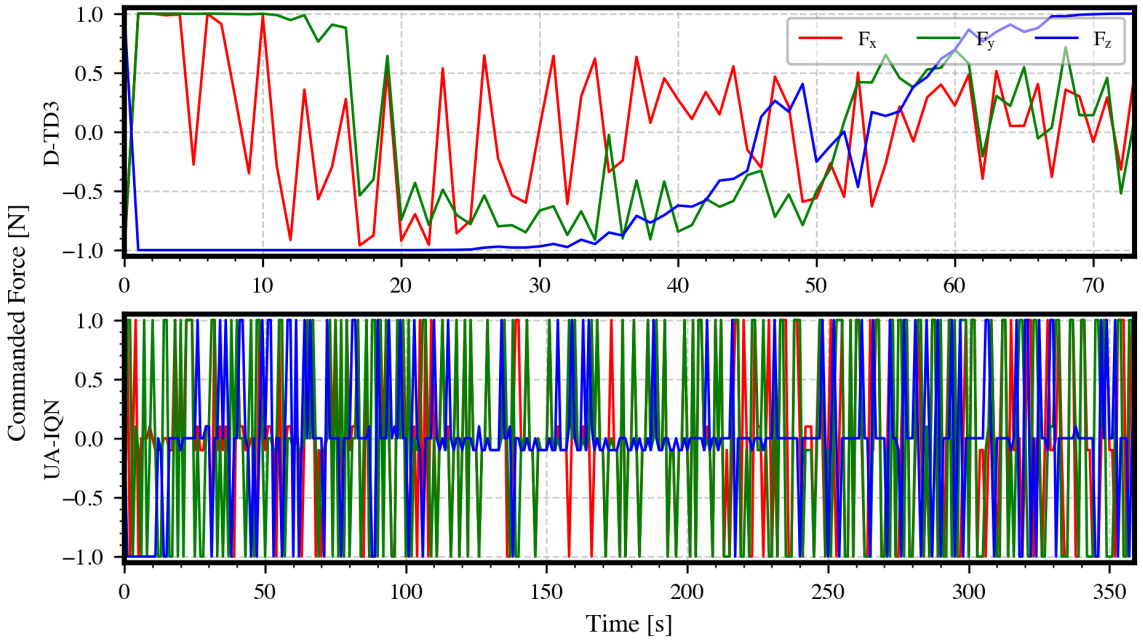


Figure 6.20: Force Commands from the D-TD3 and UA-IQN controllers during test case 2

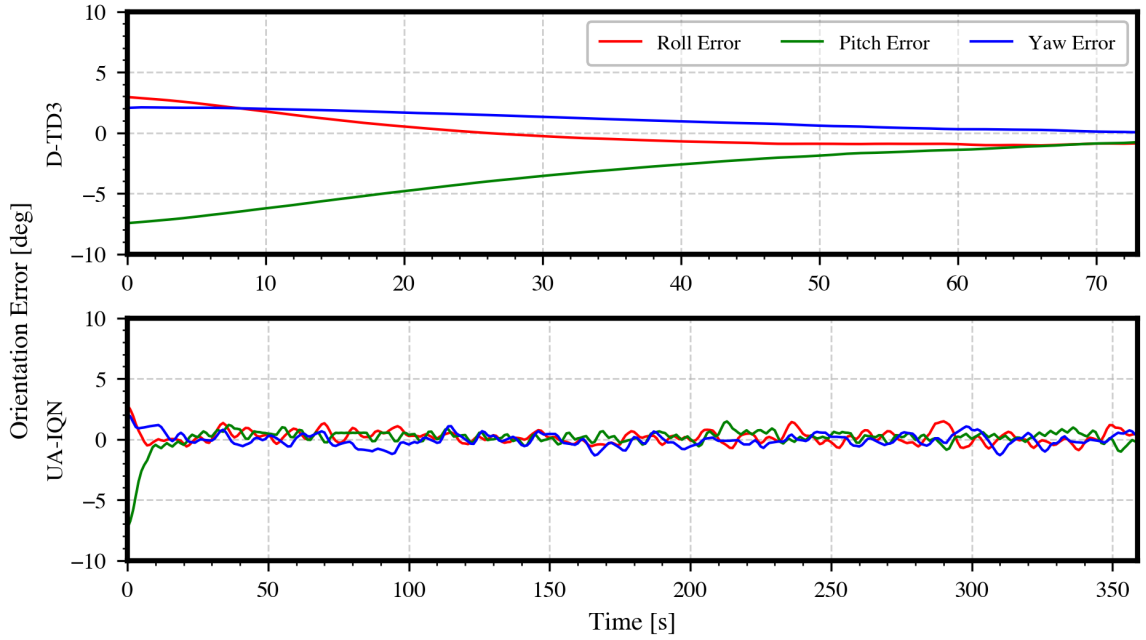


Figure 6.21: Orientation errors for the D-TD3 and UA-IQN controllers during test case 2

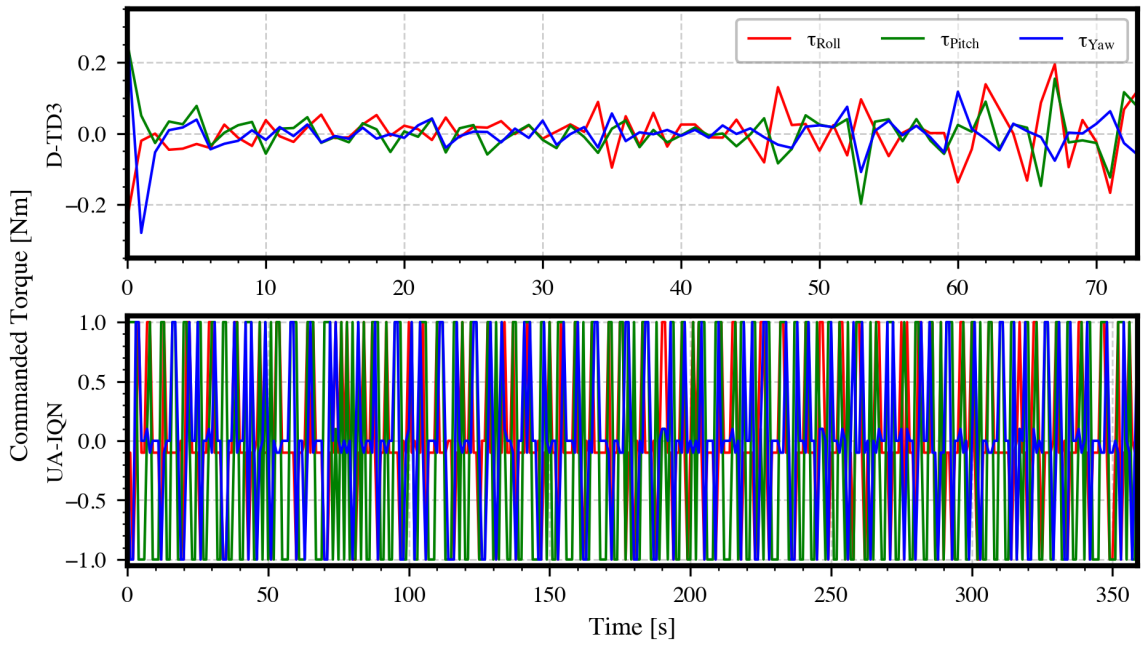


Figure 6.22: Torque Commands from the D-TD3 and UA-IQN controllers during test case 2

In Case 2, the chaser begins closer to the target, with a 2.5 m separation and more aggressive angular offsets. Figure 6.19 shows that D-TD3 completes the maneuver in approximately 70 s, following a more direct, higher-energy path, while UA-IQN requires just over 350 s to reach the docking point. Final lateral errors for D-TD3 are 0.23cm in x and -2.53cm in y, while UA-IQN ends with 1.30cm and -1.10cm, respectively. D-TD3 closes the gap quickly but with some overshoot before braking; UA-IQN converges more slowly, but with minimal transients. As shown in Figure 6.20, D-TD3 again issues larger, less frequent impulses, especially around lateral corrections, while UA-IQN retains its toggling behavior, albeit with more restraint than in Case 1.

The attitude control results echo those seen in the first scenario. D-TD3's convergence is slower and less consistent, particularly in pitch, which initially drifts before stabilizing. Its final orientation errors are -0.87°, -0.78°, and 0.04° for roll, pitch, and yaw, respectively, shown in Figure 6.21. UA-IQN quickly suppresses all angular errors within the first minute, maintaining sub-degree accuracy through the remainder of the trial, with final errors of 0.45°, -0.27°, and 0.35°. As shown in Figure 6.22, its torque commands remain high-frequency and discrete, but consistently bounded. D-TD3 continues to prioritize smoother, lower-magnitude torque outputs, trading off reactivity for control economy.

Overall, both controllers successfully completed the HIL docking tasks, demonstrating performance consistent with prior SIL evaluations. D-TD3 consistently favored faster convergence in position but had a tendency to overshoot. In contrast, UA-IQN exhibited more stable behavior, leveraging frequent discrete control updates to maintain tight bounds on attitude and position errors despite limited action resolution. Notably, the UA-IQN agent appeared to distinguish between the risks of translational and rotational domains, applying smoother corrections in position while aggressively stabilizing orientation, suggesting a degree of risk awareness emerging from its uncertainty-aware structure. While neither controller was dominant across all metrics, both achieved sub-centimeter and sub-degree accuracy by the end of each test, validating their applicability in a realistic, closed-loop proximity operations testbed.

6.6 Summary

This chapter introduced a distributional reinforcement learning framework for uncertainty-aware satellite guidance and control. Building on the foundations established in previous

chapters, the proposed approach extends the capability of DRL-based controllers by moving beyond point estimates of expected return to model the full return distribution. This richer representation enables the agent to reason about both the expected outcome and its variability, supporting risk-sensitive decision-making.

The Uncertainty-Aware Implicit Quantile Network (UA-IQN) architecture developed in this work combines the flexibility of quantile-based distributional learning with a variance-penalised action selection mechanism. By incorporating return variance into the control loop, the controller systematically favours actions with more predictable outcomes. The approach was implemented for a 6-DoF final-approach rendezvous task using modular translational and rotational agents, each trained under a dense, physically motivated reward structure.

Through Monte Carlo evaluation and hardware-in-the-loop testing, the UA-IQN controller demonstrated stable, repeatable performance across a wide range of initial conditions, achieving task success comparable to the baseline D-TD3 controller but with more consistent trajectory behaviour. Across 100 randomized docking trials, both controllers achieved successful docking with sub-centimeter lateral accuracy at the docking threshold (UA-IQN mean off-track 0.004m, mean cross-track 0.006m) and sub-degree attitude errors at docking (mean roll/pitch/yaw 0.263°/0.175°/0.277°). Consistent with its uncertainty-aware design, UA-IQN tended to adopt smoother, more conservative translational approach profiles while maintaining stable final orientations, reflecting a coherent risk-aware control strategy that is cautious and deliberate in translation but agile and decisive in rotation.

Overall, this work advances the application of deep reinforcement learning for autonomous satellite operations by integrating uncertainty estimation directly into the control policy. The results demonstrate that distributional RL can produce risk-sensitive behaviors without architectural or computational complexity beyond existing methods. This lays the groundwork for RL controllers that can reason explicitly about confidence and reliability, marking a noteworthy step toward trustworthy learning-based autonomy in safety-critical space systems.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis set out to develop and evaluate deep reinforcement learning control architectures for autonomous spacecraft close-proximity operations that are robust, scalable, and suitable for real-time deployment. In line with the objectives stated in Chapter 1, the work aimed to: (i) design learning-based control architectures capable of accurate and reliable 6-DoF spacecraft guidance and control; (ii) demonstrate that the controllers can maintain performance under uncertainty, modelling errors, disturbances, and degraded sensing or actuation; (iii) ensure safe behaviour by respecting operational constraints throughout execution; and (iv) develop architectures that are computationally efficient, scalable, and compatible with real-time implementation. The thesis addressed these objectives progressively, moving from architectural design baselines to safety, sparse-reward learning, and uncertainty-aware decision making, with validation in simulation and hardware-in-the-loop where applicable.

Chapter 1 outlined the motivation for this research: as autonomous satellites become more critical for tasks like on-orbit servicing, inspection, and debris removal, there is an urgent need for controllers that can operate reliably in uncertain, dynamic environments with minimal supervision. Traditional controllers can struggle to adapt in real time, especially under failure or model mismatch. DRL offers an alternative: a way to learn adaptable policies from data rather than rely solely on pre-modeled assumptions.

Chapter 2 provided the technical foundation for this thesis. It began with a high-level overview of reinforcement learning and key DRL algorithms used throughout the work. The second part of the chapter establishes the kinematics and dynamics models

that underpin the simulation environment used to train and evaluate all proposed controllers.

Chapter 3 addressed the first objective by providing a comparative study of centralised and decentralised DRL architectures for spacecraft control. Two TD3-based controller architectures were developed, a centralised controller that learned a single policy over the full 6-DoF action space, and a decentralised controller that split translational and rotational control into separate agents. Using a shared reward structure and controlled simulation environment, the research demonstrated that the decentralised controller consistently outperformed the centralised version in terms of stability, precision, and convergence across 100 randomised trials, while also reducing control effort and producing lower position and orientation errors. This validated a key hypothesis of the thesis: that modular control architectures can lead to more stable and interpretable learning, especially in high-dimensional, multi-objective systems.

Building on that modular foundation, Chapter 4 focused on the second and third objectives by introducing a DRL architecture explicitly designed for safety and robustness. It combined adaptive domain randomisation (ADR) with relaxed control barrier functions (CBFs) to improve generalisation and impose formal safety constraints during both training and execution. Three controller variants (Baseline, Adaptive, and Safe) were trained and evaluated on a 6-DoF rendezvous task under nominal conditions and a progressive fault cascade involving actuator and sensor degradation. Under nominal conditions, all controllers reached the terminal thresholds; however, under the failure cascade, the Baseline controller diverged rapidly (position error exceeding 45 m and reaching up to 1000 m across trials), whereas the Safe controller consistently bounded position error within 5 m and maintained a controlled attitude response. The reported CBF activation patterns during degradation further highlighted the role of the safety layer as sensing and actuation degraded, supporting stable behaviour in off-nominal conditions.

Chapter 5 addressed a key challenge of learning under sparse rewards and reducing reliance on manually engineered reward functions, addressing the first and fourth objectives. A goal-conditioned hierarchical deep reinforcement learning (HDRL) framework was developed to decompose the control problem into high-level subgoal generation and low-level actuation, enabling learning from binary terminal rewards using hindsight experience replay and subgoal relabelling. In simulation, the decentralised HDRL controller achieved mission-compliant terminal accuracy across 100 Monte Carlo trials (final position error 2.76 ± 1.11 m and final orientation error 0.95 ± 0.42 °),

comparable to the PID baseline. To support real-time implementation considerations, a hardware-in-the-loop testbed was designed and built, combining a reaction-wheel-actuated CubeSat mock-up on a spherical air-bearing with a 6-DoF robotic arm for translational emulation. Both HDRL and D-TD3 controllers were deployed on embedded hardware and maintained closed-loop stability under real-time sensing, actuation, and communication conditions. In HIL testing, both controllers reduced position error from approximately 110 m to below 5 m, with the hierarchical controller reaching this threshold by approximately 90 s compared with around 120 s for D-TD3, while both achieved attitude regulation within 5° by roughly 25 s. In contrast, the PID controller was unable to stabilise the vehicle under the same HIL conditions.

Chapter 6 addressed the second and third objectives by introducing uncertainty-aware, risk-sensitive decision making within a DRL control architecture. The Uncertainty-Aware Implicit Quantile Network (UA-IQN) framework was developed using distributional RL to model the full return distribution, with variance-penalised action selection to favour more predictable outcomes. Applied to a 6-DoF final-approach docking task using modular translational and rotational agents, UA-IQN demonstrated stable, repeatable performance across a wide range of initial conditions in both simulation and HIL evaluation. Across 100 randomised docking trials, UA-IQN achieved task success comparable to the baseline D-TD3 controller, with sub-centimetre lateral accuracy at the docking threshold (mean off-track 0.004 m, mean cross-track 0.006 m) and sub-degree attitude errors at docking (mean roll/pitch/yaw 0.263°/0.175°/0.277°). Consistent with its uncertainty-aware design, the controller produced smoother and more conservative translational approach profiles while maintaining stable final orientations.

Together, the contributions of this thesis demonstrate a pathway toward making DRL-based satellite controllers operationally viable. By way of modular design, safety integration, sparse-reward learning, and risk-sensitive decision-making, the proposed architectures advance the frontier of autonomous space systems. While challenges remain in terms of verification, trust, and hardware integration, this work lays a solid foundation for future research and development in learning-based satellite control.

7.2 Future Work

Building on the results of this thesis, several promising directions emerge for further research. These directions aim to bridge the gap between experimental validation and operational deployment, and to extend the proposed methods to more complex and

realistic mission scenarios.

7.2.1 Toward More Trustworthy Reinforcement Learning

While Chapters 4 and 6 addressed safety and uncertainty-awareness in DRL, future work could focus on making these agents more trustworthy by explicitly quantifying confidence during execution. One promising direction is to extend the Uncertainty-Aware Implicit Quantile Network (UA-IQN) architecture to output calibrated measures of confidence or epistemic uncertainty. This could enable downstream systems to reason over the trustworthiness of the agent's decisions. Techniques from Bayesian deep learning, ensemble methods, or distributional policy modeling could be incorporated to better quantify uncertainty in both the agent's policy and value function. Ultimately, a more transparent treatment of uncertainty would improve safety and reliability in high-stakes, real-time operations.

7.2.2 Hybrid DRL-Classical Control for Adaptive Gain Tuning

A promising avenue for future work is the integration of deep reinforcement learning with classical control methods, specifically using DRL to adapt the parameters or gains of traditional controllers such as PID, LQR, or MPC. While DRL offers strong adaptability and generalisation, classical controllers provide well-understood structure, stability guarantees, and predictable behaviour. Combining the two could offer the best of both: real-time adaptability to changing conditions, with the robustness and interpretability of model-based control. This hybrid approach would retain the low-latency, certifiable execution path of classical control, while introducing a learning-based outer loop that improves performance in uncertain or dynamic scenarios. Such architectures could be particularly valuable for missions where control authority must remain tightly bounded, but adaptability is still critical.

7.2.3 Extending to Multi-Agent and Formation Flying Scenarios

All control architectures developed in this thesis focused on single-agent proximity operations. A natural extension is to apply these methods to multi-agent scenarios such as cooperative inspection, formation flying, or dual-satellite rendezvous. The

modular agents developed in Chapters 3-6 could be adapted to coordinate with each other via learned or explicit communication protocols. This would introduce new challenges in decentralized control, inter-agent safety constraints, and shared reward structures. Techniques from multi-agent reinforcement learning, centralized training with decentralized execution, or distributed learning architectures could be leveraged. Exploring these scenarios would extend the applicability of DRL-based control to a broader class of autonomous space missions involving coordination and collaboration between satellites.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[2] A. Weiss, U. V. Kalabić, and S. Di Cairano, "Station keeping and momentum management of low-thrust satellites using mpc," *Aerospace Science and Technology*, vol. 76, pp. 229–241, 2018.

[3] Z. Huang, W. Zhang, T. Chen, H. Wen, and D. Jin, "Characterizing an air-bearing testbed for simulating spacecraft dynamics and control," *Aerospace*, vol. 9, p. 246, 05 2022.

[4] Y. B. Gershom, B. Onodera, M. Rayno, and J. Ang, "Design and development of a 3u cubesat adcs testing assembly with matching inertia tensor," in *Proceedings of the AIAA/USU Conference on Small Satellites*, (Logan, Utah, USA), 2017.

[5] D. Modenini, A. Bahu, G. Curzi, and A. Togni, "A dynamic testbed for nanosatellites attitude verification," *Aerospace*, vol. 7, p. 31, 03 2020.

[6] W. Commons, "File:european proximity operations simulator (epos) test facility in oberpfaffenhofen.jpg — wikimedia commons, the free media repository," 2025. [Online; accessed 28-October-2025].

[7] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," 2018.

[8] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," 2017.

[9] M. Tipaldi, R. Iervolino, and P. Massenio, "Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges," *Annual Reviews in Control*, 08 2022.

[10] G. Behrendt, M. Hale, A. Soderlund, S. Phillips, and E. Kain, "Time-constrained model predictive control for autonomous satellite rendezvous, proximity operations, and docking," 2025.

[11] K. Dong, J. Luo, and D. Limon, "A novel stable and safe model predictive control framework for autonomous rendezvous and docking with a tumbling target," *Acta Astronautica*, vol. 200, pp. 176–187, 2022.

[12] R. Bevilacqua, T. Lehmann, and M. Romano, "Development and experimentation of lqr/apf guidance and control for autonomous proximity maneuvers of multiple spacecraft," *Acta Astronautica*, vol. 68, no. 7, pp. 1260–1275, 2011.

[13] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 5064–5078, 2024.

[14] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, 2023.

[15] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," 2021.

[16] G. Fereoli, H. Schaub, and P. Di Lizia, "Meta-reinforcement learning for spacecraft proximity operations guidance and control in cislunar space," *Journal of Spacecraft and Rockets*, vol. 61, pp. 1–13, 10 2024.

[17] D. Du, N. Qi, Y. Liu, and W. Pan, "Deep bayesian reinforcement learning for spacecraft proximity maneuvers and docking," 2024.

[18] A. Tammam, A. Zenati, and N. Aouf, "Deep reinforcement learning for rendezvous and attitude control of cubesat class satellite," in *2023 7th International Conference on Automation, Control and Robots (ICACR)*, pp. 140–146, 2023.

[19] A. Tammam and N. Aouf, "Hierarchical deep reinforcement learning for cubesat guidance and control," *Control Engineering Practice, an IFAC Journal*, vol. 156, p. 106213, 2025.

[20] A. Tammam and N. Aouf, "Safe spacecraft guidance using adaptive domain randomization and relaxed control barrier functions," in *2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2025. To appear.

[21] A. Tammam and N. Aouf, "Uncertainty-aware distributional reinforcement learning for satellite final-approach and rendezvous," *Submitted to IEEE Transactions on Aerospace and Electronic Systems*, 2025.

[22] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. England: Pearson Education, third ed., 2016.

[23] T. M. Mitchell, *Machine Learning*. USA: McGraw-Hill, Inc., 1 ed., 1997.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[25] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[27] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[28] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," 2016.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[30] O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 10 2019.

[31] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[34] K. Hornik, M. B. Stinchcombe, and H. L. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[35] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[36] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Adv. Neural Inf. Process. Syst*, vol. 12, 02 2000.

[37] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[38] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[39] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, p. 58–68, 1995.

[40] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[41] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," 10 1993.

[42] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2019.

[43] 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," 2019.

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

[45] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[46] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[47] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016.

[48] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," *31st International Conference on Machine Learning, ICML 2014*, vol. 1, 06 2014.

[49] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[50] P. Hughes, *Spacecraft Attitude Dynamics*. Dover Books on Aeronautical Engineering, Dover Publications, 2004.

[51] J. Wertz, *Spacecraft Attitude Determination and Control*. Astrophysics and Space Science Library, Springer Netherlands, 1978.

[52] W. H. CLOHESSY and R. S. WILTSHIRE, "Terminal guidance system for satellite rendezvous," *Journal of the Aerospace Sciences*, vol. 27, no. 9, pp. 653–658, 1960.

[53] J. A. Starek, E. Schmerling, G. D. Maher, B. W. Barbee, and M. Pavone, "Fast, safe, propellant-efficient spacecraft motion planning under clohessy–wiltshire–hill dynamics," *Journal of Guidance, Control, and Dynamics*, vol. 40, p. 418–438, Feb. 2017.

[54] R. E. ROBERSON, "Generalized gravity-gradient torques," in *Torques and Attitude Sensing in Earth Satellites* (S. F. SINGER, ed.), vol. 7 of *Applied Mathematics and Mechanics*, pp. 73–82, Elsevier, 1964.

[55] S. R. Starin and J. Eterno, "Attitude determination and control systems," Preprint NASA/TM-20110007876, NASA Goddard Space Flight Center, Greenbelt, MD, United States, 2011. Public Use Permitted.

[56] S. Gupta, M. X. J. Raj, and R. K. Sharma, "Short-term orbit prediction with j2 and mean orbital elements," *International Journal of Astronomy and Astrophysics*, vol. 1, no. 3, pp. 135–146, 2011.

[57] T. Guffanti, S. D'Amico, and M. Lavagna, "Long-term analytical propagation of satellite relative motion in perturbed orbits," in *Proceedings of the 27th AAS/AIAA Space Flight Mechanics Meeting*, (San Antonio, Texas, USA), American Astronautical Society (AAS), February 2017. AAS Paper 17-263.

[58] G. Zeng, M. Hu, and H. Yao, "Relative orbit estimation and formation keeping control of satellite formations in low earth orbits," *Acta Astronautica*, vol. 76, pp. 164–175, 2012.

[59] A. Aman, R. Arelhi, and N. Khan, "Studying the effects of disturbance torques on a 2u cubesat in low earth orbits," *Journal of Physics: Conference Series*, vol. 1152, p. 012024, 01 2019.

[60] Z. Tudor, "Design and implementation of attitude control for 3-axes magnetic coil stabilization of a spacecraft," 01 2011.

[61] P. Servidia and R. Sánchez-Peña, "Thruster design for position/attitude control of spacecraft," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 38, pp. 1172 – 1180, 11 2002.

[62] K. Godard, Kumar and A.-M. Zou, "A novel single thruster control strategy for spacecraft attitude stabilization," *Acta Astronautica*, vol. 86, pp. 55–67, 05 2013.

[63] Y. Wu, X. Cao, Y. Xing, P. Zheng, and S. Zhang, "Relative motion decoupled control for spacecraft formation with coupled translational and rotational dynamics," in *2009 International Conference on Computer Modeling and Simulation*, pp. 63–68, 2009.

[64] Y. K. Nakka, R. C. Foust, E. S. Lupu, D. B. Elliott, I. S. Crowell, S.-J. Chung, and F. Y. Hadaegh, *A Six Degree-of-Freedom Spacecraft Dynamics Simulator for Formation Control Research*. AIAA, Aug 2018.

[65] A. Taheri and N. Assadian, "Dynamic-programming-based failure-tolerant control for satellite with thrusters in 6-dof motion," *Advances in Space Research*, vol. 65, p. 2857–2877, June 2020.

[66] H. Yang, C. Wang, and F. Zhang, "Brief paper: A decoupled controller design approach for formation control of autonomous underwater vehicles with time delays," *Control Theory & Applications, IET*, vol. 7, pp. 1950–1958, 10 2013.

[67] H. Liu, L. Guo, and Y. Zhang, "An anti-disturbance pd control scheme for attitude control and stabilization of flexible spacecrafts," *Nonlinear Dynamics*, vol. 67, 02 2012.

[68] L. N. Afifa and Azhari, "Review on attitude determination and control system of cubesat," *International Journal of Research in Aeronautical and Mechanical Engineering*, vol. 4, pp. 10–16, Dec. 2016.

[69] X. Cubillos and L. Souza, "Using of h-infinity control method in attitude control system of rigid-flexible satellite," *Mathematical Problems in Engineering*, vol. 2009, 01 2009.

[70] J. G. Elkins, R. Sood, and C. Rumpf, "Autonomous spacecraft attitude control using deep reinforcement learning," in *Proceedings of the 71st International Astronautical Congress (IAC)*, (The CyberSpace Edition), International Astronautical Federation (IAF), October 2020.

[71] A. Mahfouz, A. Valiullin, A. Lukashevichus, and D. Pritykin, "Reinforcement learning for attitude control of a spacecraft with flexible appendages," in *Proceedings of the 73rd International Astronautical Congress (IAC)*, (Paris, France), International Astronautical Federation (IAF), September 2022.

[72] J. L. Loettgen, M. Ceriotti, G. Aragon-Camarasa, and K. Worrall, "Deep reinforcement learning for spacecraft attitude tracking manoeuvres," in *Proceedings of the Aerospace Europe Conference 2023: Joint 10th EUCASS and 9th*

CEAS Conference, (Lausanne, Switzerland), European Conference for Aeronautics and Space Sciences (EUCASS) and Council of European Aerospace Societies (CEAS), July 2023.

[73] X. Wang, G. Wang, Y. Chen, and Y. Xie, "Autonomous rendezvous guidance via deep reinforcement learning," in *2020 Chinese Control And Decision Conference (CCDC)*, pp. 1848–1853, 2020.

[74] N. Hamilton, K. Dunlap, and K. Hobbs, "Investigating the impact of choice on deep reinforcement learning for space controls," 05 2024.

[75] S. Angelucci, H. Dürr, A. Straub, and L. Tarabini, "Lunarsat adcs european orbiter to the moon attitude determination and control system," in *1999 European Control Conference (ECC)*, pp. 4870–4875, 1999.

[76] L.-L. Show, J.-C. Juang, C.-T. Lin, and Y.-W. Jan, "Spacecraft robust attitude tracking design: Pid control approach," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 2, pp. 1360–1365 vol.2, 2002.

[77] M. J. Sidi, *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Cambridge, United Kingdom: Cambridge University Press, 1997.

[78] A. Souza and L. Souza, "Design of a controller for a rigid-flexible satellite using the h-infinity method considering the parametric uncertainty," *Mechanical Systems and Signal Processing*, vol. 116, pp. 641–650, 02 2019.

[79] G. Rigatos, M. Abbaszadeh, K. Busawon, and L. Dala, "A nonlinear optimal control method for attitude stabilization of micro-satellites," *Guidance, Navigation and Control*, vol. 02, 08 2022.

[80] C.-H. Won, "Comparative study of various control methods for attitude control of a leo satellite," *Aerospace Science and Technology*, vol. 3, no. 5, pp. 323–333, 1999.

[81] K. Young, V. Utkin, and U. Ozguner, "A control engineer's guide to sliding mode control," *IEEE Transactions on Control Systems Technology*, vol. 7, no. 3, pp. 328–342, 1999.

[82] E. Abdulhamitbilal and E. Jafarov, "Performances comparison of linear and sliding mode attitude controllers for flexible spacecraft with reaction wheels," in

International Workshop on Variable Structure Systems, 2006. VSS'06., pp. 351–358, 2006.

[83] F. Terui, "Position and attitude control of a spacecraft by sliding mode control," in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, vol. 1, pp. 217–221 vol.1, 1998.

[84] J. Lee and V. Utkin, "Chattering suppression methods in sliding mode control systems," *Annual Reviews in Control*, vol. 31, pp. 179–188, 12 2007.

[85] M. Navabi and M. Radaei, "Attitude adaptive control of space systems," in *2013 6th International Conference on Recent Advances in Space Technologies (RAST)*, pp. 973–977, 2013.

[86] J.-J. Sheen and R. Bishop, "Adaptive nonlinear control of spacecraft," in *Proceedings of 1994 American Control Conference - ACC '94*, vol. 3, pp. 2867–2871 vol.3, 1994.

[87] J.-E. Slotine and M. Di Benedetto, "Hamiltonian adaptive control of spacecraft," *IEEE Transactions on Automatic Control*, vol. 35, no. 7, pp. 848–852, 1990.

[88] C.-H. Cheng, S.-L. Shu, and P.-J. Cheng, "Attitude control of a satellite using fuzzy controllers," *Expert Systems with Applications*, vol. 36, pp. 6613–6620, 04 2009.

[89] R. Cihang and J.-S. Jang, "Fuzzy logic attitude control for cassini spacecraft," pp. 1532 – 1537 vol.3, 07 1994.

[90] M. Tipaldi and L. Glielmo, "A survey on model-based mission planning and execution for autonomous spacecraft," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3893–3905, 2018.

[91] M. Shirobokov, S. Trofimov, and M. Ovchinnikov, "Survey of machine learning techniques in spacecraft control design," *Acta Astronautica*, vol. 186, 05 2021.

[92] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," 2018.

[93] J. Song, D. Rondao, and N. Aouf, "Deep learning-based spacecraft relative navigation methods: A survey," *Acta Astronautica*, vol. 191, p. 22–40, Feb. 2022.

[94] J. G. Elkins, R. Sood, and C. Rumpf, "Adaptive continuous control of spacecraft attitude using deep reinforcement learning," in *Proceedings of the 2020 AAS/AIAA Astrodynamics Specialist Conference*, (Lake Tahoe, Virtual Conference), American Astronautical Society (AAS), August 2020.

[95] S. Oghim, J. Park, H. Bang, and H. Leeghim, "Deep reinforcement learning-based attitude control for spacecraft using control moment gyros," *Advances in Space Research*, vol. 75, 08 2024.

[96] J. L. Löttgen, M. Ceriotti, G. Aragon-Camarasa, and K. Worrall, "Application of deep reinforcement learning for attitude control of a satellite in the presence of uncertainties," in *Proceedings of the 73rd International Astronautical Congress (IAC)*, (Paris, France), International Astronautical Federation (IAF), September 2022.

[97] F. Vedant, J. T. Allison, M. West, and A. Ghosh, "Reinforcement learning for spacecraft attitude control," in *Proceedings of the 70th International Astronautical Congress (IAC)*, (Washington, D.C., United States), International Astronautical Federation (IAF), October 2019.

[98] M. Hariry, A. Cini, G. Mellone, and A. Balossino, "Deep reinforcement learning policies for underactuated satellite attitude control," 04 2025.

[99] G. El-Dalahmeh, M. Jabbarpour, B. Vo, and R. Kowalczyk, "Intelligent control of spacecraft reaction wheel attitude using deep reinforcement learning," 07 2025.

[100] V. Tan, J. L. Labrador, and M. C. R. Talampas, "Mata-rl: Continuous reaction wheel attitude control using the mata simulation software and reinforcement learning," in *Proceedings of the 35th Annual AIAA/USU Small Satellite Conference*, (Logan, Utah, USA), Utah State University, August 2021.

[101] W. Retagne, J. Dauer, and G. Waxenegger-Wilfing, "Adaptive satellite attitude control for varying masses using deep reinforcement learning," *Frontiers in Robotics and AI*, vol. 11, 07 2024.

[102] H. Yang, *Manned Rendezvous and Docking Technology*, pp. 185–196. Singapore: Springer Singapore, 2021.

[103] J. Goodman, "History of space shuttle rendezvous and proximity operations," *Journal of Spacecraft and Rockets - J SPACECRAFT ROCKET*, vol. 43, pp. 944–959, 09 2006.

[104] R. Zanetti, "Optimal glideslope guidance for spacecraft rendezvous," *Journal of Guidance, Control, and Dynamics*, vol. 34, pp. 1593–1597, 09 2011.

[105] D. J. Pearson, "The glideslope approach," in *Advances in the Astronautical Sciences, Vol. 69*, (Greenbelt, Maryland, USA), April 1989.

[106] L. Mazal, D. Pérez, R. Bevilacqua, and F. Curti, "Spacecraft rendezvous by differential drag under uncertainties," *Journal of Guidance, Control, and Dynamics*, vol. 39, pp. 1–13, 04 2016.

[107] Y. zhong Luo and G. jin Tang, "Spacecraft optimal rendezvous controller design using simulated annealing," *Aerospace Science and Technology*, vol. 9, no. 8, pp. 732–737, 2005.

[108] C. L. Karr and L. Freeman, "Genetic-algorithm-based fuzzy control of spacecraft autonomous rendezvous," *Engineering Applications of Artificial Intelligence*, vol. 10, no. 3, pp. 293–300, 1997.

[109] G. Ortega, "Fuzzy logic techniques for rendezvous and docking of two geostationary satellites," *Telematics and Informatics*, vol. 12, no. 3, pp. 213–227, 1995. Advanced Space Technologies For Systems Autonomy.

[110] Q. Li, J. Yuan, B. Zhang, and C. Gao, "Model predictive control for autonomous rendezvous and docking with a tumbling target," *Aerospace Science and Technology*, vol. 69, 07 2017.

[111] A. Fear and E. G. Lightsey, "Implementation of small satellite autonomous rendezvous using model predictive control," in *AIAA SCITECH 2022 Forum*, (San Diego, CA, USA), American Institute of Aeronautics and Astronautics, 2022.

[112] S. Shao, D. Zhou, G. Sun, W. Ma, and R. Deng, "Asymmetric deep reinforcement learning-based spacecraft approaching maneuver under unknown disturbance," *Aerospace*, vol. 12, no. 3, 2025.

[113] A. Brandonisio, L. Capra, and M. Lavagna, "Deep reinforcement learning spacecraft guidance with state uncertainty for autonomous shape reconstruction of uncooperative target," *Advances in Space Research*, vol. 73, 07 2023.

[114] H. Holt, S. Origer, and D. Izzo, "Comparing behavioural cloning and reinforcement learning for spacecraft guidance and control networks," 07 2025.

[115] B. Space, "100mn hpgp thruster."

[116] B. C. Technologies, "Blue Canyon Technologies Reaction Wheels," 2023. [Accessed 21-02-2026].

[117] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[118] Z. Ding, T. Yu, Y. Huang, H. Zhang, L. Mai, and H. Dong, "Rlzoo: A comprehensive and adaptive reinforcement learning library," 09 2020.

[119] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *CoRR*, vol. abs/1703.06907, 2017.

[120] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *CoRR*, vol. abs/1710.06537, 2017.

[121] O. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, p. 027836491988744, 11 2019.

[122] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[123] T. Dai, K. Arulkumaran, S. Tukra, F. Behbahani, and A. Bharath, "Analysing deep reinforcement learning agents trained with domain randomisation," 12 2019.

[124] B. Mehta, M. Diaz, F. Golemo, C. Pal, and L. Paull, "Active domain randomization," 04 2019.

[125] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Data-efficient domain randomization with bayesian optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 911–918, 2021.

[126] G. Tiboni, K. Arndt, and V. Kyrki, "Dropo: Sim-to-real transfer with offline domain randomization," 01 2022.

[127] S. Tafazoli, "A study of on-orbit spacecraft failures," *Acta Astronautica - ACTA ASTRONAUT*, vol. 64, pp. 195–205, 02 2009.

[128] M. Alidadi and A. Rahimi, "Fault diagnosis of lubrication decay in reaction wheels using temperature estimation and forecasting via enhanced adaptive particle filter," *Sensors*, vol. 23, no. 3, 2023.

[129] W. E. Bialke and E. Hansell, "A newly discovered branch of the fault tree explaining systemic reaction wheel failures and anomalies," 2017.

[130] P. Yanhui, L. Qiang, L. Zhi, and L. Xiaoying, "Degradation analysis of analog sun sensor used by satellite in orbit," in *2021 IEEE International Conference on Emergency Science and Information Technology (ICESIT)*, pp. 657–659, 2021.

[131] M. Fazelinia, S. Ebadollahi, and S. Ganjefar, "Stochastic analysis of drift error of gyroscope in the single-axis attitude determination," *Measurement*, vol. 237, p. 115136, 06 2024.

[132] D. Ye, M. Chen, and K. Li, "Observer-based distributed adaptive fault-tolerant containment control of multi-agent systems with general linear dynamics," *ISA Transactions*, vol. 71, 06 2017.

[133] B. Guo and Y. Chen, "Adaptive fast sliding mode fault tolerant control integrated with disturbance observer for spacecraft attitude stabilization system," *ISA Transactions*, vol. 94, 04 2019.

[134] B. Li, Q. Hu, Y. Yu, and G. Ma, "Observer-based fault-tolerant attitude control for rigid spacecraft," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 5, pp. 2572–2582, 2017.

[135] J. Chang, J. Cieslak, Z. Guo, and D. Henry, "On the synthesis of a sliding-mode-observer-based adaptive fault-tolerant flight control scheme," *ISA Transactions*, vol. 111, pp. 8–23, 2021.

[136] W. Zhang, D. Xu, B. Jiang, and P. Shi, "Virtual-sensor-based model-free adaptive fault-tolerant constrained control for discrete-time nonlinear systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 10, pp. 4191–4202, 2022.

[137] J. Han, X. Liu, X. Wei, and S. Sun, "A dynamic proportional-integral observer-based nonlinear fault-tolerant controller design for nonlinear system with partially unknown dynamic," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 8, pp. 5092–5104, 2022.

[138] M. Zanon and S. Gros, "Safe reinforcement learning using robust mpc," *IEEE Transactions on Automatic Control*, vol. 66, p. 3638–3652, Aug. 2021.

[139] S. Kamthe and M. P. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," 2018.

[140] T. Koller, F. Berkenkamp, M. Turchetta, J. Boedecker, and A. Krause, "Learning-based model predictive control for safe exploration and reinforcement learning," 2019.

[141] K. P. Wabersich, L. Hewing, A. Carron, and M. N. Zeilinger, "Probabilistic model predictive safety certification for learning-based control," 2021.

[142] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," 2018.

[143] T. Perkins and A. Barto, "Lyapunov design for safe reinforcement learning control," *J. Mach. Learn. Res.*, vol. 3, pp. 803–, 05 2003.

[144] J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath, "Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions," 2020.

[145] Y. Emam, G. Notomista, P. Glotfelter, Z. Kira, and M. Egerstedt, "Safe reinforcement learning using robust control barrier functions," *IEEE Robotics and Automation Letters*, vol. 10, no. 3, pp. 2886–2893, 2025.

[146] Z. Marvi and B. Kiumarsi, "Safe reinforcement learning: A control barrier function optimization approach," *International Journal of Robust and Nonlinear Control*, vol. 31, pp. 1923–1940, 08 2020.

[147] R. Cheng, G. Orosz, R. Murray, and J. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3387–3395, 07 2019.

[148] S. Liu, L. Liu, and Z. Yu, "Safe reinforcement learning for affine nonlinear systems with state constraints and input saturation using control barrier functions," *Neurocomputing*, vol. 518, 11 2022.

[149] Serenum Space, "Rw25 cubesat reaction wheel," 2024.

[150] AAC Clyde Space, "Rw222 cubesat reaction wheel," 2025.

[151] ECAPS, "100 mn hpgp thruster," 2025.

[152] Rubicon Space Systems, "0.1n high throughput (ht) thruster," 2024.

[153] Stable Baselines3 Contributors, "Td3 — twin delayed ddpg," 2025.

[154] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, p. e253, 2017.

[155] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, (San Francisco, CA, USA), p. 278–287, Morgan Kaufmann Publishers Inc., 1999.

[156] J. Clark and D. Amodei, "Faulty reward functions in the wild." `https://blog.openai.com/faulty-reward-functions`, 2016.

[157] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.

[158] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2018.

[159] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *Autonomous Mental Development, IEEE Transactions on*, vol. 2, pp. 230 – 247, 10 2010.

[160] M. Yuan, "Intrinsically-motivated reinforcement learning: A brief introduction," 2022.

[161] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," 2016.

[162] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," 2019.

[163] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," 2018.

[164] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, pp. 41–77, 2003.

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

[166] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," *CoRR*, vol. abs/1609.05140, 2016.

[167] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," *CoRR*, vol. abs/1802.06070, 2018.

[168] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," 2019.

[169] P. Dayan and G. Hinton, "Feudal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 5, 09 2000.

[170] J. A. de Vries, T. M. Moerland, and A. Plaat, "On credit assignment in hierarchical reinforcement learning," 2022.

[171] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," 1999.

[172] B. Hengst, "Discovering hierarchy in reinforcement learning with hexq," *Proceedings of the Nineteenth International Conference on Machine Learning*, 08 2002.

[173] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," 2017.

[174] A. Newton, *Design, Development, and Experimental Validation of a Nanosatellite Attitude Control Simulator*. PhD thesis, 09 2021.

[175] A. M. Ali, "Development of an experimental setup for cubesat attitude determination and control," master's thesis, Khalifa University, 2022.

[176] B. A. M. D. Ribeiro, "Development of an air-bearing-based satellite attitude simulator and testing of an adcs solution," master's thesis, Instituto Superior Técnico, University of Lisbon, March 2021.

[177] K. R. Cooper, "Computer vision system and experiment design for parabolic flight demonstration of hard disk drives as cubesat reaction wheels," master's thesis, University of California, Davis, 2023.

[178] V. Pesce, A. Bellanca, M. Lavagna, R. Benvenuto, P. Lunghi, and S. F. Rafano Carnà, "Satleash - parabolic flight validation of tethered-tugs dynamics and control for reliable space transportation applications," 02 2017.

[179] C. Menon, A. Aboudan, S. Cocuzza, A. Bulgarelli, and F. Angrilli, "Free-flying robot tested on parabolic flights: Kinematic control," *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, vol. 28, pp. 623–630, 07 2005.

[180] H. Kojima, K. Hiraiwa, Y. Yoshimura, and K. Hidaka, "Experimental study on line-of-sight (los) attitude control using control moment gyros under microgravity environment," 06 2017.

[181] Y. Kubo, T. Ando, H. Kawahara, S. Miyata, N. Uchiyama, K. Ito, and Y. Sugawara, "Model evaluation of a transformable cubesat for nonholonomic attitude reorientation using a drop tower," 2025.

[182] J. M. Wodrich, "Zero gravity research facility," March 2025.

[183] W. Whitacre, "An autonomous underwater vehicle as a spacecraft attitude control simulator," in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, Nevada), American Institute of Aeronautics and Astronautics, January 2005. Session: NSC-1, AIAA Foundation National Student Conference - Undergraduate Division. Accessed: 2025-11-03.

[184] C. Sun, S. Chen, J. Yuan, and Z. Zhu, "A six-dof buoyancy tank microgravity test bed with active drag compensation," *Microgravity Science and Technology*, vol. 29, 10 2017.

[185] T. Rybus and K. Seweryn, "Planar air-bearing microgravity simulators: Review of applications, existing solutions and design parameters," *Acta Astronautica*, vol. 120, 12 2015.

[186] G. Yang and O. Ma, "Validation of satellite docking simulations using an air-bearing-based experimental testbed," *Int. J. of Space Science and Engineering*, vol. 1, pp. 253 – 267, 01 2013.

[187] W. Zhu, Z. Pang, and G. Zhu, "Planar air-bearing microgravity testing for maneuverable space net deployment and retrieval with distributed cooperative control," *Acta Astronautica*, vol. 236, 07 2025.

[188] New Way Air Bearings, "Spherical air bearings standard line," 2025.

[189] Physik Instrumente (PI) Ltd, "Precision motion solutions and piezo technology," 2025.

[190] P. Yanyachi, A. Mamani Saico, X. Wang, and B. Espinoza, "Development of an air-bearing testbed for nanosatellite attitude determination and control systems," *IEEE Access*, vol. PP, pp. 1–1, 01 2024.

[191] G. Corsi, "Design, prototyping and testing of a reaction wheel assembly for an air-bearing spacecraft attitude simulator," master's thesis, Politecnico di Milano, Milan, Italy, April 2022. Supervised by Francesco Topputo; Co-supervised by Gianfranco Di Domenico; Academic Year 2020/2021. Accessed: 2025-11-03.

[192] T. Kwan, K. M. B. Lee, J. Yan, and X. Wu, "An air bearing table for satellite attitude control simulation," pp. 1420–1425, 06 2015.

[193] D. Thomas and J. Black, "Cubesat attitude control simulator design," 01 2018.

[194] N. Jovanovic, J. Pearce, and J. Praks, "Design and testing of a low-cost, open source, 3-d printed air-bearing-based attitude simulator for cubesat satellites," *Journal of Small Satellites*, 01 2019.

[195] K. Rahnamai, T. Searles, and R. Parker, "Design and development of a low-cost cubesat attitude control system testing platform," in *2018 IEEE Aerospace Conference*, pp. 1–6, 2018.

[196] S. Chesi, O. Perez, and M. Romano, "A dynamic hardware-in-the-loop three-axis simulator of nanosatellite dimensions," *Journal of Small Spacecraft*, vol. 4, pp. 315–328, 01 2015.

[197] G. Cervettini, S. Pastorelli, H. Park, D. Y. Lee, and M. Romano, "Development and experimentation of a cubesat magnetic attitude control system testbed," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 2, pp. 1345–1350, 2021.

[198] A. Bahu, *Development of a Ground Testing Facility and Attitude Control for Magnetically Actuated Nanosatellites*. Ph.d. dissertation, Alma Mater Studiorum – Università di Bologna, Bologna, Italy, May 2021.

[199] R. Yeşilay, A. Ankaralı, and M. Afşar, "A review paper: The dynamics, kinematics, design and control of satellite simulators with spherical air bearing," in *Proceedings of the Global Conference on Engineering Research (GLOBCER'21)*, (Balıkesir, Türkiye), June 2021.

[200] L. Granados-Cruz, R. Chávez-Moreno, J. Ferrer-Perez, C. Romo-Fuentes, and J. Ramírez-Aguilar, "Historical review of simulators for satellite position and orientation control," *Journal of Physics: Conference Series*, vol. 2804, p. 012013, 07 2024.

[201] L. Santaguida and Z. H. Zhu, "Development of air-bearing microgravity testbed for autonomous spacecraft rendezvous and robotic capture control of a free-floating target," *Acta Astronautica*, vol. 203, pp. 319–328, 2023.

[202] J. J. Cookson, "Experimental investigation of spacecraft rendezvous and docking by development of a 3 degree of freedom satellite simulator testbed," master's thesis, York University, Toronto, Canada, May 2020.

[203] Z. Wei, H. Wen, H. Hu, and D. Jin, "Ground experiment on rendezvous and docking with a spinning target using multistage control strategy," *Aerospace Science and Technology*, vol. 104, p. 105967, 06 2020.

[204] C. Mietner, "European proximity operations simulator 2.0 (epos) - a robotic-based rendezvous and docking simulator," *Journal of Large-Scale Research Facilities JLSRF*, vol. 3, 04 2017.

[205] T. Boge and O. Ma, "Using advanced industrial robotics for spacecraft rendezvous and docking simulation," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1–4, 2011.

[206] O. Ma, A. Flores Abad, and T. Boge, "Use of industrial robots for hardware-in-the-loop simulation of satellite rendezvous and docking," *Acta Astronautica*, vol. 81, p. 335–347, 12 2012.

[207] M. Zebenay, T. Boge, R. Lampariello, and D. Choukroun, "Satellite docking simulation based on hil hybrid contact model," in *Proceedings of the 12th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2013)*, (Noordwijk, The Netherlands), DLR – German Aerospace Center, January 2013.

[208] H. Frei, F. Rems, and T. Boge, "Development and hardware-in-the-loop test of a guidance, navigation and control system for on-orbit servicing," *Acta Astronautica*, vol. 102, pp. 67–80, 09 2014.

[209] W. Gomes dos Santos, E. Marconi Rocco, T. Boge, H. Frei, and F. Rems, "Hardware-in-the-loop rendezvous tests of a novel actuators command concept," *The Journal of the Astronautical Sciences*, vol. 63, pp. 287–307, 09 2016.

[210] K. Klionovska and M. Burri, "Hardware-in-the-loop simulations with umbra conditions for spacecraft rendezvous with pmd visual sensors," *Sensors*, vol. 21, p. 1455, 02 2021.

[211] Satellite Applications Catapult, "In-orbit servicing, assembly, and manufacturing (isam) facility," 2025.

[212] J. He and M. Shen, "Hybrid force/velocity control for simulating contact dynamics of satellite robots on a hardware-in-the-loop simulator," *IEEE Access*, vol. 10, pp. 59277–59289, 2022.

[213] A. Nicolae, T. SAVU, D. GUTA, and A. Ioniţă, "Industrial robotics for spacecraft rendezvous and docking simulation," *INCAS BULLETIN*, vol. 11, pp. 27–36, 12 2019.

[214] H. Yan, C. Qi, X. Zhao, and Q. Wang, "A force and moment compensation method for a hardware-in-the-loop docking simulator based on the stiffness identification of the docking mechanism," *Mechatronics*, vol. 76, p. 102513, 06 2021.

[215] M. Wilde, C. Clark, and M. Romano, "Historical survey of kinematic and dynamic spacecraft simulators for laboratory experimentation of on-orbit proximity maneuvers," *Progress in Aerospace Sciences*, vol. 110, p. 100552, 08 2019.

[216] I. Gavrilovich, *Development of a robotic system for CubeSat Attitude Determination and Control System ground tests*. Phd thesis, Université Montpellier, December 2016. NNT: 2016MONTT329. Tel: tel-01816985.

[217] Kinova Inc., "Kinova gen3 ultra-lightweight robotic arm," 2025.

[218] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1312–1320, PMLR, 07–09 Jul 2015.

[219] California Polytechnic State University, "Cubesat design specification, revision 14.1," tech. rep., California Polytechnic State University, San Luis Obispo, February 2022.

[220] Markforged Inc., "Onyx pro™ composite 3d printer," 2025.

[221] FAULHABER Group, "B-flat sc series 2610t006b sc brushless dc-flat motors with integrated speed controller," 2025.

[222] E. Oland and R. Schlanbusch, "Reaction wheel design for cubesats," in *2009 4th International Conference on Recent Advances in Space Technologies*, pp. 778–783, 2009.

[223] N. S. Krishna, S. Gosavi, S. Singh, N. Saxena, A. Kailaje, V. Datla, and P. Shah, "Design and implementation of a reaction wheel system for cubesats," in *2018 IEEE Aerospace Conference*, pp. 1–7, 2018.

[224] Formlabs Inc., "Fuse series sls 3d printing platform," 2025.

[225] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," 2017.

[226] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[227] D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, and T. Liu, "Fully parameterized quantile function for distributional reinforcement learning," 2020.

[228] B. Charpentier, R. Senanayake, M. Kochenderfer, and S. Günnemann, "Disentangling epistemic and aleatoric uncertainty in reinforcement learning," 2022.

[229] Q. Liu, Y. Li, Y. Liu, M. Chen, S. Lv, and Y. Xu, "Exploration via distributional reinforcement learning with epistemic and aleatoric uncertainty estimation," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 2256–2261, 2021.

[230] Q. Liu, Y. Li, S. Chen, K. Lin, X. Shi, and Y. Lou, "Distributional reinforcement learning with epistemic and aleatoric uncertainty estimation," *Information Sciences*, vol. 644, p. 119217, 05 2023.

[231] M. Valdenegro-Toro and D. Saromo, "A deeper look into aleatoric and epistemic uncertainty disentanglement," 2022.

[232] F. B. Smith, J. Kossen, E. Trollope, M. van der Wilk, A. Foster, and T. Rainforth, "Rethinking aleatoric and epistemic uncertainty," 2025.

[233] A. Sedlmeier, T. Gabor, T. Phan, L. Belzner, and C. Linnhoff-Popien, "Uncertainty-based out-of-distribution classification in deep reinforcement learning," in *Proceedings of the 12th International Conference on Agents and Artificial Intelligence*, SCITEPRESS - Science and Technology Publications, 2020.

[234] M. Teng, M. van de Panne, and F. Wood, "Exploration with multi-sample target values for distributional reinforcement learning," 2022.

[235] Y. Jiang, J. Z. Kolter, and R. Raileanu, "Uncertainty-driven exploration for generalization in reinforcement learning," in *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[236] B. Mavrin, H. Yao, L. Kong, K. Wu, and Y. Yu, "Distributional reinforcement learning for efficient exploration," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 4424–4434, PMLR, 09–15 Jun 2019.

[237] Y. Wu, W. Li, W. Huang, and C. P. Ho, "Drl-ora: Distributional reinforcement learning with online risk adaption," 2025.

[238] H. Eriksson, D. Basu, M. Alibeigi, and C. Dimitrakakis, "Sentinel: Taming uncertainty with ensemble-based distributional reinforcement learning," 2022.

[239] S. H. Lim and I. MALIK, "Distributional reinforcement learning for risk-sensitive policies," in *Advances in Neural Information Processing Systems* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), vol. 35, pp. 30977–30989, Curran Associates, Inc., 2022.

[240] OptiTrack, "Optitrack primex 22 motion capture camera," 2025. Accessed: 2025-11-03.

[241] KUKA, "Kr iontec industrial robot," 2025.

[242] L. Tarabini Castellani, J. Llorente, J. Ibarz, M. Ruiz, A. Mestreau-Garreau, A. Cropp, and A. Santovincenzo, "Proba-3 mission," *Int. J. of Space Science and Engineering*, vol. 1, pp. 349 – 366, 01 2013.

[243] A. Levy, R. P. Jr., and K. Saenko, "Hierarchical actor-critic," *CoRR*, vol. abs/1712.00948, 2017.

Appendix A

Algorithms

Algorithm 1 Hierarchical Actor-Critic [243]

Require: Number of levels n, episode limit M, subgoal testing flag B, replay buffers $\{R_0, \ldots, R_n\}$
  Initialize actor-critic networks $[(\pi_0, Q_0), \ldots, (\pi_n, Q_n)]$
  Initialize replay buffers $[R_0, \ldots, R_n]$
  for episode $= 1$ to M **do**
    Sample actual goal G and initial state s_0
    for $t_n = 1$ to T_n **do**
      Testing boolean $B \leftarrow \{0, 1\}$
      Sample layer n subgoal: $g_{t_n} \leftarrow \pi_n(s(t_n), G) + B \cdot \mathcal{N}(0, 1)$
      for $t_0 = 1$ to T_0 **do**
        Sample primitive action: $a_{t_0} \leftarrow \pi_0(s(t_0), g_1) + B \cdot \mathcal{N}(0, 1)$
        Execute a_{t_0}, observe r_{t_0+1} and s_{t_0+1}
        ProcessTrans$(s_{t_0}, a_{t_0}, r_{t_0+1}, s_{t_0+1}, g_{t_1})$
        if $g_{t_1}, \ldots, G$ achieved or $t_0 = T_0$ **then**
          Perform HER on layer 0 transitions
          break
        end if
      end for
      ProcessTrans$(s_{t_n}, g_{t_n}, r_{t_n+1}, s_{t_n+1}, G)$
      if G achieved or $t_n = T_n$ **then**
        Perform HER on layer n transitions
        break
      end if
    end for
    Update actor-critic networks
  end for

Algorithm 2 Transition Processing [243]

function ProcessTrans($s_{t_i}, a_{t_i}, r_{t_i+1}, s_{t_i+1}, g_{t_{i+1}}$)
  if slayer i is a subgoal layer **and** a_{t_i} not achieved **then**
    $R_i \leftarrow (s_{t_i}, g'_{t_i}, r_{t_i+1}, s_{t_i+1}, g_{t_{i+1}})$
    if Testing **then**
      $R_i \leftarrow (s_{t_i}, a_{t_i}, -T_i, s_{t_i+1}, g_{t_{i+1}})$
    end if
  else
    $R_i \leftarrow (s_{t_i}, a_{t_i}, r_{t_i+1}, s_{t_i+1}, g_{t_{i+1}})$
  end if
  if layer i is a subgoal layer **then**
    Store HER transition $(s_{t_i}, g'_{t_i}, \text{TBD}, s_{t_i+1}, \text{TBD})$
  else
    Store HER transition $(s_{t_i}, a_{t_i}, \text{TBD}, s_{t_i+1}, \text{TBD})$
  end if
end function

Algorithm 3 Uncertainty-Aware Implicit Quantile Network (UA-IQN)

Require: Replay buffer R, batch size B, number of quantiles N, target quantiles N', target update frequency C, discount factor γ, quantile threshold κ, variance sensitivity λ

Initialize online network parameters θ and target network parameters $\theta' \leftarrow \theta$

Initialize replay buffer R

for each training step **do**

Sample batch $\{(s, a, r, s')\}_{i=1}^{B}$ from R

Sample $\{\tau_i\}_{i=1}^{N}, \{\tau'_j\}_{j=1}^{N'} \sim \mathcal{U}[0, 1]$

for each s' in batch **do**

Compute $Z_{\theta'}(s', a', \tau'_j)$ for all $a' \in \mathcal{A}$ and τ'_j

Compute $Q(s', a') \approx \frac{1}{N'} \sum_{j=1}^{N'} Z_{\theta'}(s', a', \tau'_j)$

Compute $\mathrm{Var}(s', a') \approx \frac{1}{N'} \sum_{j=1}^{N'} \left(Z_{\theta'}(s', a', \tau'_j) - Q(s', a')\right)^2$

Select $a^* = \arg\max_{a'} \left[Q(s', a') - \lambda \cdot \mathrm{Var}(s', a')\right]$

end for

Compute target quantiles: $T(\tau'_j) = r + \gamma \cdot Z_{\theta'}(s', a^*, \tau'_j)$

Compute predicted quantiles: $Z_\theta(s, a, \tau_i)$

for each pair (τ_i, τ'_j) **do**

Compute $\delta_{ij} = T(\tau'_j) - Z_\theta(s, a, \tau_i)$

Compute quantile Huber loss:

$$\rho_{\tau_i}^{\kappa}(\delta_{ij}) = \begin{cases} |\tau_i - \mathbb{I}_{\{\delta_{ij}<0\}}| \, \frac{1}{2}\delta_{ij}^2, & \text{if } |\delta_{ij}| \le \kappa \\ |\tau_i - \mathbb{I}_{\{\delta_{ij}<0\}}| \, \kappa(|\delta_{ij}| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases}$$

end for

Compute total loss: $\mathcal{L}(\theta) = \frac{1}{BNN'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_{\tau_i}^{\kappa}(\delta_{ij})$

Perform gradient descent on $\mathcal{L}(\theta)$ and update θ

if step $\%$ $C = 0$ **then**

Update target network: $\theta' \leftarrow \theta$

end if

end for