UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2025-141

Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 19 December 2025 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Matteo El Hariry

Born on the 3rd of January 1994 in Cantù (Italy)

Deep Reinforcement Learning Control for Autonomous Robots Mobility in Highly Uncertain Environments

Affidavit / Statement of originality

I declare that this thesis:

- is the result of my own work. Any contribution from any other party, and any use of generative artificial intelligence technologies have been duly cited and acknowledged;
- is not substantially the same as any other that I have submitted, and;
- is not being concurrently submitted for a degree, diploma or other qualification at the University of Luxembourg or any other University or similar institution except as specified in the text.

With my approval I furthermore confirm the following:

- I have adhered to the rules set out in the University of Luxembourg's Code of Conduct and the Doctoral Education Agreement (DEA)[1], in particular with regard to Research Integrity.
- I have documented all methods, data, and processes truthfully and fully.
- I have mentioned all the significant contributors to the work.
- I am aware that the work may be screened electronically for originality.

I acknowledge that if any issues are raised regarding good research practices based on the review of the thesis, the examination may be postponed pending the outcome of any investigation of such issues. If a degree was conferred, any such subsequently discovered issues may result in the cancellation of the degree.

Approved on 2025-11-24

“The Analytical Engine has no pretensions to originate anything.

It can do whatever we know how to order it to perform.”

– Ada Lovelace

To my family and friends.

Index

List of Figures

List of Tables

Acronyms

API Application Programming Interface.

CaT Constraints as Terminations.

CFD Computational Fluid Dynamics.

CMDP Constrained Markov Decision Process.

CPU Central Processing Unit.

DoF Degrees of Freedom.

DRIFT Deep Reinforcement Learning for Intelligent Floating Platforms Trajectories.

DRL Deep Reinforcement Learning.

DVS Dynamic Vision Sensor.

EKF Extended Kalman Filter.

EV Event-based Vision.

FALCON-S Fixed-wing Aerodynamics and Learning Control Suite.

FoV Field of View.

FP Floating Platform.

GPU Graphics Processing Unit.

IMU Inertial Measurement Unit.

ISS International Space Station.

LEO Low Earth Orbit.

LQR Linear Quadratic Regulator.

LS Least Squares.

MBRL Model-Based Reinforcement Learning.

MDP Markov Decision Process.

MPPI Model Predictive Path Integral.

ODE Ordinary Differential Equation.

PID Proportional–Integral–Derivative.

PPO Proximal Policy Optimization.

RANS Reinforcement-learning-based Autonomous Navigating Spacecraft.

RL Reinforcement Learning.

RMS Root Mean Square.

RoboRAN A Unified Robotics Framework for Reinforcement Learning-Based Autonomous Navigation.

RW Reaction Wheel.

SOTA	State of the Art.
SRB	Space Robotics Bench.
ST	Star Tracker.
TD	Temporal Difference.
UGV	Unmanned Ground Vehicle.
UKF	Unscented Kalman Filter.
USV	Unmanned Surface Vehicle.
w.r.t.	with respect to.
WIG	Wing-In-Ground Effect.

Abstract

This thesis explores the use of deep reinforcement learning (DRL) for enabling robust, autonomous control of robotic systems operating in highly uncertain environments. Motivated by space applications and the need for generalizable learning pipelines, we develop a series of simulation frameworks and experimental platforms that progressively expand the scope, realism, and generalization capability of DRL-based controllers.

We begin by introducing GPU-accelerated simulation tools tailored to spacecraft-like dynamics (RANS), showing that physically grounded models and disturbance injection can yield transferable control policies. These findings are validated through DRIFT, a framework presenting an holonomic floating platform testbed where learned controllers achieve sub-centimeter trajectory tracking despite stochastic disturbances. Building on this, we propose RoboRAN, a modular IsaacLab-based framework that decouples robot and task specifications, enabling reproducible training across diverse platforms such as ground robots, USVs, and microgravity analogs. Sim-to-real evaluations confirm the framework's effectiveness for low-level policy transfer. Finally, FALCON-S broadens this research direction to fixed-wing platforms in ground-effect regimes by integrating a full 6-DoF aerodynamic model, actuator dynamics, and unified CPU–GPU backends. The framework accommodates both learning-based and classical control schemes, allowing systematic benchmarking, ablation studies, and cross-validation.

Together, these contributions demonstrate that DRL can be scaled, generalized, and validated across a range of robotic platforms, provided that simulation fidelity, modularity, and hardware alignment are preserved. Additional studies explore visual policy learning for spacecraft inspection and sensor-driven estimation for satellite angular dynamics, broadening the thesis impact. We conclude by outlining directions toward continual learning, sim-to-real-to-sim adaptation, and integrated world model architectures for real-world deployment.

Chapter 1

Introduction

Preface

The field of autonomous robotics has seen rapid advancements over the past decade, driven by progress in machine learning, computational hardware, and open-source simulation environments. Among various machine learning paradigms, Reinforcement Learning (RL) stands out as a promising tool for enabling agents to acquire control policies through interaction with their environment. RL's potential to solve complex sequential decision-making problems makes it particularly appealing for robotic systems operating in unstructured, dynamic, and uncertain environments.

However, deploying RL in real-world robotics remains non-trivial. The community has made significant strides in simulation-based performance, but a large gap persists between benchmark success and practical deployment. This thesis addresses this gap by focusing on generalizable and robust RL control for autonomous robots operating in highly uncertain environments—such as satellites, water vessels, ground and vehicles—through the development of scalable simulation tools, systematic evaluation frameworks, and real-world validation.

1.1 Motivation

Reinforcement Learning has emerged as a powerful paradigm for developing control policies for complex robotic systems. Its ability to learn directly from interaction, without requiring explicit models, makes it particularly appealing for domains where accurate modeling is difficult or where environmental uncertainty is dominant. In space robotics, maritime systems, and aerial vehicles, this adaptability is crucial. These systems often operate in high-dimensional, stochastic, and partially observable environments, with delayed or limited feedback. Despite the promise of RL, deploying such agents on physical platforms remains a significant challenge, particularly in scenarios with severe dynamics, limited sensing, and real-time constraints. The development of robust and generalizable RL agents is hindered by several factors:

(i) Poor transferability from simulation to real-world systems. Policies trained in idealized simulation environments often fail when exposed to the unmodeled dynamics, sensing noise, and disturbances present in the real world. This is known as the sim-to-real gap. It stems from the fact that simulators typically provide deterministic or low-noise feedback, whereas real environments are rife with stochasticity, delay, and partial observability [1, 2, 3]. For example, tactile-based robotic manipulation policies that perform well in simulation often degrade sharply in real-world tests unless heavily domain-randomized. Additionally, even accurate physics engines fail to model the nonlinearities of friction, compliance, or fluid-structure interactions, which can dominate certain tasks (e.g., underwater or aerial control).

(ii) Lack of standardization in benchmarking environments and evaluation protocols. A major bottleneck in the field stems from the limited standardization of RL benchmarks and evaluation protocols. The development of *OpenAI Gym* [4] marked a turning point by establishing a unified API and environment structure for training and evaluating RL agents. Gym became the de facto foundation for modern RL research, standardizing the interface between agents and environments and enabling reproducible comparisons across algorithms. However, most of its tasks—such as Atari [5] and MuJoCo locomotion environments [6]—were originally designed for algorithmic validation rather than embodied robotics.

Building upon Gym's success, a new generation of simulation frameworks emerged to bridge the gap between abstract tasks and physically realistic robotic scenarios. Among these, *Isaac Gym* [7] introduced GPU-accelerated physics for large-scale training, *Brax* [8] provided differentiable and parallelizable rigid-body dynamics, and *Habitat* [9] targeted embodied visual navigation and indoor mobility. While these toolkits significantly improved performance and realism, they differ substantially in physics fidelity, scalability, and supported robot morphologies. Moreover, few are coupled with real-world evaluation protocols or standardized metrics across robotic systems. As a result, reproducibility, comparability, and cross-domain generalization in robotics-focused RL remain open challenges.

(iii) Complexity of deploying RL agents across diverse robot morphologies. Real-world robotic systems differ substantially in actuation, sensing, dynamics, and failure modes. Transferring RL policies across such platforms is hindered by the need to carefully redesign observation and action spaces, tailor rewards, and re-tune network architectures and hyperparameters [10, 11]. For example, a policy trained to navigate a differential drive robot cannot be reused as-is for a holonomic mobile base, let alone a surface vehicle or drone. This lack of portability contrasts sharply with the generalization ambitions of RL. Moreover, RL libraries and frameworks (e.g., RLlib [12], SB3 [13], RL-Games [14]) often require significant customization for integration with robotic middleware such as ROS [15], and lack standardized wrappers or diagnostic tools for sim-to-real transfer.

Addressing these issues requires a structured and systematic approach—one that embraces simulation as a key enabler while also validating solutions in physical scenarios. In this thesis, we explore how to design scalable simulators, robust policy training strategies, and unified benchmarking tools to bridge the gap between reinforcement learning research and deployment in real-world autonomous robotic platforms.

The resulting methodologies are validated in four robotic domains that present a range of control and sim-to-real challenges:

1. **Spacecraft ground simulators (floating platforms):** These systems emulate planar spacecraft dynamics using air-bearing platforms constrained to frictionless two-dimensional motion. The platforms operate by generating a thin cushion of air between the robot base and the ground, allowing free-floating dynamics akin to microgravity conditions. Such setups are commonly used for attitude and orbit control prototyping in academic and industrial research [16, 17, 18]. In simulation, the dynamics are replicated using lightweight physics engines such as Isaac Gym [7], enabling high-throughput training of controllers under disturbances, actuator dynamics, and realistic sensing models. These platforms serve as valuable proxies for validating autonomous control strategies in space-relevant environments.

2. **Wheeled mobile robots (Turtlebot2):** Differential-drive wheeled robots like the Turtlebot2 are widely used in indoor robotic navigation and academic benchmarking. They exhibit non-holonomic dynamics and are typically equipped with low-cost sensors such as odometry, inertial measurement units, and monocular or depth cameras. These platforms enable testing of visual navigation and control policies in real-world cluttered environments [15, 19]. Their modular hardware and broad software support make them ideal for evaluating sim-to-real transfer under perception and actuation noise.

3. **Unmanned surface vehicles (USVs):** Surface vehicles such as the Kingfisher M200 represent over-actuated aquatic systems subject to complex hydrodynamics. These include drag, inertia, buoyancy, and environmental disturbances like wind and waves. Accurate simulation of such vehicles requires extending standard planar dynamics with water resistance and coupling models [20]. USVs offer a testbed for assessing generalization of control policies to domains with continuous drift and nonlinear damping, distinct from terrestrial or aerial robotics.

4. **Fixed-wing aerial vehicles in ground effect:** Fixed-wing vehicles flying close to the ground

experience nonlinear aerodynamic phenomena known as ground effect, where increased lift and reduced drag alter their dynamics. Simulating these behaviors necessitates specialized flight models, sometimes based on semi-empirical data or computational tools such as XPlane [21] or MATLAB Simulink [22]. Ground-effect vehicles pose unique challenges for control algorithms due to unstable dynamics and strong coupling—making them ideal for evaluating both classical and learning-based flight controllers.

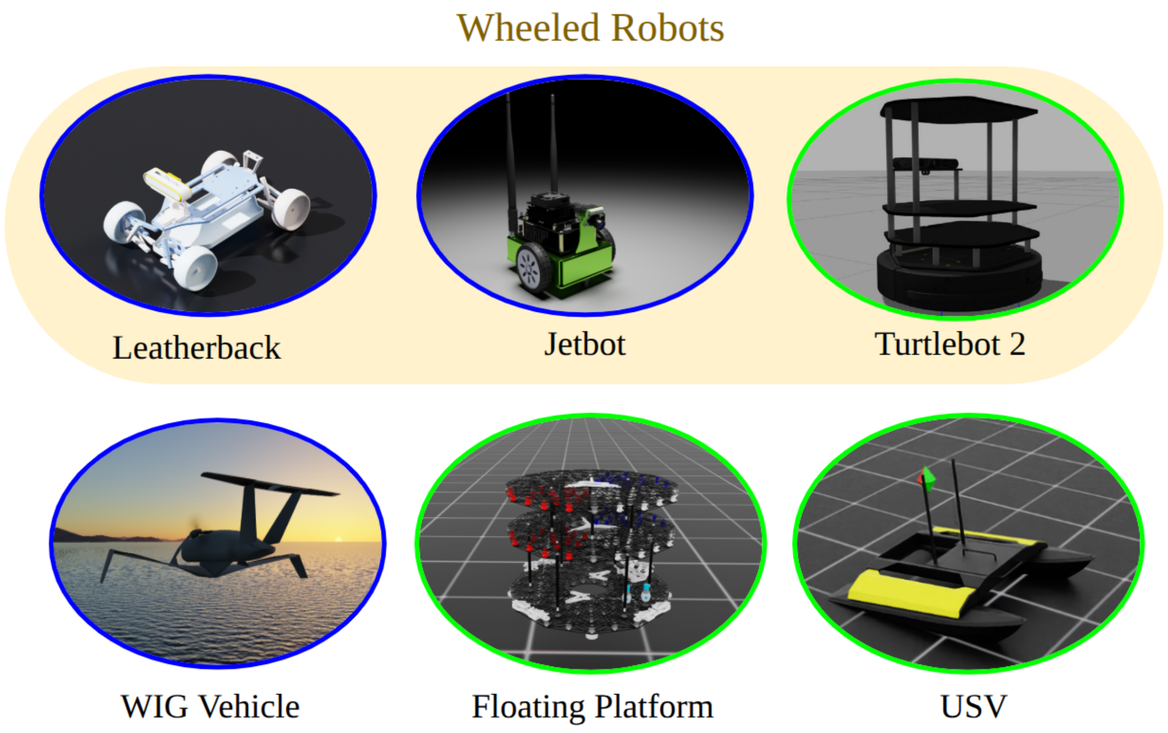


Figure 1.1: Robotic systems considered in this thesis. The five domains include spacecraft platforms (2D/3D), floating platforms (Zero-G Lab), wheeled robots (Turtlebot2), surface vehicles (USV), and fixed-wing aerial vehicles (WIG). Real-world deployments were performed on systems marked in green.

1.2 Research Scope and Objectives

This dissertation lies at the intersection of reinforcement learning, robotics, spacecrafts autonomy and simulation engineering. Its overarching aim is to develop scalable tools, simulation frameworks, and learning pipelines that enable RL-based control of autonomous robots operating in highly uncertain, dynamic, and partially observable environments. Rather than focusing on the development of novel RL algorithms, this work emphasizes the design of structured environments and evaluation protocols that support algorithmic generalization and deployment in real systems.

This choice of focus aligns with what Abel et al. [23] identify as the first foundational dogma of modern RL: the *environment spotlight*, depicted in Figure 1.2. This dogma critiques the field's traditional emphasis on environments and problem-solving over agents themselves. In this thesis, we embrace this environment-centric perspective—not as a limitation, but as a deliberate methodological stance. By engineering domain-randomized, physically grounded simulators and cross-domain benchmarks, we aim to construct the "stage" upon which diverse RL agents can be tested, improved, and transferred into reality. This work acknowledges that progress in RL for robotics requires not only richer, more representative and well-structured environments, but also smarter agents and novel algorithms and techniques. This latter topics are presented in the last chapter, where future work with focus on the Agent block of the RL loop is discussed.

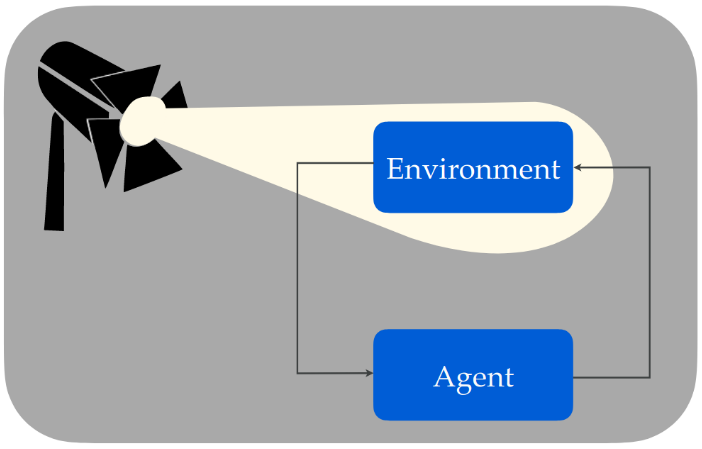


Figure 1.2: The environment spotlight from [23]: the field of RL has historically focused more on problem formulations and environments than on agent modeling. This thesis aligns with that perspective by emphasizing simulation infrastructure and task design.

The core intellectual pursuit of this work is driven by the following three high-level research questions:

Research Question 1

How can we design simulation frameworks that support scalable, physically realistic, and task-agnostic RL for autonomous robots?

Research Question 2

To what extent can reinforcement learning policies generalize across tasks, robots, and environmental conditions?

Research Question 3

Which techniques most effectively bridge the simulation–reality gap in uncertain environments?

To address these questions, the thesis pursues the following structured research objectives:

Research Objective 1

Develop modular and scalable simulation environments for reinforcement learning in robotics.

A fundamental objective of this thesis is to design and implement simulation environments that combine physical fidelity with large-scale computational efficiency. The simulators must support parallelized training, accurate dynamics modeling, and efficient GPU execution to enable thousands of environments to run in real time. Particular emphasis is placed on reproducing realistic actuation and sensing, incorporating latency, noise, and model uncertainty to better approximate real-world conditions. Each environment is structured with clear interfaces for observation, action, and reward functions, adhering to widely adopted standards such as OpenAI Gym [4], Isaac Gym [7] and Isaac Lab [24]. The resulting platforms provide a modular and extensible foundation for evaluating diverse control algorithms across multiple robotic domains, from spacecraft to ground and marine systems.

Research Objective 2

Train generalizable and robust policies through domain variation and algorithmic diversity.

This objective focuses on the development of learning pipelines that expose agents to structured variability in their training conditions. By combining techniques such as Domain Randomization (DR) [25] and Curriculum Learning [26], policies are trained to withstand uncertainty in system dynamics, sensory feedback, and environmental perturbations. The study includes comparative evaluation of different neural architectures—feedforward, recurrent (PPO-RNN), and latent world-model approaches (DreamerV3)—to assess how memory, abstraction, and temporal reasoning contribute to robustness. Training procedures are designed to promote generalization rather than overfitting to specific initial conditions, thus encouraging the emergence of adaptive and transferable control behaviors suitable for real-world operation.

Research Objective 3

Establish a unified training and evaluation framework for multi-robot and multi-task pipelines.

A central contribution of this thesis is the creation of a standardized software framework that enables systematic comparison of RL algorithms across different robot morphologies and task types. The framework integrates modular configuration, logging, and evaluation interfaces, ensuring reproducibility and scalability. It supports multi-robot training setups spanning micro-gravity platforms, wheeled and surface vehicles, and fixed-wing aircraft, as well as multiple task families such as point-to-point navigation, trajectory tracking, and pose regulation. By unifying data collection, training, and evaluation under a common protocol, this benchmark suite facilitates fair cross-algorithm analysis and accelerates the development of robust, transferable control policies. Moreover, it contributes to the broader reproducibility effort in reinforcement-learning-based robotics research.

Research Objective 4

Validate learned policies on real robotic platforms and rigorously assess sim-to-real transfer.

A last objective of this work is to evaluate the real-world applicability of policies trained in simulation. Selected control policies are deployed on physical systems introduced in Section 1.1, including air-bearing floating platforms and wheeled mobile robots. The validation process examines policy transferability across sensing noise, actuator delays, and unmodeled disturbances, providing an empirical measure of robustness. Comparative baselines based on classical control techniques (e.g., LQR) are used to contextualize the learning performance. This objective ultimately seeks to identify the conditions under which domain-randomized simulation yields transferable behaviors and to derive general insights for bridging the gap between synthetic and physical robotic environments.

1.3 Thesis Contributions

This thesis presents a research program advancing the use of Reinforcement Learning (RL) for robotic control in uncertain, dynamic, and partially observable environments. Its contributions span the design of simulation tools, training frameworks, and empirical studies across multiple robot morphologies and physical domains, rather than proposing new algorithms. Each contribution corresponds to a peer-reviewed publication or submitted manuscript and is unified by the shared objective of improving the scalability, generalization, and reproducibility of RL-based control in robotics. A visual overview is presented in Figure 1.3.

1. **RANS and DRIFT – Scalable Simulation for Generalizable RL Control of Spacecraft and Floating Platforms.** These works establish a foundation for scalable, domain randomized simulation of robotic systems that approximate space and planar microgravity dynamics. RANS introduces a high-performance parallel simulation pipeline for spacecraft

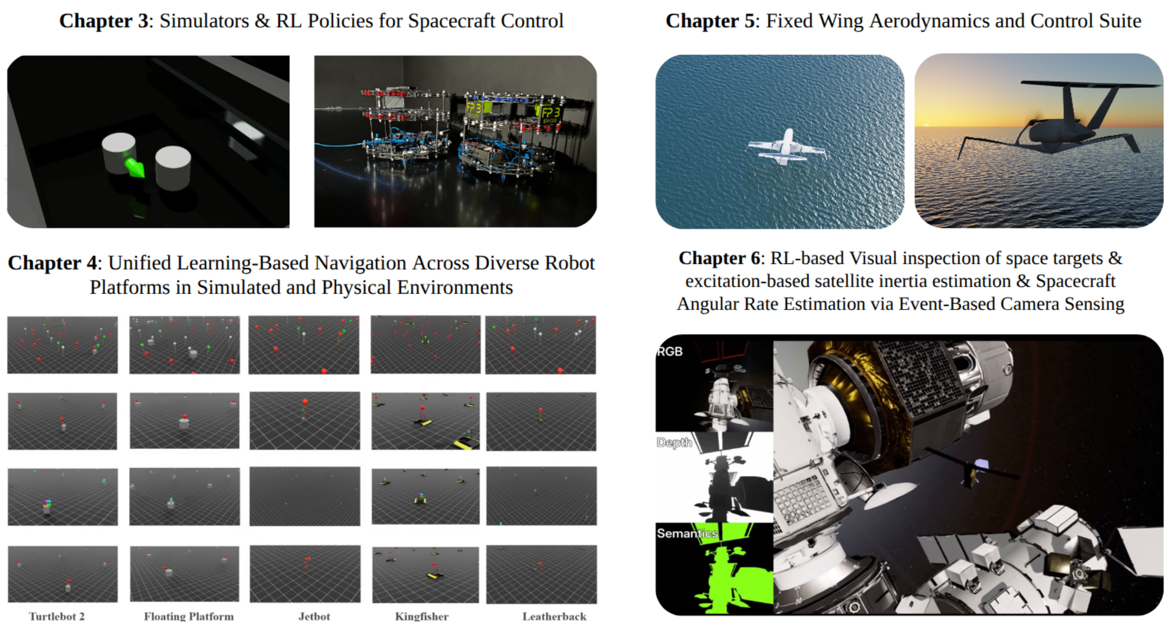

Chapter 3: Simulators & RL Policies for Spacecraft Control
Chapter 5: Fixed Wing Aerodynamics and Control Suite
Chapter 4: Unified Learning-Based Navigation Across Diverse Robot Platforms in Simulated and Physical Environments
Chapter 6: RL-based Visual inspection of space targets & excitation-based satellite inertia estimation & Spacecraft Angular Rate Estimation via Event-Based Camera Sensing
RGB
Depth
Semantics
Turtlebot 2
Floating Platform
Jetbot
Kingfisher
Leatherback

Figure 1.3: Overview of the thesis contributions and their mapping to dissertation chapters. The work evolves from simulation infrastructure (RANS, DRIFT [Chapter 3]) to benchmarking and multi-robot evaluation (RoboRAN, [Chapter 4]), domain-specific benchmarking for aerial vehicles (FALCON-S [Chapter 5]), and complementary sensing and estimation studies (Inertia-ID, RL-AVIST, Event-Based Inertia Estimation [Chapter 6]).

training, enabling large-scale data generation and multi-scenario evaluation. DRIFT extends this framework to real-world validation, demonstrating Deep RL control of a physical air-bearing floating platform under uncertainty and comparing it against optimal control baselines. Together, these contributions provide the first complete sim-to-real demonstration of DRL for near-frictionless robotic platforms and constitute a reproducible foundation for subsequent developments in this thesis.

2. **RoboRAN – A Unified Robotics Framework for Reinforcement Learning-Based Autonomous Navigation.** RoboRAN contributes a modular and extensible benchmarking suite that unifies training, logging, and evaluation across heterogeneous robots and navigation tasks. It provides standardized configuration, metric tracking, and reproducibility tools for RL research in robotics, integrating simulation back-ends and real-world interfaces.

The framework supports experiments on spacecraft analogs (floating platforms), wheeled robots, and surface vessels, enabling systematic assessment of robustness, generalization, and cross-domain transfer.

3. **FALCON-S – Fixed-Wing Aerodynamics and Learning Control Suite.** FALCON-S introduces the first benchmark suite dedicated to learning-based control of fixed-wing vehicles operating under ground-effect conditions. It integrates multiple simulation back-ends (Python-CPU, GPU-based Warp, MATLAB, and XPlane) with standardized APIs, making it possible to compare model-free (PPO), model-based (DreamerV3), and classical (LQR, MPPI) controllers on consistent aerodynamic models. The benchmark emphasizes reproducibility and extensibility, allowing comparative evaluation of control algorithms across various flight regimes, disturbances, and aircraft geometries.

4. **RL-AVIST: Reinforcement Learning for Autonomous Visual Inspection of Space Targets**: introduces a learning-based framework for 6-DOF proximity operations around large orbital assets. Built on the SpaceRobotics Bench [27], RL-AVIST employs model-based RL (DreamerV3) and model-free baselines (PPO [28], TD3 [29]) to train inspection policies under realistic spacecraft dynamics and visual feedback. This work demonstrates the potential of latent-model RL for efficient trajectory tracking, visual inspection, and multi-morphology generalization in orbital scenarios, marking a step toward perception-aware, long-duration autonomy in space operations.

5. **Complementary Contributions in Sensing and System Identification.** To complement the core simulation and control studies, two auxiliary works have been proposed to increase the realism and scope of the proposed frameworks:

   - *Active Excitation-Based Dynamic Inertia Identification in Satellites* [30]: develops a data-driven approach for estimating spacecraft inertia tensors using Least Squares and EKF filters, supporting simulator fidelity and model validation under realistic torque inputs.

- *Event-Based Angular Rate Estimation for Spacecraft* [31]: proposes a novel vision based angular velocity estimation method leveraging neuromorphic cameras observing star fields, extending sensing capabilities for future attitude-estimation pipelines.

Complete list of publications:

- ”RANS: Highly-Parallelised Simulator for Reinforcement Learning based Autonomous Navigating Spacecrafts.” Matteo El-Hariry , Antoine Richard, and Miguel A.O Mendez. *17th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA'23). 2023.*

- ”Drift: Deep reinforcement learning for intelligent floating platforms trajectories.” El-Hariry, Matteo, et al. *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024.*

- ”RoboRAN: A Unified Robotics Framework for Reinforcement Learning-Based Autonomous Navigation.” Matteo El-Hariry and Antoine Richard and Ricard M. Castan and Luis F. W. Batista and Matthieu Geist and Cedric Pradalier and Miguel Olivares-Mendez. 2025, arXiv, cs.RO. *Published in Transactions on Machine Learning Research (TMLR) (11/2025).*

- ”FALCON-S: Fixed-wing Aerodynamics and Learning Control Suite.”, Matteo El-Hariry, Pedro Lima, Andrej Orsula, Antoine Richard, Matthieu Geist, Miguel Olivares-Mendez. *Under review at – ICLR 2026 Conference.*

- ”RL-AVIST: Reinforcement Learning for Autonomous Visual Inspection of Space Targets.”. Matteo El-Hariry, Andrej Orsula, Matthieu Geist, Miguel Olivares-Mendez. *Internation Astronautical Congress (IAC) 2025, arXiv preprint arXiv:2510.22699 (2025).*

- ”Towards Active Excitation-Based Dynamic Inertia Identification in Satellites.”. El-Hariry, Matteo, Vittorio Franzese, and Miguel Olivares-Mendez. *Accepted at International Conference on Space Robotics (iSPaRo) 2025.* arXiv preprint arXiv:2510.16738 (2025).

- ”Spacecraft Angular Rate Estimation via Event-Based Camera Sensing”, Vittorio Franzese, Matteo El Hariry. *Under review at Advances in Space Research (ASR)*.

Chapter 2

Background and Related Work

This chapter establishes the context for the research conducted in this thesis, which lies at the intersection of deep reinforcement learning (DRL), simulation engineering, and robotic autonomy under uncertainty. It begins by introducing the core theoretical principles of reinforcement learning for continuous control, focusing on policy optimization methods and their extensions for handling partial observability and temporal dependencies. The chapter then examines the unique challenges posed by real-world robotic systems, including sample inefficiency, system diversity, sim-to-real mismatches, and safety-critical constraints. Building on this, we review the primary techniques used to bridge simulation and physical deployment, with emphasis on domain randomization, curriculum-based training, and memory-augmented policies.

Subsequently, the chapter surveys the landscape of simulation tools and environments that underpin DRL research in robotics, contrasting traditional physics engines with recent GPU-accelerated frameworks, and outlining the requirements that motivated the development of the custom simulation stacks presented in this thesis. A dedicated section then explores the specific robotic domains targeted in this work—ranging from floating platforms and spacecraft testbeds to wheeled robots and fixed-wing aircraft—with a focus on the dynamics, sensing, and control characteristics that distinguish each. Finally, the chapter discusses recent critiques and emerging methodological shifts in the RL community, particularly the emphasis on structure and agent design as articulated in the "Three Dogmas of RL". This synthesis helps situate the thesis contributions within a broader movement toward robust, scalable, and generalizable learning in robotics.

2.1 Reinforcement Learning for Continuous Control

Reinforcement Learning (RL) provides a mathematical framework for sequential decision-making, where an agent interacts with an environment in order to maximize cumulative reward. This framework is particularly well suited to robotics, where actions affect the system state over time, and control policies must account for delayed, noisy, or partial feedback. However, the application of RL to continuous control problems in robotics presents specific challenges—chief among them are high-dimensional state-action spaces, sample inefficiency, and sensitivity to hyperparameters and initialization.

2.1.1 The Markov Decision Process Formalism

The agent-environment interaction is commonly modeled as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is the set of possible states,
- $\mathcal{A}$ is the set of actions,
- $P(s'|s, a)$ is the transition probability distribution,
- $R(s, a)$ is the reward function,
- and $\gamma \in [0, 1)$ is the discount factor.

At each timestep t, the agent observes state s_t, selects an action $a_t \sim \pi(a_t|s_t)$ according to its policy π, and receives a scalar reward r_t while transitioning to a new state s_{t+1} (Figure 2.1). The goal is to learn a policy that maximizes the expected cumulative discounted reward:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right].$$

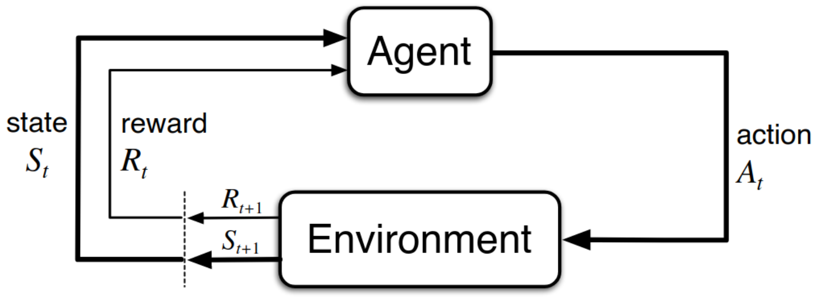


Figure 2.1: Standard agent–environment interaction loop in reinforcement learning [32].

2.1.2 Policy Optimization for Continuous Actions

Many real-world robotic control tasks require continuous actions—e.g., motor torques, thrust levels, or velocity commands—which cannot be effectively handled by discrete action methods like Q-learning [33] or Deep Q-Networks (DQN) [34]. As a result, modern deep reinforcement learning in robotics predominantly relies on *policy optimization* approaches, where a policy is directly learned to maximize expected cumulative rewards.

A widely adopted structure for this class of methods is the **actor–critic architecture** (Figure 2.2), which decouples the task of selecting actions from that of evaluating them. The *actor* is a parameterized policy $\pi_\theta(a|s)$ that outputs an action distribution conditioned on the current state. The *critic* is a value function (e.g., $V^\pi(s)$ or $Q^\pi(s, a)$) trained to estimate the expected return under the current policy. These two components work in tandem: the critic provides a learning signal—typically the advantage function or a temporal difference error—that guides the gradient updates of the actor.

This design has several advantages for robotics:

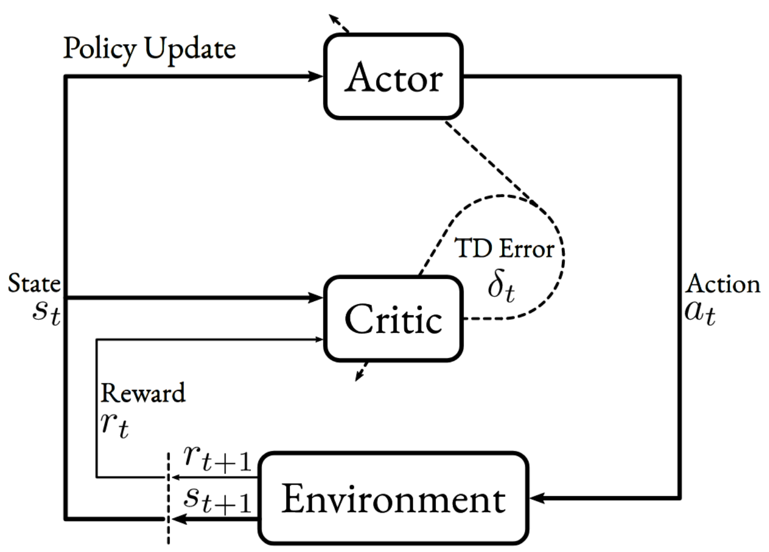


Figure 2.2: Actor–critic reinforcement learning loop. The actor updates its policy using feedback from the critic, which evaluates actions via the temporal-difference (TD) error derived from environment rewards and state transitions.

- It enables learning stochastic policies directly in continuous action spaces.
- It allows for low-variance updates through the use of bootstrapped value estimates.
- It is compatible with modular policy/value architectures tailored to different sensing and actuation modalities.

Among actor–critic methods, **policy gradient algorithms** have proven especially effective in continuous control settings. These algorithms compute an estimate of the gradient of the expected return $J(\pi_\theta)$ with respect to the policy parameters θ, and perform stochastic gradient ascent:

$$\nabla_\theta J(\pi_\theta) \approx \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \hat{A}_t \right],$$

where $\hat{A}_t$ is an estimate of the advantage of action a_t in state s_t under the current policy. This formulation enables end-to-end learning of both perception and control from raw or processed sensory inputs.

A widely used policy gradient method is **Proximal Policy Optimization (PPO)** [28], due to its ease of implementation, strong empirical performance, and stability across a wide range of domains. PPO introduces a trust region mechanism by clipping the policy update to prevent large deviations from the current policy. The surrogate objective function is:

$$\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the likelihood ratio between the new and old policies, and ϵ is a hyper-parameter controlling the maximum update step.

PPO forms the algorithmic backbone of all learning experiments in this thesis. It is used in its standard feedforward form, as well as in its recurrent variant (PPO-RNN), allowing policies to integrate temporal information across partial observations.

2.1.3 Handling Temporal Dependencies: PPO-RNN

Robotic systems often exhibit partial observability, delayed dynamics, or unmeasured disturbances. Standard feedforward policies struggle in such settings. To overcome this, recurrent architectures such as PPO-RNN extend PPO by incorporating memory via recurrent neural networks (RNNs, e.g., GRU or LSTM), enabling the policy to condition on a sequence of past observations:

$$a_t \sim \pi(a_t|o_{0:t}),$$

where $o_{0:t}$ represents the history of observations. This enhances robustness in real-world tasks, especially when sensors are noisy or state estimation is imperfect.

2.1.4 Model-Based RL and World Models

Model-based RL (MBRL) introduces, as shown in Figure 2.3 an explicit model of the environment, enabling planning or simulated rollouts to improve sample efficiency. While promising in theory, traditional MBRL has limited success in robotics due to the challenge of learning accurate transition models. Recent advances, such as **DreamerV3** [35], address this by learning compact latent dynamics models and training policies entirely in imagination.

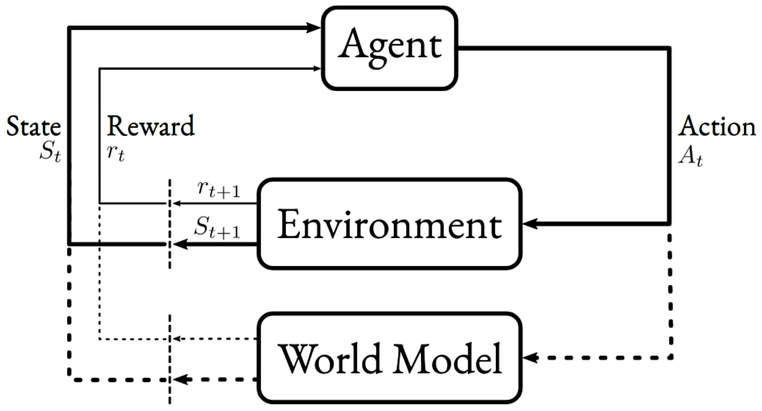


Figure 2.3: Model-based reinforcement learning loop. In addition to interacting with the environment, the agent learns a world model to simulate future states and rewards, supporting planning and policy optimization through imagined rollouts.

In this thesis, we incorporate Dreamer in the context of fixed-wing flight (FALCON-S benchmark), evaluating its performance against model-free baselines. Although still sensitive to hyperparameters and architecture choices, Dreamer represents a promising direction for efficient learning in physics-based domains.

2.2 Challenges for Real-World Reinforcement Learning

Despite the promise of deep reinforcement learning (DRL) in simulated domains, deploying such agents on real-world robotic systems remains a considerable challenge. Several bottlenecks hinder reliable transfer of learning-based control policies from virtual environments to physical platforms. This section highlights four critical challenges: (1) data efficiency, (2) simulation-reality mismatch, (3) task and morphology transferability, and (4) safe exploration, that are particularly relevant to the robotic domains addressed in this thesis. Each subsection presents the challenge and discusses practical mitigation strategies adopted throughout the thesis.

2.2.1 Data Efficiency and Exploration Costs

In contrast to game-like environments or purely virtual control problems, real-world robotic systems face hard constraints on the amount and quality of data that can be collected. Physical trials are slow, expensive, and subject to wear-and-tear. Safety concerns and limited availability of the hardware further restrict the number of interactions that can be executed. As a result, algorithms that require millions of environment steps, such as standard model-free DRL methods, are often infeasible for direct deployment.

Even in simulation, where data is cheap, many robotic environments involve complex dynamics and sparse rewards, exacerbating exploration difficulties. This motivates the use of parallelized simulation frameworks (e.g., RANS [36]) and curriculum learning strategies to mitigate sample inefficiency. Techniques such as reward shaping, parameterized initial conditions, and dense auxiliary objectives can further reduce training time and improve policy convergence.

Example: Data Bottlenecks in Air-Bearing Platforms

In the Zero-G Lab [16] floating platform testbed, each real-world episode requires physical setup, safety checks, and actuator reset, taking several minutes per trial. This makes iterative learning in-the-loop impractical and highlights the need for scalable simulation-first pipelines.

Curriculum Learning for Sample Efficiency. One of the strategies explored in this thesis to mitigate sample complexity is curriculum learning, gradually increasing task difficulty during training, as shown in Figure 2.4. This is especially effective in sparse-reward settings or under complex failure modes. By first training in simplified conditions and progressively introducing variability (e.g., more noise, tighter targets), convergence is improved and unsafe behavior in early stages is avoided.

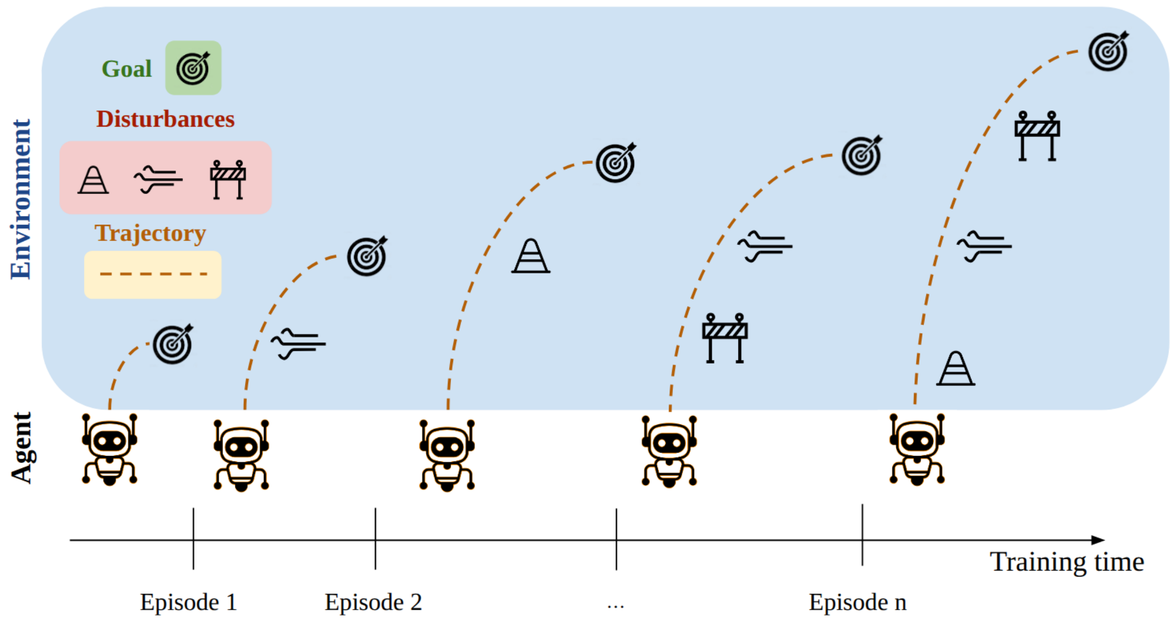


Figure 2.4: Illustrative curriculum progression: training starts in simplified conditions (short goal distances, no drift) and gradually increases complexity over time by introducing disturbances in the environment (e.g., actuator noise, wind, tighter tolerances, obstacles).

2.2.2 Simulation–Reality Mismatch and Dynamics Uncertainty

A core obstacle to sim-to-real transfer is the mismatch between modeled and real-world dynamics. Simulation often fails to capture the full complexity of sensors (e.g., latency, noise, occlusions), actuators (e.g., backlash, delay, saturation), and environmental interactions (e.g., ground effect, friction asymmetries). These discrepancies result in policies that overfit to idealized physics and perform poorly when deployed.

To address this, several strategies have been proposed:

- **Domain Randomization (DR)**: Introduces variability in environment parameters during training (e.g., mass, damping, latency) to encourage robustness [25].
- **Observation Corruptions**: Models sensor noise and delay stochastically to simulate degraded sensing conditions [37].

- **Stochastic Actuator Models**: Adds delay, saturation, or variability to control inputs, better matching physical actuators [38].

Despite these measures, there remains no guarantee of successful deployment, and most policies require careful tuning to bridge the residual sim-to-real gap. In this thesis, we empirically evaluate DRL controllers under various mismatch conditions across floating platforms, aquatic vehicles, and mobile robots.

2.2.3 Morphology and Task Transferability

RL policies are typically trained for a single robot, task, or domain. However, in practical robotics, it is often desirable to re-use or fine-tune policies across platforms with different morphologies (e.g., from floating to wheeled to aquatic robots) or across related tasks (e.g., point navigation to velocity tracking). Transfer learning in this context remains difficult due to several factors:

- Inconsistent observation and action spaces across robots.
- Differing control frequencies, delays, and dynamics regimes.
- Divergent reward function definitions and task formulations.

Even when high-level policies or encoders are shared, the inductive biases of each morphology require significant adaptation. Frameworks like RoboRAN [39] attempt to mitigate this by defining a common structure and logging interface, enabling modular training pipelines and evaluation across morphologies. Still, full generalization remains elusive without retraining.

Challenge: Reusing RL Policies Across Robots

Policies trained on the Turtlebot2 may fail when applied to USVs or floating platforms due to mismatched inertia, drift, and heading dynamics. Despite shared tasks (e.g., point navigation), the control strategies must adapt to morphology-specific disturbances.

Architectural Support for Transfer. One approach used in this thesis is to train shared policy architectures (e.g., PPO with MLP encoders) across robot types using aligned state-action spaces and task encodings. Additionally, training with morphology-specific domain randomization helps expose agents to wider dynamics, enabling partial transfer.

2.2.4 Safety and Constraint Encoding

Unlike in simulation, mistakes in real-world RL carry physical risks of damaging hardware, violating safety constraints, or entering irrecoverable states. Standard RL algorithms do not natively account for these risks. Instead, safety must be enforced through:

- **Reward shaping**: Penalizing unsafe behaviors (e.g., high angular velocities or collisions) during training.
- **Action clipping or bounds**: Hard constraints on actuation to prevent over-control.
- **Control-theoretic prefilters**: Adding stability layers outside the learned policy.
- **Constrained RL or shields**: Explicit incorporation of constraints in the learning objective (not adopted in this thesis).

A critical difficulty is that many constraints, such as actuator limitations or minimum safety margins, are discontinuous, hard to model, or not differentiable. While constrained RL [40, 41, 42, 43] is an active research area, the methods used in this thesis rely instead on structured training curricula, domain randomization, and reward design to indirectly promote safer behaviors. Empirical tests are performed under realistic disturbances and sensor noise to assess robustness post-training.

Summary: When Does Sim-to-Real Transfer Work?

While sim-to-real transfer remains an open problem, the findings of this thesis suggest that:

- Domain randomization and stochastic modeling significantly improve robustness when properly calibrated.
- Curriculum learning reduces early training instabilities and improves convergence in complex tasks.
- Transferability across robot morphologies remains difficult without re-training, despite modular architectures.

Takeaway

Robust sim-to-real transfer is achievable when structured training (DR, curriculum), architectural support, and task-aware simulation design are combined. However, generalization across robots or tasks still requires significant effort, motivating the need for unified frameworks like RoboRAN.

2.3 Simulation Frameworks for RL in Robotics

Simulation is a critical enabler for reinforcement learning (RL) in robotics. It allows safe, reproducible, and accelerated training of control policies that would otherwise be impractical or risky to develop directly on hardware. However, the design of such simulators presents a trade-off between physical fidelity, computational performance, and scalability. This section reviews the evolution of simulation tools in robotics and describes the custom frameworks developed in this thesis to support large-scale, generalizable RL.

2.3.1 Traditional vs GPU-Accelerated Simulators

Early simulators such as Gazebo [44], PyBullet [45], and V-REP [46] prioritized realism and extensibility for robotic prototyping. However, their CPU-bound architecture limited the number of concurrent environments that could be simulated, often leading to prohibitively long training times in DRL applications. While accurate and modular, these tools were not originally designed for thousands of parallel rollouts, making them ill-suited for modern data-hungry policy gradient methods.

To address this, recent efforts have introduced GPU-accelerated simulators such as Isaac Gym [7] (now Isaac Lab [24]), Warp [47], Brax [8], and MuJoCo [6] with GPU backends. These frameworks exploit massively parallel physics kernels to simulate hundreds to thousands of environments simultaneously, drastically reducing training wall time. In particular, **Isaac Lab** and **Warp** support tensor-based simulation of rigid body dynamics and are tightly integrated with PyTorch, allowing for seamless GPU-to-GPU data pipelines during RL training.

Nonetheless, GPU-based simulators come with trade-offs: they often require simplified contact models, limited sensor realism, and manual tuning of timestep stability. This is primarily because high-throughput simulation on GPUs prioritizes vectorized computation and rigid scheduling over detailed physics fidelity. For example, soft contacts, frictional instabilities, and sensor latency are computationally expensive to model accurately in parallelized GPU pipelines, which are optimized for deterministic, synchronous updates across thousands of environments. This motivates hybrid architectures and modular extensions to better balance realism with speed—e.g., using simplified GPU simulation for training and switching to CPU-based or high-fidelity simulators (e.g., Isaac Sim, X-Plane) for validation and fine-tuning.

2.3.2 Custom Simulation Tools Developed in this Thesis

This thesis contributes three modular and scalable simulation tools, each tailored to a specific robotic class and research objective:

(1) RANS and DRIFT: Parallel Simulation for Floating Platforms and Spacecraft Control

- **RANS** [36] is a highly-parallelized environment suite for spacecraft-relevant planar motion (x, y, yaw), designed to support domain-randomized training of DRL agents on floating platforms.

- **DRIFT** [37] builds on RANS by integrating actuator failure modes, observation corruptions, and curriculum learning for progressive training across difficulty levels.
- Both simulators integrate with Isaac Gym and IsaacLab [24] backends and support transfer to the physical air-bearing platform used at the Zero-G Lab [16].

(2) RoboRAN: Multi-Robot and Multi-task Sim-to-Real Framework

- **RoboRAN** [39] enables training and evaluation across heterogeneous robots (floating platform, Turtlebot2, Kingfisher USV) and task types (GoToPose, trajectory following).
- It includes shared configuration schemas, observation/action space adapters, and task-agnostic logging for reproducible benchmarking.
- The modular design ensures that new robots or tasks can be added with minimal effort, promoting generalization studies across morphologies.

(3) FALCON-S: Fixed-Wing Aerodynamics and Ground Effect Simulator

- **FALCON-S** [38] is a lightweight, extensible benchmark for multi-aircraft designs and modular physics simulation. It includes:
  - A CPU-based physics core (Python, Warp) for scalable DRL training.
  - MATLAB and X-Plane simulators for validation and high-fidelity comparisons.
  - PID, LQR, PPO, PPO-RNN, and DreamerV3 controllers for baseline evaluation.
- The suite provides an end-to-end pipeline for evaluating learning and classical control strategies on nonlinear flight dynamics.

2.3.3 The Need for Scalable, Structured Benchmarks

As DRL for robotics matures, the field increasingly demands standardized, structured, and scalable benchmarks that go beyond single-task, single-robot training. Unlike game environments or locomotion tasks, real-world robotics involves diverse sensing modalities, control regimes, and failure modes—requiring modular simulation tools and consistent evaluation pipelines.

Existing efforts such as OpenAI Gym [4], RLBench [48], Habitat [9], and IsaacLab [24] each offer valuable abstractions, but are limited in their ability to cover:

- Heterogeneous robot morphologies with shared evaluation logic.
- Realistic actuator and sensor models for sim-to-real research.
- Multi-task curricula with systematic difficulty variation.

To this end, our frameworks (especially RoboRAN and FALCON-S) address a key research gap by providing reusable environments, configurable agent wrappers, and cross-domain evaluation tools. They allow benchmarking DRL policies on equal footing with classical controllers and support structured experiments on generalization, transferability, and robustness.

2.4 Robotic Control Domains Addressed in this Thesis

This thesis investigates reinforcement learning for autonomous control across a diverse set of robotic platforms. Each domain—spacecraft-inspired floating systems, ground and surface robots, and fixed-wing aerial vehicles—poses distinct modeling, control, and transfer challenges. The selection of these domains reflects both their practical relevance and their role as structured testbeds for evaluating DRL generalization under uncertainty.

2.4.1 Spacecraft Simulators and Floating Platforms

Simulating spacecraft dynamics for learning-based control presents a unique opportunity: the dynamics are relatively clean (few contact discontinuities), yet highly sensitive to model uncertainty, actuator delays, and observation noise. Real-world testing is infeasible in microgravity, but planar analogues can emulate key aspects.

In this thesis, we use floating platforms—air-bearing systems that move frictionlessly on a planar surface—to replicate 2D spacecraft dynamics. These platforms exhibit near-inertial motion, are actuated by discrete on-off thrusters, and feature drift-dominant behavior with minimal passive damping. Control must account for long time constants, slow convergence, and frequent actuator saturation.

Simulation is performed using custom rigid-body dynamics integrated into Isaac Gym and later IsaacLab, with extensions to support:

- **Binary and low-thrust actuation models** with time delay and saturation.
- **Noisy sensing** for pose, velocity, and angular rates.
- **Task variability**, including point-to-point, pose alignment, and trajectory following.

Real-world experiments are conducted on the Zero-G Lab air-bearing platform [16], validating PPO and LQR controllers trained entirely in simulation. These experiments demonstrate successful transfer under actuator faults, drift, and delayed sensing.

2.4.2 Ground and Surface Robots (Wheeled and USV)

Terrestrial and marine robots present a complementary challenge: although easier to deploy and instrument, they feature heterogeneous dynamics, noisy proprioception, and varying actuation strategies. In this work, two types of robots are considered:

Wheeled Robots. We use the Turtlebot2 as a representative differential-drive robot with non-holonomic constraints and odometry-based sensing. Policies are trained in simulation and executed onboard the physical robot using the RoboRAN framework. Additional wheeled robots (JetBot, Leatherback) are tested in sim to verify morphology generalization. Navigation tasks include:

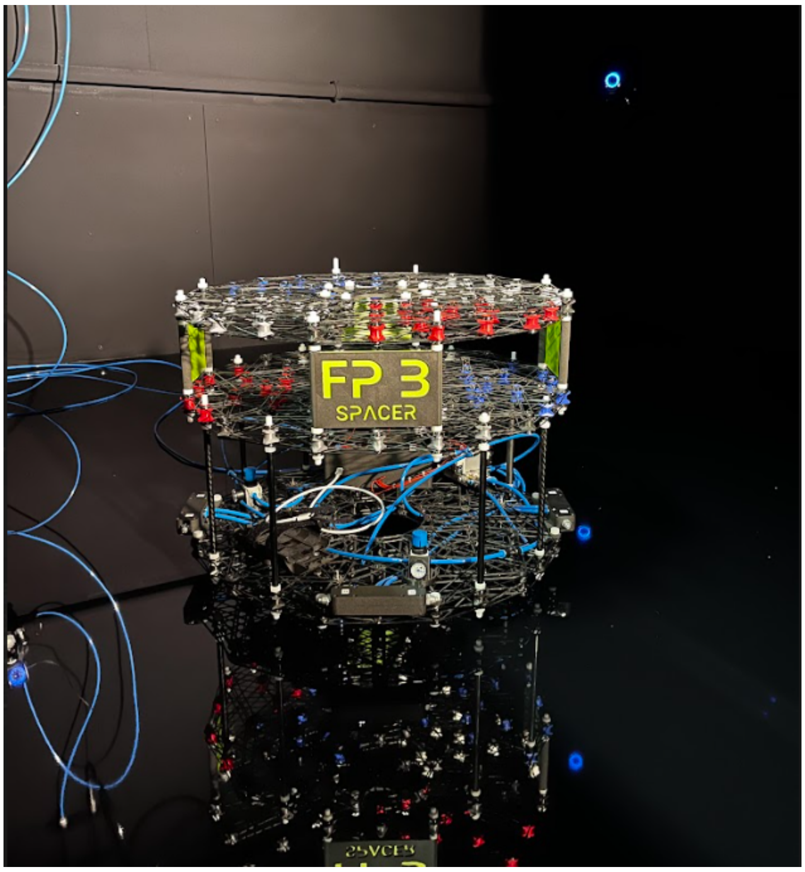


Figure 2.5: Photo of the floating platform system: 8 binary thrusters enable planar maneuvering under drag and inertia.

- **Go-To-Pose:** Reaching a specific target position and orientation.
- **Trajectory Tracking:** Following a predefined path in clutter-free environments.

Unmanned Surface Vehicles (USV). The Kingfisher USV is a twin-hull vessel with high inertia and hydrodynamic coupling. It is simulated using custom planar dynamics with added buoyancy, damping, and wind-like perturbations. Control tasks include waypoint tracking under disturbance and drift.

Cross-Robot Policy Evaluation

The RoboRAN framework enables training policies that are portable across floating, wheeled, and aquatic platforms by standardizing task interfaces and logging. However, significant domain-specific tuning remains necessary to achieve robust generalization.

2.4.3 Fixed-Wing Vehicles in Ground-Effect

Fixed-wing aerial vehicles operating near the ground exhibit highly nonlinear, coupled dynamics due to the ground effect phenomenon, which modifies lift and drag characteristics. These effects are hard to model and pose challenges for both classical and learning-based control.

The FALCON-S benchmark introduced in this thesis provides the first structured suite for RL in ground-effect flight. It supports multiple simulation backends:

- **Python and Warp-based rigid body dynamics** for fast training.
- **MATLAB Simulink and XPlane** for high-fidelity validation.

Control policies are evaluated on:

- **Altitude keeping** near the ground plane.
- **Go-to-position** tasks under wind perturbations.
- **Flight path stabilization** across vehicle geometries.

Both model-free (PPO, PPO-RNN, Dreamer) and model-based (LQR, MPPI) controllers are tested. Ground-effect interaction is randomized to promote robustness. This domain showcases the challenge of partial observability and the benefit of recurrent policies and domain randomization.

2.5 Trends and Critiques in RL for Robotics

The field of reinforcement learning (RL) has undergone a significant shift in recent years, moving from purely theoretical formulations toward large-scale experimentation, real-world deployment, and critical introspection. In the context of robotics, this has led to several emerging themes: a rethinking of agent-environment abstraction, a focus on structure and priors over brute-force optimization, and a broader reflection on the scientific framing of RL.

2.5.1 The "Three Dogmas" and Environment-Centric Design

In their influential position paper, de Masi et al. [23] identify three "dogmas" that have constrained progress in RL:

1. **Tabula Rasa Learning**: Agents are expected to learn from scratch in each environment, ignoring past experience or structure.
2. **Scalar Reward Maximization**: Learning is driven by a single reward signal, often sparse or misaligned with the task.
3. **Episodic Learning**: The MDP formulation assumes well-defined resets and episodic returns, which do not reflect real-world operation.

A central theme in this critique is the overemphasis on the environment, which has led to what the authors term the **environment spotlight**. Most RL systems are designed around a fixed environment structure, treating the agent as a passive optimizer rather than a grounded, adaptive entity.

This thesis embraces this critique by shifting focus toward the *design of environments, simulators, and evaluation suites*. Rather than proposing new RL algorithms, we emphasize:

- Carefully constructed simulation pipelines.
- Realistic physical modeling and sensor/actuator noise.

- Transferable training curricula.
- Modular task-agent interfaces.

These decisions aim to support generalizable policy learning under realistic constraints, grounded in the actual needs of robotic deployment.

2.5.2 Structure, Priors, and Generalization over Optimization

As RL systems scale up, purely reward-driven optimization becomes less practical. In robotics, rewards are often sparse, safety constraints are hard to encode, and environment resets are infeasible. Recent research trends reflect a growing emphasis on *structure and inductive priors*, including:

- **Architectural priors:** Using recurrent networks [49], residual policies [50], or world models [35] to bias the learning process.
- **Curriculum and task design:** Structuring the learning process with gradually increasing complexity [26].
- **Multi-task and transfer setups:** Sharing representations or policies across tasks and morphologies [1].

These methods often outperform "pure" optimization when generalization is required. In this thesis, these insights are adopted in the form of:

- Domain-randomized simulation environments.
- Recurrent policy architectures (e.g., PPO-RNN).
- Multi-robot benchmarking frameworks (e.g., RoboRAN).

Rather than optimizing each agent-environment pair in isolation, we prioritize *designing systems that generalize across settings*: a goal that aligns with the broader shift in RL research.

2.5.3 Positioning This Thesis in the Evolving Landscape

This dissertation responds to the field's current challenges not by contributing a new RL algorithm, but by structuring the conditions under which existing algorithms can succeed in robotics. Specifically, it addresses the need for:

- **Structured simulation** to support efficient and robust learning pipelines.
- **Unified benchmarks** for evaluation across robot morphologies and tasks.
- **Physical deployment** as a validation step for real-world applicability.

The systems studied span spacecraft emulators, surface and ground robots, and fixed-wing aerial vehicles—each serving as a testbed for evaluating generalization, robustness, and transferability. Across all domains, the thesis emphasizes robust agent-environment co-design and realistic training signals, reflecting a pragmatic and grounded response to the theoretical critiques of RL.

2.6 Summary

This chapter has surveyed the theoretical foundations, practical challenges, and recent critiques of reinforcement learning for real-world robotics. We began by introducing the policy optimization framework for continuous control and discussed the limitations of current DRL methods when applied to physical systems. The chapter then addressed the key bottlenecks—data inefficiency, simulation mismatch, transferability, and safety—and reviewed strategies to mitigate them, such as domain randomization and curriculum design.

We also introduced the simulation tools developed in this thesis and outlined the three robotic control domains investigated. Finally, we positioned this work in the broader RL discourse, drawing from recent critiques such as the Three Dogmas of RL to argue for environment- and structure-aware learning systems.

The next chapters present the core contributions of the thesis, detailing simulation frameworks, benchmark design, and empirical results across multiple robotic platforms.

Chapter 3

Generalizable RL for spacecrafts ground simulators

3.1 Motivation and Scope

Autonomous navigation and control systems are essential for the success and resilience of future space missions. As spacecraft become increasingly compact, distributed, and autonomous, the need for robust guidance, navigation, and control (GNC) algorithms grows accordingly. This trend is particularly evident in the rise of nano- and micro-satellites [51, 52], which impose stringent constraints on onboard power, actuation, and computation [53]. These platforms must be able to make independent decisions under uncertainty, without human oversight, especially in remote or delay-sensitive space scenarios.

The traditional approaches to spacecraft control typically rely on optimal control and model-based schemes [53], which, while powerful, often assume idealized conditions and require accurate modeling of the system dynamics. Moreover, these techniques tend to operate in open-loop or assume predefined flight trajectories, which limits their ability to adapt to changing mission requirements or external disturbances. When applied to real-world space robotics applications—where sensing noise, actuator faults, and partial observability are common—these limitations hinder reliability and performance.

Meanwhile, the recent rise of artificial intelligence (AI), deep learning (DL), and reinforcement learning (RL) in terrestrial robotics has opened promising avenues for space applications as well [54]. Reinforcement learning, in particular, has demonstrated the potential to autonomously learn complex control policies through interaction, without requiring hand-crafted models. Relevant applications include planetary landing [55], spacecraft trajectory planning in unknown gravitational fields [56], terrain navigation [57], and mapping during orbital operations [58].

Despite these advances, the deployment of RL in space systems presents substantial challenges:

- RL policies require extensive training data—unavailable on physical spacecraft—and are difficult to test safely.
- Existing simulation tools for space applications (e.g., GMAT [59], SPICE [60], Trajectory Browser [61]) are not designed to support RL-style training.
- Modern physics simulators for robotics (e.g., Gazebo [62], MuJoCo [6], Webots [63], Isaac Sim [24]) often focus on ground-based or articulated robots and lack support for thrust-based spacecraft dynamics.

To address these gaps, this thesis presents two core contributions:

1. **RANS** [36], an open-source GPU-parallel simulator for reinforcement learning-based spacecraft control. RANS allows fast training of RL agents for 2D and 3D free-floating vehicles using realistic force-based dynamics, fault profiles, and domain randomization.
2. **DRIFT** [37], a structured extension of RANS enabling real-world policy deployment on floating air-bearing platforms. DRIFT introduces more complex evaluation scenarios, supports real-world sensing and actuation profiles, and allows benchmarking of RL versus classical optimal control methods (e.g., LQR).

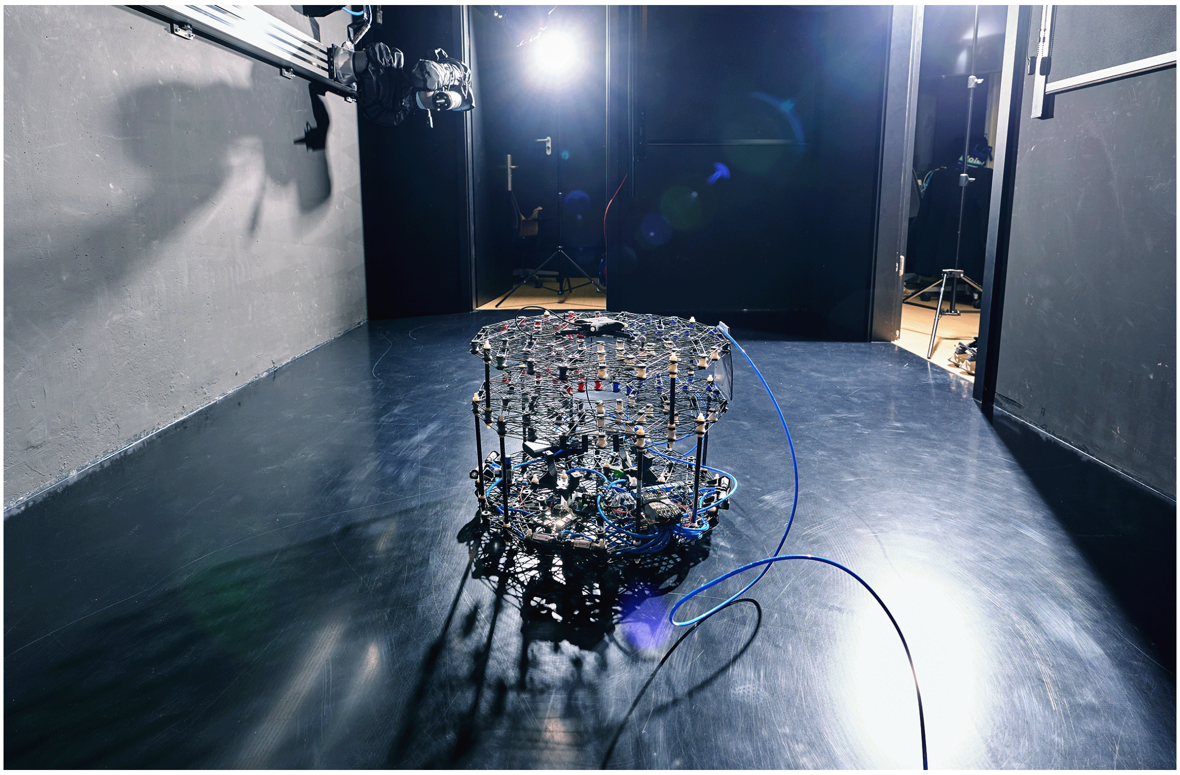


Figure 3.1: Floating platform testbed used in this thesis, located at the Zero-G Lab. It enables validation of RL policies in real-world microgravity-like environments using 3DoF thrust-actuated planar motion.

Together, RANS and DRIFT form a two-stage simulation-and-deployment pipeline for spacecraft-like platforms. While RANS focuses on fast, scalable training with generalization across domains, DRIFT targets physical deployment with realistic system modeling and policy robustness under uncertainty. Figure 3.1 shows the floating platform used for the sim-to-real validation.

This chapter presents both frameworks and their validation across multiple tasks: pose-reaching, velocity tracking, fault recovery, and robustness testing. We compare DRL policies with classical baselines and highlight key factors influencing sim-to-real transfer performance. These contributions aim to bridge the current gap between RL research and the development of practical autonomous mobility systems in space.

3.2 Related Work

3.2.1 Simulation Tools for Space Robotics

Several general-purpose mission planning and simulation tools have been developed to support spacecraft navigation. These include:

- **GMAT** [59], an open-source tool for orbit dynamics and trajectory planning.
- **SPICE** [60], used for science data analysis and spacecraft ephemeris management.
- **Trajectory Browser** [61], which allows multi-body trajectory optimization.

However, none of these frameworks are designed for RL-based control or interactive learning. They assume predefined system models and are not compatible with modern deep learning pipelines.

On the other hand, popular physics simulators for terrestrial robotics—such as Gazebo [62], MuJoCo [6], PyBullet [45], and Isaac Sim [24]—offer plug-and-play tools for RL experimentation. Notably, Isaac Sim supports GPU acceleration and universal scene description (USD) files, making it highly extensible. Still, most existing environments target manipulation or ground robots. None of them extends to thrust-based control or spacecraft analogs.

RANS and DRIFT address this gap by providing fully parallelized and spacecraft-specific RL simulation tools, integrating domain randomization, fault injection, and real-world transfer pipelines.

3.2.2 Floating Platforms and Air-Bearing Systems

Floating platforms provide a practical way to emulate microgravity conditions on Earth. Air bearings reduce planar friction and allow free-floating 3DoF motion [64]. These systems have been widely adopted for rendezvous, docking, and spacecraft servicing research [65, 66, 67].

Recent innovations have introduced vision-based sensing [65], dual-arm manipulation [68], and more robust thruster actuation [69]. However, most control approaches remain grounded in trajectory optimization or PID-like strategies. Few works explore fully learned policies.

3.2.3 RL for Spacecraft and Floating Platform Control

The use of RL for spacecraft control remains limited, especially in real-world settings. Most works report simulation-only results [55, 70, 56]. The few real-world studies—e.g., [71, 72]—use hybrid setups, where RL provides a reference signal to a classical controller. Our work differs in two ways:

- We deploy a fully learned RL policy that directly produces low-level control commands.
- We introduce a structured evaluation methodology for comparing learned vs classical control across both simulation and real-world trials.

To the best of our knowledge, this is the first open-source framework to systematically train, deploy, and benchmark RL spacecraft controllers on a floating platform in real-world experiments.

3.3 RANS: Highly-Parallelised Simulator for Reinforcement learning based Autonomous Navigating Spacecrafts

To enable the training and validation of RL-based guidance and control policies for free-flying spacecraft systems, we introduce **RANS**—a GPU-accelerated simulator built on Isaac Gym [7] designed specifically for thrust-based spacecraft dynamics. RANS is developed to bridge the gap between existing space mission design tools and modern deep reinforcement learning frameworks, providing a scalable and modular environment for training RL agents on planar and 3D navigation tasks.

Unlike traditional mission simulators such as GMAT [59] or SPICE [60], RANS supports direct control of low-level thrust vectors, realistic force dynamics, and randomized physical param-

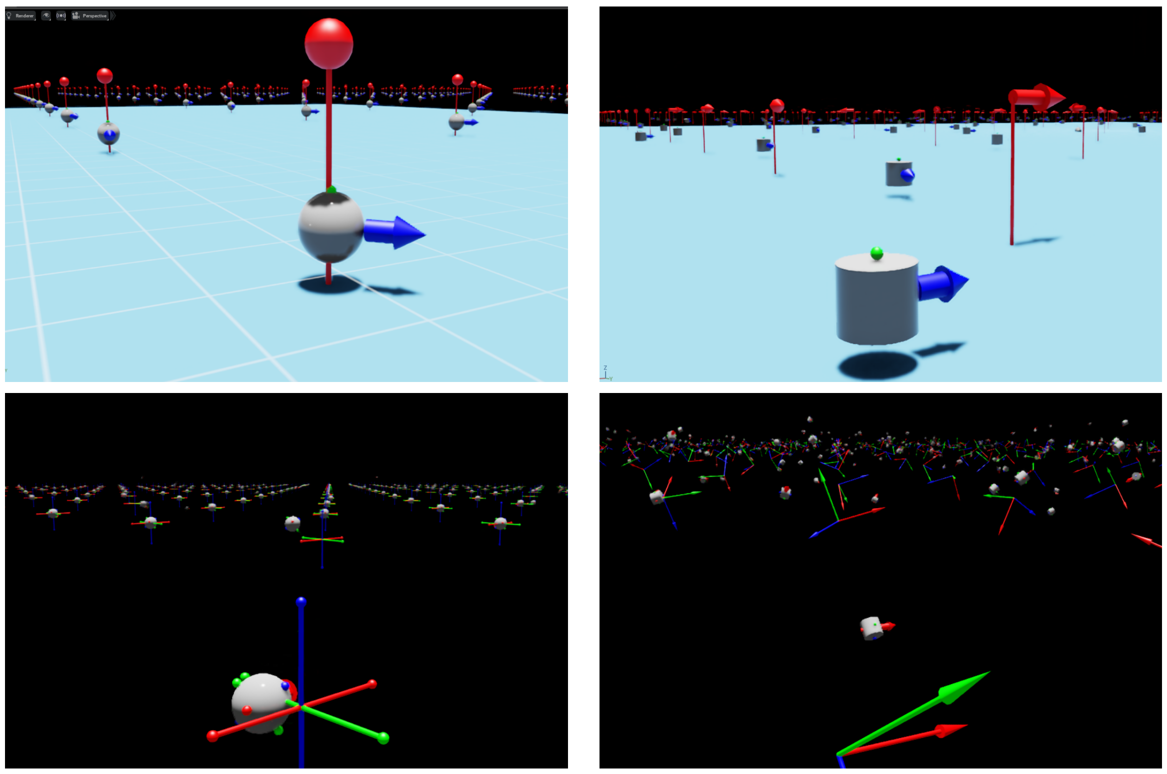


Figure 3.2: Sample renders of RANS tasks in 3DoF (top) and 6DoF (bottom) settings. The simulated agents (cylindrical or spherical) must reach target locations and orientations marked by arrows or pins. Tasks vary in translational and rotational complexity.

eters. It is fully compatible with popular RL libraries (e.g., RL-Games [14]), and includes both headless and GUI-rendered modes. Figure 3.2 illustrates the visual interface for task visualization.

3.3.1 Parallelism and GPU Efficiency

RANS is built on NVIDIA Isaac Gym and utilizes its GPU-native physics engine (PhysX) to enable highly parallelized simulation. This design allows for training thousands of independent agents simultaneously, drastically reducing wall-clock time for policy optimization. Such efficiency is particularly valuable for spacecraft control tasks, where long horizons and sparse rewards increase training time. Figure 3.3 summarizes community feedback on Isaac Sim, the broader ecosystem from which Isaac Gym originates. Practitioners highlight its GPU performance and extensibility

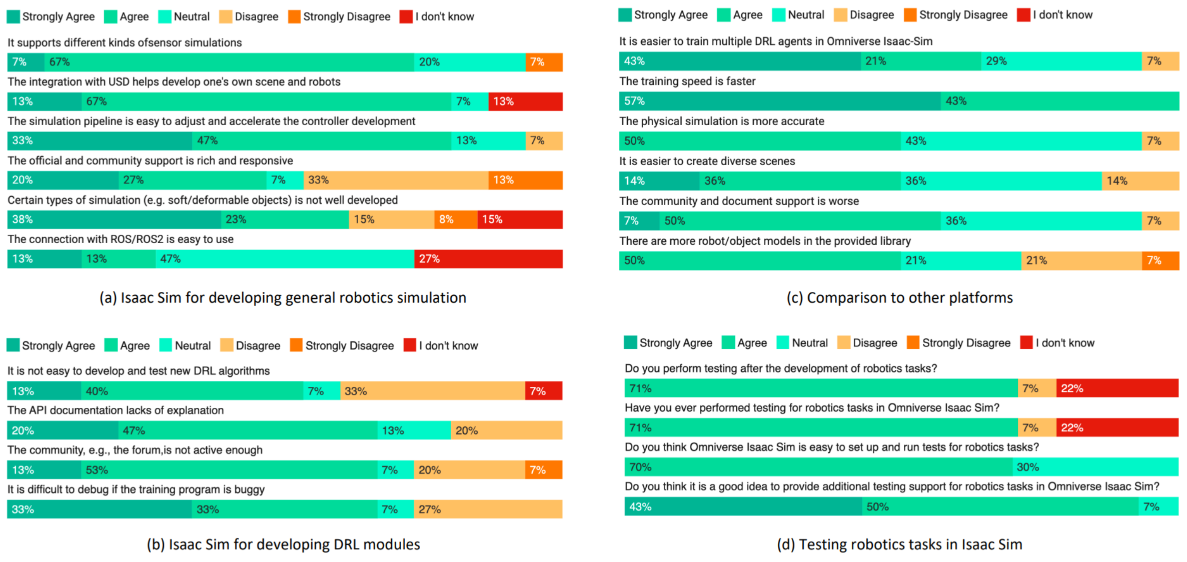


Figure 3.3: Isaac Sim survey on DRL suitability [73]. Although Isaac Gym offers high parallelism and flexibility, challenges remain in documentation and community support. RANS builds on this foundation with domain-specific extensions for spacecraft control.

as key strengths, yet note gaps in documentation and community support [73]. RANS builds on this foundation by providing domain-specific extensions for spacecraft and floating-platform dynamics, tailored observation/action spaces, and ready-to-use PPO interfaces.

Unlike traditional CPU-based simulators like PyBullet [45] or Gazebo [62], which simulate agents sequentially or with limited multi-threading, RANS exploits the GPU's parallel compute architecture to run up to 16000 environments concurrently. This facilitates:

- Fast policy iteration and hyperparameter tuning.
- Scalable evaluation over randomized initial states.
- Real-time observation of learned behaviors across task variants.

Combined with the efficient PPO implementation from the `rl-games` library [14], RANS enables simulation-to-training cycles at a scale suitable for robust control under uncertainty.

Parallel Training Efficiency

RANS supports running as many as 16,000 environments concurrently on a single GeForce RTX 4090 GPU. A PPO agent can solve the GoToXY and GoToPose tasks to high accuracy within 15 minutes of training time, demonstrating the advantage of GPU-accelerated RL workflows.

3.3.2 Simulator Design and Task Suite

The RANS simulator provides a flexible and high-performance environment tailored for thrust-based control of spacecraft and floating platforms in 3DoF and 6DoF. The simulation is implemented using NVIDIA Isaac Gym's PhysX backend, enabling massive GPU-parallel simulations across thousands of agents. This allows reinforcement learning (RL) policies to be trained efficiently in domains with complex, continuous dynamics.

Simulation Physics. The simulation timestep is typically set to 10–20 ms, with substepping used to maintain numerical stability. Agent policies actuate at 5–10 Hz, while the physics engine runs at 10x higher frequency to avoid instability. Gravity is disabled in 3DoF scenarios to emulate planar floating motion, while in 6DoF tasks, free-space motion is fully unconstrained.

For thrust application, multiple rigid bodies are created and forces applied at the relative positions of the thrusters. This circumvents limitations in OmniIsaacGym where consecutive force applications would overwrite previous inputs.

Environment Definition as an MDP. Each task in RANS is formalized as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, r, \mathcal{T}, \gamma)$:

- $\mathcal{S}$: State space includes orientation (e.g., 2D angle or 6D rotation representation), linear and angular velocities, task flags, and target deltas.
- $\mathcal{A}$: Discrete thrust activation vector (binary action per thruster), either 8 (3DoF) or 16 (6DoF) dimensions.

- r: Dense reward function penalizing distance to target, angular deviation, and control effort.
- $\mathcal{T}$: Transition dynamics based on PhysX physics.
- γ: Discount factor, typically 0.99.

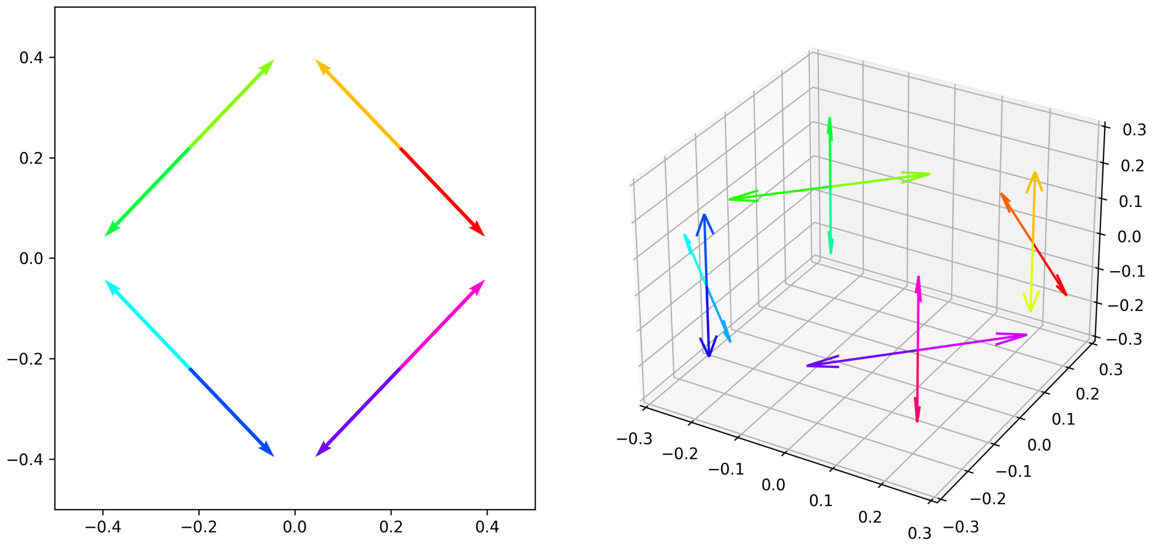
Tasks and Configurations. RANS supports multiple control tasks: In the 3 DoF scenarios, the simulator includes a default system configuration with 8 thrusters (Fig. 3.4 (a)) and allows users to customize various parameters, such as mass and thruster positions, via configuration files. Similarly, in the 6 DoF scenario, the simulators comes with a default 16 thrusters configuration (Fig. 3.4 (b)). The tasks defined for position control and position-attitude control are:

- **GoToXY / GoToPose (3DoF)**: Reach a target (x, y) and optionally a heading angle θ.
- **TrackXYVel / TrackXYOVel (3DoF)**: Match linear or angular velocity commands.
- **GoToXYZ / GoToPose-3D (6DoF)**: Reach a 3D target position and orientation using full control of all degrees of freedom.

The observation space varies between 10 and 22 dimensions depending on the task, and includes terms such as: $\cos(\theta), \sin(\theta), v_{xy}, \omega_z$, target deltas, and task-specific identifiers. Table 3.1 summarizes the task-dependent observations.

Table 3.1: Task-specific deltas appended to the observation vector.

Task	**Observations td**	**Target Types**
GoToXY	$\Delta x, \Delta y$	Position
GoToPose-2D	$\Delta x, \Delta y, \Delta\theta$	Position + Angle
TrackXYVel	$\Delta v_x, \Delta v_y$	2D Velocity
TrackXYZVel	$\Delta v_x, \Delta v_y, \Delta v_z$	3D Velocity
GoToXYZ	$\Delta x, \Delta y, \Delta z$	Position 3D
GoToPose-3D	$\Delta x, y, z + \Delta R$ (rotation matrix)	Full 6D pose

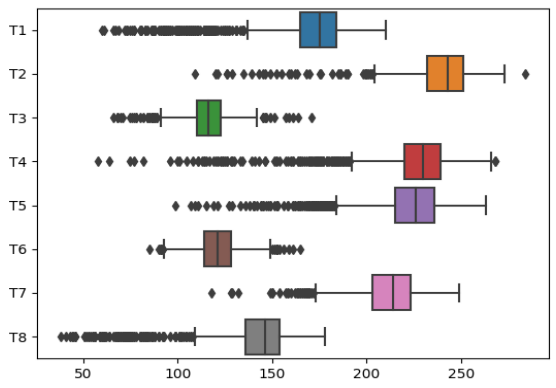


(a) 3DoF (b) 6DoF

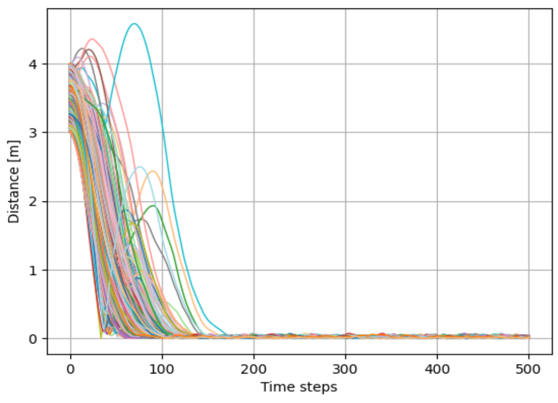
Figure 3.4: Arrows indicate the directions of the forces applied by the thrusters mounted on the system. The center of mass is located at $(0, 0, 0)$.

3.3.3 Baseline Agents and Experimental Results

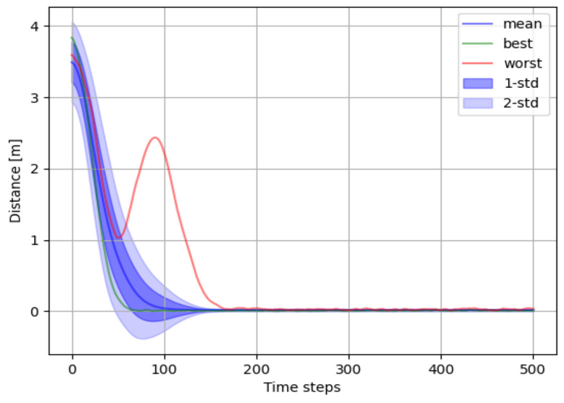
All RL agents are trained using PPO [28] with binary action heads corresponding to each thruster. The policies are implemented as feedforward neural networks with either two or three hidden layers (128 or 256 units), depending on the dimensionality of the control problem (3DoF or 6DoF). Training is performed for 2000 epochs using 1024 parallel environments, leveraging GPU acceleration for fast convergence.



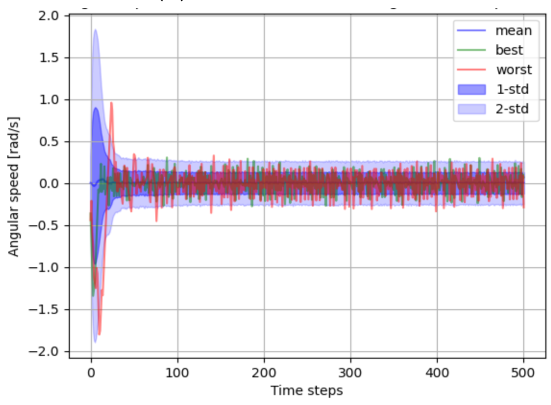
(a) Actions count



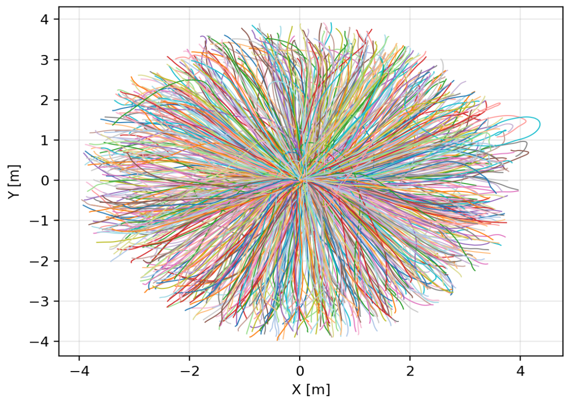
(b) Position distances



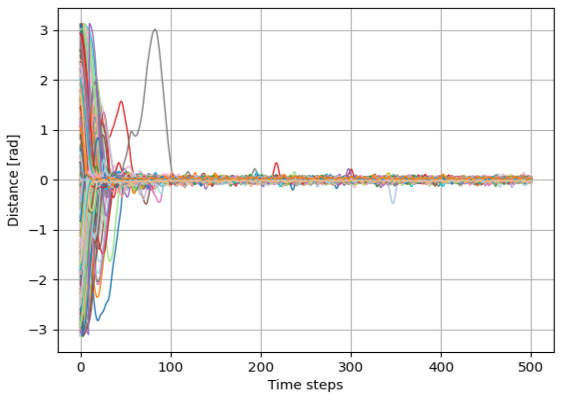
(c) Position distances summary



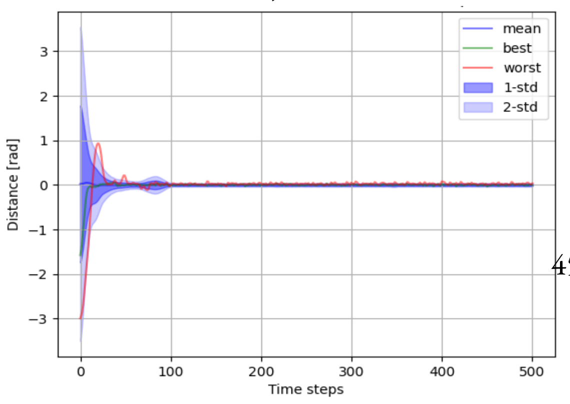
(d) Angular velocities summary



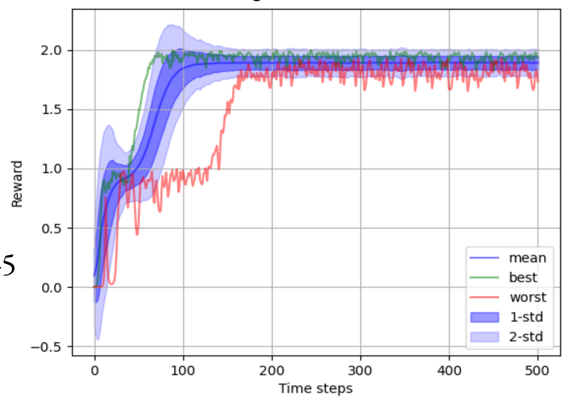
(e) Trajectories



(f) Angle distances



(g) Angle distances summary



(h) Rewards summary

Figure 3.5: Evaluation for the "go to pose" task, with 1024 parallel agents running for 500 steps (25 s), each with randomized initial conditions.

3.3.4 3 DoF Pose Evaluation results

The evaluation encompassed spawning a trained policy in random poses around the target, within distances of 3 to 4 meters. Evaluation metrics, including distance-to-target over time and equivalent planar trajectories, were utilized to quantify and visualize the performance of the trained agents. Figure 3.5 shows the results of the evaluation of an agent trained for the GoToPose task over 1024 runs under nominal conditions. In (a) the average number of thrusts activation per episode shows a high energy demand, needed to achieve both position and orientation control, which can need constant compensation as there is distinguished actuator for that (e.g. reaction wheels). (b) and (c) illustrate the fast and stable convergence to the target position through the distance lines starting from a random position between 3 and 4 meters. Similarly, the plots (f) and (g) demonstrates the convergence to the target orientation. In (d) the mean and standard deviation of the angular velocity show the rotation ranges, after an initial spike, tend to quickly converge to zero, or oscillate around 0.3 radians in the worst case. All the trajectories can be seen in the 2D plane (e), where starting from a random position they all converge to the center. Finally, the reward best, mean and worst cases in (h) interestingly display the agents learned behavior to first collect the orientation rewards by adjusting the attitude, then moving to the target position. Robustness

3.3.5 3 DoF Linear Velocity Tracker results

The results of the linear velocity tracking policy trained with PPO are demonstrated through a set of trajectory following examples. To make the agent follow the trajectories, we use a simple look-ahead planner. In these examples, showed in Fig. 3.6, the RL agents exhibit the ability to accurately track target velocities, enabling them to follow both simple and more complicated predefined trajectories with precision. This planner acquires the farthest point of the trajectory within a 25cm radius, or if there are no points within 25cm, the closest point to the system. Using the position of this point and the position of the system, we compute vector between these two points, normalize it, multiply it by the desired system velocity (0.25m/s) and the resulting vector is given to the agent as the velocity to be tracked. Three distinct trajectories are showcased, including circle, a spiral, and a square. On the circle trajectory, we can see that the agent can easily track sinusoidal velocity commands, though the measured velocities are noisy. Similarly, the spiral shows the agent successfully tracking sinusoidal velocities with different frequencies. The most challenging trajectory to track is the square, this results in step-like velocity commands which the agents match fairly well. However, we can see that the positive and negative velocities have less overshoot than the null velocities. This could be linked to the reward design.

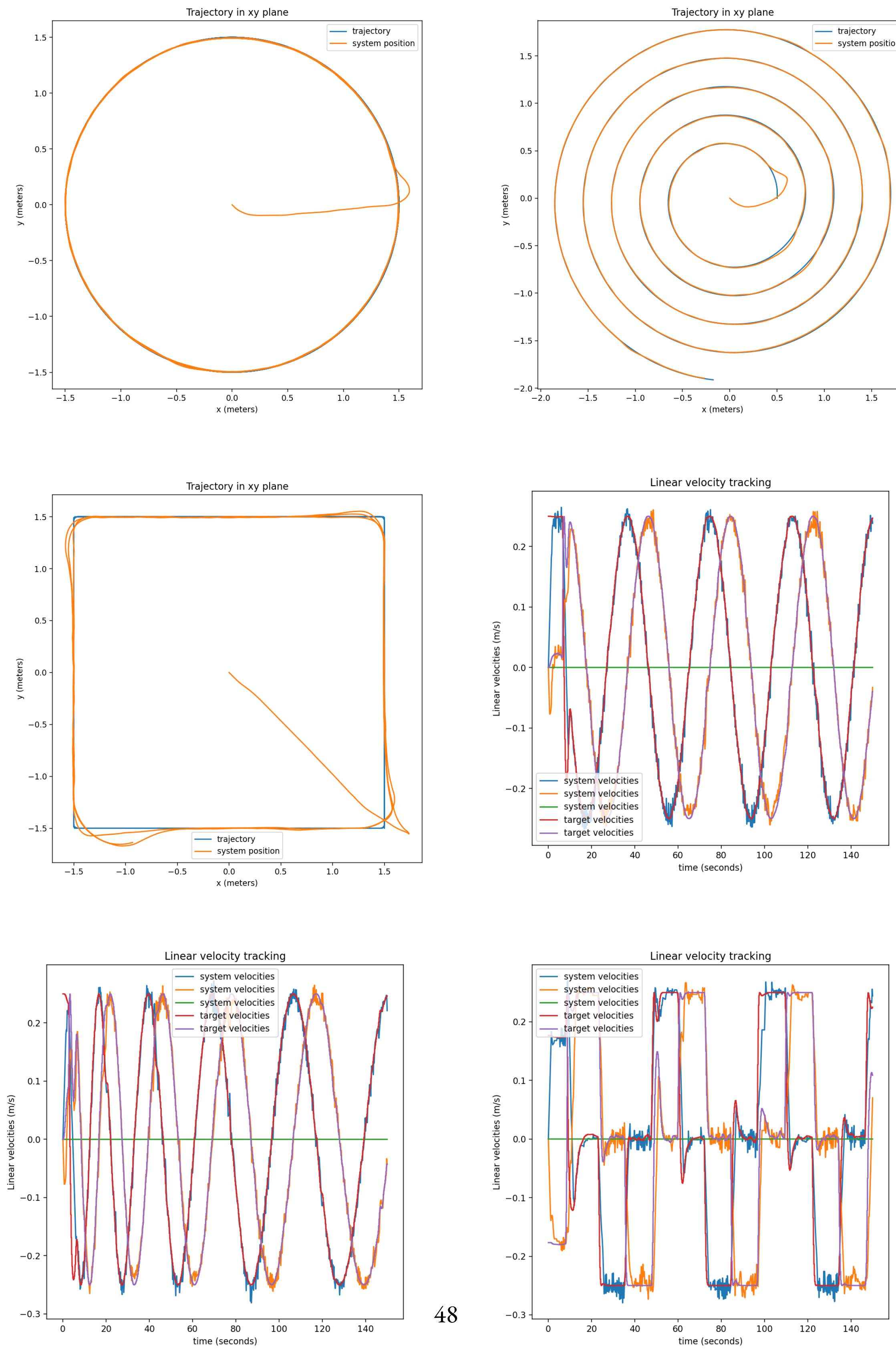


Figure 3.6: Example of trajectory following using agents trained to track velocities. The first three plots show circular, spiral, and square trajectories executed with a simple look-ahead controller that provides target velocities to the RL agents. The last three plots show the policy tracking sinusoidal and square continuous reference signals.

3.3.6 6 DoF GoToXYZ Evaluation results

Presented here is an illustrative demonstration of the policy's behavior trained for the 6DoF GoToXYZ task. The agent's initialization occurs on a spherical surface centered around the target, randomly positioned within a radius of 1 to 5 meters and an angle ϕ ranging from $-\pi$ to π. During the evaluation episode, a set of 1024 parallel agents is spawned for the task, converging swiftly towards the target. Occasionally, some agents display small overshooting or high angular speeds. Overall, the PPO agent's performance is acceptable, exemplified by a rendered trajectory illustrated in Figure 3.7.

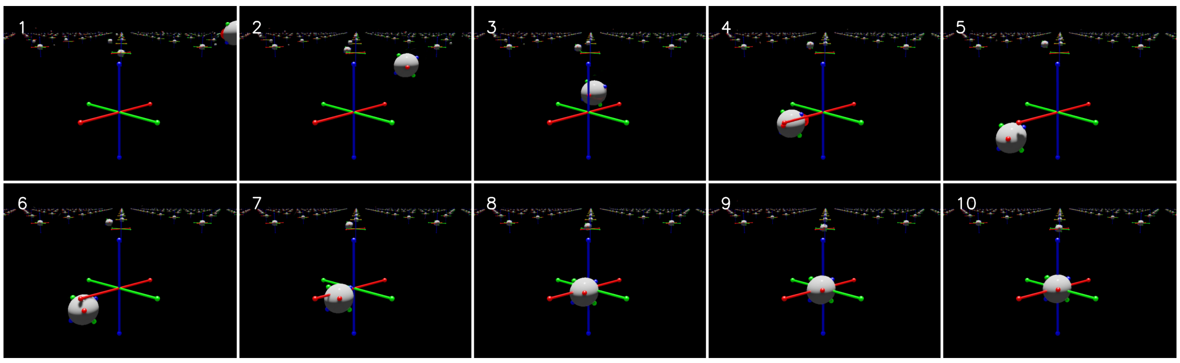


Figure 3.7: Visualization of a 10-frame sequence from a rendered episode depicting the 6-DoF GoToXYZ task. Progressing from top-left to bottom-right, the sequence shows one of the 1024 agents approaching and stabilizing at the designated target location.

3.4 Discussion and Summary

The RANS simulator constitutes a foundational component of this thesis, enabling the development, evaluation, and benchmarking of reinforcement learning agents for spacecraft control tasks in both 3DoF and 6DoF settings. Through extensive experiments, we demonstrated the ability of PPO-based agents to solve complex pose and velocity control tasks with high accuracy across thousands of parallel simulated environments.

Built on Isaac Gym, RANS leverages GPU-parallelism to drastically reduce training times and support systematic experimentation over randomized dynamics, actuator faults, and noise profiles.

The simulation framework is explicitly tailored to thrust-based control scenarios, allowing agents to learn low-level motor policies through binary action heads per thruster. This design supports fine-grained investigation of actuator interactions and fault resilience.

Key Features of RANS

- GPU-accelerated physics with support for 3DoF and 6DoF spacecraft models.
- Modular task suite with structured observations and actions.
- Compatible with modern DRL libraries and scalable to 6000+ agents.
- Built-in support for custom dynamics, actuator topologies, and logging tools.

While RANS already enables meaningful experimentation, the results obtained are still limited to simulation and focus primarily on nominal conditions. Future development will extend the simulator with richer failure modes (e.g., partial thrust loss, sensor degradation), multi-task training support, and seamless integration with real-world testbeds. This work sets the stage for the DRIFT platform, which builds on RANS to evaluate policies in a physical floating platform environment, closing the sim-to-real loop.

3.5 DRIFT: Deep Reinforcement Learning for Intelligent Floating Platforms

DRIFT builds upon the RANS simulator and extends it to support complex control tasks, realistic simulation disturbances, and direct deployment on physical air-bearing platforms. This section details the DRIFT methodology, simulator architecture, task formulation, reinforcement learning training, and a comparison with optimal control baselines in both simulation and hardware.

3.5.1 Problem Formulation

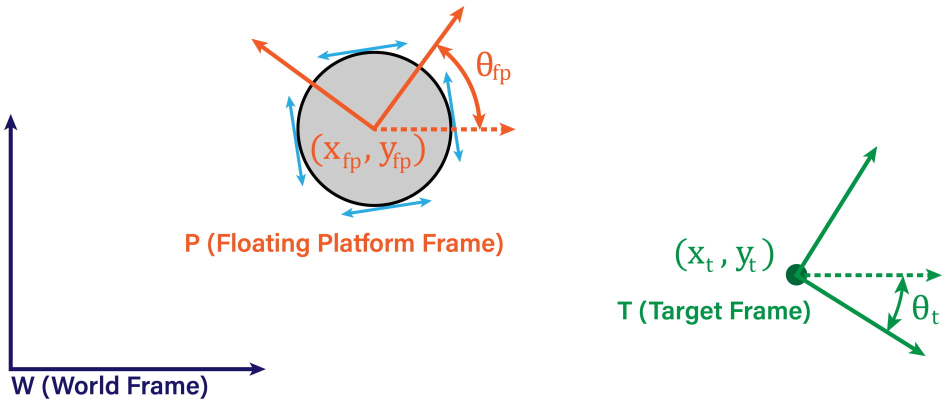


Figure 3.8: Floating platform and target in the global reference frame.

Similarly to RANS, in DRIFT the task of guiding a FP's maneuvers is modelled as a sequential decision-making problem. To facilitate and demonstrate the practical applicability of RL from sim to real-world scenarios, the complex orbital dynamics is simplified into a two-dimensional kinematic model. As illustrated in Figure 3.8, a global reference frame (denoted W) is used. This allows for consistent and absolute measurements of the position and heading errors. The framework also allows for the use of local coordinates whenever considered convenient.

Within this framework the control policy must learn the optimal sequence of actions by observing state transitions, thereby minimizing the task-specific error. We define the different tasks as: *(i) Go to pose*, starting from a random initial position in the plane, reach the given pose (position and orientation θ); *(ii) Track velocity*, track the given velocity vector, which can in turn be used to follow a trajectory.

For both tasks the control policy is required to minimize the error metrics derived from the current state observations of the floating platform and the target. Regarding the "go to pose" task, the positional error is defined as the Euclidean distance between the FP's current position, $p_{fp} = (x_{fp}, y_{fp})$, and the target position, $p_t = (x_t, y_t)$, Eq. (3.1), while the heading error is calculated based on the difference between the platform's current orientation θ_{fp} and the target heading θ_t, Eq. (3.2):

$$e_p = \|\mathbf{p}_{\text{t}} - \mathbf{p}_{\text{fp}}\|_2 \tag{3.1}$$

$$e_\theta = \arctan 2\left(\sin(\theta_{\text{t}} - \theta_{\text{fp}}), \cos(\theta_{\text{t}} - \theta_{\text{fp}})\right) \tag{3.2}$$

For the "track velocity" task the angular and linear velocity errors ($\mathbf{e}_v, \mathbf{e}_\omega$) are determined by subtracting the FP's current velocities ($\mathbf{v}_{fp}$) from the target velocities ($\mathbf{v}_t$), Eq. (3.3) and (3.4).

$$\mathbf{e}_v = \mathbf{v}_{\text{t}} - \mathbf{v}_{\text{fp}} \tag{3.3}$$

$$\mathbf{e}_\omega = \boldsymbol{\omega}_{\text{t}} - \boldsymbol{\omega}_{\text{fp}} \tag{3.4}$$

The floating platform system [74] system is defined by a 10-dimensional state space, Eq. 3.5. At each discrete time step t, the state variables include the FP's heading (θ), its linear velocities (v_x and v_y), angular velocity (ω_z), a task flag (f) indicating the current task, and four additional variables (d_{1-4}) representing task-specific data such as distances to the target position and heading:

$$s_t = (\cos(\theta), \sin(\theta), v_x, v_y, \omega_z, \text{f}, \text{d}_1, \text{d}_2, \text{d}_3, \text{d}_4)^\top. \tag{3.5}$$

Task-specific data, written d_{1-4}, is detailed in Table 3.1, where Δ denotes the vector norm distance between the variables (such as position, velocity, or angle) and their respective target values. This configuration of the observation space is intentionally designed to facilitate the future extension of this work to learn policies capable of handling multiple tasks simultaneously.

For the control of the platform, our agents use an 8-dimensional action space that corresponds to a binary activation of 8 "on-off thrusters". These share the same pressure line, such that, at every step of the control loop, the maximum force generated by each thruster is $\frac{1}{n}$ N where n is the number of active thrusters. Simply put, if only one thruster is turned on, it will output 1 Newton, if 2 thrusters are activated they generate 0.5 N each, etc.

To guide the optimization process for the control policies, aas shown in Figure 3.9, an expo-

Table 3.2: State task-specific data.

Task	**f**	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
Go to pose	1	Δx	Δy	$\cos(\Delta\theta)$	$\sin(\Delta\theta)$
Track velocity	2	Δv_x	Δv_y	-	-

nential reward structure was adopted, as after empirical evaluation it was found to yield faster and more accurate convergence. In particular, Eq. (3.6) for the "go to pose" task and Eq. (3.7) for the "track velocity" task were used:

$$R_{po} = \exp\left(-\frac{e_p}{0.25}\right) \cdot S_p + \exp\left(-\frac{e_\theta}{0.25}\right) \cdot S_\theta - p \tag{3.6}$$

$$R_v = \exp\left(-\frac{e_v}{0.25}\right) \cdot S_p + \exp\left(-\frac{e_\omega}{0.25}\right) \cdot S_\theta - p \tag{3.7}$$

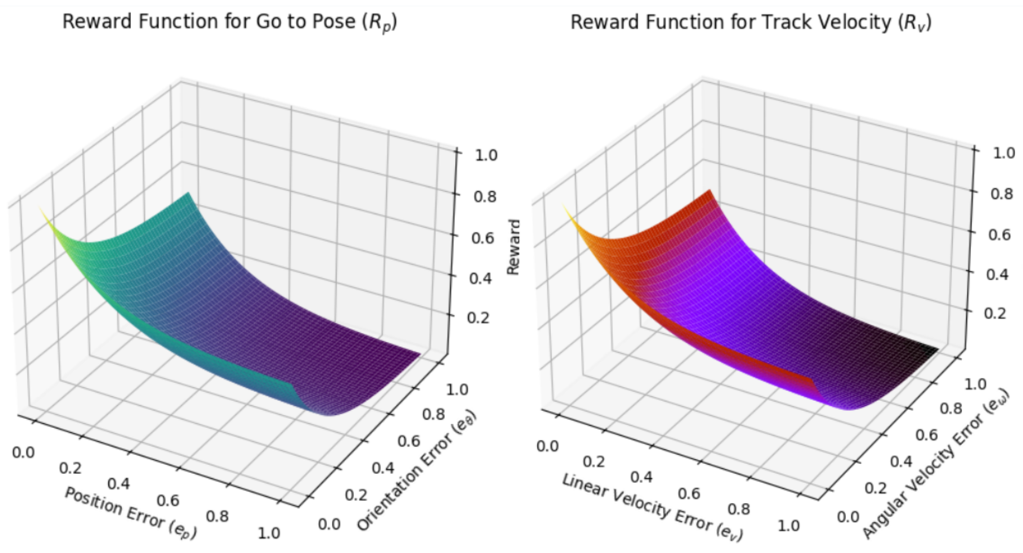


Figure 3.9: Reward functions represented in a 3D plot, showing the shape of the scalar exponential signal provided to the agent for the GoToPose (left) and TrackVelocity (right) tasks.

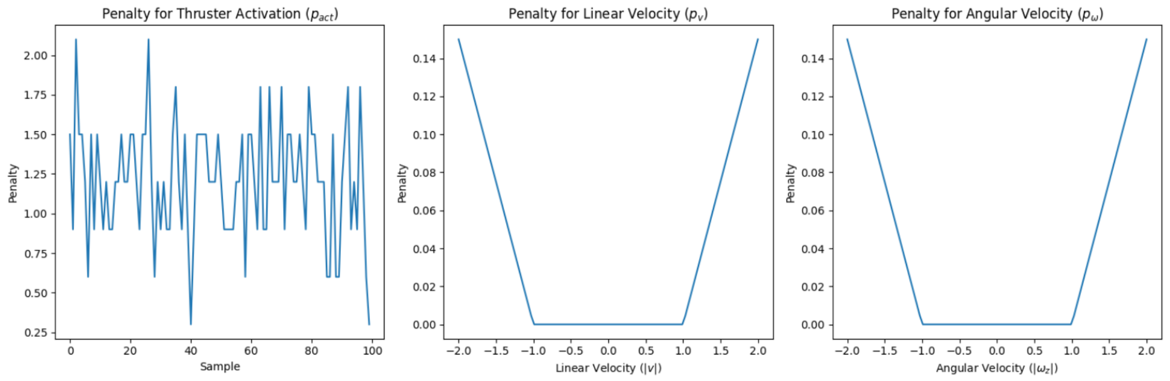


Figure 3.10: Penalty signals represented in 2D plots: thruster activation (left), linear velocity (center), and angular velocity (right) penalties.

In this context, errors are quantified as the norm distance from the specified targets, with e_v denoting the linear velocity error, and e_p and e_θ representing the errors in position and orientation, respectively. Scaling coefficients S_p and S_θ, which adjust the impact of position and orientation errors, were both set to 0.5 in our experiments. Additionally, p sums up to three penalties (p_{act}, p_{vel}, p_ω) designed to discourage excessive thruster activation or reaching states with elevated linear and angular velocities. A representation of the penalties is displayed in Figure 3.10. Our experimentation with various penalty configurations led us to adopt a penalty for thruster activation, Eq. (3.8) as well as excessive angular velocities, Eq. (3.9). Here, T stands for an indicator function reflecting the on-off states of the thrusters.

$$p_{act} = 0.3 \sum_{i=1}^{8} T_i \tag{3.8}$$

$$p_\omega = 0.15 \max(0, |\omega_z| - 1) \tag{3.9}$$

3.5.2 Simulation

In the original RANS framework, only nominal system and environmental conditions were present. This hindered the ability of the agents to adapt to non-ideal conditions, which are usually common when using the real FP systems. To mitigate this gap, we introduce RANS v2.0

which includes the following extensions: (1) parameterized rewards and penalties, to allow easy fine-tuning of the control policies; (2) analogue kinematic model in Mujoco [6], to allow easy evaluation of both traditional and RL-based controllers in a non-Torch depended environment; (3) disturbance generation module, that allows the injection of: (a) Action Noise (AN): a random disturbance force of $\pm\ an$ N applied to every thruster; (b) Velocity Noise (VN): $\pm\ vn$ m/s added to the state velocities; (c) Uneven Floor (UF): uf N of force, added to simulate the floor unevenness, applied to the FP body throughout the episode, either with a constant direction or through a sinusoidal generated direction; (d) Torque Disturbance (TD): td Nm of torque applied to the body's center of mass; (e) Random Thrusters Failure (RTF): a zeroing mask over the output actions to simulate one or multiple thruster failures which remains the same throughout the episode. Some disturbance are illustrated in Figure 3.11 for better clarity.

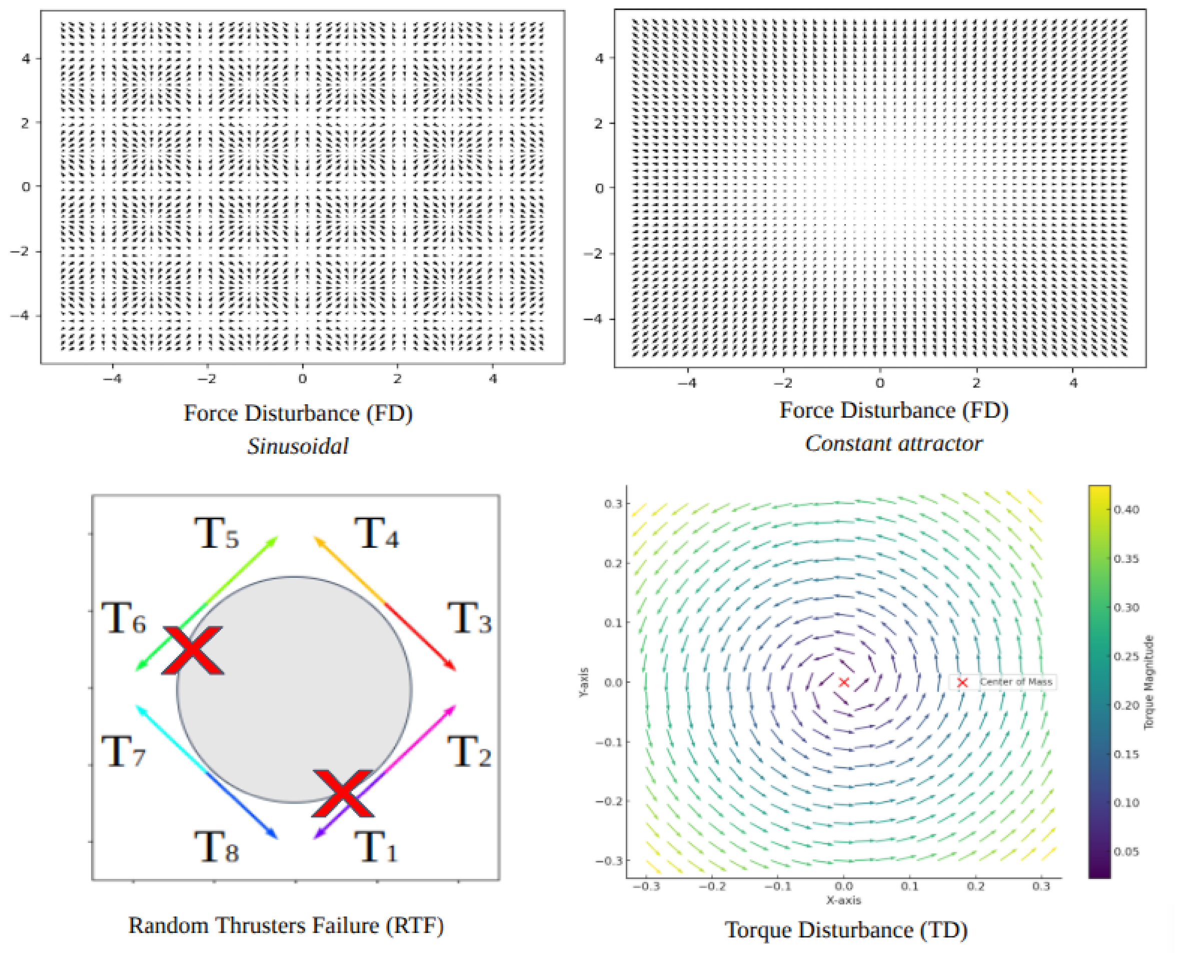


Figure 3.11: Visualizations of the disturbance models used in simulation: (top-left) sinusoidal force field (FD) introducing position-dependent perturbations; (top-right) constant force field pulling the agent toward the origin; (bottom-left) Random Thruster Failure (RTF) where a subset of actuators is disabled during training; (bottom-right) torque disturbance (TD) field inducing rotational drift around the center of mass.

RANS v2.0, requires 30 minutes to train an agent on an RTX 4090. Achieving a throughput of more than 40,000 steps per second with all disturbances enabled, which is very close to its previous version. Furthermore, it enables large-scale testing by swiftly evaluating thousands of initial conditions in seconds. It offers rich visualization options, including metric tracking during training through the WandB API [75], and comprehensive evaluation metrics presented through tables and plots. The library uses the OpenAI Gym [4] format to define the RL loop, including the standard normalization of the observation space. Additionally, the integration of a ROS interface enhances the versatility of our framework, allowing easy integration and deployment of the control policies within real-world robotic systems.

3.5.3 Training Procedure

We reworked the PPO implementation from the RL Games library [14] as the foundation of our training procedure. This implementation utilizes GPU acceleration to vectorize observations and actions, enabling parallelization within the simulator by having both the simulation and the policy training residing on GPU. Our agents are designed as actor-critic networks with two hidden layers, each consisting of 128 units. This makes them light and fast enough to be ran at high frequency on embedded devices. The hyper-parameters are listed in Table 3.3 in the appendix. The agents train in their respective environments for 2000 epochs (approximately 130M steps). Table 3.3 outlines the key parameters used in the adapted version of the Proximal Policy Optimization (PPO) algorithm for training our models.

3.5.4 Benchmark comparison with an Optimal Controller

DRIFT aims to provide a benchmark comparison between deep reinforcement learning and optimal control approaches, LQR in particular, for addressing the control problem of the floating platform in various scenarios. The objective is not to establish the superiority of one method over the other, but rather to gain insights into the strengths and weaknesses of each approach under different environmental conditions and task requirements.

An infinite horizon discrete-time LQR controller [76] is used as a preliminary comparison with the DRL algorithm to control the FP. The LQR technique utilizes linearized dynamics to comprehensively model system behavior, providing optimal solutions with long-term stability while handling minor disturbances [77]. Their adaptability and relatively straightforward implementation have resulted in their adoption for numerous space applications [78, 79, 80]. In the case of a FP, the position, linear velocities, orientation quaternions, and angular velocities in the

Parameter	**Value**
Algorithm	PPO
Network Type	Actor-Critic MLP
Separate Networks	True
MLP Units	[128, 128]
Activation Function	tanh
Initializer	Identity
Regularizer	None
Learning Rate	$1e-4$
Gamma (γ)	0.99
Tau (τ)	0.95
Entropy Coefficient	0.0
Horizon Length	16
Minibatch Size	8192
Mini Epochs	8
Critic Coefficient	0.5
Gradient Clipping Norm	1.0
KL Threshold	0.016
Critic Coefficient	0.5

Table 3.3: PPO training parameters.

2D plane are considered state variables of the system, $\mathbf{X}_i$. Since a FP operates at a relatively high frequency, a linearized system dynamics, defined as (3.10)

$$\mathbf{X}_{k+1} = \mathbf{A}\mathbf{X}_k + \mathbf{B}\mathbf{U}_k \tag{3.10}$$

is sufficient to predict the control output for incremental steps. The linearized system matrices, represented by **A** and **B**, are the partial derivatives of the state vector at the final time step, denoted as $\mathbf{X}_{k+1}$, with respect to the current time step, $\mathbf{X}_k$, and the control input $\delta\mathbf{U}_k$, respectively. This computation leverages the central differencing technique, where the effects on the final states are evaluated in response to deliberate and minor perturbations applied to both the states and control inputs within the kinematic model simulated in Mujoco. To better account for the disturbances endured by the FP, the system matrices are updated at regular intervals. The LQR controller minimizes the cost function:

$$J = \sum_{k=0}^{\infty} \mathbf{X}_k^{\mathrm{T}} \mathbf{Q} \mathbf{X}_k + \mathbf{U}_k^{\mathrm{T}} \mathbf{R} \mathbf{U}_k$$

where **Q** and **R** are weighting matrices that penalize state errors and control outputs. Minimizing the aforementioned cost function delivers an optimal control sequence given by:

$$\mathbf{U}_k = -\mathbf{K}\mathbf{X}_k$$

where **K** is the control feedback gain matrix defined by:

$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{A}$$

such that **P** is a positive definite matrix that is a solution for the Algebraic Riccati equation, as in:

$$\mathbf{P} = \mathbf{Q} + \mathbf{A}^{\mathrm{T}}\mathbf{P}\mathbf{A} - \mathbf{A}^{\mathrm{T}}\mathbf{P}\mathbf{B}\mathbf{K}.$$

The optimal control output, $\mathbf{U}_k$, is an eight-dimensional array with real numbers. Note that the control outputs correspond to the actuation of the eight thrusters on the FP, hence an alternate vector $\mathbb{U}_k$ is implemented that is a least squares solution to:

$$\min \quad || \, \mathbb{U}_k - \mathbf{U}'_k \, ||^2$$

where $\mathbf{U}'_k$ is the normalized vector of $\mathbf{U}_k$ with values between 0 and 1. Moreover, for $\mathbb{U}_k = [u_1, u_2, ..., u_8]$, each u_i for $i \in \{1, 2, ..., 8\}$ represents a binary variable, i.e., $u_i \in \{0, 1\}$ signifying the actuation state of each thruster as either "on" or "off".

Table 3.4 summarizes the parameters of the Discrete LQR Controller used. The controller is made planar compatible, indicating a restriction to the 2D plane.

Parameter	**Value**
Name	LQR
Q (State cost matrix)	[0.0001, 1e-05, 100, 100, 1e-06, 1e-06, 1]
R (Control cost matrix)	[0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]
W (Disturbance weight matrix)	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
Make planar compatible	Yes
Control type	LQR

Table 3.4: Parameters for the Discrete LQR Controller

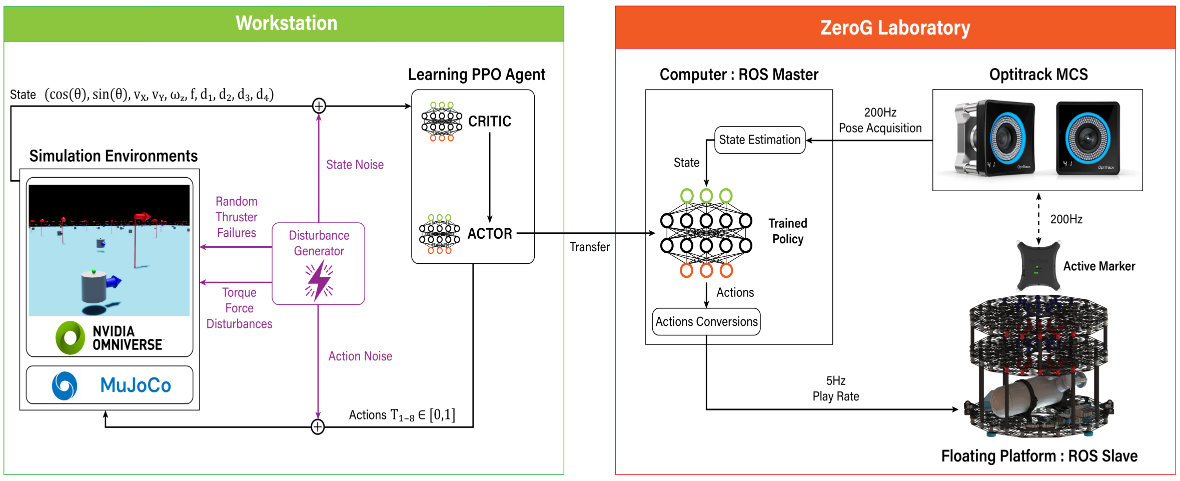


Figure 3.12: Framework employed for training and evaluation. Left: agent interaction during training and evaluation in simulation, including disturbance injection. Right: deployment of the trained policy performing open-loop control on the real floating-platform system.

3.5.5 Laboratory Experiment Setup

To validate our approach in a real-world scenario, we conducted experiments using the physical air bearings platform [74] located within the ZeroG Laboratory at the University of Luxembourg. This specific platform floats on an epoxy floor, weighs 5.32 kg and measures 31 cm in radius and 45 cm in height, a detailed representation is shown in Figure 3.13. It is equipped with a Raspberry Pi 4 for onboard control and communication. The ZeroG Lab contains an Optitrack Motion Capture System (MCS) that precisely tracks the platform's pose at a frequency of 200 Hz. We derive linear and angular velocities through simple forward differencing, that estimate the rate of change of positions and orientations over consecutive time-steps. Thanks to the relatively high accuracy of the MCS, and a reasonable averaging window, concerns about noise sensitivity are negligible. Our experimental setup maintains a connection between a laptop, the MCS, and the FP through a local network. The laptop serves as the ROS (Robot Operating System) master node on the network, subscribing to the Optitrack node to acquire pose data and publishing the actions of the trained agents at a rate of 5 Hz. This action frequency is deliberately constrained to prevent damage to the solenoid valves controlling the thrusters on the floating platform. Figure 3.12 illustrates the key components interacting during the simulated training and validation phase (on the left), and those interacting during the closed-loop control tests of the real FP system in the Lab (on the right).

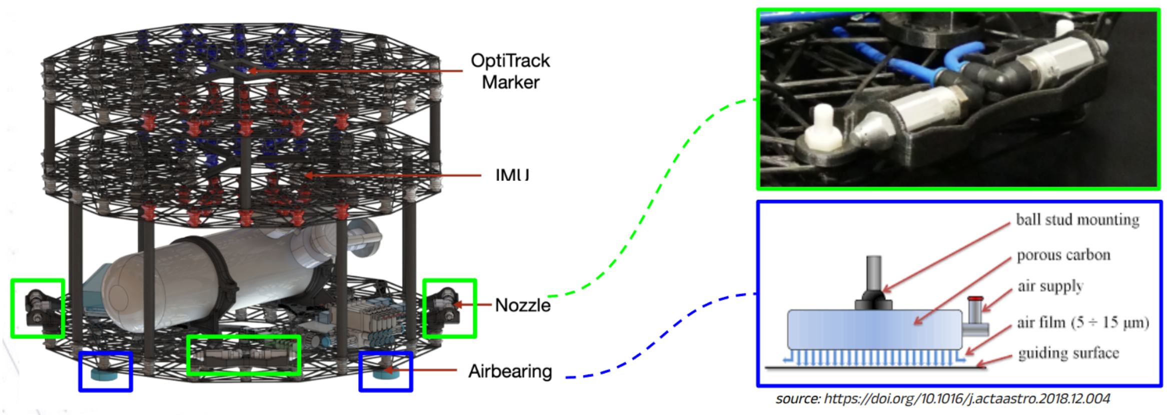


Figure 3.13: Rendered floating platform used for the experiments, showing detailed positioning and zoomed views of the air bearings (blue boxes) and thrusters (green boxes).

3.5.6 Experimental Setup

Our experiments encompass both numerical simulation-based evaluations and real-world validations. For the evaluation, each trained policy was tested across a diverse set of scenarios defined by various environmental conditions.

Performance Metrics

To evaluate the performance of the pose task in numerical simulations, we record 9 metrics: The percentage of time the agent spends under a given distance threshold during a single trajectory. This measure is then averaged across all experiments. For instance, PT_5 denotes the percentage of time spent under 5 cm, we also record this for 2 cm (PT_2) and 1 cm (PT_1). This measure is also applied to the heading of the agents when performing the pose task. In this case, OT_5 is the percentage of time spent under 5 degrees, this measure is also done for 2 degrees (OT_2), and 1 degrees (OT_1). Finally, we also record the absolute average linear velocity (ALV) and absolute average angular velocities (AAV). These metrics are compiled per trajectory, and averaged on the whole of them. This enables us to estimate how dynamic the agent's movements are. Furthermore, we monitor the average number of actions used per step (AAS), to evaluate the efficiency of the policy.

To evaluate the pose task in the lab, we only use the position and orientation error, since we do not have enough experiments to compile more complete statistics. However, we do provide complete trajectories to better understand the behavior of the RL agent and LQR controller.

Finally, for the velocity tracking, we chose to apply the controllers on a trajectory tracking task. For that, we wrote a simple trajectory tracker, that generates a velocity vector to track, based on a sequence of points to follow. This vector is computed by taking the closest point that intersect with a circle of radius r centered around the system. This radius, is a look-ahead-distance which can be tuned to adjust the speed of the tracker. The velocity is considered fixed for the whole of the trajectory, meaning that the instructed velocity is not reduced even if there are sharp corners. This controller is then applied on 3 shapes, a circle, a square and a infinite. For these trajectories, we measure the error in velocity, and the averaged trajectory tracking error.

Real-World Experimental Validations

To validate the real-world applicability of our simulation-trained control policies, we used the physical floating platform with the laboratory setup described in section 3.5.5 to perform a series of experiments. Each test run, for the same policy, initiated the FP from different initial conditions,

Table 3.5: Benchmark of the RL model and LQR controller under disturbances. For PT and OT, higher values indicate better performance; for ALV, AAV, and AAS, lower is better. Colors in the table indicate the drop in performance relative to each method's ideal (no-disturbance) conditions: blue (0–20%), green (20–40%), yellow (40–60%), red (60–80%), purple (80–100%). LQR dynamics parameters are tuned without noise or disturbances enabled.

Conditions	Controllers	Disturbances				Metrics								
		VN (m/s)	UF (N)	TD (N·m)	RTF (-)	PT5 (%)	PT2 (%)	PT1 (%)	OT5 (%)	OT2 (%)	OT1 (%)	ALV (m/s)	AAV (rad/s)	AAS (-)
Ideal	RL	-	-	-	-	64	34	6	94	89	73	0.08	0.12	0.29
	LQR	-	-	-	-	73	41	17	27	11	5	0.07	0.16	0.10
Velocity Noise	RL	0.02	-	-	-	64	30	7	94	90	72	0.08	0.12	0.31
	RL	0.04	-	-	-	61	21	6	94	89	66	0.09	0.13	0.31
	LQR	0.02	-	-	-	53	21	6	4	1	0	0.09	0.49	0.23
	LQR	0.04	-	-	-	14	3	0	2	1	0	0.15	0.56	0.29
Constant Torque	RL	-	-	0.05	-	63	24	2	94	86	61	0.08	0.12	0.35
	LQR	-	-	0.05	-	57	20	6	3	1	0	0.07	0.43	0.35
Constant Force	RL	-	0.20	-	-	63	29	7	94	90	74	0.09	0.12	0.30
	RL	-	0.40	-	-	52	19	5	94	89	72	0.09	0.12	0.31
	LQR	-	0.20	-	-	66	17	4	28	12	6	0.07	0.15	0.12
	LQR	-	0.40	-	-	23	0	0	30	13	6	0.08	0.16	0.15
Constant Force & Torque	RL	-	0.20	0.05	-	62	24	5	94	86	61	0.08	0.12	0.35
	LQR	-	0.20	0.05	-	13	2	0	3	1	0	0.07	0.44	0.32
Thruster Failures	RL	-	-	-	1	32	15	6	70	55	36	0.10	0.12	0.28
	RL	-	-	-	2	15	6	2	45	31	20	0.16	0.15	0.25
	LQR	-	-	-	1	40	17	5	20	8	4	0.10	0.21	0.16
	LQR	-	-	-	2	12	4	1	11	4	2	0.14	0.28	0.22

namely position and orientation within the lab.

3.5.7 Results

Simulation-based experiments demonstrate the efficacy of the PPO-based approach in achieving the defined tasks. The agent exhibits rapid task completion, stability in control, and adaptation to various scenarios. Quantitative metrics and qualitative visualizations substantiate the agent's high-performance capabilities.

Numerical Simulation RL & LQR

In this section, we explore the behaviour of an RL agent trained to perform the "go to pose" task, and compare it to the LQR controller. We chose the "go to pose" task as it is a representative example, allowing us to assess the behaviour of different policies while controlling both the position and the orientation of the FP. To characterize the controllers' behaviors we expose them to a range of disturbances. Neither the RL agents nor the LQR are specifically adapted to incorporate methods from robust RL or robust optimal control theory. Yet, it is important to acknowledge that the RL agent was trained with some domain randomization to learn how to deal with force disturbances up to 0.25 N. Both of them are evaluated in MuJoCo, with similarly randomized initial conditions. In Table 3.5, each line corresponds to an experiment, with various disturbances applied, and was compiled using 256 trajectories of 250 steps each.

First, the two test models are analyzed under ideal conditions with no disturbances. From the PT metrics, it is evident that the LQR controller converges faster in position with better accuracy than the RL, owing to substantially longer durations where the LQR maintains a position error under 1 cm. We can also see that the RL controller first aligns its heading with the goal, as it spends almost all its time under the $5°$ threshold. This is a byproduct of its reward shaping, which incentivizes the convergence of the heading as much as the position. Hence, to score the maximum of points, aligning the heading first is a sound strategy as it is the easiest under ideal conditions. Finally, AAS values show that the LQR is a lot more fuel efficient in these conditions, with 66% less fuel used than the RL agent.

When considering the Velocity Noise (VN), it is observed that with the lowest noise level, the RL performances remain unchanged, while the LQR struggles, in particular with attitude control. With 0.04 m/s of noise, the performance of both controllers decreases. However, the RL controller is more resilient than the LQR controller to this kind of disturbance, even though it was not trained for it. In the interest of brevity, we do not report action noise value in the table, as we found their effect to be negligible on both controllers.

Furthermore, when examining the Torque Disturbance (TD) of 0.05 N·m, equivalent to 1/6-th of the total torque capacity of the platform, the performance of both controllers experiences a noticeable reduction, particularly for the LQR controller. A similar pattern is observed with the force disturbance (UF), which would be equivalent to an uneven floor in the lab. In this case, starting by applying 0.2N of force on the platform (equivalent to 1/5th of its maximum thrust), results in the performance of both controllers being close to the ideal conditions, with a small performance drop of the LQR in fine positioning. When doubling it (0.4 N), the RL policy remains close to its baseline, but the LQR performance decreases, making it unable to maintain positions under the 2.5 cm threshold. Similar behaviours are observed upon the addition of both force and torque disturbances.

Finally, the thruster failures impact the performance of both controllers in the same manner. With a single failed thruster, both controllers perform relatively well, but the addition of a second thruster failure impedes the controller's ability to drive the FP to its defined goals.

Overall, while the LQR controller demonstrates greater efficiency and precision in position control with our current tuning, it encounters challenges when subjected to the selected range of disturbances. In contrast, RL exhibits a lower degree of energy conservation but offers stronger resilience when subject to a wide range of disturbances. It is possible that with a different cost function, better tuning of its weights, and a robust optimal control approach, the LQR becomes adept with these disturbances. Similarly, the RL agent could be induced to learn more conservative policy that uses less actions throughout the episodes, via adequate reward shaping. However, the RL agent is not using a robust RL approach either, and domain randomization was only applied on force disturbances up to 0.25N, which is less than the disturbances it can overcome.

ZeroG Laboratory

For experiments with the real FP system, we report tests using both the RL and LQR methods for the "go to pose" task, and tests using the RL agent only for the "track velocity" task.

Go to pose The controllers are run on the FP, which is connected to a constant air supply through a tether. This tether applies some light unknown disturbances such as a small torque and force to the platform. Moreover, the system velocities are derived from the optitrack system. The observed velocities include minor noise and small delays due to network communication.

Figure 3.14 illustrates the performance of each controller. The first row shows the trajectories of the FP, and the second row shows the distance to the goal in position and orientation. The first two columns have the rough same initial pose: Init1, while the two last share the same initial pose: Init2.

From the last row, it is evident that the LQR controller converges faster in position than the RL controller. This aligns well with the behaviours observed in the simulation benchmark, with an LQR controller converging faster. However, it is also apparent that the LQR solution exhibits a minor overshoot. Such an observation is also in line with the simulation benchmark, as the uneven floor in the lab likely disrupts the LQR controller by applying a subtle constant force, preventing it from reaching its simulation baseline performance. Looking at the top row, we can see that the LQR is also overshooting a bit. Of course, the behaviour can be adjusted by modifying the weights associated with the importance of the error in position in the cost matrix. It is also worth noting that if these weights are not large enough, the LQR controller struggles to converge toward the goal. It is also worth noting that the LQR controller is sensitive to the weights; smaller weights do not incentivize the FP motion toward the goal. Compared to the simulation, we had to adjust these weights to make the controller more aggressive in order to have satisfactory performances. In comparison to the simulation, it was deemed necessary to alter the weights of the LQR controller to yield a more aggressive approach to achieve satisfying performances. As for the RL agent, it can be seen that it is first aligning its heading and then slowly converging towards the goal. As for the RL agent, it is noticeable that the FP initially aligns its heading and then gradually converges toward the goal. Consistently with the results from the simulation, the RL controller is significantly more accurate in terms of heading while achieving a position accuracy similar to that of the LQR controller. Overall, both controllers performed well in the lab, reaching their expected performances.

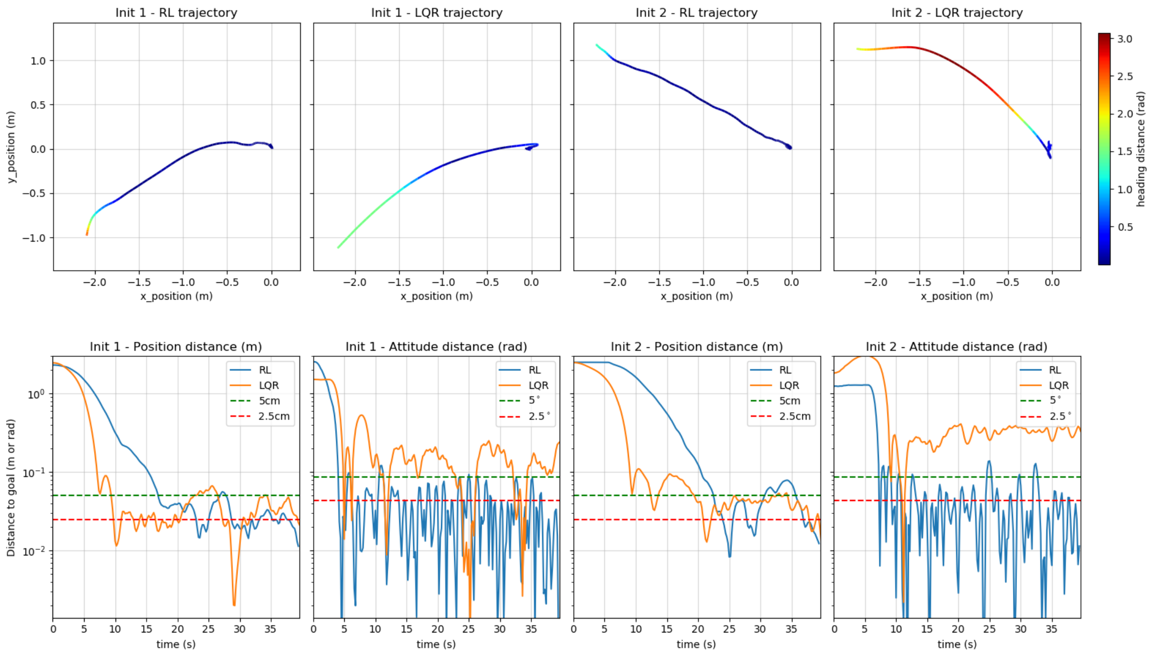


Figure 3.14: Comparison of the RL and LQR controllers on two different initial poses in the Zero-G Lab. Init 1 denotes the first initial pose and Init 2 the second. For the trajectory plots, the y-axis is shown on a logarithmic scale for improved visualization.

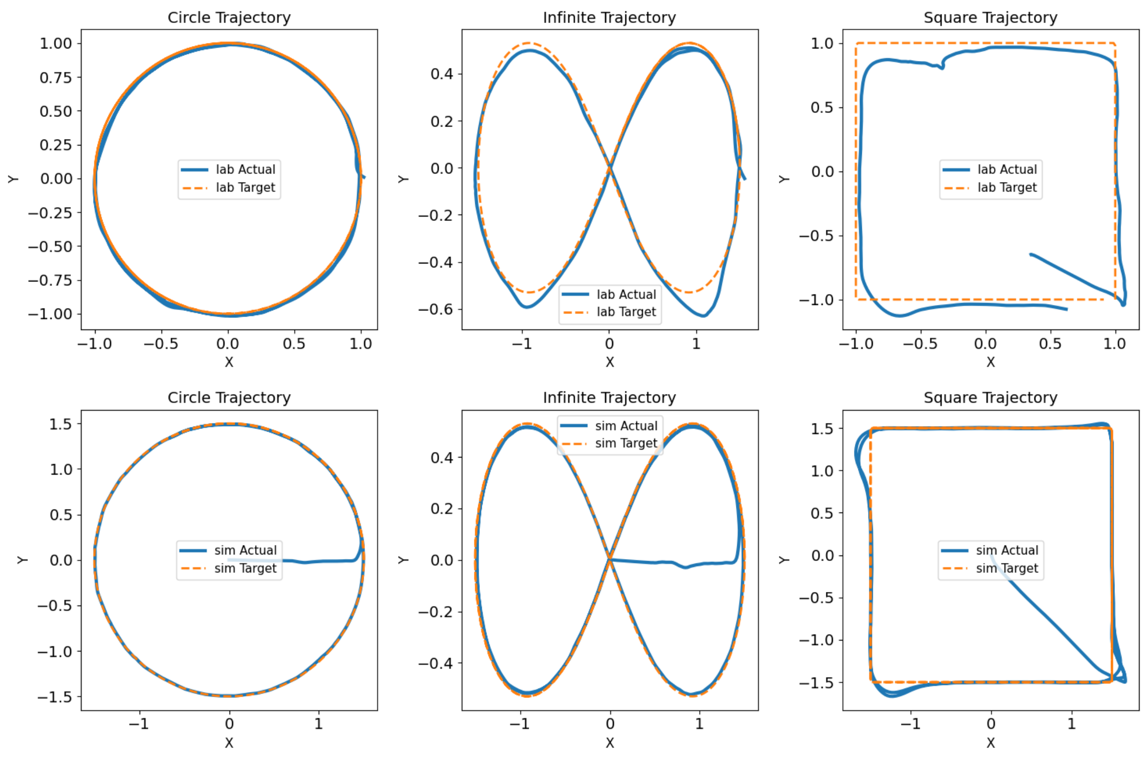


Figure 3.15: RL agent performing velocity tracking in simulated (bottom) and lab (top) environments. From left to right, circular, lemniscate, and square trajectories are used as references.

Track velocity

In the tests performed for this task in the lab, the objective is to assess the simulation-trained policy ability to adhere to a set of predetermined target velocities. Since the LQR model relies on both position and velocity states as input, while the RL agent only requires velocity, we opted to present the RL policy results for this specific task. Both numerical-simulation and lab tests are displayed to validate the sim-to-real transfer.

Similar to the “go to pose” experiments, the FP was subjected to un-modeled disturbances affecting both linear and angular motion. An additional challenge in these tests was the accurate estimation of velocities, affected by slight measurement noise and communication delays. The pre-generated trajectories to be tracked by the policy were designed to test the FP’s response accuracy

Shape	**Lab Error** ($\mu \pm \sigma$) [m/s]	**Sim Error** ($\mu \pm \sigma$) [m/s]
circle	0.03 ± 0.02	0.01 ± 0.01
infinite	0.04 ± 0.03	0.01 ± 0.01
square	0.07 ± 0.05	0.05 ± 0.08

Table 3.6: Comparison of velocity errors between lab and simulation environments for the Track-Velocity task. All trajectories are tracked at 0.2 m/s.

and agility.

Figure 3.15 illustrates the target trajectory and the FP's position for the circle, square and infinite shapes. It is clearly visible that the hardest task was to follow a squared-shaped trajectory. This is due to the sharp turns that require precise maneuvering and acceleration adjustments, which could be induced by reducing the look-ahead-distance and target velocity of the tracking when close to corners. The performance metric used is the linear velocity error e_v expressed as $\mu \pm \sigma$, where μ is the mean and σ is the standard deviation during the test duration. Table 3.6 reveals that the lab environment generally presents higher velocity errors compared to the simulation environment, particularly notable in the square shape with a lab error of 0.07 ± 0.05 m/s versus a sim error of 0.05 ± 0.08 m/s, the difficulty of real-world transfer. For the infinite trajectory, we observed a slight overshoot in the path's lower regions, caused by the irregularities in the epoxy floor, which are significant in that area of the laboratory, affecting the FP's motion. This can also be seen on the square, and to less of a degree on the circle. In our case, there is a slope pulling free-floating objects towards negative y.

3.6 Discussion and Summary

This chapter presented DRIFT, an experimental framework and physical testbed enabling the validation of DRL-based control policies on air-bearing floating platforms. Building upon the RANS simulator, DRIFT introduced a sim-to-real pipeline that supports both position–orientation control and velocity tracking in a 2D planar space, emulating the drag-free conditions of microgravity.

We demonstrated that PPO-based agents trained entirely in simulation could be successfully transferred to the physical floating platform with minimal degradation in performance. Comparative experiments against optimal control techniques such as LQR revealed that while traditional methods remain effective under nominal conditions, DRL policies exhibit greater robustness in the presence of stochastic force fields, actuator delays, and model mismatch. This confirms the potential of DRL for real-world space autonomy, particularly when accurate modeling is infeasible or when onboard adaptability is required.

Key Contributions of DRIFT

- Introduced a sim-to-real pipeline for 2D floating platform control.
- Demonstrated task transfer of PPO policies under domain uncertainties.
- Compared DRL with LQR under force disturbances, showing better adaptability of learning-based policies.
- Released an extended simulation framework with richer dynamics, visual tools, and task diversity.

Despite the encouraging results, limitations remain. The current DRL agents use feedforward policies, which may underperform in highly delayed or partially observable settings. In future work, we plan to investigate memory-based architectures (e.g., LSTMs or Transformers) and augment the simulation with additional sensors and disturbances. Moreover, extending the platform to support articulated morphologies—such as legged robots with reaction thrusters—could open new avenues for microgravity locomotion and hopping-based mobility.

Together, RANS and DRIFT lay the groundwork for reproducible research in space-relevant RL control, from scalable simulation to real-world deployment. These efforts inform the unified training and evaluation frameworks described in the next chapters, where we extend from spacecraft scenarios to multi-robot navigation and aerial-ground control.

Chapter 4

Unified Learning-Based Navigation Across Diverse Robot Platforms in Simulated and Physical Environments

4.1 Introduction and Motivation

The previous chapters have highlighted the growing role of simulation frameworks in advancing reinforcement learning (RL) methods for robot navigation. While task-specific environments like RANS and DRIFT provided efficient training grounds for spacecraft and microgravity platforms, the broader robotics field still lacks standardized frameworks to develop, compare, and deploy RL agents across heterogeneous platforms and environments.

This chapter introduces **RoboRAN**, a multi-domain simulation and evaluation framework designed to scale up RL-based navigation research across diverse robotic systems. The work builds on the IsaacLab [24] ecosystem and extends it with modular abstractions for tasks, robots, evaluation protocols, and deployment pipelines. RoboRAN is a direct response to the limitations of isolated, domain-specific benchmarks that hinder generalization and reproducibility in robot learning research.

Unlike previous chapters, which focused on single-domain systems (e.g., spacecraft, floating platforms), RoboRAN targets cross-medium extensibility—supporting thruster-based, wheeled, and aquatic robots under a unified RL stack. It enables flexible robot-task combinations, allowing RL practitioners to train and evaluate one policy per pair under standardized configurations, reward structures, and performance metrics.

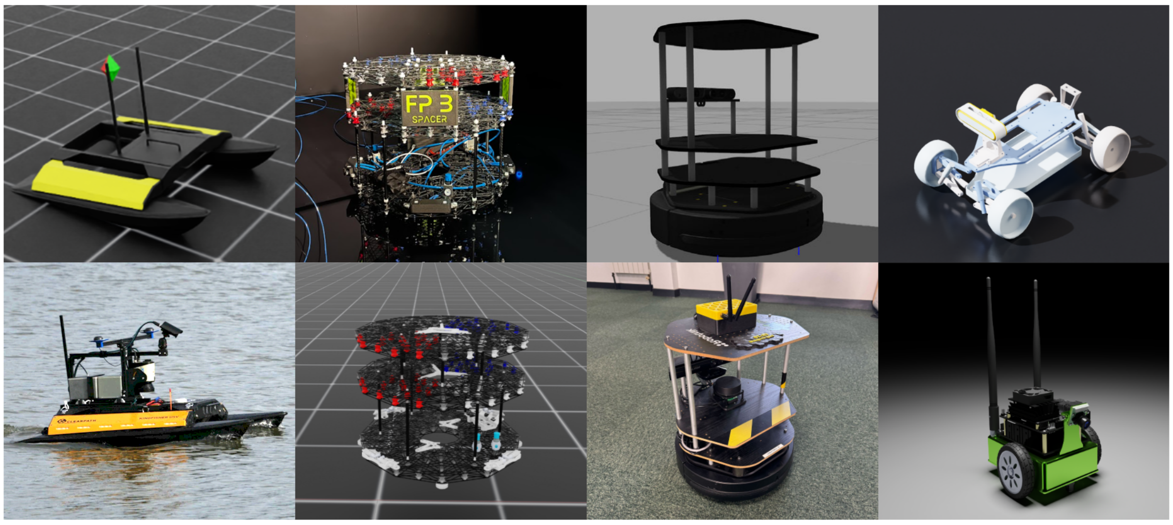


Figure 4.1: RoboRAN supports RL-based navigation tasks across a variety of robotic platforms, including the Kingfisher USV, Floating Platform, Turtlebot2, Leatherback, and JetBot. The first three were also evaluated in real-world conditions.

Acknowledgments. This framework was developed as part of a collaborative effort. While I led the design, implementation, and testing of the simulation stack—including all robot and task abstractions and domain-randomized training—I contributed only marginally to the real-world deployment of trained policies. These hardware experiments were conducted by my co-authors, who also developed the ROS 2-based deployment infrastructure and supervised the physical tests on the Floating Platform, USV, and Turtlebot2 robots.

Main Contributions

This chapter presents four key contributions, structured around design, training, deployment, and evaluation:

1. **A modular and scalable RL framework** that enables robot–task interchangeability through standardized APIs. This supports the seamless integration of new tasks and robotic platforms with minimal code duplication or structural changes.

2. **Sim-to-real transfer across three distinct physical robots**, leveraging a shared training pipeline with domain randomization and physically plausible noise models. Experiments span microgravity emulation (floating platform), aquatic (Kingfisher USV), and terrestrial (Turtlebot2) systems.

3. **The first open-source deployment interface for IsaacLab-trained policies**, bridging the gap between simulation and physical execution. The interface supports models trained with `rl_games` [14] and `skrl` [81], includes lightweight inference routines, and integrates with Docker [82] and ROS 2 [83] for field deployment.

4. **A unified evaluation suite for navigation tasks** across different domains, using consistent metrics (e.g., distance-to-goal, heading error, constraint violation rates). This allows reproducible comparisons and cross-domain policy benchmarking.

4.2 Related Work

Reinforcement Learning (RL) has emerged as a powerful paradigm for control tasks, demonstrating its ability to learn complex policies directly from sensorimotor data. This has led to significant advancements across various domains, including robotic manipulation [84], humanoid locomotion [85], and the control of legged robots [86]. While benchmark development has primarily centered on manipulation [87, 88, 89, 90], navigation remains a fundamental aspect of embodied intelligence that has gained increasing attention [91, 36].

Table 4.1: Comparison of RoboRAN with existing RL frameworks.

Benchmark	Domain Diversity	Task Types	Sim-to-Real	Robustness	Sensor / Env. Realism	Modularity	Backend
RoboRAN (Ours)	**Land / Water / Orbital**	**Navigation (4+)**	**✓(3 robots)**	**Partial**	**Moderate (realistic physics)**	✓	IsaacLab
RL-Nav (Xu et al., 2023) [98]	Ground	Navigation (1)	(1 robot)	Partial	Moderate (realistic physics)	Partial	Gazebo
Habitat 2.0 [93]	Indoor	Rearrangement / Manipulation	—	Limited	High (photorealism, articulation)	Partial	Bullet
RRLS [99]	Sim (MuJoCo)	Continuous control	—	✓(worst-case)	Low	Moderate	MuJoCo
Robust Gymnasium [100]	Sim (varied tasks)	Control / Safe RL / Multi-agent	—	✓(disruptions)	Medium	✓	Gymnasium
FlightBench [95]	Aerial (quadrotors)	Ego-vision navigation	✓(1 robot)	Partial	High (occlusion, motion blur)	—	Custom
BARN [94]	Ground	Reactive / Safe navigation	✓(1 robot)	✓(safety/uncertainty)	Medium / Low	Partial	ROS / Gym
iGibson 0.5 [97]	Indoor	Interactive navigation	—	Limited	High (realistic sensors)	Partial	Gibson + PyBullet
Aquatic Benchmark [96]	Water (aquatic)	Point-to-point navigation	—	Partial	Moderate (hydrodynamics, drift)	—	Unity3D

To facilitate learning-based navigation, numerous simulation environments and physics engines have been developed. Frameworks such as MuJoCo [6], PyBullet [45], Webots [63], and Isaac Gym [7] provide efficient and scalable platforms for RL training, but are often constrained to single-domain settings or specific robot morphologies. IsaacLab [24] extends Isaac Gym by supporting diverse robotic platforms, though it lacks both a structured evaluation suite for benchmarking RL policies across tasks and domains and the flexibility for interchangeable training of multiple tasks across multiple robots. Several recent benchmarks have addressed learning-based navigation under specific environmental and sensory constraints. Habitat [92, 93] targets high-level planning and mobile manipulation in photorealistic indoor environments. The BARN challenge [94] focuses on low-level control in cluttered scenes, while FlightBench [95] benchmarks ego-vision-based navigation for agile quadrotors. Aquatic navigation tasks are considered in [96], and iGibson 0.5 [97] provides an interactive benchmark in household environments. These efforts, however, are typically domain-specific and lack support for robot–task interchangeability or sim-to-real evaluation.

Robustness and generalization have become important in recent benchmark development. RRLS [99] introduces worst-case robust control evaluation using adversarial domains in MuJoCo, while Robust Gymnasium [100] defines modular disruption models across 60+ tasks. Although these environments are well-suited to studying resilience in policy learning, they remain simulation-bound and are limited in the diversity of robotic embodiments.

Prior work such as [98] identifies four key desiderata for RL in robotics (uncertainty handling, safety guarantees, data efficiency, and generalization) and provides valuable evaluation metrics and insights. The Gazebo-based simulation environment used in this work supports algorithm comparisons, but is limited to a single navigation task and robot. In contrast, RoboRAN emphasizes multi-robot, multi-domain flexibility within a high-throughput, GPU-accelerated simulation stack. Its modular design and extensibility enable future integration of safety-focused features such as policy and environment constraints.

While robustness and safety-centric studies like RRLS [99], Robust Gymnasium [100], and [98] focus on domain shifts or guarantees, RoboRAN provides complementary value by supporting simple real-world deployment and modular task-robot definitions, allowing practitioners to easily integrate different robot morphologies and new navigation tasks across diverse physical environments. Table 4.1 compares RoboRAN with existing RL benchmarks along axes such as domain diversity, task types, sim-to-real validation, robustness testing, realism, and modularity. We highlight our benchmark's cross-domain reach, support for real-world deployment, and modular structure enabling extensibility.

4.3 RoboRAN Overview

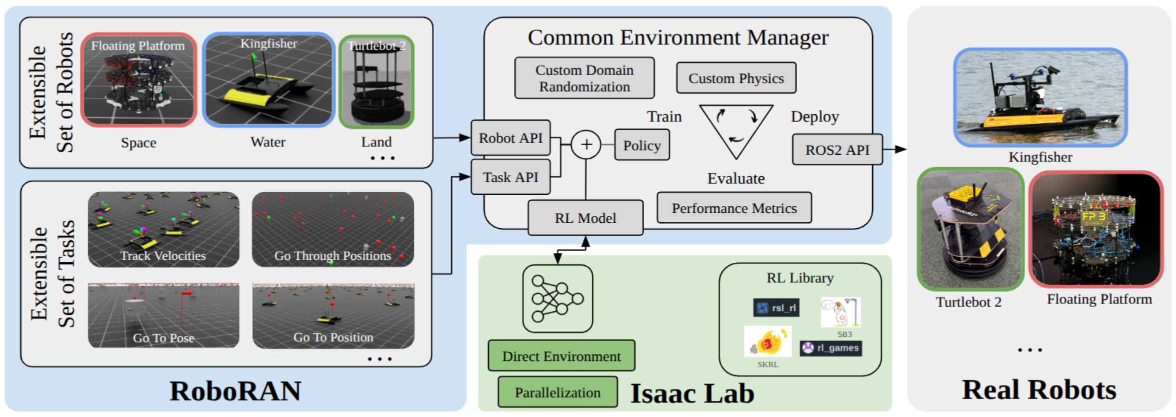


Figure 4.2: RoboRAN framework: the Navigation Tasks and Simulation Robots modules, together with a chosen RL library, are the only inputs required by the Environment Manager to train a policy in simulation, producing a ready-to-deploy network for the real robot counterpart.

RoboRAN is designed to train and evaluate robotic navigation tasks across a variety of operational settings. We introduce a unified structure where diverse robots can be evaluated on a shared set of tasks, using consistent interfaces and metrics. This design uniquely enables seamless interchangeability between agents and environments across different physical domains.

Our environment, formulated as a standard Markov Decision Process (MDP) [101], is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P(s' \mid s, a)$ is the transition probability function, $r(s, a)$ is the scalar reward function, and $\gamma \in [0, 1]$ is the discount factor. At each time step t, an agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to its policy $\pi(a_t \mid s_t)$, receives a reward $r_t = r(s_t, a_t)$, and transitions to a new state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. The environment thus provides at each step an observation $o_t \in \mathcal{O}$, a reward r_t, and a done signal $d_t \in \{0, 1\}$ indicating termination. The goal of the agent is to maximize the expected return $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]$ over episodes of length T.

Figure 4.2 depicts the main components of our framework:

The **Common Environment Manager** instantiates a specified task-robot pair, dynamically configuring the simulation assets, physics parameters, and task constraints based on the robot's specific characteristics and operational medium. This modular design is a key contribution of our framework, as it enables full interchangeability between tasks and robots, and sub-module addresses a distinct aspect of this flexibility.
The **Custom Physics** module computes custom dynamics and actuation forces through parameterized thruster/propeller models. For instance, it applies hydrodynamic and propeller models for surface vessels, or microgravity and frictionless dynamics for the floating platform. It also enables flexible rewards to platform-specific constraints, such as penalizing rapid thruster actuation.
The **Custom Domain Randomization** module implements the disturbances detailed in subsection 4.3.4 which are required to achieve sim-to-real transferability.
During training or evaluation, the *Performance Metrics* layer attach task-specific logging hooks enabling both on-line navigation metric updates and uniform post-hoc evaluation.
The **ROS2 API** simplifies policy deployment by using a ready-to-use inference node that exposes standard ROS2 interfaces, eliminating manual policy export steps that differ across RL libraries.

Together, these sub-modules enables flexibility and streamline the pipeline that shortens the loop between Training, Deployment and Simulation for diverse types of robots.

4.3.1 Robots

RoboRAN supports all robots presented in Figure 4.1. Among them, we selected three representative robots for further evaluation and field tests. Their characteristics and control properties are summarized in Table 4.2.

Land

We selected the Turtlebot2, an open source platform with differential drive system with non-holonomic dynamics. To demonstrate the extensibility of our stack, two more wheeled robots (Leatherback and JetBot) are supported and tested in simulation.

Water

We use the Kingfisher M200[1], a surface vessel with high inertial properties featuring a catamaran hull configuration and is driven by two fixed propellers, one on each hull. To simulate aquatic dynamics, we override IsaacLab's default planar physics with custom hydrodynamics and hydrostatics models, enabling more accurate motion behaviors influenced by water resistance.

Space

We implement a floating platform, a thruster-actuated system constrained to planar movement, mimicking spacecraft-like motion with force-based control. This robot, through air bearings mounted on its base, generates a microgravity effect by pushing a constant airflow against the floor to lift and levitate in a free-floating fashion. To simulate this effect, we implement a custom frictionless dynamics to approximate free-floating orbital behavior, which is not natively supported by IsaacLab.

[1]https://clearpathrobotics.com/

Table 4.2: Comparison of robot properties in RL navigation tasks. Control inputs are expressed as mathematical spaces.

Robot	Actuation Type	Degrees of Freedom	Control Input Space	Motion Constraints
Floating Platform	Thruster-based (binary)	3 (x, y, yaw)	$\{0,1\}^8$	No rolling/pitching, planar motion
Kingfisher	Water-based thrusters	3 (x, y, yaw)	$\mathbb{R}^2$ (left/right thrust)	Drag and inertia effects, smooth but slow
Turtlebot2	Differential drive	3 (x, y, yaw)	$\mathbb{R}^2$ (v, ω)	No lateral movement, limited turn speed

Table 4.3: Summary of navigation tasks, objectives, and observation space.

Task	Objective	Obs Dim	Obs Components	Obs Variables
GoToPosition	Reach a target position	6	Base Velocities, Target Info	$[v_x, v_y, \omega]$, $[d, \cos(\theta), \sin(\theta)]$
GoToPose	Reach a target 3DoF pose	8	Base Velocities, Target Info, Target Heading	$[v_x, v_y, \omega]$, $[d, \cos(\theta), \sin(\theta)]$, $[\cos(\psi), \sin(\psi)]$
GoThroughPositions	Follow a sequence of waypoints	$6 + 3n$	Base Velocities, Target Info, Future Goals	$[v_x, v_y, \omega]$, $[d, \cos(\theta), \sin(\theta)]$, $[d_i, \cos(\theta_i), \sin(\theta_i)]$
TrackVelocities	Maintain a set velocity	6	Error Terms, State	$[e_v, e_l, e_\omega]$, $[v_x, v_y, \omega]$

These three representative robots (FloatingPlatform, Kingfisher, Turtlebot2) are described in the main evaluation; additional wheeled platforms supported by RoboRAN are summarized next.

Additional wheeled platforms

In addition to the TurtleBot2 (used for real-world deployment and sim-to-real evaluation), RoboRAN supports two further wheeled platforms in simulation: the **Leatherback** (NVIDIA research platform) and the **JetBot** (open-source educational robot). Including multiple wheeled platforms highlights RoboRAN's modular robot API and allows us to evaluate how learned navigation controllers generalize across differential- and Ackermann-style locomotion without modifying the task definitions or training pipeline.

4.3.2 Tasks

While IsaacLab designs tasks around fixed robot models, RoboRAN decouples robot and task definitions, allowing consistent training and evaluation pipelines for any supported robot across all tasks. Our framework includes a suite of four navigation tasks designed to evaluate robotic motion in different environments and actuation methods. Each task leverages a structured observation space, detailed in Table 4.3, providing essential state information such as *base velocities* $[v_x, v_y, \omega]$, which capture the linear and angular velocity of the agent. To enhance temporal reasoning, we augment the observation vector with the previous action a_{t-1}, enabling the policy to infer dynamic transitions and improve stability in control. In all tasks, the observations are provided in the robot's own frame and apply Domain Randomization that mimics the noise of real sensors commonly used for state estimation.

GoToPosition task requires the agent to reach a randomly initialized 2D position using the *target information* $[d, \cos(\theta), \sin(\theta)]$, representing the Euclidean distance and bearing to the goal. The relative angular position of the goal, is provided as a cos and sin of the angle to ensure the observations are continuous [102].

GoToPose task is similar to *GoToPosition*, but also requires orientation alignment. Therefore, the observation space incorporates the *target heading* as $[\cos(\psi), \sin(\psi)]$ to provide the angular distance to the desired final orientation.

GoThroughPositions task involves sequential navigation through a series of n waypoints, introducing *future goals* $[d_i, \cos(\theta_i), \sin(\theta_i)]$ in the observation space to ensure smooth trajectory planning.

TrackVelocities task requires the agent to follow a time-varying velocity reference in both linear and angular components. The observation space includes *velocity error terms* $[e_v, e_l, e_\omega]$ capturing deviations from the desired forward, lateral, and angular velocities. While no explicit path planner is embedded in the control policy, the velocity references can be derived from any arbitrary trajectory generator, including spline interpolators or MPC-based local planners. In this sense, the generator acts as a lightweight path planner, and the learned policy serves as a robust low-level controller that tracks planned motion commands across diverse robot morphologies and terrains.

These tasks provide a flexible evaluation suite for RL-based navigation, adaptable to use-cases such as autonomous docking, inspection, formation control, and trajectory tracking. While the core experiments in this paper focus on fundamental control-oriented tasks without obstacles or perceptual inputs, the framework is designed to support more complex scenarios. Thanks to its modular architecture, features such as obstacle avoidance, moving targets, and real-world sensing modalities can be integrated with minimal code changes.

4.3.3 Reward Formulation

The reward function combines task-specific objectives with general regularization terms to ensure consistent goal-directed behavior and control smoothness across robot types. Its unified form is shown in Eq. 4.1, where d_p, d_h, and d_b denote the distance to the goal position, heading misalignment, and boundary proximity respectively. The terms v_j represent linear and angular velocities clipped to task-defined ranges, Δd_p is the signed progress along the goal direction, and $\mathbb{1}_{\text{goal}}$ provides a terminal bonus when the goal is reached. The term r_t^{robot} adds optional robot-specific shaping such as control regularization.

$$r_t = \sum_{i \in \{p,h,b\}} w_i e^{-d_i/\lambda_i} + \sum_{j \in \{v,\omega\}} w_j \operatorname{clip}(v_j, v_{\min}, v_{\max}) + w_{\text{pg}} \Delta d_p + w_{\text{bns}} \cdot \mathbb{1}_{\text{bonus}} + r_t^{\text{robot}} \tag{4.1}$$

The weights w_i, w_j, w_{pg}, and w_{succ} vary by task, and are denoted in Table 4.4 as α_i for *GoToPosition*, β_i for *GoToPose*, ϕ_i for *GoThroughPositions*, and γ_i for *TrackVelocities* with decay constants λ_i shared across tasks. For example, $\alpha_1 e^{-d_p/\lambda_1}$ encourages position convergence in *GoToPosition*, while $\beta_1 e^{-d_p/\lambda_1} e^{-d_h/\lambda_4}$ jointly rewards alignment in *GoToPose*. Similarly, progress is captured by $\phi_1 \Delta d_p$ in *GoThroughPositions*, and *TrackVelocities* uses $\gamma_i e^{-e_i/\lambda_5}$ to penalize velocity tracking errors. All coefficients were tuned for balance and stability across robots, and are reported in Table 4.4 for full reproducibility.

4.3.4 Domain Randomization

To support sim-to-real transfer, we apply domain randomization in three key areas: (i) robot mass properties (mass, center of mass location, inertia tensor), (ii) actuation noise via Gaussian perturba-

Table 4.4: Reward parameters for PPO training. Task-specific coefficients and decay values used in Equation 4.1.

GoToPosition		**GoToPose**		**GoThroughPos.**		**TrackVelocities**	
α_{i1} (pos)	1.0	β_{i1} (pose align)	1.0	ϕ_{i1} (progress)	1.0	γ_{i1} (lin vel err)	-1.0
α_{i2} (head)	0.25	β_{j1} (lin vel)	-0.05	ϕ_{i2} (head)	0.05	γ_{i2} (ang vel err)	-0.5
α_{j1} (lin vel)	-0.05	β_{j2} (ang vel)	-0.05	ϕ_{j1} (lin vel)	0.0	γ_{i3} (bonus)	0.0
α_{j2} (ang vel)	-0.1	β_{bns1} (boundary)	-10.0	ϕ_{j2} (ang vel)	-0.05	γ_{bns1} (boundary)	-10.0
α_{bns1} (bonus)	-10.0	β_{pg1} (progress)	0.2	ϕ_{bns1} (bonus)	-10.0	—	—
$\lambda_1 = 1.0$ (dist)		$\lambda_2 = 0.25$ (head)		$\lambda_3 = 1.0$ (bnd)		$\lambda_4 = 1.0$ (vel err)	

tions to commanded actions, and (iii) external disturbances modeled as random wrenches applied to the robot's base. The amount of randomization is chosen at random at every reset. We ensure reproducibility through a per-environment seed-controlled random number generation (RNG) using Warp [47], allowing fine-grained domain randomization across parallel training environments. We apply moderate randomizations to simulate real-world uncertainties. For the Turtlebot2, we vary its mass by ± 0.1 kg and CoM by ± 0.05 m (std = 0.01), reflecting typical manufacturing variances. For the Kingfisher, which operates in a fluid environment, we use broader mass (± 2.0 kg) and CoM (± 0.05 m) perturbations, and apply random body wrenches (forces $\in [0, 0.25]$ N, torques $\in [0, 0.05]$ Nm) to account for water currents. For the Floating Platform, we use intermediate mass (± 0.25 kg) and similar CoM and wrench ranges to model small-scale system variations and external disturbances.

4.3.5 Training

We train RL algorithms using the skrl [81] library, with PPO [28] as the training algorithm. PPO was selected due to its stability in high-dimensional continuous control and its widespread use in RL robotics settings. Rather than comparing algorithms, our focus is on demonstrating the decoupling of robot-task development within a unified framework. All experiments were run on a single NVIDIA RTX 4090. PPO was trained with default hyperparameters, and each robot-task pair converged in $\sim$ 15 minutes on average. The final set of policies trained and used for evaluation

Table 4.5: **Task success criteria and thresholds.** Each task defines success based on reaching position, orientation, velocity, or time-based constraints.

Task	Success Condition	Threshold
GoToPosition	Final position error $\leq \epsilon_p$	$\epsilon_p = 0.1m$
GoToPose	ϵ_p and orientation error $\leq \epsilon_\theta$	$\epsilon_p = 0.1m$, $\epsilon_\theta = 10°$
GoThroughPositions	Waypoints reached within ϵ_{tp}	$\epsilon_{tp} = 0.2m$
TrackVelocities	Maintain ϵ_v, ϵ_w	$\epsilon_v = 0.2m/s$, $\epsilon_w = 10°/s$

are 12 (3 robots and 4 tasks).

4.3.6 Deployment (summary)

While I did not contribute directly to the real-robot deployment, our co-authors developed and tested a ROS2-based deployment stack enabling policies trained in IsaacLab to control the Floating Platform, Turtlebot2, and Kingfisher in the field. The architecture includes state abstraction, policy inference, and goal interface modules, integrated with Docker and ROS 2. Detailed deployment and system integration are available in the original publication and appendix.

4.4 Simulation Results

We evaluate our RL-trained policies in simulation across representative robot–task pairs, reporting results for three multi-domain robots: *Floating Platform*, *Kingfisher*, and *Turtlebot2*. These cover a diverse range of actuation models and navigation challenges, ensuring a broad evaluation scope.

4.4.1 Experimental Setup

Each policy is trained for 3200 epochs using PPO, over 5 random seeds per robot-task pair. During evaluation, we use GPU-accelerated IsaacLab rollouts with parallel environments to collect performance data from 4096 evaluation episodes per run. All results are reported as mean ± std across environments. We define task-specific success as percentage of trajectories that satisfy the task specific metrics. Each metric is associated to a set of thresholds (ϵ_p, ϵ_θ, ϵ_{tp}, ϵ_v, and ϵ_w) that are listed in Table 4.5):

GoToPosition: distance to goal $< \epsilon_p$ within a fixed time budget.

GoToPose: both distance $< \epsilon_p$ and heading error $< \epsilon_\theta$ must be satisfied.

GoThroughPositions: count of waypoints reached in sequence within ϵ_{tp} tolerance before time-out.

TrackVelocities: mean absolute tracking error for linear and angular velocity must stay below ϵ_v and ϵ_w.

In addition to success rate (defined as the percentage of episodes that meet task-specific thresholds), we report continuous evaluation metrics to capture control precision and stability:

Final Distance Error (m): Euclidean distance to the goal at the end of the episode.

Heading Error (°): Absolute orientation difference at the final timestep (GoToPose only).

Time to Target (s): Duration required to reach the target precision threshold. Lower values reflect faster convergence.

Velocity Tracking Error (m/s): Mean absolute error between target and actual linear/angular velocities (TrackVelocities only).

Control Signal Variation (unitless): Standard deviation of control signals over the episode, reflecting smoothness or abruptness of control.

Goals Reached: Total number of intermediate targets successfully reached during sequential waypoint tasks (GoThroughPositions).

All these metrics are aggregated in Table 4.7, enabling a multi-dimensional comparison across tasks and robots.

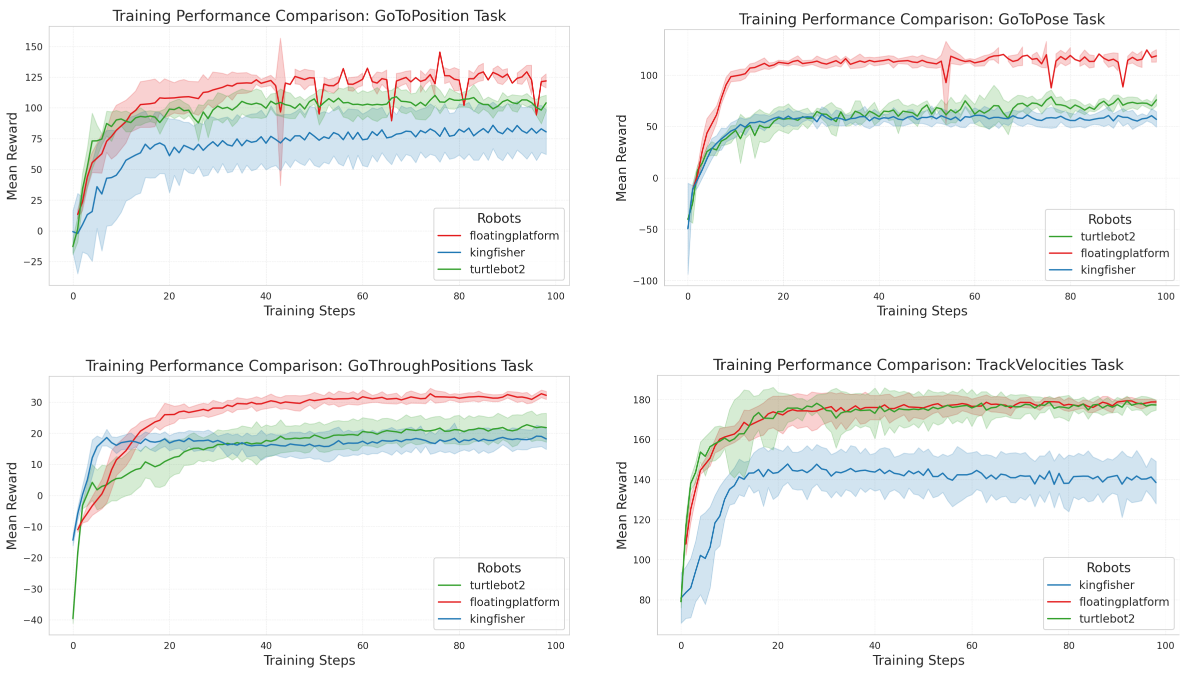


Figure 4.3: **Learning curves** showing rewards (mean $\pm$ std) over 5 seeds per robot, compared by task.

4.4.2 Training Efficiency and Learning Trends

Figure 4.3 shows the training reward across 5 seeds, highlighting learning speed and convergence per robot. The *FloatingPlatform* achieves the highest asymptotic rewards, benefiting from direct actuation despite its discrete thrust model. *Turtlebot2* converges reliably with moderate final returns, aided by low-dimensional control. *Kingfisher* shows slower and less stable learning, likely due to its hydrodynamic complexity and inertia. Table 4.6 reports average wall-clock time per training run. The *Kingfisher* requires the longest training time, consistent with its complex dynamics. *Turtlebot2* trains fastest among wheeled platforms. The unexpectedly short time for the *FloatingPlatform* suggests beneficial interaction between its discrete control structure and IsaacLab's GPU-based parallelization. These differences motivate further study into simulation efficiency under varying robot dynamics.

Table 4.6: **Wall-clock time per robot–task pair (mean $\pm$ std over 5 seeds) in minutes [m].**

Task	Floating Platform	Kingfisher	Turtlebot2
GoToPosition	7.35 ± 0.02m	13.91 ± 0.19m	11.55 ± 0.14m
GoToPose	5.46 ± 0.00m	—	11.53 ± 0.16m
TrackVelocities	5.08 ± 0.18m	13.47 ± 0.07m	11.28 ± 0.02m
GoThroughPositions	5.51 ± 0.13m	13.47 ± 0.35m	11.20 ± 0.03m

Table 4.7: **Simulation evaluation metrics per task and robot (mean $\pm$ std across 4096 envs, PPO–skrl).** Metrics include: success rate (%), final distance error (m), heading error (°), time to target (s), velocity tracking error (m/s), control signal variation (unitless), and number of goals reached. "—" indicates non-applicable metrics.

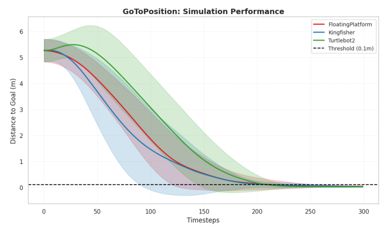
Task	Robot	Success Rate ↑	Dist Err ↓	Heading Err ↓	Time to Target ↓	Lin Vel Err ↓	Ang Vel Err ↓	Ctrl Var ↓	Goals Reached ↑
GoToPosition	FloatingPlatform	0.94 ± 0.04	0.05 ± 0.01	—	87.05 ± 3.38	—	—	0.62 ± 0.04	—
	Kingfisher	0.59 ± 0.29	1.06 ± 0.73	—	176.11 ± 60.86	—	—	0.75 ± 0.45	—
	Turtlebot2	0.99 ± 0.01	0.07 ± 0.00	—	92.60 ± 4.67	—	—	0.43 ± 0.26	—
GoToPose	FloatingPlatform	0.99 ± 0.01	0.02 ± 0.01	0.78 ± 0.01	92.38 ± 2.59	—	—	0.69 ± 0.05	—
	Kingfisher	0.66 ± 0.09	0.23 ± 0.06	7.07 ± 3.08	126.80 ± 31.29	—	—	0.48 ± 0.29	—
	Turtlebot2	0.84 ± 0.04	0.14 ± 0.01	4.39 ± 1.56	131.49 ± 2.16	—	—	0.63 ± 0.38	—
GoThroughPositions	FloatingPlatform	1.00 ± 0.00	2.35 ± 0.25	—	65.18 ± 1.03	—	—	0.32 ± 0.04	13.57 ± 0.33
	Kingfisher	1.00 ± 0.00	2.41 ± 0.79	—	93.29 ± 18.56	—	—	0.43 ± 0.24	10.70 ± 2.84
	Turtlebot2	1.00 ± 0.00	1.79 ± 0.05	—	101.50 ± 12.25	—	—	0.13 ± 0.06	11.01 ± 0.12
TrackVelocities	FloatingPlatform	0.93 ± 0.18	—	—	—	0.05 ± 0.07	0.03 ± 0.01	0.45 ± 0.04	—
	Kingfisher	0.48 ± 0.03	—	—	—	0.03 ± 0.00	0.24 ± 0.02	0.62 ± 0.37	—
	Turtlebot2	0.77 ± 0.01	—	—	—	0.02 ± 0.01	0.11 ± 0.01	0.15 ± 0.09	—

4.4.3 Task Success and Performance Analysis

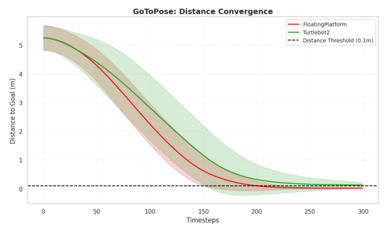
To complement the reward learning curves shown in Figure 4.3, we conduct a detailed quantitative evaluation across all robot-task pairs. This evaluation uses standardized success metrics and control efficiency indicators (Table 4.7) collected over 4096 parallel trajectories per setting.

Figure 4.4 presents the convergence curves for each robot-task pair. Shared tasks (*GoToPosition*, *GoThroughPositions*, and *TrackVelocities*) are plotted together for comparison, while specialized tasks (*GoToPose*) are shown separately.

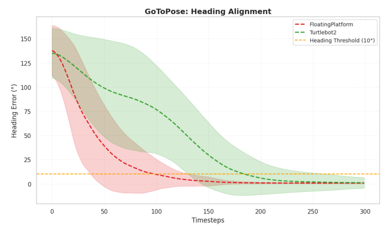
GoToPosition and GoToPose The *Turtlebot2* achieves the highest success rate in *GoToPosition* with **0.99 ± 0.01**, benefiting from its differential-drive system and precise low-speed control. The *FloatingPlatform* follows with 0.94 ± 0.04, while the *Kingfisher* lags at 0.59 ± 0.29 due to inertia and limited turning agility. These trends are confirmed in Figure 4.4a, where Turtlebot2 reaches



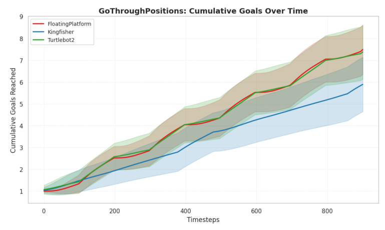
(a) GoToPosition: all robots.



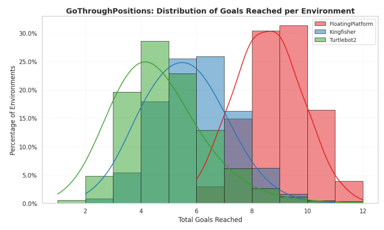
(b) GoToPose (distance): FloatingPlatform, Turtlebot2.



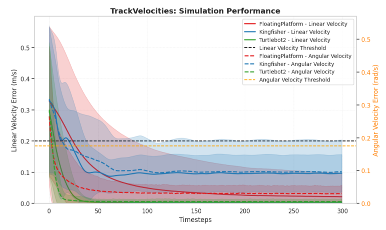
(c) GoToPose (heading): FloatingPlatform, Turtlebot2.



(d) GoThroughPositions: goals achieved (all robots).



(e) GoThroughPositions: goals distribution (4096 evaluation envs, all robots).



(f) TrackVelocities: linear (black) and angular (orange) velocity errors (all robots).

Figure 4.4: **Simulation results across robots and tasks.** Performance comparisons for *GoToPosition*, *GoToPose*, *GoThroughPositions*, and *TrackVelocities*. (a) All robots for GoToPosition. (b, c) FloatingPlatform and Turtlebot2 on GoToPose (distance, heading). (d) Number of goals achieved in GoThroughPositions (all robots). (e) Goals distribution over 4096 parallel evaluation environments (all robots). (f) Linear velocity error in TrackVelocities (all robots).

the goal region fastest, followed by FloatingPlatform and Kingfisher. In the *GoToPose* task, both *FloatingPlatform* and *Turtlebot2* succeed in reaching the target, with success rates of 0.99 ± 0.01 and 0.84 ± 0.04, respectively. *Kingfisher* is not evaluated due to its lack of heading control. FloatingPlatform achieves superior orientation control, with a heading error of **0.78° ± 0.01°**, compared to Turtlebot2's 4.39° ± 1.56°, as shown in Figure 4.4c. Distance convergence is also faster and more precise for FloatingPlatform (0.02 ± 0.01 m vs 0.14 ± 0.01 m, Fig. 4.4b).

GoThroughPositions All three robots successfully complete partial trajectories (100% success rate), but differ in the number of goals reached. FloatingPlatform achieves the highest average at **13.57** ± **0.33**, while Turtlebot2 and Kingfisher reach **11.01** ± **0.12** and **10.70** ± **2.84** respectively. These differences are reflected in Figure 4.4d (cumulative goals) and Figure 4.4e (distribution), where *FloatingPlatform*'s performance is both higher and more consistent. Turtlebot2 shows smoother trajectories but fails to reach all waypoints within the time constraints, while Kingfisher's performance is more variable due to inertia limiting sharp turns.

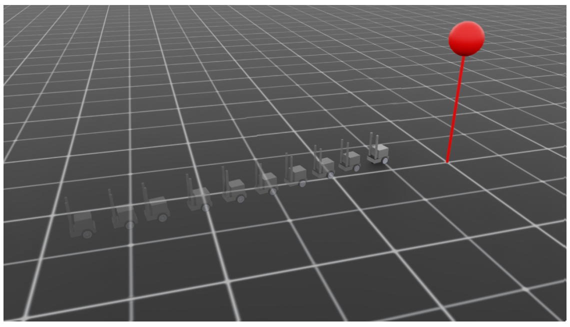
TrackVelocities *FloatingPlatform* demonstrates moderate success in tracking target velocities. Its linear velocity error is **0.05** ± **0.07**, and angular velocity error is **0.03** ± **0.01**, better than both Turtlebot2 (0.02 ± 0.01, 0.11 ± 0.01) and Kingfisher (0.03 ± 0.00, **0.24** ± **0.02**), as detailed in Table 4.7 and shown in Figure 4.4f. The high angular error for Kingfisher highlights the difficulty of fast heading corrections in water due to drag and momentum.

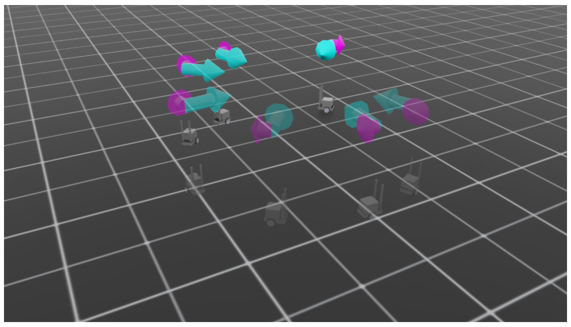
Success Rate Summary Table 4.7 confirms these observations across tasks. Turtlebot2 dominates in *GoToPosition*, FloatingPlatform leads in *GoThroughPositions*, and both Turtlebot2 and FloatingPlatform perform comparably in *GoToPose*. In *TrackVelocities*, all robots achieve reasonable success, but Kingfisher exhibits the highest angular tracking errors, limiting its overall precision. These trends are visible in Figure 4.4, supporting our conclusion that control effectiveness varies not only across robots but also across tasks.

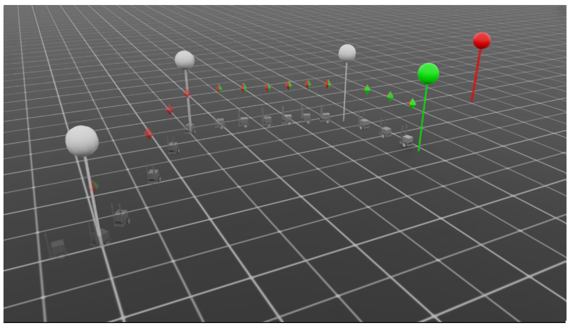
Additional results Figure 4.5 shows the other wheeled robots available in the simulation stack, while they solve the the available tasks GoToPosition, TrackVelocities, GoThroughPositions, and GoToPositionWithObstacles.

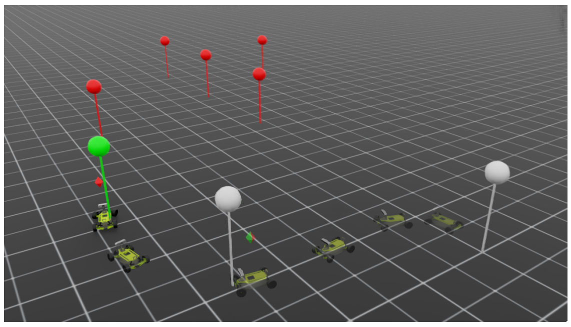
Extended Environments and Task Variations RoboRAN supports more complex environments and objectives. Its modular task interface allows users to easily introduce obstacles, moving targets, or perception-driven goals without altering the robot implementation or training pipeline.

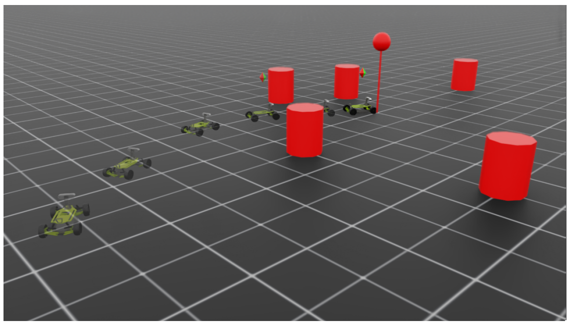
To demonstrate flexibility under environmental constraints, we evaluate the *GoToPosition* task with static obstacles placed between the robot and its sequence of goals. Using the same reward structure and PPO hyperparameters as in the base task, without task-specific tuning, all three robots (FloatingPlatform, Kingfisher, Turtlebot2) successfully learn to navigate around obstacles


(a) JetBot — GoToPose


(b) JetBot — TrackVelocities


(c) JetBot — GoThroughPositions


(d) Leatherback — GoThroughPositions


(e) Leatherback — GoToPositionWithObstacles

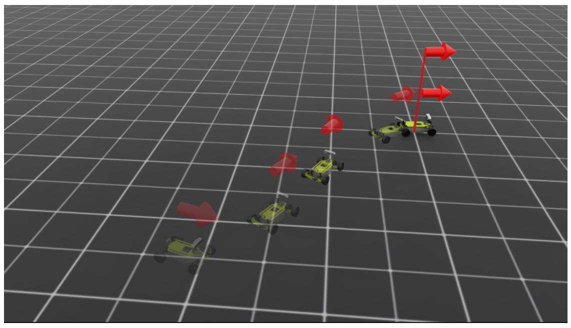

(f) Leatherback — GoToPose

Figure 4.5: Examples of JetBot and Leatherback in simulated tasks.

(Fig. 4.7). Training curves (Fig. 4.6) show rapid improvement followed by stable convergence, with final mean rewards over the last 50 steps of ~ 93.4 (Kingfisher), ~ 75.4 (Turtlebot2), and ~ 73.0 (FloatingPlatform). The observation space is extended from the base task by appending the positions of the three closest objects in addition to the original six dimensions.

Additional complex scenarios, such as manipulation-inspired tasks (e.g., push-block for

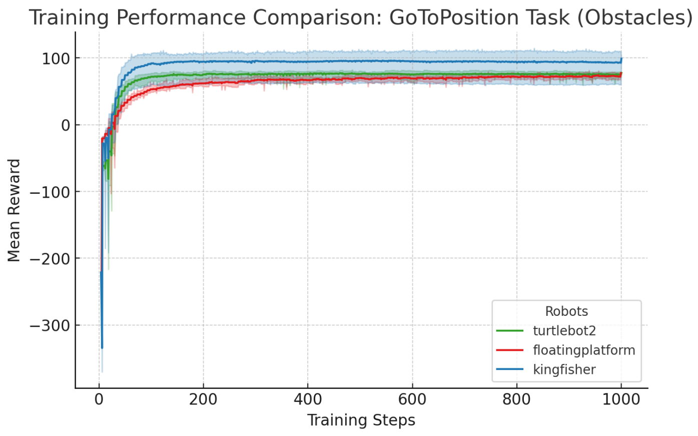


Figure 4.6: Training performance on *GoToPosition* with static obstacles for three robots. Curves show mean and std reward over 10 seeds. All robots learn stable obstacle-avoiding behaviors; the final-50-step mean rewards are **Kingfisher** ≈ 93.4, **Turtlebot2** ≈ 75.4, and **FloatingPlatform** ≈ 73.0.

wheeled robots), are also supported but are left out of the main scope. These can be integrated with minimal code changes and will be shared in future iterations of the framework.

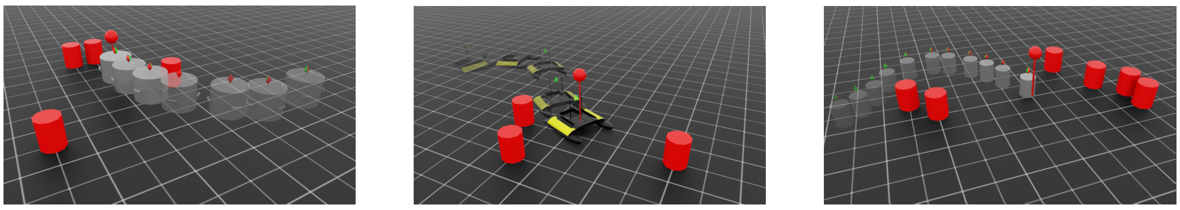
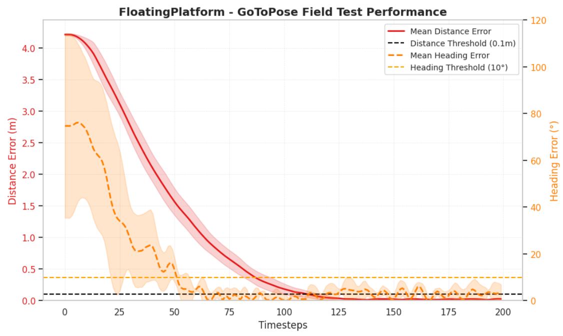


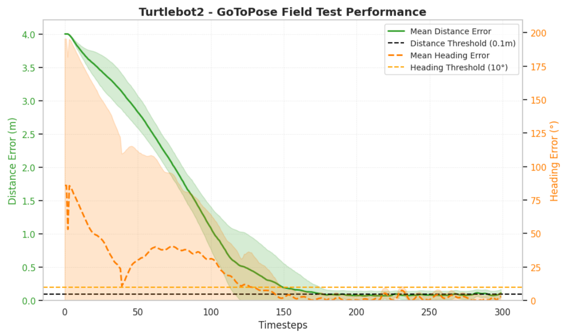
Figure 4.7: FloatingPlatform, Kingfisher, and Turtlebot2 in GoToPositionWithObstacles

4.4.4 Discussions

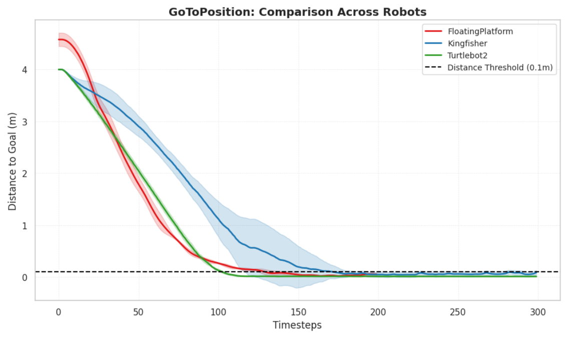
While RL policies achieve high success rates, several robot-specific failure cases were observed. The *FloatingPlatform* experiences oscillations near target positions due to force-based control lag. The *Kingfisher* struggles with understeering in tight waypoint sequences, making sharp turns difficult. The *Turtlebot2*, despite overall fast learning, exhibits difficulty in precise in-place rotations, leading to longer turning maneuvers in the *GoToPose* task. These challenges highlight the need for refined reward shaping and constraint definitions to improve task execution. Overall, the successful train-



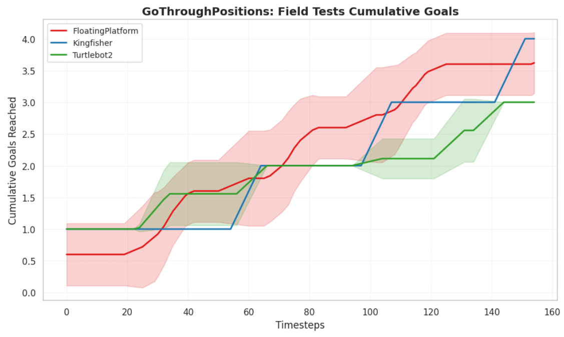
(a) GoToPose: FloatingPlatform.



(b) GoToPose: Turtlebot2.



(c) GoToPosition: all robots.



(d) GoThroughPositions: all robots.

Figure 4.8: **Field test results for navigation tasks.** Performance evaluation for *GoToPose* (FloatingPlatform, Turtlebot2), *GoToPosition* (all robots), and *GoThroughPositions* (all robots).

ing of diverse robots on shared tasks, despite their differing actuation and mobility constraints, demonstrates the viability of **unified cross-medium pipeline**. The *Turtlebot2*'s rapid convergence, the *FloatingPlatform*'s discrete thrust limitations, and the *Kingfisher*'s inertia-driven control difficulties highlight the importance of evaluating RL policies across heterogeneous platforms.

4.5 Sim-to-Real Results (Summary)

The deployment of RoboRAN policies on physical robots (FloatingPlatform, Kingfisher, and Turtlebot2) was led by collaborators and is described in detail in the original paper. While I contributed to the simulator side and overall benchmarking structure, the real-world integration and tests were outside the main scope of my thesis contribution. Nonetheless, their results confirm the effectiveness of domain-randomized training and structured evaluation pipelines for policy transfer.

4.6 Conclusions

This chapter presented **RoboRAN**, a unified and scalable reinforcement learning framework designed to benchmark and compare navigation policies across robots with different locomotion modalities—wheeled, aquatic, and air-bearing systems. By decoupling robot and task definitions through standardized interfaces and modular simulation backends, RoboRAN enables consistent training, evaluation, and deployment pipelines, addressing the long-standing fragmentation in RL for robotics.

The chapter demonstrated the effectiveness of this modular design across four core navigation tasks—GoToPosition, GoToPose, TrackVelocity, and GoThroughPositions—and three heterogeneous robotic platforms: the Turtlebot2 (wheeled), the Kingfisher USV (aquatic), and the Floating Platform (microgravity emulator). Policies trained entirely in simulation were evaluated using uniform metrics and thresholds, enabling side-by-side comparisons across embodiments. The resulting policies achieve high success rates and centimeter-level accuracy in both simulated and real-world conditions, showcasing the framework's potential for *sim-to-real generalization*.

While real-world deployment of the policies revealed strong qualitative transfer, residual performance gaps—especially in heading alignment and failure recovery under high-momentum transitions—highlighted the importance of dynamics fidelity and morphology-aware domain randomization. These observations reinforce the thesis' broader theme: robust real-world generalization depends not only on algorithmic strength but also on the structured design of simulation environments and training protocols.

Importantly, this work also underscores a key methodological insight: *policy benchmarking across modalities* is not merely a matter of comparing numerical metrics, but requires a carefully structured simulation and evaluation interface. RoboRAN addresses this need and provides a principled foundation for future research in generalization, transfer learning, and multitask control.

Looking forward, RoboRAN's modularity opens the door to several extensions: support for new robots (e.g., aerial or articulated systems), richer observation modalities (e.g., vision-based inputs), and more complex navigation tasks involving obstacle avoidance, constrained motion, or adversarial environments. Moreover, RoboRAN provides a fertile testing ground for algorithmic innovations such as curriculum learning, policy distillation, or learning-to-learn approaches. In this sense, RoboRAN represents a convergence point between robust engineering practices and scalable RL research, and contributes directly to the thesis' broader goal of enabling reliable deep reinforcement learning for autonomous robotics across real-world domains.

Chapter 5

FALCON-S – Fixed Wing Aerodynamics And Control Suite

5.1 Introduction

The previous chapters introduced modular frameworks for reinforcement learning (RL) in robotic navigation, spanning floating platforms, ground vehicles, and water-surface vessels. These contributions emphasized sim-to-real transfer, scalable multi-robot pipelines, and modularity in simulation and control. In this chapter, we expand the scope of autonomous control to fixed-wing aerial vehicles operating in ground effect—a particularly challenging and underexplored domain for deep reinforcement learning.

The motivation behind this work stems from the observation that most existing simulators either oversimplify aerial vehicle dynamics or are tailored to pilot training and do not provide the flexibility and scalability required for modern RL-based control. For instance, while tools like JSBSim [103], X-Plane [21], and Flightmare [104] offer partial realism, they lack essential features such as GPU-accelerated simulation, modular controller integration, or fine-grained modeling of aerodynamic phenomena like ground effect. As a result, deploying RL policies to real aerial platforms remains limited by the fidelity gap between simulation and real-world dynamics.

To address these limitations, we introduce **FALCON-S** (Fixed-wing Aerial Learning and Control with Open-Source Newtonian Simulation), a modular, physics-rich simulation framework designed for training and benchmarking both learning-based and classical control algorithms on fixed-wing aircraft in low-altitude flight scenarios. The key features of FALCON-S include:

- **High-throughput dual-backend simulation:** A GPU-accelerated physics engine built on NVIDIA Warp and a CPU fallback, allowing efficient training and evaluation at scale—with support for millions of parallel environments at real-time step rates.
- **Modular control stack:** Support for interchangeable controllers including Proximal Policy Optimization (PPO), DreamerV3, LQR, and MPPI, enabling controlled benchmarking across control paradigms.
- **Physically-grounded dynamics:** Accurate modeling of aerodynamic forces (including ground effect), actuator and sensor dynamics, and environmental disturbances such as wind and noise.
- **Cross-platform validation:** Interfaces to MATLAB/Simulink and X-Plane for co-simulation and validation in commercial-grade high-fidelity environments.

FALCON-S builds upon the prior contributions of this thesis—namely the emphasis on simulation fidelity, control generalization, and benchmarkability—but shifts the focus to aerial vehicles, which pose new challenges in terms of high-speed dynamics, actuator constraints, and sensitivity to environmental effects. The framework serves both as a scientific tool for controlled experimentation and as an engineering platform for RL deployment pipelines.

By offering fully open and customizable infrastructure, FALCON-S aims to bridge the gap between traditional flight control and modern machine learning, supporting the development of robust controllers for aerial autonomy. The remainder of this chapter details the simulator design, flight tasks, physics modeling, and experimental benchmarking, concluding with a discussion of limitations and directions for future aerial learning research.

Acknowledgments. The work presented in this chapter is the result of collaborative efforts. While the simulation architecture and reinforcement learning experiments were primarily conducted by the author, key contributions to the development of classical control algorithms and the integration with MATLAB/Simulink backends were carried out in close collaboration with co-authors. In particular, we acknowledge the substantial technical support and insight provided by Pedro Lima, whose work was instrumental in enabling the comparative evaluation between traditional and learning-based controllers.

5.2 Related Work

Simulation of fixed-wing flight dynamics.

Simulators such as *JSBSim*, *FlightGear*, and *X-Plane* have long supported fixed-wing aircraft modeling, but are primarily designed for pilot training or certification, and lack native support for reinforcement learning or scalable training. Recent research platforms such as QPlane [105] and NeuralPlane [106] address this limitation by exposing lightweight and configurable interfaces suitable for policy learning. QPlane wraps JSBSim for Gym-based RL experiments, while NeuralPlane introduces a parallel GPU-based pipeline for efficient large-scale simulation. However, both frameworks simplify critical aspects of flight dynamics, often using 3DoF or attitude-only models, with limited actuator fidelity and minimal environmental realism.

Flight control benchmarks and learning environments.

While platforms like AirSim [107], Flightmare [104], and RotorS [108] have successfully advanced learning-based control for multirotor drones, fixed-wing benchmarks remain scarce due to the increased complexity of forward-flight dynamics, non-holonomic constraints, and sensitivity to external disturbances. Most existing learning environments focus on hover-capable vehicles, leaving limited support for lift-based platforms. Our work addresses this gap by introducing a unified simulation suite tailored to fixed-wing aircraft operating near the ground, combining realistic 6DoF dynamics with actuator and sensor models, ground effect, and wind disturbances. It supports both classical and learning-based controllers and achieves millisecond-scale single-step performance through GPU acceleration, enabling rigorous, scalable, and physically grounded benchmarking.

Combining classical control with deep learning.

There is increasing interest in combining optimal control methods with reinforcement learning [109, 110]. Works like Basescu et al. [111] show how model predictive control can be extended with learned aerodynamic models to achieve aggressive post-stall landings. Similarly, residual RL and hybrid policy architectures have been used to improve control generalization while retaining safety guarantees. Our environment supports both classical baselines and learning-based controllers, enabling direct comparisons and hybrid control studies under consistent dynamics.

Modular, accelerated simulators for RL.

Efficient learning requires simulators that are both fast and customizable. GPU-accelerated simulators like WarpDrive [112] and IsaacGym [7] have become increasingly popular in robotics research, but few have targeted flight vehicles. Flightmare [104] provides GPU acceleration via Unity, yet focuses on quadrotor dynamics. Our Warp-based simulator offers domain-specific GPU acceleration for fixed-wing vehicles with detailed aerodynamics, supporting large-scale training without compromising physical realism.
To contextualize our contribution, Table 5.1 presents a detailed comparison between our simulation platform and several prominent aircraft simulation frameworks, including NeuralPlane [106], QPlane [105], JSBSim [103] and XPlane [21]. While prior systems offer valuable capabilities, such as high-fidelity physics engines, Gym-compatible RL integration, or large-scale parallelism, most fall short in supporting near-ground aerodynamic effects or unified, extensible control pipelines. In contrast, our platform combines realistic 6-DoF flight dynamics with explicit ground effect

Table 5.1: Comparison of our platform with existing aircraft simulation frameworks. Our system combines realistic near-ground fixed-wing aerodynamics with modular flight tasks and supports advanced controllers in a reinforcement learning context, with rich sensor and actuator modeling, while enabling expandability for sim-to-real transfer. [✓] fully supported, [*] partially or optionally supported, [–] not supported.

Feature	Ours	NeuralPlane	QPlane	JSBSim	XPlane
Open-source	✓	✓	✓	✓	–
Physics-based FDM	✓ (WIG, 6DoF)	✓ (fixed-wing only)	✓ (JSBSim/XPlane)	✓	✓
Ground Effect Model	✓ (semi-empirical)	–	* (depends on JSBSim)	*	✓
GPU Acceleration	✓ (Warp)	✓ (PyTorch)	–	–	–
Multi-agent Support	–	✓	✓	*	* (via UDP)
Multiple Flight Tasks	✓	✓	✓	*	*
Controller Support	✓	✓	✓	*	*
Realism	High	Medium	High (if X-Plane)	High	High
Visualization Tools	✓	*	*	* (via FlightGear)	✓
Sim-to-Real Ready	*	*	*	✓	✓

modeling, modular control integration (classical and learning-based), precise actuator and sensors modeling and support for advanced aerodynamic modeling and realistic disturbance injection, like wind turbulence, atmospheric pressure (for high altitude flight conditions) and simplified computation of aerodynamics coefficients with OpenVSP [113].

5.3 Preliminaries

We consider the control of a rigid fixed-wing vehicle flying in proximity to the ground, modeled as a six-degrees-of-freedom (6DoF) system with coupled translational and rotational dynamics. The vehicle is subject to forces from gravity, aerodynamics, and propulsion, and its motion is described in the body frame. The state vector $\mathbf{x} \in \mathbb{R}^9 \times S^3$ (or $\mathbf{x} \in \mathbb{R}^{12}$) comprises the position $\mathbf{p} \in \mathbb{R}^3$, orientation (represented as an unit quaternion $\mathbf{q} \in S^3 = \{\mathbf{q} \in \mathbb{H} : ||\mathbf{q}|| = 1\}$ or Euler angles $(\phi, \theta, \psi) \in \mathbb{R}^3$), linear velocity $\mathbf{v} \in \mathbb{R}^3$, and angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$. Control inputs include throttle and actuator deflections for the elevator, rudder, and ailerons. The equations of motion follow Newton-Euler rigid body dynamics:

$$m\dot{\mathbf{v}} = \mathbf{F}_g + \mathbf{F}_a + \mathbf{F}_t - \boldsymbol{\omega} \times m\mathbf{v}, \tag{5.1}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau}_a + \boldsymbol{\tau}_t - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}, \tag{5.2}$$

where m is the vehicle mass, **J** is the inertia tensor, $\mathbf{F}_g$ is the gravitational force, $\mathbf{F}_a$ and $\boldsymbol{\tau}_a$ are aerodynamic forces and moments, and $\mathbf{F}_t$ and $\boldsymbol{\tau}_t$ are thrust-generated force and moment vectors. We assume constant mass and neglect gyroscopic effects. Each vehicle is modeled as a rigid body with a body-fixed frame $\{b\}$ rigidly attached at the centre of mass, and motion is described relative to an inertial north–east–down (NED) frame $\{I\}$.
Aerodynamic forces and moments are computed using semi-empirical models based on the vehicle's angle of attack α, sideslip β, Reynolds number Re and control surface deflections. Lift, drag, and side force coefficients are computed from look-up tables or parametric expressions derived from geometric tools such as OpenVSP [113]. The effect of actuator dynamics is captured using first- or second-order response models, governed by user-defined time constants and damping ratios. This introduces realistic response delays and rate limits to control surface inputs. Thrust is generated by propellers whose outputs are mapped from normalized throttle commands via first-order response curves. In asymmetric thrust configurations, this can introduce differential yaw moments. Our simulator also supports ground effect modeling, which alters the lift and drag characteristics of the vehicle when flying close to the surface. This effect is modeled through empirical corrections [114] to the aerodynamic coefficients as a function of height-over-span ratio, tamper ration and aspect ratio.

The detailed derivation and parameterizations for the physical model are presented next.

5.3.1 Physical Modeling Details

This appendix provides the mathematical details behind the simulation environment used in the main paper. The simulator integrates a high-fidelity 6DoF flight model with realistic actuator dynamics, wind turbulence (Dryden), and optional ground-effect modeling. These models are configured via a modular system that supports controlled ablation studies and toggling of physical phenomena.

Aerodynamic Coefficients and Ground Effect

The aerodynamic forces and moments are computed from lookup tables or polynomial fits, using the local flow conditions:

$$\mathbf{F}_{\text{aero}} = qS \begin{bmatrix} -C_D \\ C_Y \\ -C_L \end{bmatrix}, \quad \mathbf{M}_{\text{aero}} = qS \begin{bmatrix} bC_l \\ cC_m \\ bC_n \end{bmatrix}, \tag{5.3}$$

where $q = \frac{1}{2}\rho V_a^2$ is the dynamic pressure, S is the reference wing area, b and c are the wingspan and chord, and C_i are the aerodynamic coefficients dependent on angle of attack α, sideslip β, and control surfaces δ_a (ailerons), δ_e (elevator) and δ_r (rudder).

To model ground effect, the lift and drag coefficients C_L and C_D are corrected via empirical terms, following [114]:

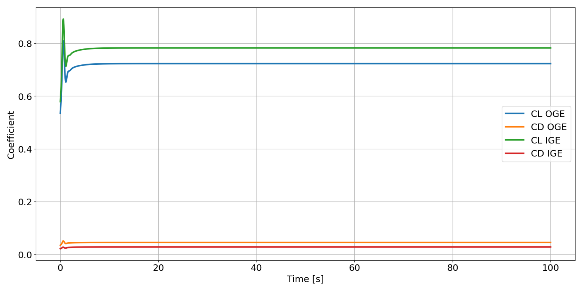
$$C_L = C_L^{\infty} \left(1 + \mu_L(h/b)\right), \tag{5.4}$$

$$C_D = C_D^{\infty} \left(1 - \mu_D(h/b)\right), \tag{5.5}$$

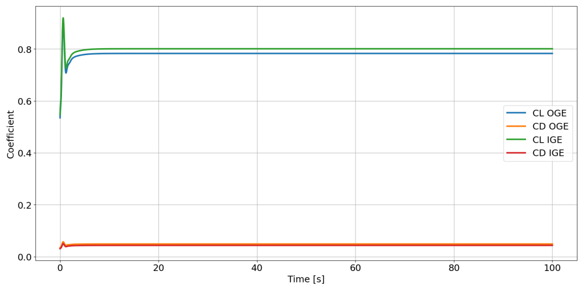
where μ_L, μ_D are ground effect modifiers parameterized as functions of the height ratio h/b, and C_L^{∞}, C_D^{∞} denotes the out-of-ground-effect coefficients. These modifiers can be toggled to assess the effect of WIG-specific dynamics.

Table 5.2 presents the influence of ground effect on the performance of the LQR controller. As expected, operating close to the ground leads to a noticeable reduction in overall commanded thrust. The increase in C_L reduces the required angle of attack (α) for the same airspeed to maintain steady flight. Together with the higher μ_D in ground effect, this results in a substantial decrease in

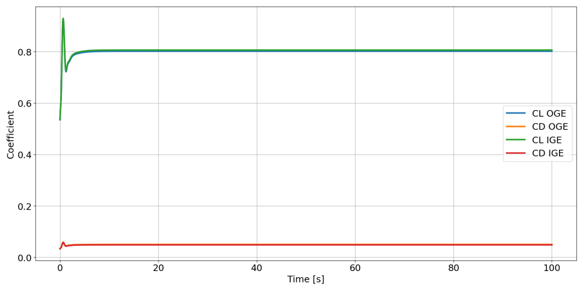
C_D, which lowers the overall drag and, consequently, the thrust required to sustain steady flight. These effect quickly become negligible once $(h/b) \geq 1$ (figure 5.1).



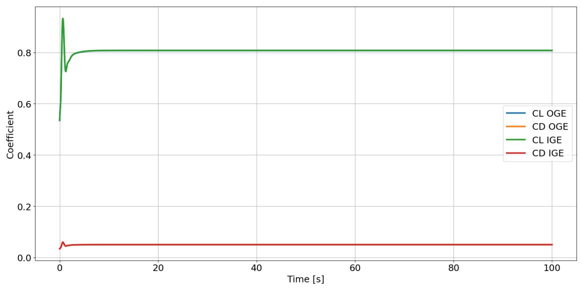
(a) 1.0 m constant altitude



(b) 2.5 m constant altitude



(c) 5.0 m constant altitude



(d) 10.0 m constant altitude

Figure 5.1: Variation of C_L and C_D with altitude due to ground effect, using the airship with the LQR controller.

Table 5.2: LQR performance metrics (RMSE, settling time, overshoot, error mean + std, and energy utilization) for the Airship vehicle at various altitudes.

Altitude (m)	**RMSE (m)**	**Settling Time (s)**	**Overshoot (m)**	**Error (mean $\pm$ std) (m)**	**Energy Utilization**
1.0	0.008	0.01	0.187	0.002 ± 0.014	0.303
2.5	0.011	0.01	0.247	0.002 ± 0.020	0.590
5.0	0.012	0.01	0.265	0.003 ± 0.021	0.689
10.0	0.013	0.01	0.271	0.003 ± 0.022	0.723
100.0	0.013	0.01	0.280	0.003 ± 0.022	0.743

Actuator Dynamics

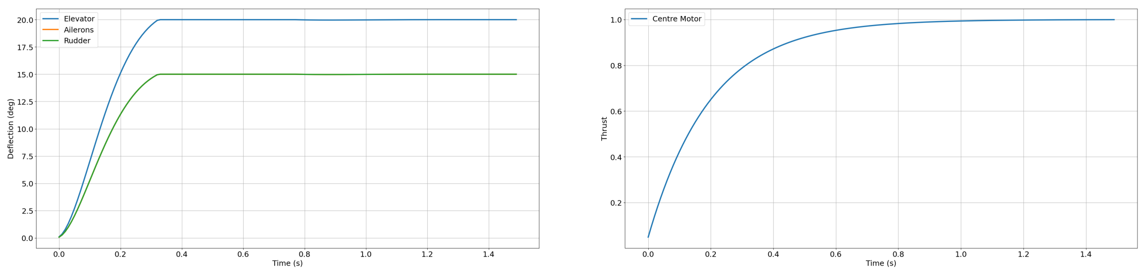
Actuator systems (control surfaces and motors) are modeled via first- or second-order transfer functions with configurable time constants and damping ratios:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \quad \text{(2nd-order)}, \tag{5.6}$$

or

$$H(s) = \frac{1}{\tau s + 1}, \quad \text{(1st-order)}, \tag{5.7}$$

where ω_n is the natural frequency, ζ is the damping ratio, and τ is the time constant. Each actuator group (e.g., elevator, ailerons, motors) can use a different response model based on configuration.



(a) Control surfaces step response

(b) Motor step response

Figure 5.2: Control surfaces and motor unit step responses for the Airship.

Wind and Turbulence Modeling

Environmental disturbances include: - **Constant wind** in the inertial frame (NED), rotated to the body frame. - **Dryden turbulence** [115], implemented via the MIL-F-8785C model using band-limited white noise through forming low-pass filters (see table 5.4).

For low altitude flights ($h < 1000\ ft$), the turbulence scale lengths and intensities are defined as

$$L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}, \quad L_w = h \tag{5.8}$$

Table 5.3: Dryden turbulence velocity spectral filters.

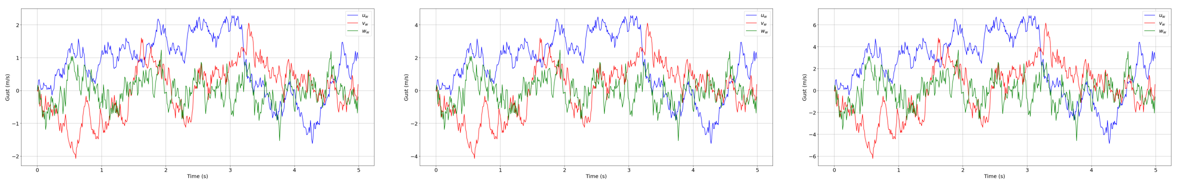
Table 5.4: Dryden turbulence velocity spectral filters.

	Longitudinal	Lateral	Vertical
Filter	$G_u(s) = \dfrac{\sigma_u K_u}{(1+T_u s)^2}$	$G_v(s) = \dfrac{\sigma_v K_v(1+\sqrt{3}T_v s)}{(1+T_v s)^2}$	$G_w(s) = \dfrac{\sigma_w K_w(1+\sqrt{3}T_w s)}{(1+T_w s)^2}$
Constants	$K_u = \sqrt{\dfrac{2L_u}{\pi U_0}},\quad T_u = \dfrac{L_u}{U_0}$	$K_v = \sqrt{\dfrac{L_v}{\pi U_0}},\quad T_v = \dfrac{L_v}{U_0}$	$K_w = \sqrt{\dfrac{L_w}{\pi U_0}},\quad T_w = \dfrac{L_w}{U_0}$

and

$$\sigma_u = \sigma_v = \frac{\sigma_w}{(0.177 + 0.000823h)^{0.4}}, \qquad \sigma_w = 0.1W_{20}, \tag{5.9}$$

where h represents the altitude in feet, and W_{20} is the chosen wind speed at 20 meters, which defines the intensity of the turbulence.



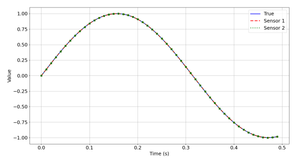
(a) Light Dryden turbulence (b) Moderate Dryden turbulence (c) Severe Dryden turbulence

Figure 5.3: Longitudinal, lateral, and vertical effects of different Dryden turbulence intensities at 2.5 m altitude and 28 m/s airspeed, using the same random seed.

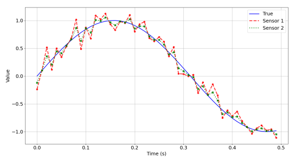
Sensor Realism and Noise

To simulate realistic perception pipelines, our framework includes configurable sensor models affected by various imperfections: additive noise, constant bias, scaling errors, clipping (limits), quantization (resolution), reduced sampling rates, and delay. Figure 5.4 illustrates these effects on sensor outputs compared to the ground truth signal. Each disturbance can be toggled or combined for robustness testing and sim-to-real transfer studies.

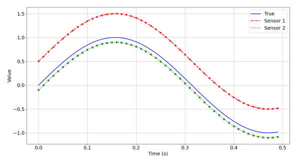
Overall, the simulator produces time-continuous dynamics that are discretized using a config-



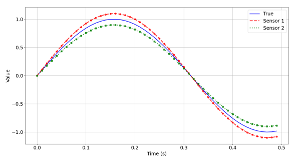
(a) Perfect sensors



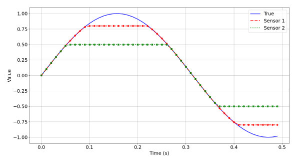
(b) Sensor noise



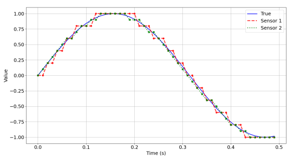
(c) Sensor bias



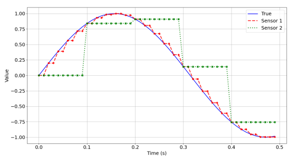
(d) Sensor scaling factor



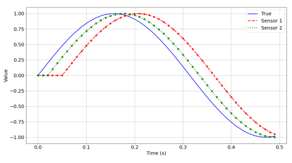
(e) Sensor limits



(f) Sensor resolution



(g) Sensor sampling rate



(h) Sensor delay

Figure 5.4: Illustration of different sensor effects implemented in the simulator.

urable integration scheme (e.g., Euler or RK4) and exposed through a modular interface supporting both CPU and GPU implementations. These dynamics form the basis for the environments used in training classical and learning-based controllers.

5.4 FALCON-S Framework

Our simulation platform is designed to support the development, training, and evaluation of flight control strategies for fixed-wing aircraft operating in near-ground environments. The architecture, illustrated in Figure 5.5, consists of two primary modules: the *agent* and the *environment*.

5.4.1 Agent module

The agent module supports a wide range of control models, including classical approaches such as Linear Quadratic Regulator (LQR) and Model Predictive Path Integral (MPPI), as well as modern learning-based controllers such as PPO, LSTM-based PPO, and DreamerV3. These controllers can be executed on either CPU or GPU for both evaluation and large-scale training, enabling both quick debugging of simulated flight conditions and heavy-duty batched experiments, where millions of trajectories can be collected to train and evaluate control performance.

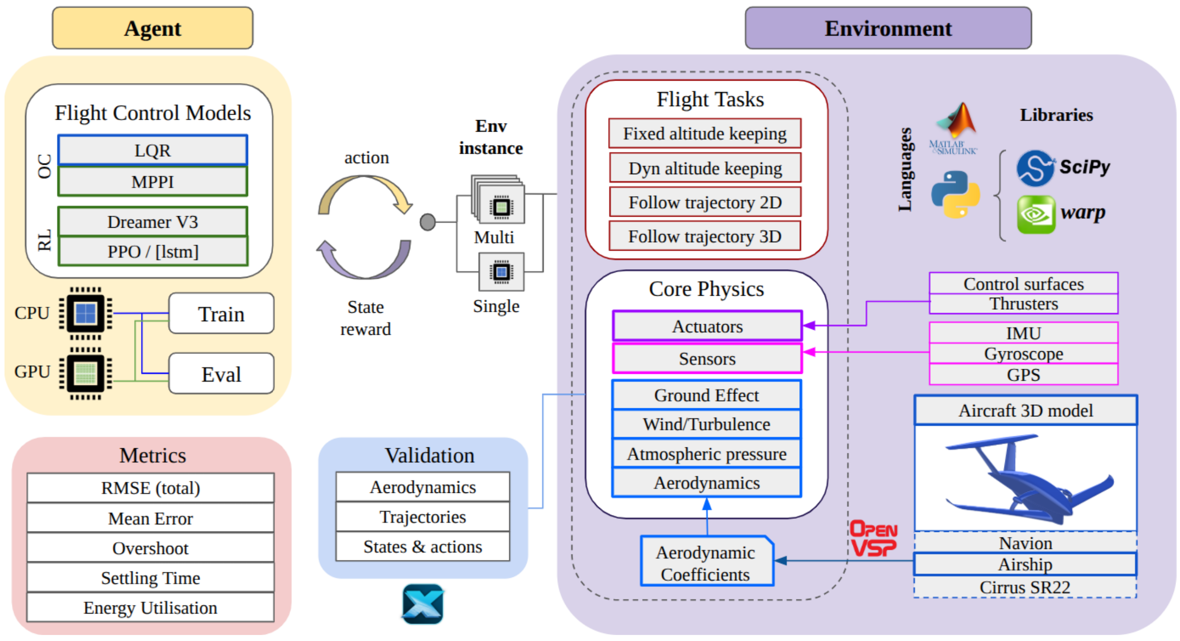


Figure 5.5: Overview of our FALCON-S simulation platform architecture. The environment module includes aerodynamic modeling, actuator dynamics, environmental effects such as ground effect and turbulence, and configurable sensor suites. The agent module supports both classical and learning-based controllers. Tasks, metrics, and visualization tools are modular and extensible, enabling robust benchmarking and policy training across single- and multi-agent setups.

LQR with Integral action

The LQR controller is implemented in closed form using discrete-time linearization of the vehicle dynamics around a steady trim condition. The gain matrix K is precomputed using the Riccati equation solution, and the resulting control law $u = -Kx$ is applied at each simulation step. The linearization matrices (A, B) are precomputed and approximated using numerical Jacobians based on the simulator's physics model. For LQRI (LQR with integral action) the state vector is augmented with integrator states (e.g. integrated altitude error) before forming (A_d, B_d) and solving the Ricatti equation 5.10.

The Linear Quadratic Regulator (LQR) controller was tuned using state and input weighting matrices selected to balance tracking accuracy and control effort. The state weighting matrix Q_{LQR} was defined as

$$\begin{aligned} Q_{LQR} = \text{diag}(&14.6,\ 8.2,\ 14.6,\ 8.2,\ 14.6,\ 8.2,\ 1,\ 1,\\ &0.25,\ 0.25,\ 0.25,\ 18.2,\ 18.2,\ 18.2,\\ &10^{-5},\ 10^{-5},\ 10^{-5},\ 10^{-5},\ 1,\ 400), \end{aligned} \tag{5.10}$$

where the first six entries correspond to actuator states, followed by velocity, angular velocity, imaginary quaternion components, and position. The very small weights on the quaternion error terms (10^{-5}) were introduced to avoid biasing the controller towards any one given attitude, while still ensuring stability.

To incorporate integral action, the augmented weighting matrix was defined as

$$Q_{\text{LQI}} = \text{diag}\left(Q_{\text{LQR}},\ Q_{\text{LQR}}(18\!:\!20)\right), \tag{5.11}$$

and the integral gains were set to $K_I = \begin{bmatrix} 0 & -1 & -1.5 \end{bmatrix}^\top$.

The control effort weighting matrix was chosen as

$$R_{\text{LQR}} = \text{diag}\big(1,\ 1,\ 800,\ 800,\ 800\big),$$

assigning higher penalties to thrust-related control inputs in order to limit excessive propulsive effort and improve efficiency.

The complete state-feedback gain matrix K (including integral augmentation where applicable) was computed in MATLAB using the built-in `lqr` function. The state-space matrices (A, B) required by `lqr` were obtained from the system linearization tool (linearization around the chosen trim condition). Full implementation and resulting K matrices for each aircraft can be seen in the open source repository.

MPPI

The Model Predictive Path Integral controller follows a sampling-based trajectory optimization approach. At each control step, the algorithm samples multiple control sequences sampled from a Gaussian distribution centred on the previous action sequence, propagates each action through the dynamics model, and computes an optimal control output from the weighted average of trajectories based on their cumulative cost. The implementation supports GPU-based sampling for parallelized inference, using a Warp backend. The cost function is task-specific and includes weighted penalties on tracking error, control effort, and constraint violations.

For all experiments the MPPI controller was initialized the following settings: number of sampled trajectories $N = 10^3$, planning horizon $T = 100$ time steps, temperature $\lambda = 3.0$, and control perturbation covariance $\Sigma = \text{diag}(\sigma_e^2, \sigma_a^2, \sigma_r^2, \sigma_T^2) = \text{diag}(0.10,\ 0.08,\ 0.08,\ 0.10)$, for elevator, aileron, rudder and throttle respectively.

The cost function employed in the planner is the sum of weighted penalties for: (i) altitude tracking, (ii) lateral (cross-track) tracking, (iii) ground-collision/proximity avoidance, (iv) angular-rate intensity, (v) airspeed keeping, (vi) penalization of excessive altitude, (vii) excessive angle-of-attack (α), and (viii) excessive sideslip (β).

MPPI trajectory sampling and propagation were implemented using NVIDIA Warp to JIT-compile the simulation kernels and execute large numbers of trajectories in parallel on the GPU.

The formulation and implementation follow the approach described in [116]. Exact numeric weights, cost functions and process are also available in the open-source repository.

Gymnasium interface

The environment exposes a compliant Gymnasium [117] interface through the `CoreAirshipEnv` class and its wrappers. It supports `reset()`, `step(action)`, and `render()` methods, and optionally includes `info` dictionaries with task-specific diagnostics. Observations are exposed as flat NumPy arrays and can be extended with sensor noise or delays via wrapper classes. The action space is continuous (bounded) and directly maps to control surface deflections and throttle values.

For high-performance training and evaluation, a parallelized variant of the environment is available through the Warp backend. This wrapper implements the same Gymnasium API but executes dynamics in batched form on the GPU, leveraging Warp's kernel-level integration and memory model.

Stable-Baselines3 and DreamerV3 support

We provide out-of-the-box integration with Stable Baselines3 (SB3), enabling rapid experimentation with off-the-shelf RL algorithms like PPO and SAC. Model-based RL agents are supported via a DreamerV3 [35] pipeline that wraps the simulation environment in a recurrent state-space model (RSSM). The implementation reuses the 'dreamerv3' codebase, adapted for continuous-control fixed-wing tasks. The world model is trained jointly with a policy and value network using imagined rollouts. Action sequences are optimized through learned latent trajectories. GPU acceleration is used for both training and inference.

5.4.2 Environment module

Our environment module supports multiple simulation backends and interoperation with external tools, allowing flexibility in simulation fidelity, performance, and controller design workflows. Specifically, we offer two primary physics engines in Python: one based on SciPy's numerical integration for rapid prototyping, and another leveraging NVIDIA Warp for large-scale GPU-accelerated simulation. In addition, MATLAB and Simulink can be used for validation or control design tasks, such as symbolic derivation of system matrices for LQR or linearized model identification. This dual-language and dual-backend setup enables practitioners to prototype quickly in Python and validate or deploy controllers using industry-standard tools when necessary.

Core Physics

The environment simulates full six-degree-of-freedom (6DoF) rigid-body aircraft dynamics, focusing on low-altitude scenarios where physical effects such as ground proximity and turbulence dominate. The physics module is structured around five interconnected components:

Aerodynamics: Uses precomputed aerodynamic coefficients from OpenVSP or analytical approximations. Aerodynamic forces and moment forces are adjusted dynamically based on airspeed, angle of attack, sideslip angle, height above ground and control surface deflection. Ground effect corrections are applied using semi-empirical models 5.3.1.

Actuators: Control surface deflections and thrust values are passed through first- or second-order actuator dynamics 5.3.1, allowing simulation of latency, saturation, and rate-limited responses. The actuator module outputs net forces and moments in the body frame.

Environmental Effects: Wind gusts, turbulence fields, and pressure gradients are injected into the dynamics via different noise models 5.3.1, enabling robustness testing under realistic conditions.

Sensors: An onboard sensor model simulates IMU measurements (accelerometer, gyroscope), GPS, and optional encoders 5.3.1. Sensor noise, sampling rate, resolution, or delay can be added to evaluate performance under degraded sensing.

Flight Tasks: The agent interacts with the environment through a set of modular task definitions, such as fixed-altitude keeping, dynamic climbing/descending, 2D path following, and full 3D trajectory tracking. These are defined as reward functions and success conditions on top of the raw physics simulation.

Each of these components interacts through the environment interface, which passes state transitions, sampled observations, and reward signals to the agent. Each physics component can be independently toggled or simplified, enabling ablation studies and comparative benchmarking under controlled settings.

Aircraft 3D model

Our framework supports rapid prototyping of different airframes via JSON-based configuration files. Each aircraft model (e.g., Navion, Cessna, or Airship) is described by its geometry, mass, inertia, control surface layout, propulsion system parameters, sensor configuration and environmental settings. Given the aircraft OpenVSP 3D model, using its python API, the aerodynamic coefficients can be computed as a luck-up table and then fitted to a Nth order polynomial. These models are then used for both simulation and visualization. The modular setup makes it easy to switch between vehicles and test control policies across different configurations, improving generalization and robustness.

Validation with X-Plane

To improve validation and high-fidelity visualization, FALCON-S includes an interface to the X-Plane using the Python XPlaneConnect API [118] developed by NASA. Given the same aircraft configuration (geometry and flight initial conditions), trajectories generated in our simulator can be replayed or compared within X-Plane's high-resolution rendering engine. This allows cross-verification of dynamics between our model and an industry-standard closed-source simulator. Additionally, X-Plane can be used to test the different controllers and scenarios in an different simulation environment, providing a practical robustness and check for controller performance under a different modeling physics engine. Lastly, it can be used to capture high-quality video demonstrations of trained agents flying over varied terrain.

To support high-quality visualization and cross-simulator validation, FALCON-S includes a Python interface to X-Plane. This allows us to reproduce the same control task shown in previous experiments—such as dynamic altitude keeping—within X-Plane's rendering engine using the same aircraft configuration and reference trajectory. This enables visual inspection, qualitative validation, and future extensions toward sim-to-real transfer.

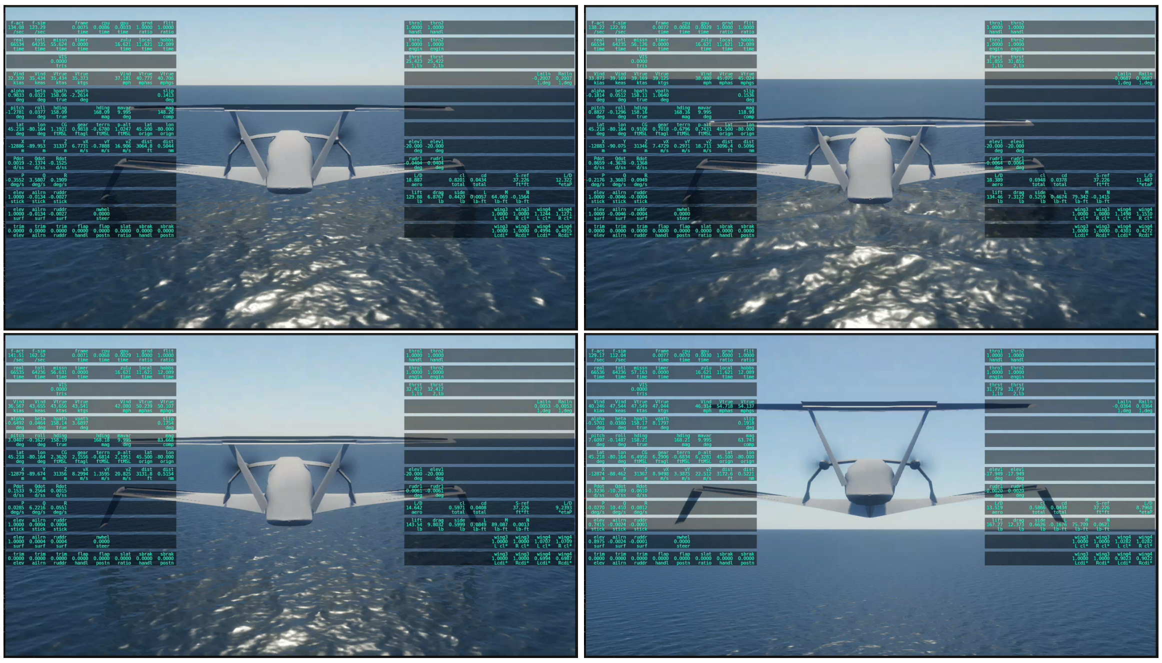


Figure 5.6: Rendered views of the Airship performing a dynamic altitude-keeping task in X-Plane, aligned with the same reference trajectory used in FALCON-S.

5.5 Experiments & Results

Our experiments are designed to highlight the flexibility and realism of the FALCON-S framework, rather than to optimize or compare specific learning or control algorithms. The primary objective is to demonstrate how the simulator supports a wide variety of use cases and provides structured tools to evaluate control performance under diverse settings. To this end, we present a set of illustrative results covering four key aspects:

- (1) **Algorithm performance illustration**: We demonstrate how FALCON-S supports consistent benchmarking by applying both classical (e.g., LQR) and learning-based (e.g., DreamerV3) controllers to standard tasks like altitude keeping.
- (2) **Multi-task generalization**: We test a single controller (e.g., MPPI or LQR) on multiple tasks (e.g., altitude regulation, 2D path tracking, 3D trajectory tracking) to show how FALCON-S supports task variation and behavioral analysis with minimal reconfiguration.

- (3) **Cross-vehicle testing**: Using the same control policy, we evaluate performance across different aircraft models (e.g., Cessna, Navion, Airship) to highlight how simulation fidelity and control difficulty change across morphologies and configurations.
- (4) **Environmental sensitivity**: We analyze the impact of physical realism features, such as wind disturbance, ground effect, sensor noise, or actuator delay, by toggling them independently and observing the effect on controller robustness and behavior.

Metrics

To evaluate controller performance, we compute a set of standard metrics from each simulated trajectory, including root mean square error (RMSE), settling time, overshoot, energy utilization, and mean error. RMSE and mean error quantify overall tracking accuracy; settling time measures how quickly the agent enters and remains within a defined error band (1m); overshoot reflects the maximum deviation from the reference; and energy utilization serves as a proxy for control effort, computed from the squared motor actions over time. These metrics, together with full trajectory and action logs, allow structured comparisons across algorithms, tasks, vehicle models, and environmental settings. Table 5.5 summarizes the performance metrics used to evaluate control strategies in FALCON-S. Each metric is computed from logged trajectories and actions, capturing accuracy, responsiveness, and control efficiency.

For interpretation, lower values of tracking error and overshoot indicate higher accuracy, while shorter settling times reflect faster convergence. Control smoothness metrics (e.g., input rate penalties) capture responsiveness without excessive actuator usage. Energy consumption is evaluated from integrated thrust and control surface activity (range [0-1]): lower values indicate more efficient control. Conversely, excessively high energy consumption may reflect oscillatory or unstable control behavior.

Trajectories: The trajectories (a)–(f) correspond to: (a) altitude sine wave, (b) altitude ramps, (c) altitude and lateral ramps, (d) lateral sine wave, (e) altitude and lateral sine wave, and (f) spiral wave.

Table 5.5: Summary of evaluation metrics computed from trajectory and control logs.

Metric	Formula / Description
RMSE (Total)	$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N} \lVert \mathbf{e}_t \rVert^2}$ Where $\mathbf{e}_t$ is the position error at timestep t
Mean Error	$\text{Mean} = \frac{1}{N}\sum_{t=1}^{N} \lVert \mathbf{e}_t \rVert$
Overshoot	$\text{Overshoot} = \max_t \lVert \mathbf{e}_t \rVert$
Settling Time	Minimum time t such that $\lVert \mathbf{e}_t \rVert \leq \delta$ and remains within the band $\forall t' \geq t$ for at least 10% of the episode length. Default threshold: $\delta = 1\text{m}$
Energy Utilization	$\text{Energy} = \frac{1}{T}\int_0^T \lVert \mathbf{u}_{\text{motors}}(t) \rVert^2 \, dt$ Where $\mathbf{u}_{\text{motors}}$ are normalized motor inputs and T is total time

Each subsection below presents a brief experiment showcasing these capabilities. We leave detailed quantitative benchmarking and algorithm tuning to future work.

5.5.1 Demonstrating Learning-Based Control with Dreamer

To illustrate how FALCON-S supports modern reinforcement learning pipelines, we trained a DreamerV3 agent to perform altitude regulation. The task consists of maintaining flight along a forward trajectory while matching a time-varying altitude reference. Figure 5.7 shows the learned behavior over multiple rollouts, with 3D trajectory tracking, orientation stabilization, position evolution, and linear velocity regulation. The results indicate stable control behavior and successful learning of the target altitude profile, albeit with slight oscillations due to limited policy tuning. Performance metrics across representative environments are summarized in Table 5.6, demonstrating tracking accuracy in the sub-meter range with energy usage that can be improved.

Table 5.6: Mean $\pm$ standard deviation across five DreamerV3 runs for dynamic altitude tracking.

RMSE (m)	Alt. RMSE (m)	Overshoot (m)	Error (mean ± std) (m)	Energy Util.
0.756 ± 0.763	1.309 ± 1.231	1.540 ± 1.246	1.270 ± 1.369	0.743 ± 0.066

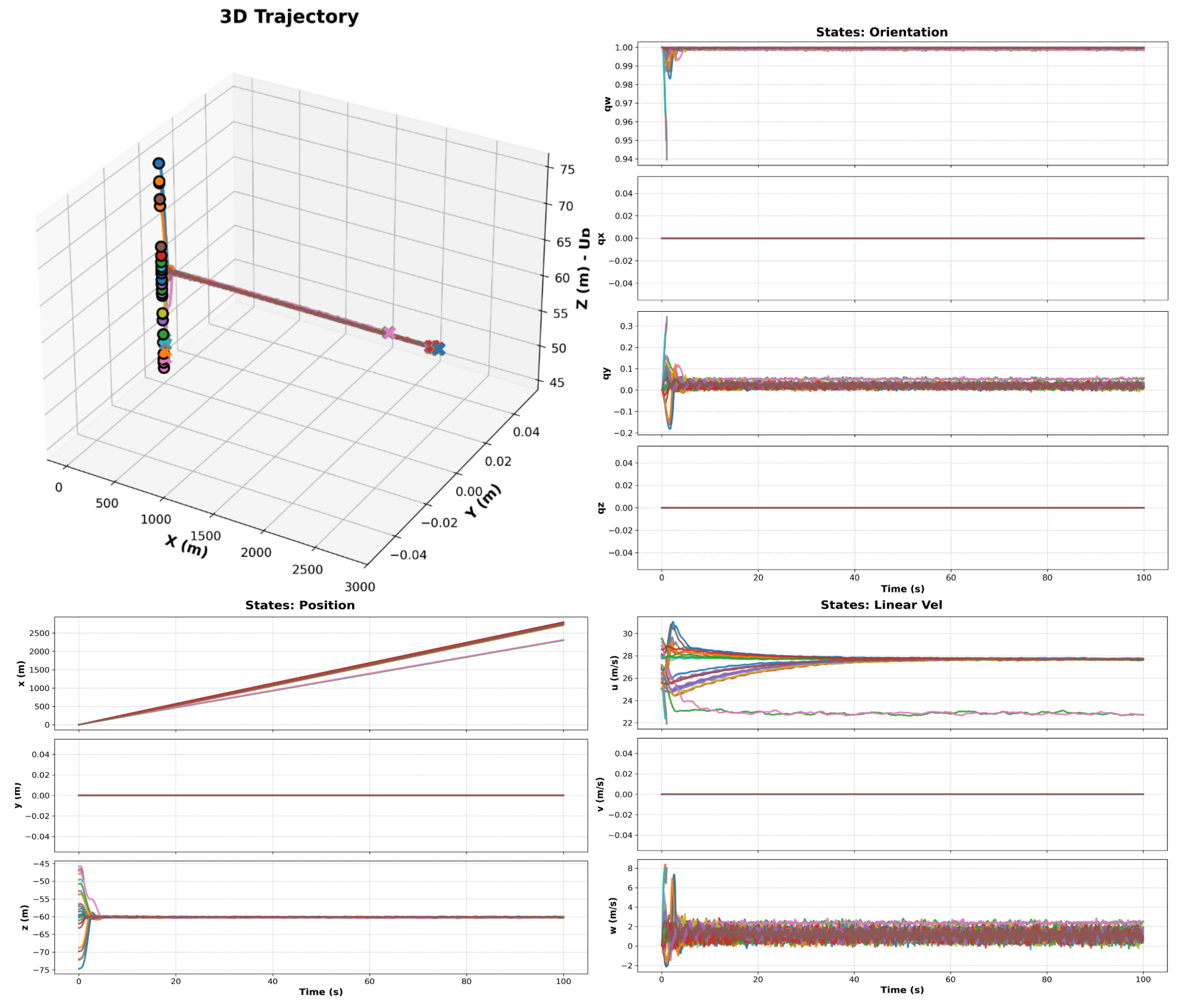


Figure 5.7: DreamerV3 agent controlling the airship along a dynamic altitude-keeping trajectory. Top-left: 3D trajectory tracking. Top-right: orientation convergence. Bottom-left: position over time. Bottom-right: body-frame velocity components.

5.5.2 Single Controller Across Multiple Tasks

We evaluate the LQR controller on six trajectory tracking tasks of increasing complexity using the same airship model. As shown in Figure 5.8 and Table 5.8, the controller maintains low RMSE and smooth behavior on simpler tasks such as single-axis sine waves (a, d) and low-frequency ramps (b, e), with minimal overshoot and low energy usage. Performance degrades in more challenging 3D or fast-changing trajectories (c, f), where the controller exhibits larger errors and reduced stability.

These results demonstrate the ability of our framework to highlight task-dependent control limitations and enable fine-grained benchmarking across diverse reference profiles.

Performance metrics for the MPPI controller can be seen in table 5.7.

Table 5.7: Performance metrics (RMSE, settling time, σ, overshoot, mean error, and energy utilization) for the Airship vehicle for constant altitude and tasks (a)–(f) with the MPPI controller. Runs marked with '*' indicate simulations that terminated prematurely due to instability (crash or stall).

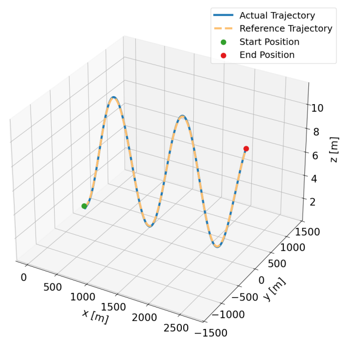
Task	**RMSE (m)**	**Settling Time (s)**	**Overshoot (m)**	**Error (mean $\pm$ std) (m)**	**Energy Utilization**
2.5 m altitude	0.010	0.84	0.119	0.013 ± 0.010	0.787
60.0 m altitude	0.013	1.08	0.158	0.017 ± 0.013	0.786
(a)	0.012	0.01	0.070	0.017 ± 0.011	0.780
(b)*	1.075	–	3.836	1.330 ± 1.303	0.367
(c)*	3.046	–	11.256	3.837 ± 3.621	0.340
(d)	0.008	0.01	0.081	0.011 ± 0.009	0.525
(e)	0.010	0.01	0.088	0.014 ± 0.010	0.757
(f)*	0.814	–	4.581	0.971 ± 1.023	0.419

Table 5.8: Performance metrics (RMSE, settling time, σ, overshoot, mean error, and energy utilization) for the Airship vehicle in tasks (a)–(f) with the LQR controller.

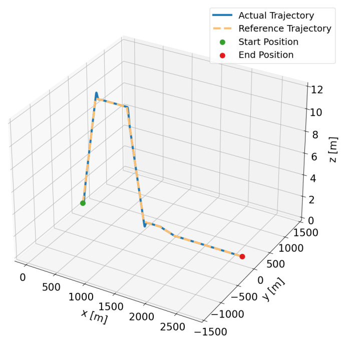
Task	**RMSE (m)**	**Settling Time (s)**	**Overshoot (m)**	**Error (mean $\pm$ std) (m)**	**Energy Utilization**
(a)	0.025	1.48	0.211	0.037 ± 0.024	0.663
(b)	0.052	41.55	0.671	0.025 ± 0.087	0.370
(c)	0.213	44.06	1.708	0.144 ± 0.339	0.379
(d)	0.017	2.94	0.213	0.014 ± 0.026	0.303
(e)	0.019	2.40	0.209	0.024 ± 0.021	0.583
(f)	0.149	–	0.704	0.231 ± 0.114	0.565

5.5.3 Cross-Aircraft Evaluation

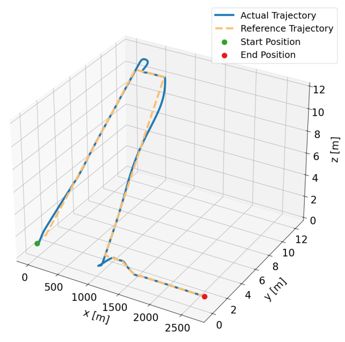
To evaluate generalization across vehicle morphologies, we test the same LQR controller on three aircraft models, Airship (A), Cirrus SR22 (B), and Navion (C), across all six trajectory tracking tasks. As shown in Table 5.9, performance varies significantly with aircraft dynamics. The Airship (A), for which the controller was tuned, consistently achieves the lowest RMSE and overshoot, indicating good stability and responsiveness. In contrast, the Cirrus (B) and Navion (C) exhibit higher errors and settling times, especially in dynamic or multi-axis tasks (e.g., tasks c and f), due



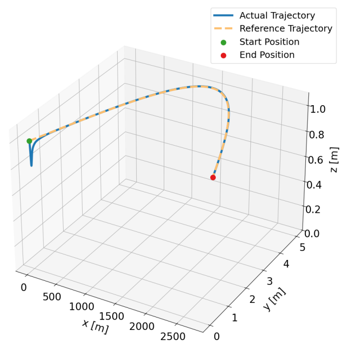
(a) Altitude sine wave



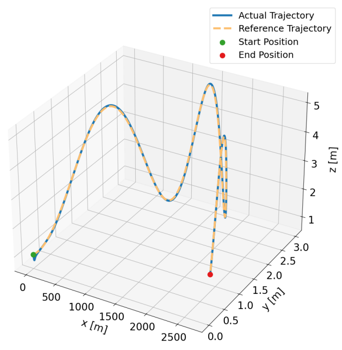
(b) Altitude ramps



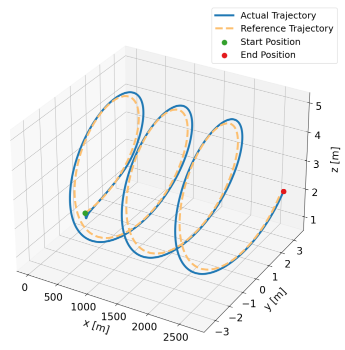
(c) Altitude and lateral ramps



(d) Lateral sine wave



(e) Altitude and lateral sine wave



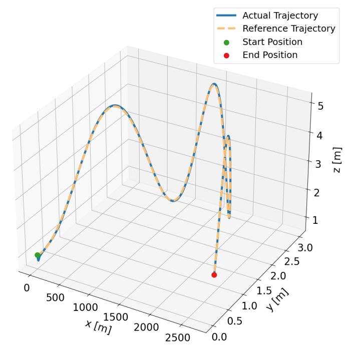
(f) Spiral wave

Figure 5.8: LQR-controlled airship response while tracking different trajectories.

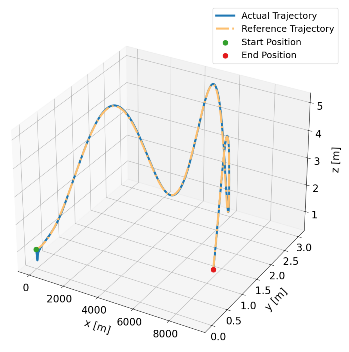
to differences in actuation and inertia properties. These results illustrate how the framework enables structured comparisons across vehicle configurations and supports benchmarking controller robustness to morphology changes.

Table 5.9: Performance metrics for scenarios (a)–(f). Columns (A), (B), and (C) correspond to the Airship, Cirrus SR22, and Navion, respectively.

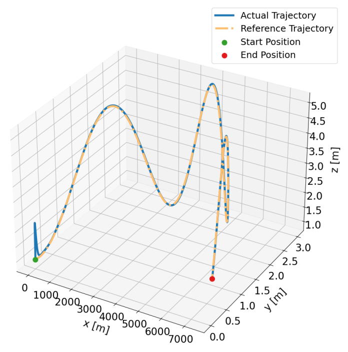
	RMSE			**Settling Time (s)**			**Overshoot**			**Error (mean ± std) (m)**			**Energy Utilization**		
Task	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)
(a)	0.025	0.024	0.055	1.48	1.63	1.51	0.211	0.419	1.227	0.037 ± 0.024	0.028 ± 0.031	0.034 ± 0.089	0.663	0.372	0.362
(b)	0.052	0.045	0.049	41.55	41.27	41.27	0.671	0.787	0.854	0.025 ± 0.087	0.019 ± 0.076	0.019 ± 0.082	0.370	0.322	0.315
(c)	0.213	0.223	0.208	44.06	46.20	46.00	1.708	1.709	1.877	0.144 ± 0.339	0.158 ± 0.352	0.148 ± 0.328	0.379	0.322	0.315
(d)	0.017	0.023	0.056	2.94	3.22	2.70	0.213	0.412	1.250	0.014 ± 0.026	0.017 ± 0.037	0.022 ± 0.094	0.303	0.290	0.289
(e)	0.019	0.022	0.055	2.40	2.59	1.95	0.209	0.416	1.237	0.024 ± 0.021	0.021 ± 0.032	0.027 ± 0.091	0.583	0.345	0.338
(f)	0.149	0.161	0.163	–	–	–	0.704	0.739	1.290	0.231 ± 0.114	0.248 ± 0.126	0.243 ± 0.143	0.565	0.340	0.335



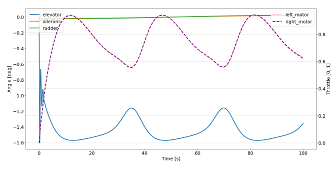
(a) Airship trajectory



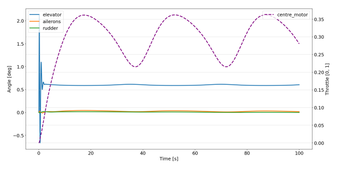
(b) Cirrus SR22 trajectory



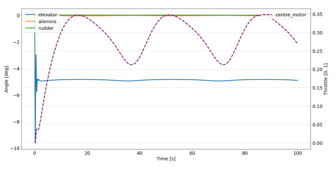
(c) Navion trajectory



(d) Airship actuator actions



(e) Cirrus SR22 actuator actions

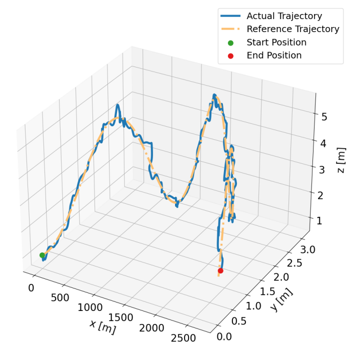


(f) Navion actuator actions

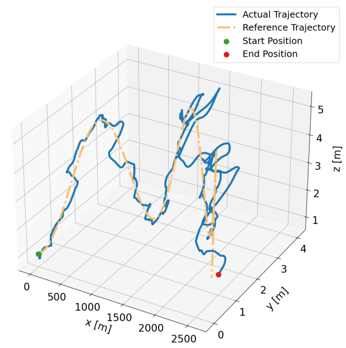
Figure 5.9: LQR-controlled Airship, Cirrus SR22, and Navion responses to altitude and lateral sine-wave trajectory tracking.

5.5.4 Robustness Under Environmental Variations

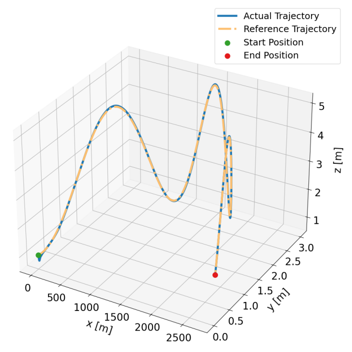
We assess the robustness of the LQR controller under different sources of environmental uncertainty: sensor noise (A), wind disturbances (B), and sensor delay (C). Table 5.10 shows that all three perturbations impact performance to varying degrees, with wind disturbances generally inducing the highest errors, overshoot, and energy usage, especially in fast-changing tasks such as (c) and (f). Sensor noise introduces more variability (e.g., increased RMSE and error variance), while sensor delay has a relatively smaller effect in most scenarios, though certain tasks (e.g., (b), (f)) remain sensitive. These results demonstrate FALCON-S's capacity to simulate realistic disturbances and evaluate controller sensitivity in a structured and reproducible way.



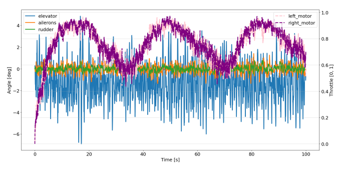
(a) Trajectory with sensor noise



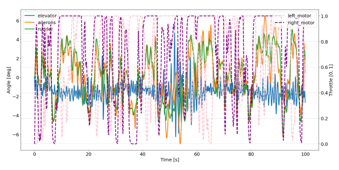
(b) Trajectory with light Dryden turbulence



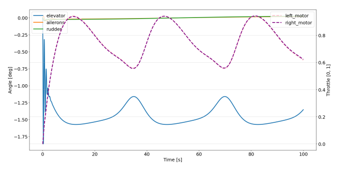
(c) Trajectory with sensor delay



(d) Actuator actions with sensor noise



(e) Actuator actions with light Dryden turbulence



(f) Actuator actions with sensor delay

Figure 5.10: LQR-controlled Airship response to altitude and lateral sine-wave trajectory tracking under sensor noise, light Dryden turbulence, or a 20 ms sensor delay.

Table 5.10: Performance metrics for scenarios (a)–(f). Columns (A), (B), and (C) correspond to the Airship under sensor noise, wind disturbances, and sensor delay, respectively.

	RMSE			**Settling Time (s)**			**Overshoot**			**Error (mean ± std) (m)**			**Energy Utilization**		
Scenario	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)
(a)	0.091	0.220	0.024	64.43	–	0.01	0.619	1.155	0.215	0.131 ± 0.089	0.312 ± 0.218	0.036 ± 0.023	0.760	0.741	0.663
(b)	0.087	0.141	0.051	31.00	42.24	30.70	0.689	0.783	0.663	0.118 ± 0.093	0.178 ± 0.168	0.024 ± 0.084	0.464	0.476	0.371
(c)	0.394*	0.250	0.210	–*	43.37	43.20	1.732*	1.753	1.686	0.473 ± 0.492*	0.249 ± 0.354	0.142 ± 0.335	0.799*	0.474	0.380
(d)	0.066	0.076	0.017	0.01	0.01	0.01	0.356	0.382	0.217	0.099 ± 0.057	0.114 ± 0.065	0.014 ± 0.026	0.425	0.430	0.303
(e)	0.068	0.194	0.018	0.01	–	0.01	0.438	1.167	0.214	0.102 ± 0.060	0.253 ± 0.220	0.023 ± 0.021	0.669	0.668	0.583
(f)	0.163	0.238	0.147	2.34	87.28	2.48	0.766	1.401	0.691	0.260 ± 0.112	0.327 ± 0.249	0.228 ± 0.113	0.649	0.665	0.565

5.6 Discussion and Conclusions

This chapter introduced **FALCON-S**, a simulation benchmark for learning and control of fixed-wing aerial vehicles operating in ground effect. The contribution builds upon the central themes of this thesis—modular simulation, physically grounded modeling, and scalable reinforcement learning—while extending them to a more aerodynamically complex domain.

FALCON-S was designed in response to the limitations of existing flight simulators in terms of modularity, aerodynamic realism, and compatibility with modern learning algorithms. Through a dual CPU-GPU architecture and a clean modular design, the framework supports both high-throughput training and physically plausible evaluation. Notably, it incorporates detailed models for six-degree-of-freedom dynamics, actuator and sensor response, and ground-effect aerodynamics, enabling systematic experimentation across a range of low-altitude flight scenarios.

From a research perspective, FALCON-S serves two complementary goals. First, it provides a robust testbed for evaluating classical and learning-based control methods under realistic flight conditions, bridging the gap between control theory and deep reinforcement learning. Second, it enables ablation studies on model fidelity, observation corruption, and dynamics variation, which are critical for understanding the limits of sim-to-real generalization.

By supporting classical controllers (e.g., LQR, MPPI) alongside state-of-the-art learning algorithms (e.g., PPO, DreamerV3), FALCON-S encourages structured comparisons and opens the door to hybrid approaches that combine model-based and model-free reasoning. This aligns with the broader thesis direction of integrating domain knowledge, structure, and data-driven learning.

Limitations and Outlook. While FALCON-S offers high simulation fidelity and control flexibility, the current framework operates entirely in simulation and assumes accurate aerodynamic parameters and actuator models. Real-world transfer remains an open challenge, particularly under actuator faults, sensor drift, and uncertain aerodynamic regimes (e.g., post-stall behavior). Future work will focus on:

- **Extending the library of tasks and vehicle models**, including path following, obstacle avoidance, and energy-efficient cruise.
- **Sim-to-real validation**, including interfaces with onboard hardware, sensors, and embedded controllers for real-world experiments.
- **Policy adaptation and generalization**, incorporating online learning, residual control, and memory-based architectures for robustness across conditions.

In the broader context of this thesis, FALCON-S complements previous chapters by tackling one of the most dynamic and unstable robotic platforms—fixed-wing aircraft—and by showing that structured, high-fidelity simulation can enable progress toward generalizable and interpretable control. It serves as a foundation for future research in aerial robotics, where data-driven methods and physical modeling must go hand-in-hand to support robust autonomy in complex, low-altitude flight regimes.

Chapter 6

Additional Studies

6.1 Introduction

While the core of this thesis revolves around the design of high-performance simulation frameworks (RANS, RoboRAN, FALCON-S) and their application to deep reinforcement learning (DRL) control in multi-modal robotic platforms, several additional studies were carried out in parallel to explore adjacent challenges in spacecraft autonomy. These works, although not fully integrated into the main simulation stack presented so far, provide complementary insights into two critical dimensions of autonomous space robotics: (i) learning-based visual inspection in complex orbital scenarios, and (ii) estimation of angular velocity and inertia properties for spacecraft operating under partial observability and limited sensing.

The first study, **RL-AVIST** (Reinforcement Learning for Autonomous Visual Inspection of Space Targets), investigates model-based DRL for proximity operations around large orbital structures such as the Lunar Gateway and the ISS. Implemented using the Space Robotics Bench (SRB) framework rather than the IsaacLab-based environments developed in this thesis, RL-AVIST demonstrates the potential of DreamerV3 and other RL algorithms for training generalist and specialist policies in 6-DoF continuous-thrust settings. Although developed independently from the RANS and DRIFT pipelines, this work shares many motivations with previous chapters—particularly the need for adaptability, robustness, and scalable learning in uncertain space environments.

The second and third studies shift focus to a more classical control domain, aiming to recover inertial properties or angular velocity from minimal on-board sensing. The **Excitation-Based Angular Rate Estimation** work explores a novel method to estimate angular velocity using injected torque profiles and observed attitude changes, offering a lightweight and interpretable alternative to learning-based estimators. A related work, currently under review and conducted in collaboration, proposes an event-camera-based approach to recover angular velocity by tracking stars in the spacecraft's visual field. Though these contributions differ in methodology from the DRL-centric core of the thesis, they remain aligned with the overarching theme of enabling autonomy under uncertainty and hardware limitations.

Taken together, these works expand the scope of this thesis toward enabling autonomy not just through control policies, but also through improved sensing, estimation, and inspection capabilities. The remainder of this chapter is organized as follows: - Section 6.2 presents the RL-AVIST framework and experimental results; - Section 6.3 introduces the torque-based angular rate estimation pipeline; - Section 6.4 provides an overview of the event-camera angular velocity estimation work.

Each section includes a summary of the problem formulation, key methodological innovations, and main findings. Throughout, we clarify the boundaries of the author's contribution and discuss how these studies relate to the broader thesis narrative.

6.2 RL-AVIST: Visual Inspection of Space Assets via Model-Based RL

While the previous chapters focused on training generalizable RL policies for terrestrial and orbital robots using unified simulation frameworks (RANS, RoboRAN, FALCON-S), this section explores a complementary but independent line of work on intelligent visual inspection in orbit.

Specifically, we present RL-AVIST (Reinforcement Learning for Autonomous Visual Inspection of Space Targets), a study leveraging the Space Robotics Bench (SRB) [119] to train spacecraft agents to maneuver around complex space structures using 6-DoF control.

This work diverges from the simulator stack used in earlier chapters, operating instead within SRB — a physics- and rendering-capable platform designed for diverse space and planetary robotics tasks. The focus here lies in benchmarking RL methods under continuous-thrust dynamics, spacecraft morphology variation, and mission trajectory complexity. Despite this deviation, RL-AVIST remains aligned with the thesis' broader goal: investigating scalable and robust RL strategies for robot control in highly uncertain environments.

6.2.1 Problem Setting and Environment

In RL-AVIST, agents learn 6-DoF control policies to autonomously perform close-range visual inspections of static orbital targets (e.g., Lunar Gateway, ISS, Venus Express). The spacecraft is equipped with 8 fixed-direction thrusters and is modeled as a free-floating rigid body initialized at a random position and orientation relative to the target.

The state space includes previous actions, body-frame velocities, and relative pose to the target. The continuous action space $\mathcal{A} \subset \mathbb{R}^8$ corresponds to normalized thrust levels for each actuator. The dynamics are fully simulated, including control latency, inertia variations, and randomized initial conditions.

Reward functions are shaped to encourage smooth, efficient, and accurate tracking of desired poses around the target. Components include position convergence, orientation alignment, action smoothness, and actuation penalties — weighted to balance fuel efficiency with task success.

6.2.2 Model-Based vs Model-Free Learning

The study compares the performance of three RL algorithms:

- **DreamerV3** [35] – A model-based algorithm learning compact latent dynamics for long-

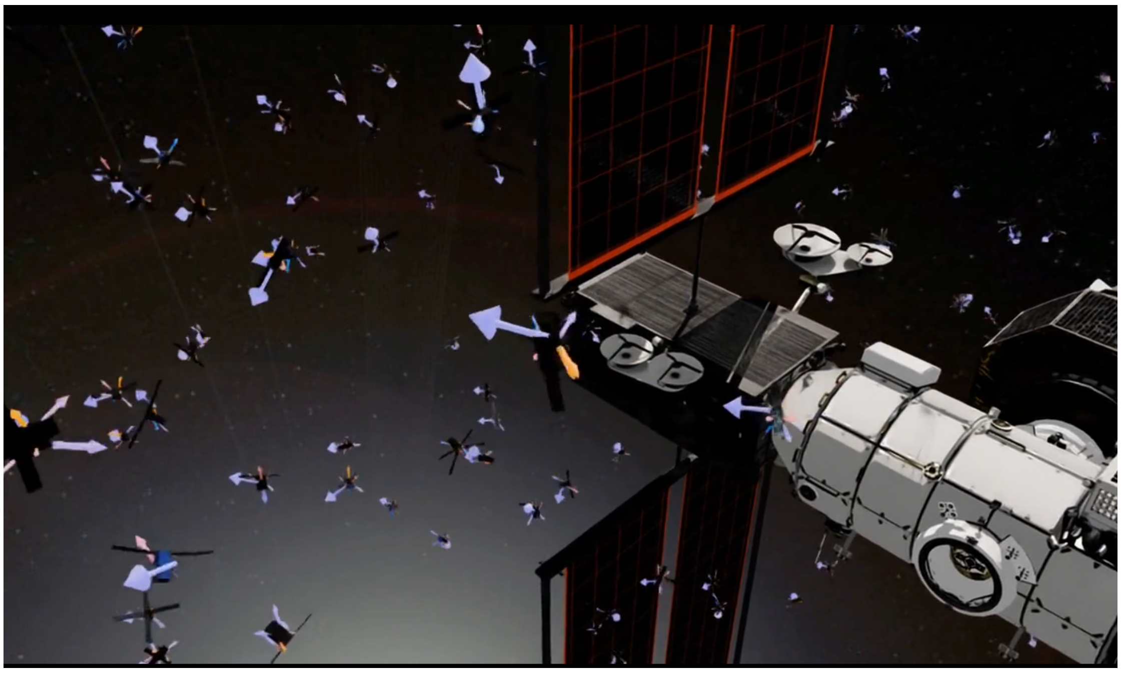


Figure 6.1: Training of multiple CubeSat morphologies to follow randomized velocity commands around the Lunar Gateway. Agents learn generalist policies by experiencing a wide variety of scenarios within SRB [119].

horizon planning.

- **PPO** [28] – A widely used model-free policy gradient baseline.
- **TD3** [29] – An off-policy model-free algorithm suitable for continuous actions.

All agents are trained in SRB using randomized spacecraft geometries and initial conditions. Figure 6.2 shows that DreamerV3 achieves faster convergence and better performance, highlighting the benefit of planning in latent space for complex, long-horizon control.

6.2.3 Trajectory Tracking and Generalization

Using DreamerV3, agents are deployed on structured inspection trajectories: circle, spiral, capsule, rectangle, lemniscate, and Lissajous patterns. Figure 6.3 illustrates accurate tracking in all scenarios, showcasing DreamerV3's adaptability and precision.

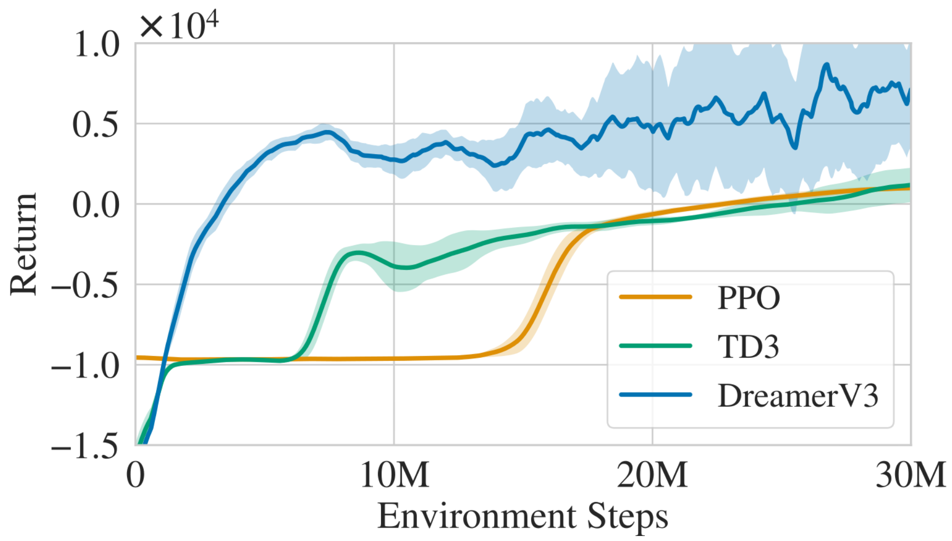


Figure 6.2: Training curves (mean ± std over 3 seeds) for different RL algorithms on generalized velocity-tracking tasks. DreamerV3 achieves higher returns and faster convergence than model-free baselines.

6.2.4 Deployment on Realistic Orbital Targets

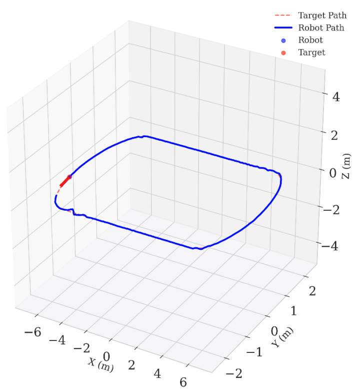
To simulate realistic mission settings, the trained agent is tested on inspection tasks around high-fidelity space assets. These include:

- **Lunar Gateway**, Fig. 6.4
- **Venus Express**, Fig. 6.5
- **International Space Station (ISS)**, Fig. 6.6

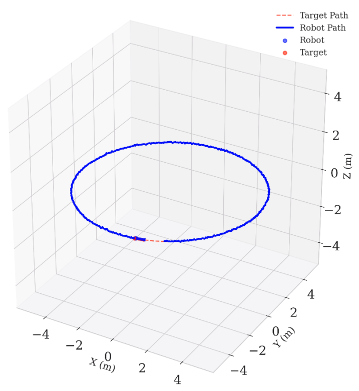
Each deployment includes RGB, depth, and semantic renderings for visual inspection, emulating the outputs of an onboard perception system.

6.2.5 Summary and Discussion

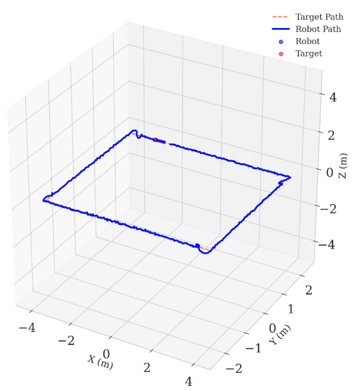
RL-AVIST provides a compelling case for applying model-based reinforcement learning in orbital visual inspection. Despite using a separate simulation stack (SRB), the work aligns with this the-



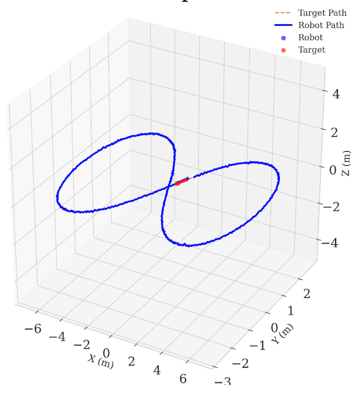
(a) Capsule



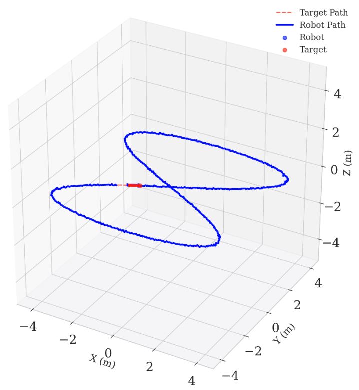
(b) Circle



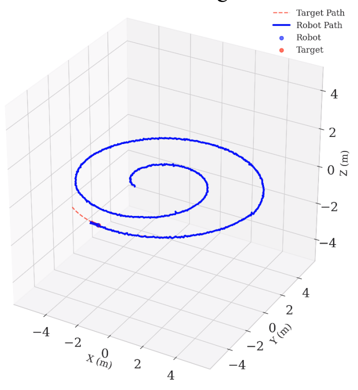
(c) Rectangle



(d) Lemniscate



(e) Lissajous



(f) Spiral

Figure 6.3: DreamerV3 agent tracking various inspection trajectories. The policy demonstrates strong generalization across geometrically diverse 3D paths.

sis by targeting intelligent control under uncertainty and using task-specific RL reward shaping, morphologically diverse training, and rigorous trajectory evaluation.

Scope and Integration Note

While RL-AVIST was developed independently of the core simulator stack used in this thesis, its methodology—task-driven RL with high-fidelity dynamics—complements our broader investigation into scalable control in uncertain environments. This study validates DreamerV3 as a sample-efficient baseline in complex orbital contexts.

Future work includes integrating perception-based policy learning and bridging the sim-to-real

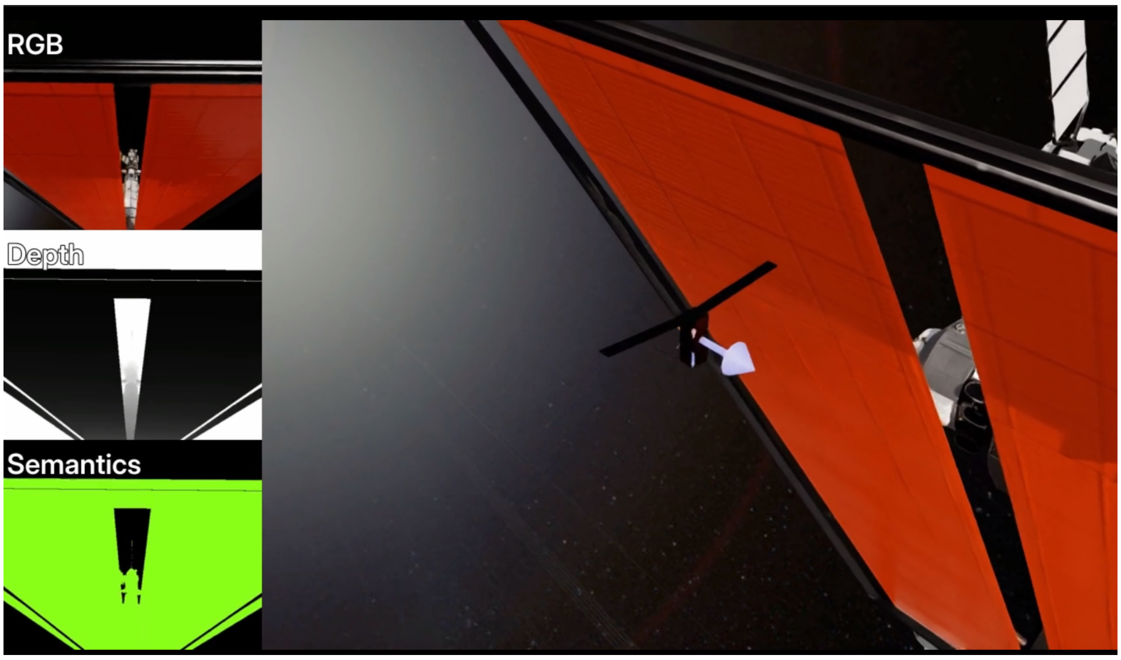


Figure 6.4: Inspection trajectory around the Lunar Gateway.

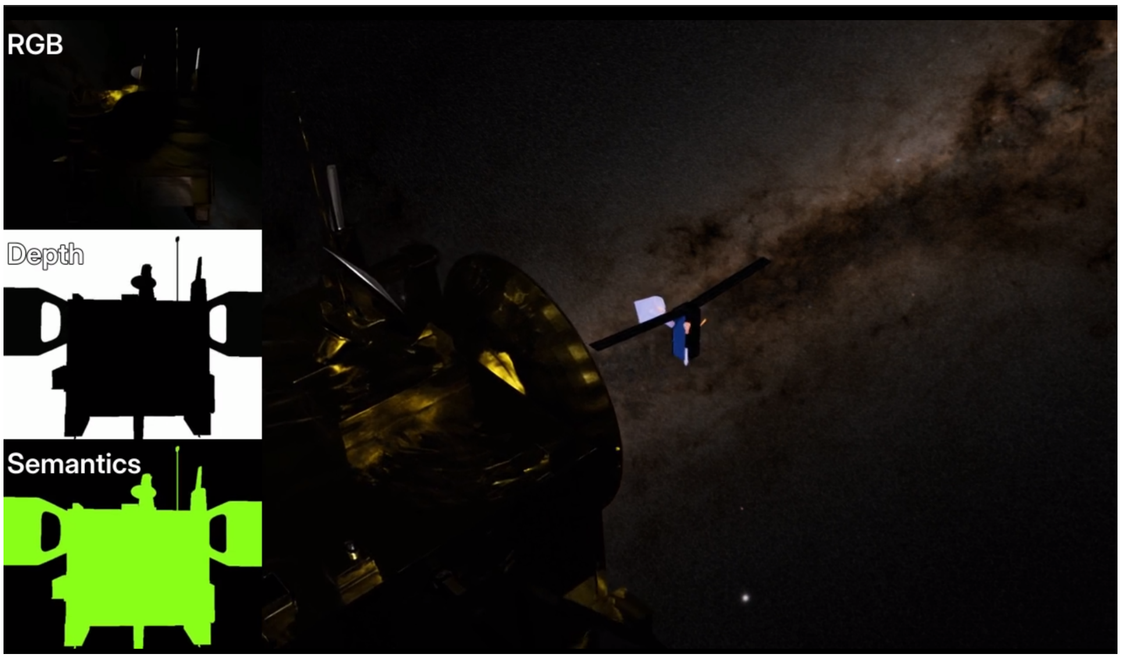


Figure 6.5: Visual inspection simulation near Venus Express.

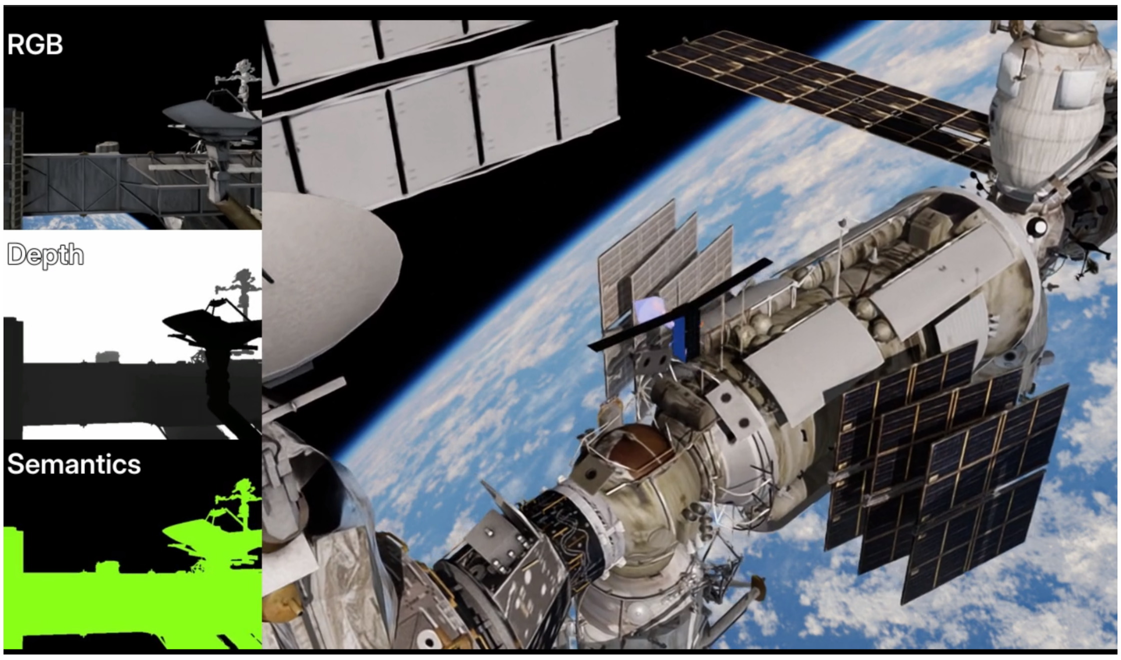


Figure 6.6: Close-proximity maneuvering around the ISS structure.

gap via hardware-in-the-loop validation or docking demonstration platforms.

6.3 Inertia Estimation via Active Excitation in Satellites

6.3.1 Motivation and Context

Accurate knowledge of a spacecraft's inertia tensor is critical for safe and precise attitude control, especially in long-duration missions or modular systems where mass distribution evolves over time. While pre-launch models are available, post-deployment changes—due to fuel consumption, payload deployment, or environmental degradation—can cause mismatches that degrade control performance. This motivates autonomous on-board identification methods based solely on available actuation and sensing.

This study investigates the use of excitation-based estimation pipelines for on-orbit inertia identification, comparing the effectiveness of two estimators—a batch Least Squares (LS) method and an Extended Kalman Filter (EKF)—under different excitation designs and satellite configurations.

The work is published in [30] and complements the thesis' broader focus on intelligent autonomy under uncertainty by addressing parameter estimation as a prerequisite to robust control.

6.3.2 Simulation Framework

The system simulates a 6-DoF rigid-body satellite equipped with orthogonal reaction wheels (RWs), modeling nonlinear attitude dynamics, RW coupling, actuator saturation, and simplified external disturbances (e.g., gravity gradient). Three spacecraft configurations are considered: a CubeSat (24kg), a Microsatellite (95kg), and a SmallSat (118kg), each with distinct RW specifications (Table 6.1). Estimation is performed over 300s episodes under varying conditions: static inertia, and three dynamic profiles—step, linear drift, and periodic variation.

Eight normalized torque excitation profiles are used to induce system observability (Figure 6.7). These range from smooth (sine, chirp) to discontinuous (multi-step, PRBS), targeting different spectral and temporal excitation characteristics.

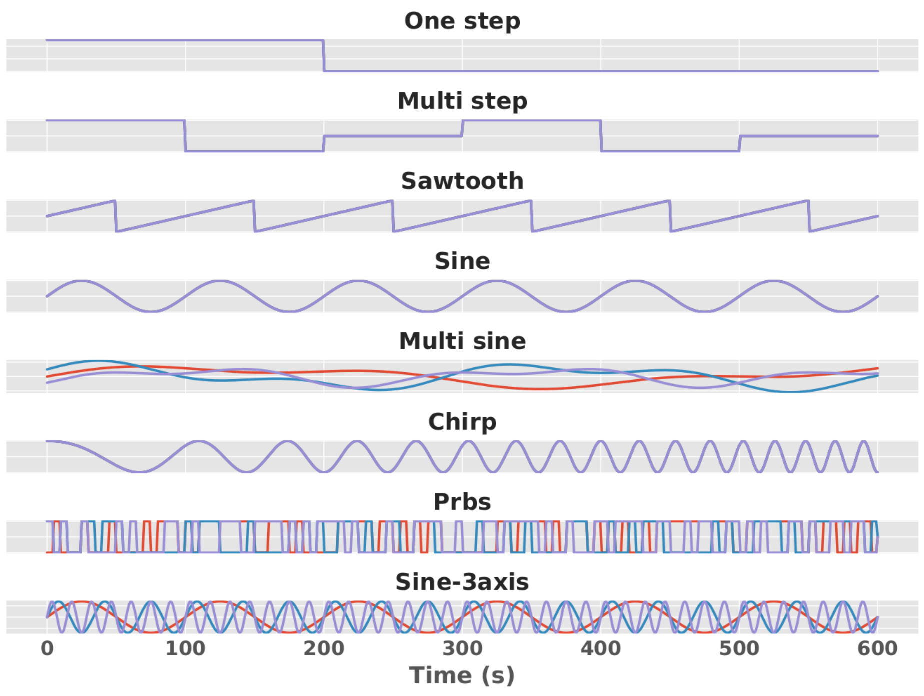


Figure 6.7: Overview of excitation profiles used to stimulate satellite dynamics for estimation.

Table 6.1: Satellite physical and reaction wheel parameters [30].

Model	**Mass [kg]**	**Inertia [kg·m^2]**	**RW Max Torque [Nm]**
CubeSat	24.0	[0.26, 0.26, 0.16]	0.01
Microsat	95.0	[6.53, 5.96, 4.53]	0.1
SmallSat	118.0	[10.6, 14.2, 15.3]	0.1

6.3.3 Estimation Methods

The LS method assumes static inertia and reconstructs angular accelerations using finite differences from gyro readings, minimizing residual errors over time. The EKF tracks angular velocity, reaction wheel speeds, and the diagonal inertia vector jointly, allowing it to handle dynamic changes by modeling inertia as a random walk state variable. Both estimators are implemented without hard physical constraints (e.g., triangle inequality), yet remain physically valid across seeds.

6.3.4 Results and Insights

Static Inertia Estimation: Figure 6.8 shows the relative error for all estimators and profiles. LS performs best under spectrally rich, smooth excitations (e.g., `chirp`, `multi-sine`), which improve regression conditioning. EKF shows better performance for high-inertia systems and temporally rich inputs (e.g., `multi-step`, `sine-3axis`) due to frequent state updates.

Time-Varying Inertia Tracking: Figure 6.9 summarizes estimation errors under step, drift, and periodic variations. EKF outperforms LS in smoothly varying scenarios (e.g., drift or sinusoidal change), especially on larger satellites. For abrupt changes (step), both degrade similarly, with EKF lagging due to adaptation delay. This confirms the EKF's capacity to adapt to in-flight variations, especially when excitation is rich and temporally persistent.

6.3.5 Discussion

This work provides a comparative foundation for selecting excitation strategies and estimation methods in in-orbit identification pipelines. Batch LS is preferable in short-duration or low-noise

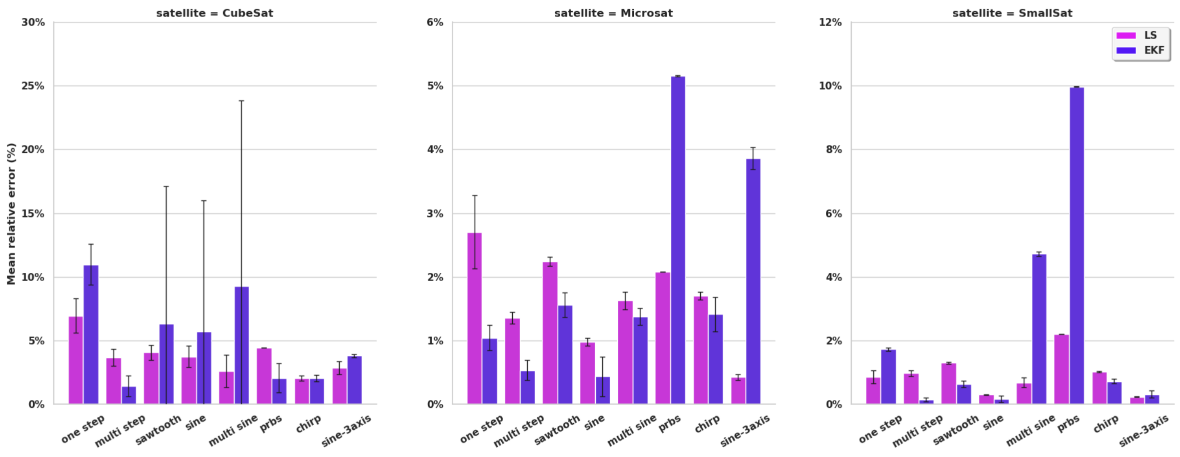


Figure 6.8: Normalized inertia estimation error across excitation profiles for static cases. Least Squares excels with smooth profiles, while the EKF benefits from dynamic excitation.

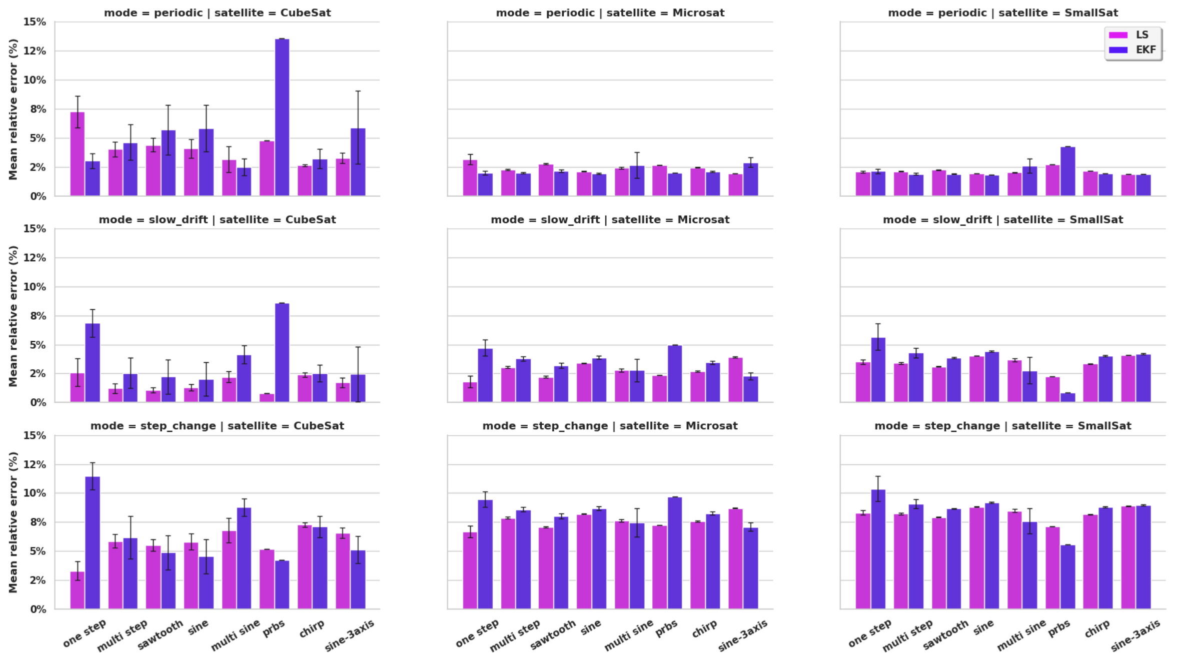


Figure 6.9: EKF vs. LS estimation errors under dynamic inertia conditions. The EKF consistently outperforms LS under smooth time-varying profiles.

contexts with smooth control inputs. EKF enables online tracking and is better suited for larger, slower-varying platforms or missions with inertia drift. The simulation framework is released open-source, and can be further extended with constrained optimization, advanced filtering, or reinforcement learning–based excitation design.

6.3.6 Summary and Thesis Integration

While distinct from the core control-focused frameworks in previous chapters, this contribution enriches the thesis narrative by addressing a critical prerequisite for learning-based control: accurate system modeling. The techniques and findings outlined here open paths to adaptive flight controllers that can re-estimate their mass properties online, and thus remain robust to mission-phase transitions, deployment, or failures. Future work includes integrating this estimation module into RL-based control stacks, either as a pre-calibration phase or as an online estimator jointly optimized with policy learning.

6.4 Event-based Angular Rate Estimation with Starfield Observations

This section summarizes a complementary study to the main contributions of the thesis, led by Franzese et al.[REF], where the author of this thesis contributed as second author. The work investigates whether event-based cameras—neuromorphic sensors that asynchronously record changes in brightness—can be used to estimate spacecraft angular velocity by observing the apparent motion of stars in the focal plane. This sensing modality offers high temporal resolution, low latency, and low power consumption, making it an attractive backup or complement to conventional gyroscopes for small satellites and resource-constrained spacecraft.

The key idea is that spacecraft rotation induces a measurable motion field of stars across the image plane. By reconstructing this apparent motion from the stream of brightness-change events, and by exploiting the geometry of the perspective projection, it is possible to infer the three angular rate components. This section provides a compact technical summary tailored for the thesis context.

6.4.1 Motion Field Model and Single-Camera Limitations

Let a star at normalized camera-frame coordinates (X, Y, Z) project to image-plane coordinates (x, y) under the pinhole model. A spacecraft rotating with angular velocity vector (p, q, r) induces an apparent motion field (u, v) in the image plane. The relationship is [120]:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \boldsymbol{F}(x, y) \begin{bmatrix} p \\ q \\ r \end{bmatrix},$$

where $\boldsymbol{F}(x, y)$ is a matrix derived from camera geometry. Each event provides one constraint on (p, q, r), and stacking N events yields the least-squares estimate:

$$\hat{\boldsymbol{\omega}} = (\boldsymbol{H}^\top \boldsymbol{H})^{-1} \boldsymbol{H}^\top \boldsymbol{y}.$$

Figure 6.10 visualizes the reference frames used (inertial, camera, and star projection).

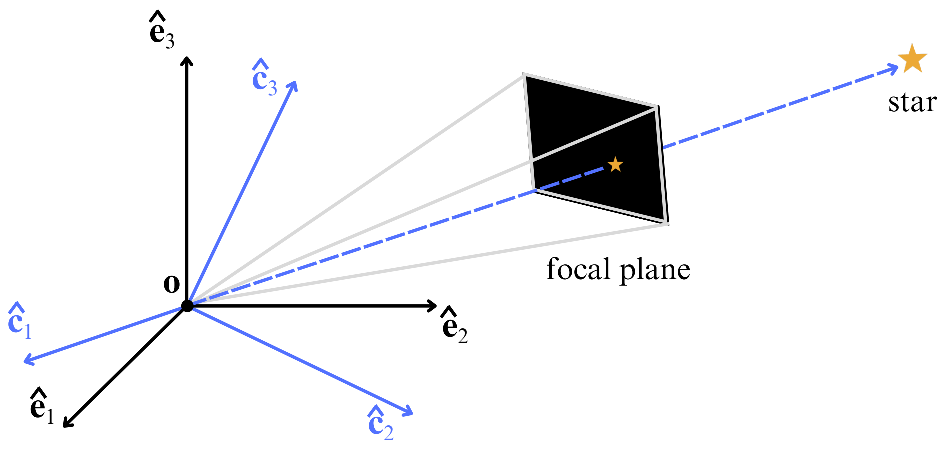


Figure 6.10: Reference frames used in the event-based angular rate estimation pipeline and projection of a star onto the camera focal plane. The inertial frame is aligned with catalogued star directions, while spacecraft rotation induces apparent star motion in the camera frame.

Since the projection model causes the roll component (r) to be weakly observable, single-camera estimation performs poorly on that axis, as shown later in Fig. 6.13.

6.4.2 Event-Based Sensing and Contrast Maximization

Event cameras report asynchronous events (x, y, t, k) when changes in brightness exceed a threshold. As stars move across the sensor due to spacecraft rotation, they produce trails of positive and negative events indicating edges of motion.

Figure 6.11 shows the principle of contrast maximization: events are "unwarped" back in time using a candidate motion field (u, v). The candidate that maximizes spatial contrast corresponds to the correct apparent velocity.

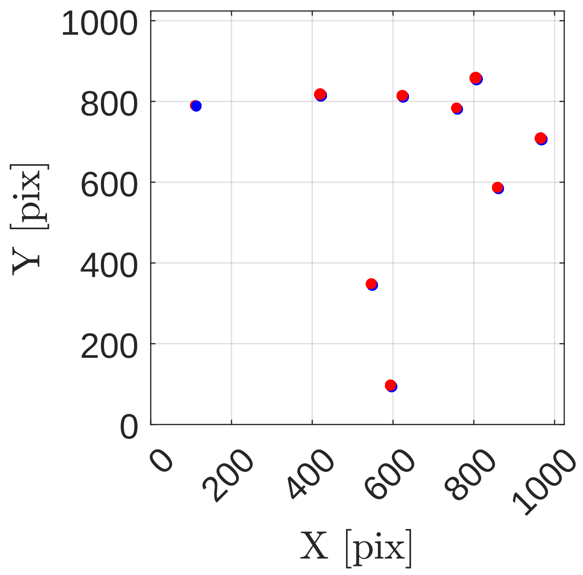


Figure 6.11: Contrast maximization for apparent star-motion reconstruction. Left: raw event stream projected onto the focal plane. Right: events warped with the optimal motion-field estimate, producing a high-contrast image.

6.4.3 Simulation Pipeline

A full synthetic pipeline was built to validate the method (Fig. 6.12), using HIPPARCOS and GAIA star catalogs, random spacecraft attitudes and angular velocities, photometric projection, event triggering, motion field estimation, and least-squares recovery of (p, q, r).

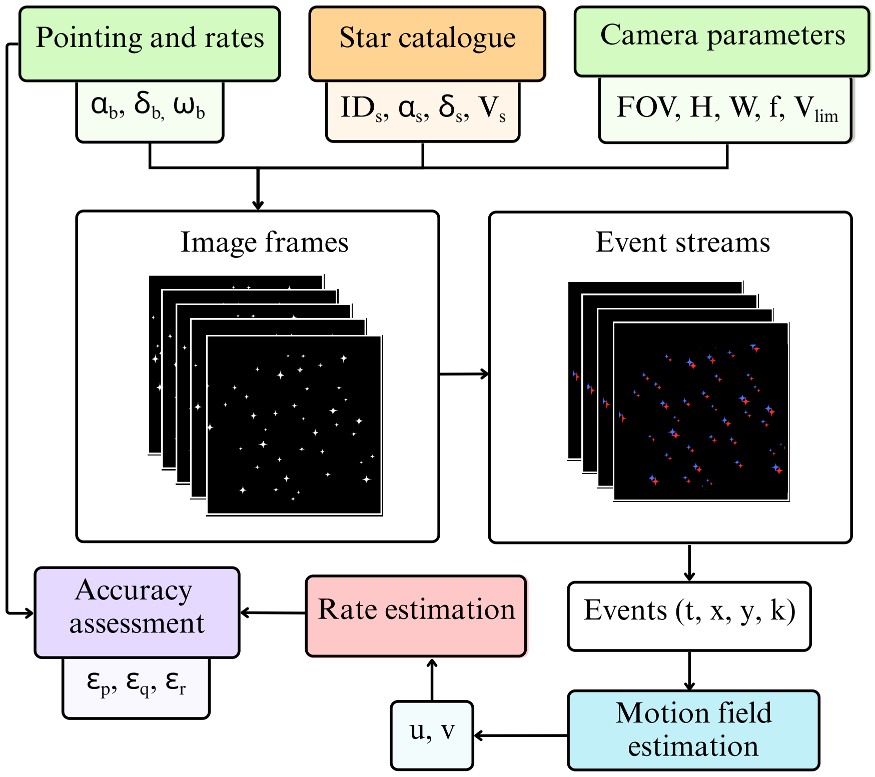


Figure 6.12: Simulation pipeline used to evaluate event-based angular rate estimation: (1) star catalog retrieval, (2) camera and boresight definition, (3) star projection and image synthesis, (4) event triggering, (5) motion-field estimation, (6) angular-rate recovery and accuracy evaluation.

6.4.4 Performance of Single-Camera Estimation

Using 100 random simulations of boresight and angular velocities, the method achieves good accuracy in p and q, but poor accuracy in the roll component r, consistent with projection-model limitations.

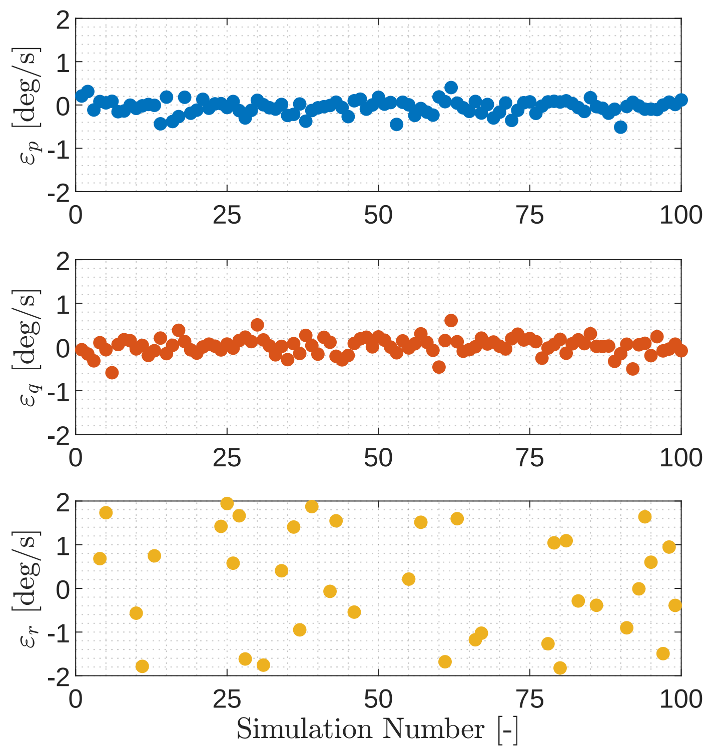


Figure 6.13: Angular velocity estimation error for single-camera setups. Pitch (p) and yaw (q) are well estimated, whereas the roll rate (r) is not reliably observable due to projection-geometry limitations.

6.4.5 Dual-Camera Configuration and Sensor Fusion

To recover full 3-axis angular velocity, an **orthogonal dual-camera setup** is introduced: each camera's weak axis becomes the strong axis of the other. The combined configuration yields complete observability.

The fusion rule blends the well-estimated axes from each sensor to recover (p, q, r) robustly.

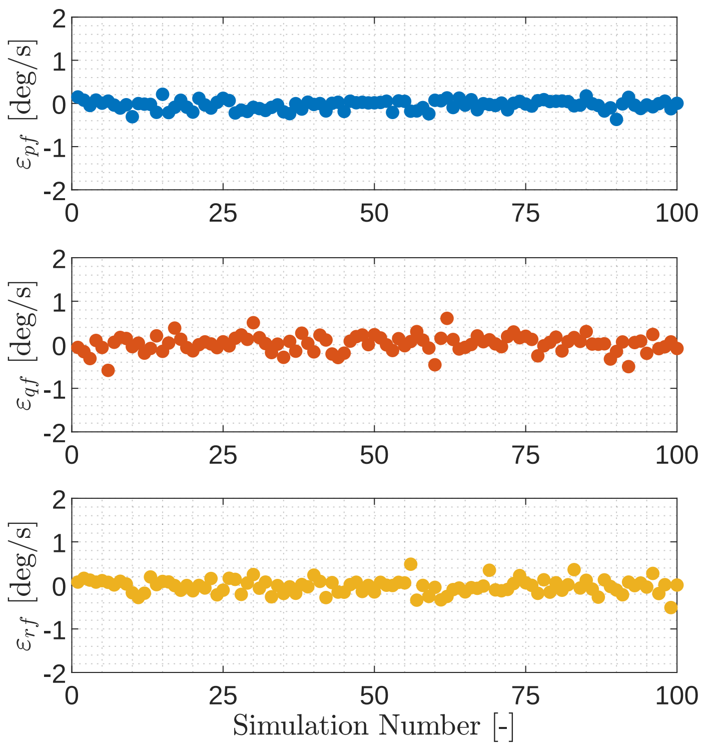


Figure 6.14: Estimation accuracy after dual-camera sensor fusion. All three axes achieve comparable performance, eliminating the roll-axis weakness present in single-camera setups.

Table 6.2 reports the final RMS error across all simulations.

Table 6.2: Angular velocity RMS error (deg/s) for single- and dual-camera configurations.

Configuration	ε_p	ε_q	ε_r	**RMS**
Single Camera	0.0165	0.0192	0.3060	0.3070
Dual Cameras	0.0115	0.0192	0.0160	0.0275

6.4.6 Discussion

The results demonstrate that event-based vision can effectively estimate angular rates by leveraging the apparent motion of stars, particularly when paired with a dual-camera configuration. This approach enables redundancy against gyroscope failures, offers microsecond temporal resolution, and is highly suitable for small spacecraft thanks to event cameras' low size, weight, and power.

6.4.7 Summary

This study illustrates a novel sensing modality for spacecraft attitude-rate estimation. Although not part of the thesis' core robotics-RL contributions, it complements the broader theme of autonomy under sensing uncertainty and highlights opportunities to integrate neuromorphic sensing with learning-based control in future work.

6.5 Conclusion of Additional Studies

This chapter presented three complementary studies that, while distinct from the main simulation and reinforcement learning frameworks developed throughout the thesis, share a common objective: advancing autonomous spacecraft operations under sensing, modeling, and control uncertainties.

The first contribution (RL-AVIST) proposed a learning-based visual inspection framework for 6DoF spacecraft motion using event-based observations. It demonstrated how generalist and specialist policies can be trained to track dynamic inspection trajectories across a variety of space targets, using the SRB simulator. This work highlights the potential of deep RL beyond ground-based systems and motivates future extensions to hardware-in-the-loop vision pipelines.

The second study introduced an excitation-based approach to inertia tensor estimation using reaction wheel torques. Through controlled system identification campaigns, it was shown that even short excitation profiles can yield accurate inertia estimates, validating their applicability in adaptive control settings. This contributes to the broader theme of learning or estimating physical models for onboard decision-making.

Finally, the third work proposed an angular rate estimation method using event-based camera data and starfield motion. The method achieved sub-0.03 deg/s accuracy using dual orthogonal sensors, demonstrating a viable alternative or backup to gyroscopes in space. This study illustrates how emerging neuromorphic sensors can be exploited for navigation tasks, expanding the perception capabilities of future spacecraft.

Collectively, these studies broaden the scope of the thesis by exploring how learning, estimation, and neuromorphic sensing can support space autonomy in scenarios where conventional models or sensors may be unavailable, inaccurate, or degraded. While developed outside the main thesis stack, they point toward promising research directions for integrating learning-based control with adaptive models and alternative sensing modalities in space robotics.

Chapter 7

Conclusion and Future Work

The work presented in this thesis set out to investigate how deep reinforcement learning (DRL), when paired with principled simulation design, can enable scalable, robust, and generalizable control of autonomous robots operating in highly uncertain environments. Across spacecraft simulators, multi-robot navigation frameworks, and aerodynamic vehicles under ground effect, this thesis has argued for a unified perspective: high-fidelity physics, structured task design, and modular software interfaces are not peripheral elements of DRL-based robotics — they are central enablers of successful learning and transfer.

Beginning with a rigorous overview of DRL foundations (Chapters 1–2), the thesis highlighted that modern actor–critic and policy gradient methods excel in continuous control but remain sensitive to modeling errors, insufficient state information, and non-stationary real-world dynamics. These challenges motivated the first major contribution: the design of specialized simulation infrastructures that more faithfully capture the physics and uncertainties faced by real systems.

Chapters 3–5 developed three such frameworks:

- **RANS and DRIFT** (Chapter 3): GPU-accelerated 3DoF/6DoF spacecraft simulators enabling domain randomization, large-scale parallel training, and sim-to-real validation on floating platforms.

- **RoboRAN** (Chapter 4): a multi-domain navigation framework for cross-robot, cross-medium generalization with a unified training and deployment stack validated on Turtlebots, Kingfisher USV, and floating platforms.
- **FALCON-S** (Chapter 5): a dual-backend (GPU/CPU) 6DoF physics suite for fixed-wing aerial vehicles in ground effect, supporting RL and optimal control with high aerodynamic fidelity.

Together, these systems enabled controlled experimentation on learning efficiency, robustness, generalization, and sim-to-real transfer. The final chapter (Chapter 6) complemented these contributions with additional studies on RL-based visual inspection around orbiting assets (RL-AVIST) and spacecraft mass-property estimation, broadening the scope of the thesis while maintaining its central focus on autonomy in uncertain conditions.

In this concluding chapter, we synthesize the scientific outcomes by revisiting the research questions introduced in Chapter 1, articulating how each was addressed by the frameworks, experiments, and analyses in the thesis. We then discuss broader implications for robotics and autonomy research, identify limitations, and outline future research directions — including sim-to-real-to-sim loops, world models, and continual-learning pipelines for long-lived autonomous robots.

7.1 Answers to Research Questions

RQ1. How can we design simulation frameworks that support scalable, physically realistic, and task-agnostic RL for autonomous robots?

Answer. This thesis addresses RQ1 by developing a suite of simulation frameworks tailored for scalable and physically grounded reinforcement learning across space-relevant robotic systems.

- **RANS (Chapter 3)** enables fast, reproducible training for spacecraft control via a GPU-accelerated backend (6,000+ envs per GPU), configurable thrust models, and modular task definitions such as TrackX, TrackXY, and TrackXYVel, all operating under user-defined disturbance profiles.

- **DRIFT (Chapter 3)** validates this approach on hardware using a floating platform testbed, confirming the fidelity of 3DoF translation and the sim-to-real consistency of disturbance-aware PPO policies.

- **RoboRAN (Chapter 4)** abstracts robot and task definitions through unified IsaacLab interfaces, enabling flexible composition and reproducible training of 16 robot–task pairs across land, water, and microgravity domains.

- **FALCON-S (Chapter 5)** highlights the value of dual backends — Warp for high-throughput learning and CPU for classical control — while introducing fine-grained aerodynamic and actuator models for fixed-wing aircraft in ground effect.

Together, these frameworks demonstrate that effective simulation infrastructure must combine: *(i) modular design, (ii) high-throughput physics backends, (iii) realistic actuation and sensing, and (iv) unified APIs enabling multi-domain extensibility.*

RQ2. To what extent can reinforcement learning policies generalize across tasks, robots, and environmental conditions?

Answer. Generalization was systematically evaluated in RoboRAN (Chapter 4) and supported by findings from RANS/DRIFT:

- **Cross-task generalization**. In RoboRAN, each robot (Turtlebot2, Kingfisher, Floating Platform) successfully learned *four distinct navigation tasks*, including GoToPosition and GoThroughPositions. Policies trained with domain randomization showed high success rates across unseen initial conditions.

- **Cross-robot generalization**. RoboRAN's unified task API enabled training the same tasks using fundamentally different mobility systems (thrusters, differential drive, water-based propulsion). This demonstrated that shared task structures extend across vehicle types when dynamics are abstracted appropriately.
- **Environmental generalization**. DRIFT showed that PPO policies trained under stochastic thrust perturbations retained stable behavior on real hardware. RoboRAN reproduced this for USV and ground robots under wind, drag, and water disturbances.
- **Generalization via world models**. RL-AVIST (Chapter 6) demonstrated that DreamerV3 generalizes better than PPO/TD3 to unseen orbital inspection paths, spacecraft morphologies, and target geometries.

In summary, the thesis shows that DRL generalizes effectively when supported by structured APIs, domain randomization, and unified learning pipelines — and that this generalization transfers across real platforms.

RQ3. Which techniques most effectively bridge the simulation–reality gap in uncertain environments?

Answer. The thesis introduced and empirically validated several strategies:

- **Physics-grounded domain randomization** (Chapters 3–4): injecting noise in thrust curves, actuator latency, water drag, inertia scaling, and sensor bias improved robustness and reduced overfitting.
- **Realistic actuator and sensor models** Enabled in RANS, DRIFT, and FALCON-S: actuator dynamics, drag models, aerodynamic coefficients, and reaction wheel coupling increased stability during transfer.

- **Unified simulation $\rightarrow$ deployment pipeline** RoboRAN's ROS2-based lightweight executor allowed IsaacLab-trained policies (skrl, rl_games) to run directly on hardware without the simulation layer.

- **Trajectory-level evaluation and debugging tools** Heatmaps, success metrics, and trajectory overlays (Chapter 4) enabled detecting transfer failure modes such as heading drift, inertia mismatch, and underdamped dynamics.

- **Real-world validation** DRIFT and RoboRAN both demonstrated successful zero-shot transfer to hardware platforms.

These findings establish that bridging sim-to-real requires a combination of modeling fidelity, structured randomization, unified interfaces, and principled evaluation protocols.

7.2 Summary of Fulfilled Objectives

This thesis was guided by the ambition to bridge the gap between simulation-based learning and physically grounded robotic autonomy in uncertain environments. The objectives outlined in the introduction — including building robust simulation platforms, enabling sim-to-real control, supporting diverse robot-task combinations, and extending DRL techniques to space-relevant scenarios — have been systematically addressed through the core contributions:

- **Design of high-fidelity, task-rich simulators:** RANS and FALCON-S provide modular and scalable environments for spacecraft and fixed-wing vehicle control, respectively, featuring 6DoF dynamics, domain randomization, environmental disturbances, and structured benchmarks. These frameworks enable training and evaluation under realistic uncertainty and dynamics constraints.

- **Generalizable learning across robots and tasks:** RoboRAN operationalizes multi-robot multi-task reinforcement learning in a reproducible and decoupled fashion. It supports fast training, real-world deployment, and systematic comparisons across terrestrial, aquatic, and microgravity platforms.

- **Effective sim-to-real transfer:** DRIFT and RoboRAN demonstrate successful deployment of policies trained in IsaacLab to real floating platforms, USVs, and ground robots. Real-world results validate simulation fidelity and the training methodology under noisy actuation, imperfect sensing, and inertia mismatches.

- **Deployment-ready pipelines for space robotics:** Through the RL-AVIST and event-based angular rate estimation studies, the thesis expands into on-orbit robotic autonomy, showing that learned perception and estimation pipelines can complement or replace classical GNC systems. These works provide a foundation for future learning-enabled space missions.

- **Benchmarking and reproducibility tools:** The developed frameworks offer consistent metrics, logging utilities (e.g., WandB), and modular controllers to enable comparative studies. These tools ensure that future work can build upon this foundation with transparency and scalability.

Taken together, these contributions fulfill the initial research objectives while laying the groundwork for broader generalization, integration with adaptive control, and lifelong learning in robotic systems.

7.3 Broader Implications and Impact

The research presented in this thesis contributes to a broader understanding of how deep reinforcement learning can be made usable, scalable, and reliable for real-world robotics. By emphasizing physics-aware simulation, multi-robot frameworks, and standardized evaluation pipelines, the thesis shows that autonomy under uncertainty is not a byproduct of learning but a property that must be scaffolded by design.

Several implications emerge:

- **Simulation as a first-class research artifact**. Tools like RANS, RoboRAN, and FALCON-S elevate simulation from an auxiliary tool to a core enabler of reproducible and generalizable research. This supports the growing recognition of simulators as benchmarks — not just pre-training environments — in RL and robotics communities.

- **Unified frameworks enable cross-domain robotics**. RoboRAN demonstrates that a single codebase can train floating platforms, USVs, and mobile ground robots across multiple tasks, showing that modular abstractions facilitate not only research efficiency but also insight transfer between domains.

- **Scalable DRL with real-world grounding is achievable**. The successful deployment of policies trained entirely in IsaacLab (DRIFT, RoboRAN) onto real robots illustrates that modern GPU simulation and structured randomization can yield deployable policies in practice.

- **DRL can extend traditional spacecraft autonomy**. The RL-AVIST and angular rate estimation studies suggest that vision-based and event-based sensing, when paired with learned policies or estimation pipelines, can offer complementary or fallback autonomy modes to classical GNC systems.

Together, these outcomes suggest a path forward where learning-based control, estimation, and sensing systems are not only academically interesting, but also practically viable across domains such as space robotics, mobile navigation, and aerial robotics.

7.4 Future Work

While this thesis establishes robust baselines and frameworks, several avenues remain open for expanding and deepening the research.

- **Sim-to-Real-to-Sim Loops**. While we demonstrated sim-to-real transfer (e.g., DRIFT, RoboRAN), the reverse process — using real-world data to refine simulators — remains underexplored. A promising direction is to iteratively adjust dynamics, noise profiles, or actuator models using real-world trajectory deviations as a supervisory signal.

- **World Models for Long-Horizon Planning**. The success of DreamerV3 in RL-AVIST suggests world models are promising for domains with sparse rewards or partial observability. Future work could integrate world models into RoboRAN or FALCON-S to enable sample-efficient learning or hybrid model-based/model-free control.

- **Continual and Lifelong Learning for Robotics**. Real-world deployment demands agents that adapt over time, without catastrophic forgetting. With RoboRAN's modular setup, it becomes possible to explore continual learning strategies — e.g., replay-based regularization, task-conditioned policies, or forward/backward transfer metrics — across tasks and robots.

- **Cross-Vehicle and Multi-Agent Generalization**. While generalization across individual robots and tasks was demonstrated, scaling to multi-agent or cross-fleet settings (e.g., heterogeneous robot swarms) could be a natural extension, especially using curriculum learning and domain-conditioned policies.

- **In-the-Loop Estimation and Control**. RL-AVIST and the Angular Rate Estimation frameworks offer strong estimation backbones. An exciting future direction is to close the loop by feeding such estimates directly into adaptive control pipelines or online policy updates — enabling perception-adaptive controllers.

Collectively, these directions aim to move from robust single-task deployment to lifelong autonomy: systems that can learn, adapt, and generalize in complex, dynamic environments.

References

[1] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. “Sim-to-real transfer in deep reinforcement learning for robotics: a survey”. In: *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE. 2020, pp. 737–744.

[2] Ilge Akkaya et al. “Solving rubik’s cube with a robot hand”. In: *arXiv preprint arXiv:1910.07113* (2019).

[3] Xue Bin Peng et al. “Sim-to-real transfer of robotic control with dynamics randomization”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[4] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[5] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[6] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[7] Viktor Makoviychuk et al. “Isaac gym: High performance gpu-based physics simulation for robot learning”. In: *arXiv preprint arXiv:2108.10470* (2021).

[8] C Daniel Freeman et al. “Brax-a differentiable physics engine for large scale rigid body simulation, 2021”. In: *URL http://github. com/google/brax* 6 (2021).

[9] Manolis Savva et al. “Habitat: A platform for embodied ai research”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9339–9347.

[10] Henry Zhu et al. “The ingredients of real-world robotic reinforcement learning”. In: *arXiv preprint arXiv:2004.12570* (2020).

[11] OpenAI: Marcin Andrychowicz et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[12] Eric Liang et al. “RLlib: Abstractions for distributed reinforcement learning”. In: *International conference on machine learning*. PMLR. 2018, pp. 3053–3062.

[13] Antonin Raffin et al. “Stable-baselines3: Reliable reinforcement learning implementations”. In: *Journal of machine learning research* 22.268 (2021), pp. 1–8.

[14] Denys Makoviichuk and Viktor Makoviychuk. *rl-games: A High-performance Framework for Reinforcement Learning*. `https://github.com/Denys88/rl_games`. May 2021.

[15] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe. 2009, p. 5.

[16] Miguel Olivares-Mendez et al. “Zero-G Lab: A multi-purpose facility for emulating space operations”. In: *Journal of Space Safety Engineering* 10.4 (2023), pp. 509–521.

[17] Trevor Hocksun Kwan et al. “An air bearing table for satellite attitude control simulation”. In: *2015 IEEE 10th conference on industrial electronics and applications (ICIEA)*. IEEE. 2015, pp. 1420–1425.

[18] Daniel Sakoda and James A Horning. “Overview of the NPS Spacecraft Architecture and Technology Demonstration Satellite, NPSAT1”. In: *Internal Report* (2002).

[19] Robin Amsters and Peter Slaets. “Turtlebot 3 as a robotics education platform”. In: *International Conference on Robotics in Education (RiE)*. Springer. 2019, pp. 170–181.

[20] Carlos Barrera et al. "Trends and challenges in unmanned surface vehicles (Usv): From survey to shipping". In: *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation* 15 (2021).

[21] Laminar Research. *X-Plane Flight Simulator*. `https://www.x-plane.com/`. Accessed: 2025-09-12. 2024.

[22] Harold Klee and Randal Allen. *Simulation of dynamic systems with MATLAB® and Simulink®*. Crc Press, 2018.

[23] David Abel, Mark K Ho, and Anna Harutyunyan. "Three Dogmas of Reinforcement Learning". In: *Reinforcement Learning Conference*. 2024.

[24] Mayank Mittal et al. "Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments". In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: `10.1109/LRA.2023.3270034`.

[25] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[26] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[27] Andrej Orsula et al. "Towards Benchmarking Robotic Manipulation in Space". In: *Conference on Robot Learning (CoRL) Workshop on Mastering Robot Manipulation in a World of Abundant Data*. 2024.

[28] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[29] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *arXiv preprint arXiv:1802.09477* (2018).

[30] Matteo El Hariry, Vittorio Franzese, and Miguel Olivares-Mendez. "Towards Active Excitation-Based Dynamic Inertia Identification in Satellites". In: *iSpaRo*. 2025.

[31] Vittorio Franzese and Matteo El Hariry. "Spacecraft Angular Rate Estimation via Event-Based Camera Sensing". In: *ASR* (2025). Under review.

[32] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

[33] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[34] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: `1312.5602 [cs.LG]`. URL: `https://arxiv.org/abs/1312.5602`.

[35] Danijar Hafner et al. "Mastering diverse control tasks through world models". In: *Nature* (2025), pp. 1–7.

[36] Matteo El-Hariry, Antoine Richard, and Miguel Olivares-Mendez. "Rans: Highly-parallelised simulator for reinforcement learning based autonomous navigating spacecrafts". In: *arXiv preprint arXiv:2310.07393* (2023).

[37] Matteo El-Hariry et al. "Drift: Deep reinforcement learning for intelligent floating platforms trajectories". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024, pp. 14034–14041.

[38] Matteo El Hariry et al. "FALCON-S: Fixed-Wing Aerodynamics and Learning Control Benchmark". In: *ICLR Datasets and Benchmarks* (2026). Under review.

[39] Matteo El-Hariry et al. "NavBench: A Unified Robotics Benchmark for Reinforcement Learning-Based Autonomous Navigation". In: *TMLR* (2025). Under review.

[40] Yongshuai Liu, Avishai Halev, and Xin Liu. “Policy learning with constraints in model-free reinforcement learning: A survey”. In: *The 30th international joint conference on artificial intelligence (ijcai)*. 2021.

[41] Weiye Zhao et al. “State-wise safe reinforcement learning: A survey”. In: *arXiv preprint arXiv:2302.03122* (2023).

[42] Shangding Gu et al. “A review of safe reinforcement learning: Methods, theories and applications”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[43] Javier Garcıa and Fernando Fernández. “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[44] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. Ieee. 2004, pp. 2149–2154.

[45] Erwin Coumans and Yunfei Bai. *Pybullet, a python module for physics simulation for games, robotics and machine learning*. 2016.

[46] Eric Rohmer, Surya PN Singh, and Marc Freese. “V-REP: A versatile and scalable robot simulation framework”. In: *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2013, pp. 1321–1326.

[47] Miles Macklin. *Warp: Differentiable Spatial Computing for Python*. 2024.

[48] Stephen James et al. “Rlbench: The robot learning benchmark & learning environment”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3019–3026.

[49] Nicolas Heess et al. “Memory-based control with recurrent neural networks”. In: *arXiv preprint arXiv:1512.04455* (2015).

[50] Tobias Johannink et al. “Residual reinforcement learning for robot control”. In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 6023–6029.

[51] J. Kopacz, R. Herschitz, and J. Roney. “Small satellites an overview and assessment”. In: *Acta Astronautica* 170 (2020), pp. 93–105. DOI: 10.1016/j.actaastro.2020.01.034.

[52] G. Curzi, D. Modenini, and P. Tortora. “Large constellations of small satellites: a survey of near future challenges and missions”. In: *Aerospace* 7 (9 2020), p. 133. DOI: 10.3390/aerospace7090133.

[53] M. B. Quadrelli et al. “Guidance, navigation, and control technology assessment for future planetary science missions”. In: *Journal of Guidance, Control, and Dynamics* 38 (7 2015), pp. 1165–1186. DOI: 10.2514/1.g000525.

[54] J. Song, D. Rondao, and N. Aouf. “Deep learning-based spacecraft relative navigation methods: a survey”. In: *Acta Astronautica* 191 (2022), pp. 22–40. DOI: 10.1016/j.actaastro.2021.10.025.

[55] Brian Gaudet, Richard Linares, and Roberto Furfaro. “Deep reinforcement learning for six degree-of-freedom planetary landing”. In: *Advances in Space Research* 65.7 (2020), pp. 1723–1741.

[56] Stefan Willis, Dario Izzo, and Daniel Hennes. *Reinforcement Learning for Spacecraft Maneuvering Near Small Bodies*. 2016.

[57] T. Tanaka, M. Cescon, and H. A. Malki. “Linear quadratic tracking with reinforcement learning based reference trajectory optimization for the lunar hopper in simulated environment”. In: *IEEE Access* 9 (2021). DOI: 10.1109/access.2021.3134592.

[58] David M. Chan and Ali-akbar Agha-mohammadi. *Autonomous Imaging and Mapping of Small Bodies Using Deep Reinforcement Learning*. 2019. DOI: 10.1109/AERO.2019.8742147.

[59] Darrel J Conway and Steven P Hughes. “The general mission analysis tool (GMAT): Current features and adding custom functionality”. In: *International Conference on Astrodynamics Tools and Techniques (ICATT)*. 2010.

[60] C Acton et al. “SPICE tools supporting planetary remote sensing”. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 41 (2016), pp. 357–359.

[61] Cyrus Foster. “Trajectory Browser: An online tool for interplanetary trajectory analysis and visualization”. In: *2013 IEEE Aerospace Conference*. IEEE. 2013, pp. 1–6.

[62] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[63] Olivier Michel. “Cyberbotics ltd. webots™: professional mobile robot simulation”. In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5.

[64] Tomasz Rybus and Karol Seweryn. “Planar air-bearing microgravity simulators: Review of applications, existing solutions and design parameters”. In: *Acta Astronautica* 120 (2016), pp. 239–259.

[65] Z. Huang et al. “Characterizing an air-bearing testbed for simulating spacecraft dynamics and control”. In: *Aerospace* 9 (5 2022). DOI: 10.3390/aerospace9050246.

[66] 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”. In: *Acta Astronautica* (2023). DOI: 10.1016/j.actaastro.2022.11.056.

[67] Cristóbal Nieto-Peroy et al. *Simulation of Spacecraft Formation Maneuvers by means of Floating Platforms*. 2021. DOI: 10.1109/AERO50100.2021.9438537.

[68] M. Sabatini, P. Gasbarri, and G. B. Palmerini. “Coordinated control of a space manipulator tested by means of an air bearing free floating platform”. In: *Acta Astronautica* 139 (2017), pp. 296–305. DOI: 10.1016/j.actaastro.2017.07.015.

[69] A. Banerjee et al. "On the design, modeling and experimental verification of a floating satellite platform". In: *IEEE Robotics and Automation Letters* 7 (2 2022), pp. 1364–1371. DOI: 10.1109/lra.2021.3140134.

[70] X. Yu, P. Wang, and Z. Zhang. "Learning-based end-to-end path planning for lunar rovers with safety constraints". In: *Sensors* 21 (3 2021), p. 796. DOI: 10.3390/s21030796.

[71] K. Hovell and S. Ulrich. *On deep reinforcement learning for spacecraft guidance*. 2020. DOI: 10.2514/6.2020-1600.

[72] D Athauda et al. "Intelligent Motion Planning for Collision Free Autonomous Docking of Satellite Emulation Platform Using Reinforcement Learning". In: *IFAC-PapersOnLine* 56.2 (2023), pp. 3354–3359.

[73] Zhehua Zhou et al. "Towards Building AI-CPS with NVIDIA Isaac Sim: An Industrial Benchmark and Case Study for Robotics Manipulation". In: *arXiv preprint arXiv:2308.00055* (2023).

[74] Bariş Can Yalçin et al. "Lightweight Floating Platform for Ground-Based Emulation of On-Orbit Scenarios". In: *IEEE Access* 11 (2023), pp. 94575–94588. DOI: 10 . 1109 / ACCESS.2023.3311202.

[75] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: $https://www.wandb.com/$.

[76] Robert F Stengel. *Optimal control and estimation*. 1994.

[77] Elisa Sara Varghese, Anju K Vincent, and V Bagyaveereswaran. "Optimal control of inverted pendulum system using PID controller, LQR and MPC". In: *IOP Conference Series: Materials Science and Engineering*. IOP Publishing. 2017.

[78] Mohsen Khosravi, Hossein Azarinfar, and Kiomars Sabzevari. "Design of infinite horizon LQR controller for discrete delay systems in satellite orbit control: A predictive controller and reduction method approach". In: *Heliyon* (2024).

[79] Vivek Muralidharan et al. “Rendezvous in cislunar halo orbits: Hardware-in-the-loop simulation with coupled orbit and attitude dynamics”. In: *Acta Astronautica* 211 (2023), pp. 556–573.

[80] Vivek Muralidharan et al. “On-ground validation of orbital GNC: Visual navigation assessment in robotic testbed facility”. In: *Astrodynamics* (2024).

[81] Antonio Serrano-Muñoz et al. “skrl: Modular and Flexible Library for Reinforcement Learning”. In: *arXiv preprint arXiv:2202.03825* (2022).

[82] Dirk Merkel. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014.239 (2014), p. 2.

[83] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: `10.1126/scirobotics.abm6074`. URL: `https://www.science.org/doi/abs/10.1126/scirobotics.abm6074`.

[84] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *arXiv:1504.00702* (2015). Available at arXiv: `https://arxiv.org/abs/1504.00702`.

[85] Xue Bin Peng et al. “DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills”. In: *arXiv:1804.02717* (2018). DOI: `10.1145/3197517.3201311`.

[86] Joonho Lee et al. “Learning Quadrupedal Locomotion over Challenging Terrain”. In: *Science Robotics 2020 Vol. 5, Issue 47, eabc5986* (2020). DOI: `10.1126/scirobotics.abc5986`. eprint: `arXiv:2010.11251`.

[87] Youngwoon Lee et al. “IKEA Furniture Assembly Environment for Long-Horizon Complex Manipulation Tasks”. In: *arXiv:1911.07246* (2019).

[88] Stephen James et al. “RLBench: The Robot Learning Benchmark & Learning Environment”. In: *arXiv:1909.12271* (2019).

[89] Yuke Zhu et al. “robosuite: A Modular Simulation Framework and Benchmark for Robot Learning”. In: *arXiv:2009.12293* (2020).

[90] Minho Heo et al. “FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation”. In: *arXiv:2305.12821* (2023).

[91] Kai Zhu and Tao Zhang. “Deep reinforcement learning based mobile robot navigation: A review”. In: *Tsinghua Science and Technology* 26.5 (2021), pp. 674–691. DOI: 10.26599/TST.2021.9010012.

[92] Manolis Savva et al. “Habitat: A Platform for Embodied AI Research”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.

[93] Andrew Szot et al. “Habitat 2.0: Training home assistants to rearrange their habitat”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021), pp. 251–266.

[94] Daniel Perille et al. “Benchmarking Metric Ground Navigation”. In: *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2020.

[95] Shu-Ang Yu et al. “FlightBench: Benchmarking Learning-based Methods for Ego-vision-based Quadrotors Navigation”. In: *arXiv preprint arXiv:2406.05687* (2024).

[96] Davide Corsi, Davide Camponogara, and Alessandro Farinelli. “Aquatic Navigation: A Challenging Benchmark for Deep Reinforcement Learning”. In: *arXiv:2405.20534* (2024).

[97] Fei Xia et al. “Interactive Gibson Benchmark (iGibson 0.5): A Benchmark for Interactive Navigation in Cluttered Environments”. In: *IEEE Robotics and Automation Letters, Vol. 5, No. 2, April 2020* (2019). DOI: 10.1109/LRA.2020.2965078. eprint: arXiv:1910.14442.

[98] Zifan Xu et al. “Benchmarking Reinforcement Learning Techniques for Autonomous Navigation”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 9224–9230. DOI: 10.1109/ICRA48891.2023.10160583.

[99] Adil Zouitine et al. “RRLS: Robust Reinforcement Learning Suite”. In: *arXiv* (2024).

[100] Shangding Gu et al. “Robust Gymnasium: A Unified Modular Benchmark for Robust Reinforcement Learning”. In: *Github* (2024).

[101] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[102] Yi Zhou et al. *On the Continuity of Rotation Representations in Neural Networks*. 2018. eprint: `arXiv:1812.07035`.

[103] Jon Berndt. “JSBSim: An open source flight dynamics model in C++”. In: *AIAA modeling and simulation technologies conference and exhibit*. 2004, p. 4923.

[104] Yunlong Song et al. “Flightmare: A flexible quadrotor simulator”. In: *Conference on Robot Learning*. PMLR. 2021, pp. 1147–1157.

[105] Christoph Richter and Ruben Calix. “QPlane: A reinforcement learning toolkit for fixed-wing aircraft simulation”. In: *Proceedings of the ACM Multimedia Systems Conference (MMSys)*. ACM. 2021, pp. 334–337.

[106] Zifan Xue et al. “NeuralPlane: Efficiently parallelizable platform for fixed-wing aircraft control”. In: *NeurIPS Datasets and Benchmarks*. 2024.

[107] Shital Shah et al. “AirSim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and Service Robotics*. 2017.

[108] Franz Furrer et al. “RotorS—A modular Gazebo MAV simulator framework”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 46–51.

[109] Felix Berkenkamp et al. “Safe model-based reinforcement learning with stability guarantees”. In: *NeurIPS* 32 (2019).

[110] Jiayuan Liu, Jemin Hwangbo, Jongwoo Lee, et al. “Impact of dynamics randomization on policy transfer for quadrotors”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5300–5307.

[111] Calin Basescu et al. “Post-stall landing of a fixed-wing UAV using learned aerodynamic models and nonlinear MPC”. In: *IEEE Robotics and Automation Letters* 8.3 (2023), pp. 1681–1688.

[112] Xingyou Pan, Yikang Cui, Yi Zhang, et al. “WarpDrive: Extremely Fast End-to-End Deep Multi-Agent Reinforcement Learning on a GPU”. In: *arXiv preprint arXiv:2108.13976* (2021).

[113] NASA OpenVSP Team. *Open VSP: NASA’s Open Vehicle Sketch Pad*. `https://openvsp.org/`. Accessed: 2025-09-12. 2025.

[114] Warren F Phillips and Douglas F Hunsaker. “Lifting-line predictions for induced drag and lift in ground effect”. In: *Journal of Aircraft* 50.4 (2013), pp. 1226–1233.

[115] HL Dryden and AM Kuethe. *Effect of turbulence in wind tunnel measurements*. Vol. 342. US Government Printing Office, 1930.

[116] Grady Williams et al. “Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving”. In: *IEEE Transactions on Robotics* 34.6 (2018), pp. 1603–1622. DOI: `10.1109/TRO.2018.2865891`.

[117] Mark Towers et al. “Gymnasium: A Standard Interface for Reinforcement Learning Environments”. In: *arXiv preprint arXiv:2407.17032* (2024).

[118] NASA Ames Research Center. *X-Plane Connect (XPC)*. `https://github.com/nasa/XPlaneConnect`. Accessed: 2025-09-12. 2025.

[119] Andrej Orsula et al. “Towards Benchmarking Robotic Manipulation in Space”. In: *Conference on Robot Learning (CoRL) Workshop on Mastering Robot Manipulation in a World of Abundant Data (MRM-D)*. 2024.

[120] Yi Ma et al. *An invitation to 3-d vision: from images to geometric models*. Vol. 26. Springer, 2004.

SNT