FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Learning Transferable Navigation Skills for USVs via Dynamics-Agnostic Behavioral Priors

Federico António Campos Barrezueta



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Andry Maykol Gomes Pinto

October 7, 2026

© Federico António Campos Barrezueta, 2026

Resumo

A navegação autónoma em ambientes marítimos continua a constituir um desafio, dado que os veículos de superfície não tripulados (USVs) operam sob dinâmicas complexas, limites de atuação e características específicas de cada plataforma. Embora a sua utilização tenha crescido em áreas como a monitorização ambiental, os levantamentos hidrográficos e a inspeção de infraestruturas marítimas críticas, a garantia de uma navegação robusta e fiável permanece difícil face à incerteza e variabilidade das condições operacionais.

Esta tese apresenta um enquadramento transferível de Aprendizagem por Reforço para competências de navegação de USVs, baseado numa fatorização em duas etapas: uma política de base geométrica aprende *behavioral priors* agnósticos à dinâmica, e um adaptador específico do veículo mapeia essas intenções geométricas para ações de controlo de baixo nível. Esta decomposição separa a geometria da tarefa das dinâmicas do veículo, melhorando a interpretabilidade e reduzindo a quantidade de retreino necessária quando a política é transferida para uma nova plataforma.

Foram desenvolvidas duas competências de navegação, nomeadamente a OrbitXY, uma tarefa de manutenção de órbita com simetria nos sentidos horário e anti-horário e a FollowPath, uma tarefa de seguimento de trajetória em referencial de Frenet, baseada em trajetórias spline. Ambas as competências são inicialmente treinadas em simulação simplificada e sem fricção, de modo a aprender a geometria subjacente da tarefa sendo posteriormente adaptadas a um modelo de veículo de maior fidelidade, com saturação de impulso, inércia e efeitos hidrodinâmicos. O treino é realizado com Proximal Policy Optimization, utilizando currículos e formulações de recompensa específicas para cada tarefa.

As políticas resultantes são avaliadas por métricas de simulação que medem sucesso, precisão de seguimento, estabilidade e suavidade do controlo, bem como através de experiências de transferência no ambiente de simulação final. Neste ambiente, a OrbitXY obteve uma taxa média de sucesso de 80% e um erro médio de distância de 0,57 metros, num conjunto de 40 condições com raios, velocidades e direção de órbita variáveis. Na FollowPath, obteve-se uma taxa média de sucesso de 92% com um erro médio de distância de 0,53 metros. Foi também realizado um teste zero-shot da transferência da política de base para um quadrotor, o que reforça a transferibilidade das geometrias aprendidas. Esse teste foi realizado com uma grelha de avaliação semelhante, tendo o drone atingido uma taxa média de sucesso de 97% e um erro médio de distância de 0,45 metros.

Os resultados mostram que a fatorização proposta permite aprender de forma estável comportamentos de navegação transferíveis e suporta a sua aplicação de simulação para cenários mais realistas. Em particular, a política de base, quando estável, é preservada, captando uma estrutura geométrica reutilizável, enquanto o adapter faz a ponte para a atuação específica do veículo. No geral, a tese demonstra que *behavioral priors* agnósticos à dinâmica constituem uma base eficaz para aprender competências de navegação simultaneamente adaptáveis e transferíveis entre plataformas.

Abstract

Autonomous navigation in maritime environments continues to be a challenge, given that unmanned surface vehicles (USVs) operate under complex dynamics, operational limits, and platform-specific characteristics. Although their use has grown in areas such as environmental monitoring, hydrographic surveys, and inspection of critical maritime infrastructure, ensuring robust and reliable navigation remains difficult in the face of the uncertainty and variability of operating conditions.

This thesis presents a transferable Reinforcement Learning framework for USV navigation skills, based on a two-step factorization: a geometrically based policy learns behavioral priors agnostic to dynamics, and a vehicle-specific adapter maps these geometric intentions to low-level control actions. This decomposition separates the task geometry from the vehicle dynamics, improving interpretability and reducing the amount of retraining required when the policy is transferred to a new platform.

Two navigation skills were developed, namely, OrbitXY, an orbit maintenance task with clockwise and counterclockwise symmetry, and FollowPath, a trajectory following task in a Frenet reference frame, based on spline trajectories. Both skills are initially trained in simplified, frictionless simulation in order to learn the underlying geometry of the task, and are subsequently adapted to a higher-fidelity vehicle model with thrust saturation, inertia, and hydrodynamic effects. Training is performed with Proximal Policy Optimization, using curricula and reward formulations specific to each task. The resulting policies are evaluated by simulation metrics that measure success, tracking accuracy, stability, and control smoothness, as well as transfer experiments in the final simulation environment. In this environment, OrbitXY achieved an average success rate of 80% and an average distance error of 0.57 meters, in a set of 40 conditions with varying radii, speeds, and orbit direction. In FollowPath, an average success rate of 92% was obtained with an average distance error of 0.53 meters. A zero-shot test of the basic policy transfer to a quadrotor was also performed, reinforcing the transferability of the learned geometries. This test was performed with a similar evaluation grid, with the drone achieving an average success rate of 97% and an average distance error of 0.45 meters.

The results show that the proposed factorization allows for the stable learning of transferable navigation behaviors and supports their simulation application for more realistic scenarios. In particular, the basic policy, when stable, is preserved, capturing a reusable geometric structure, while the adapter bridges the gap to the vehicle-specific actuation. In general, the thesis demonstrates that dynamics-agnostic behavioral priors constitute an effective basis for learning navigation skills that are simultaneously adaptable and transferable across platforms.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework for achieving a better and more sustainable future for all. It includes 17 goals[1] to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.
"You are building Innovation (9) to support Climate Action (13)."

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

SDG 13 Take urgent action to combat climate change and its impacts

SDG	Target	Contribution	Performance Indicators and Metrics
9	9.5	Advancing innovation in autonomous navigation through a transferable reinforcement learning framework that separates reusable geometry from vehicle-specific control.	Convergence rate; transfer performance across vehicles; robustness of learned behaviour in simulation.
13	13.2	Supporting climate-related maritime autonomy in simulation by enabling efficient navigation policies for monitoring and inspection tasks.	Mission efficiency in simulation; success rate; robustness under varying conditions.

[1] https://sdgs.un.org/goals

Agradecimentos

Ao meu orientador, Prof. Andry Pinto, expresso o meu sincero agradecimento pela oportunidade de desenvolver este trabalho, pela constante disponibilidade, pelo acompanhamento ao longo deste percurso e, acima de tudo, pela generosa partilha de conhecimento, que foi fundamental para a concretização desta dissertação.

Quero também deixar uma palavra de profundo agradecimento à Prof.ª Maria Inês Pereira, cuja disponibilidade, prontidão e apoio foram essenciais em vários momentos de maior dificuldade. A sua ajuda foi sempre prestada com uma enorme dedicação, pela qual lhe estou muito grato.

Por fim, mas de forma alguma menos importante, o meu mais sentido obrigado à minha família. Em especial, à minha mulher, Joana, pelo amor, pela paciência, pelo incentivo constante e por nunca deixar de acreditar em mim, mesmo nos momentos mais exigentes. Aos meus pais e ao meu irmão, agradeço todo o apoio ao longo deste último ano e, sobretudo, o amor incondicional e os valores que sempre me transmitiram. Sem eles, este percurso não teria sido o mesmo.

Federico Barrezueta

Declaração de Honra

Declaro que a presente dissertação é da minha autoria e não foi previamente apresentada e avaliada nesta ou em outra instituição. As referências a outros autores (afirmações, ideias, pensamentos), assim como a utilização de trabalhos publicados respeitam escrupulosamente as regras da atribuição, e encontram-se devidamente indicadas no texto e nas referências bibliográficas, de acordo com as normas de referenciação. Tenho consciência de que a prática de plágio ou de autoplágio constituem ilícitos académicos, conforme estabelecido no Código de Ética da U.Porto. Declaro, ainda, que utilizei ferramentas de Inteligência Artificial para a realização de parte(s) da presente dissertação, e essa utilização e os seus resultados foram objeto de cuidada verificação para deteção de erros, imprecisões, atribuições infundadas ou outras formas de enviesamento de conteúdos e argumentos, bem como de determinação de autoria.

Federico Barrezueta



"And beyond what has been said, experience, which is the mother of all things, undeceives us and removes all our doubts."

Duarte Pacheco Pereira

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Objectives . . . 2
- 1.3 Document Structure . . . 3

2 State of the art **4**
- 2.1 Reinforcement Learning: Introduction . . . 4
- 2.2 Taxonomy of RL Methods . . . 6
  - 2.2.1 Policy Optimization . . . 6
  - 2.2.2 Q-Learning . . . 7
- 2.3 Hybrid Methods . . . 8
- 2.4 Reinforcement Learning for USV Navigation . . . 9
- 2.5 Modular Policy Learning . . . 15
- 2.6 Critical Review . . . 17

3 Two-Stage Transferable Navigation Framework **18**
- 3.1 Problem Formulation . . . 18
- 3.2 Orbit Following . . . 19
  - 3.2.1 Observation space . . . 20
  - 3.2.2 Action Space . . . 22
  - 3.2.3 Termination Condition . . . 23
  - 3.2.4 Reward Function . . . 24
- 3.3 Path tracking . . . 25
  - 3.3.1 Observation and Action Spaces . . . 26
  - 3.3.2 Termination Conditions . . . 28
  - 3.3.3 Reward Function . . . 29
- 3.4 Reinforcement Learning Architecture . . . 30
  - 3.4.1 Backbone and Adapter Policies . . . 31
  - 3.4.2 PPO Parameters . . . 32
- 3.5 Implementations and Training Details . . . 34
  - 3.5.1 Environment and Vehicle Dynamics . . . 34
  - 3.5.2 Training Pipeline . . . 35
  - 3.5.3 Curriculum Learning Schedules . . . 36
  - 3.5.4 Evaluation Protocol . . . 37

4 Results **39**
- 4.1 Metrics and Test Scenario . . . 41
- 4.2 Backbone Results . . . 43

4.3 Adapter Results 49
4.4 Transfer Results 53
4.5 Backbone zero-shot 61
4.6 Ablation Results 63

5 Conclusions and Future Work **66**

References **68**

List of Figures

2.1 Agent-environment interaction . . . 5
2.2 RL Algorithm taxonomy . . . 6
2.3 APF_DQN Path Results . . . 10
2.4 MK-A3C results . . . 11
2.5 PPO-ICM Control Model . . . 12
2.6 PPO-ICM Results . . . 13
2.7 DDPG-Trajectory Data . . . 14
2.8 ATL-TD3 Results . . . 15
2.9 SAC Tracking Results . . . 16

3.1 Backbone-Adapter Architecture . . . 19
3.2 OrbitXY Backbone Adapter obs . . . 20
3.3 Observation OrbitXY Scheme . . . 21
3.4 Followpath Backbone Adapter Obs . . . 26
3.5 Observation FollowPath Scheme . . . 27
3.6 MLP Network . . . 31
3.7 Actor Critic Network . . . 32
3.8 VMAS visual inspection of trained policies for both skills and stages. . . . 35
3.9 Skill evaluation in Gazebo. This is the final simulation environment for evaluation. 35
3.10 Training pipeline. Five seeds are trained per backbone stage; the best checkpoint is frozen and used as the fixed foundation for adapter training, also with five seeds. The final model is validated in Gazebo. . . . 35

4.1 Gazebo and RViz, the final stage for test and validation.Gazebo provides a realistic simulation scenario, while RViz allows the visualization of trails, markers and sensor information. . . . 40
4.2 RViz visualizations of both skills. After the selection of the best performing seed, a visual inspection of the inference was performed, alongside quantitative log analysis. 40
4.3 OrbitXY Distance Error . . . 44
4.4 OrbitXY Speed Error . . . 44
4.5 OrbitXY Backbone Validation Success Rates . . . 45
4.6 FollowPath Backbone Metrics . . . 46
4.7 FollowPath Backbone valildation and path progress metrics . . . 47
4.8 OrbitXY Adapter Validation Success Rates . . . 49
4.9 OrbitXY Adapter Speed Error . . . 50
4.10 FollowPath Adapter Path Progress . . . 51
4.11 FollowPath Adapter Validation Success . . . 52
4.12 FollowPath Adapter Lateral Error . . . 53

4.13 OrbitXY Adapter orbit Trajectories 55
4.14 OrbitXY Adapter Distance Errors 55
4.15 OrbitXY Gazebo Transfer Test 56
4.16 Two examples of randomly generated paths for the Gazebo simulation for two different curriculum stages. 57
4.17 FollowPath Adapter Lateral Error 57
4.18 Four additional examples of randomly generated paths for the Gazebo simulation. The red line represents the intended path and the green line represents the actual path driven by the agent 59
4.19 Two examples of randomly generated paths for the Gazebo simulation for the Hector drone. Red line is the intended path , green line is the actual drone trajectory. 61
4.20 FollowPath Adapter Lateral Error 62
4.21 OrbitXY ablation 63
4.22 FollowPathh Ablation Success Rate and Lateral Error 64
4.23 FollowPath Ablation number of steps to complete 64
4.24 FollowPath Ablation Lateral Error by Difficulty Stage 65

List of Tables

3.1 Comparison of Backbone and Adapter policies. 23
3.2 Episode termination conditions ($n_{\text{orbits}} = 1$ for all experiments). Success requires both angular completion and radial tolerance; all other terminations are considered failures. 24
3.3 Grace termination parameters for backbone and adapter policies. 24
3.4 Reward components for the OrbitXY backbone and adapter policies. Dashes indicate the component is not applicable. 25
3.5 Episode termination conditions. Success requires both path completion and lateral tolerance; all other terminations are considered failures. 29
3.6 Reward components for the FollowPath adapter policy. 30
3.7 Shared PPO parameters. These parameters are shared across all 4 settings. 33
3.8 PPO hyperparameters for all four policy variants. Backbones share one regimen; adapters share another. 33
3.9 Training configuration for all backbone and adapter policies. 34
3.10 Main vehicle parameters used in adapter training and deployment. 34
3.11 Three-stage curriculum schedule for the OrbitXY backbone. 36
3.12 Four-stage curriculum schedule for the FollowPath backbone. 36
3.13 Three-stage curriculum schedule for the OrbitXY adapter. 37
3.14 Curriculum schedule for the FollowPath adapter. Stage 0 is skipped; training begins at Stage 1. 37
3.15 FollowPath adapter evaluation streams. 38

4.1 Training and evaluation metrics logged during policy training for OrbitXY. 41
4.2 Scenario parameters and evaluation limits for the OrbitXY skill. 42
4.3 Training and evaluation metrics logged during FollowPath policy training. 42
4.4 Scenario parameters and evaluation limits for the FollowPath skill. 43
4.5 Summary of OrbitXY backbone experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better). Best value per column in **bold**. 46
4.6 Summary of FollowPath backbone experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better, ≈ 0 closer to zero is better). Best value per column in **bold**. 48
4.7 Summary of OrbitXY adapter experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better). Best value per column in **bold**. 51
4.8 Summary of FollowPath adapter experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better, ≈ 0 closer to zero is better). Best value per column in **bold**. 52

4.9 OrbitXY adapter Gazebo evaluation across radius and speed conditions. Success rate averaged over CW and CCW directions. Metrics reported as mean ± std across episodes. Larger radius combined with the lowest speed produce 0% success in two cells on the grid, but this is due to timeout. 54
4.10 FollowPath adapter Gazebo evaluation across curriculum stages, speeds and path lengths. Metrics reported as mean ± std across 3 episodes. Partial success conditions shown with exact rate. 58
4.11 FollowPath backbone zero-shot transfer to Hector drone, aggregated by curriculum stage. Metrics reported as mean ± std across all speed and path length conditions within each stage. 62

List of Acronyms

COLREGs	International Regulations for Preventing Collisions at Sea
CW	Clockwise
CCW	Counterclockwise
EMA	Exponential Moving Average
HRL	Hierarchical Reinforcement Learning
ICM	Intrinsic Curiosity Module
IMU	Inertial Measurement Unit
MLP	Multi-layer Perceptron
MDP	Markov Decision Process
MPC	Model Predictive Control
PPO	Proximal Policy Optimization
PID	Proportional–Integral–Derivative
POMDP	Partially Observable Markov Decision Process
Q-Learning	Action-Value Based Reinforcement Learning Algorithm
RHRL	Receding Horizon Reinforcement Learning
RL	Reinforcement Learning
ROS	Robot Operating System
SAC	Soft Actor-Critic
SAR	Search and Rescue
UAV	Unmanned Aerial Vehicles
USV	Uncrewed Surface Vehicle
UMV	Unmanned Marine Vehicle
AGV	Autonomous Guided Vehicle
SMDP	Semi-Markov Decision Process
MASS	Maritime Autonomous Surface Ship
VMAS	Vectorized Multi Agent Simulator

Chapter 1

Introduction

1.1 Context

Precise and reliable navigation is fundamental to safe seafaring. From early advances in cartography and celestial navigation to modern sensing, control, and autonomy, maritime systems have continually evolved toward greater operational reach and reduced human intervention. In this context, Unmanned Surface Vehicles (USVs) have emerged as an important class of autonomous platforms capable of performing persistent tasks in coastal and offshore environments [1, 2].

USVs are now used in a broad range of civilian, scientific, and industrial applications, including transport, environmental monitoring, hydrographic surveying, bathymetric mapping, security-related missions [3], and infrastructure inspection [4, 5]. In fact, infrastructure inspection, together with maintenance, have been an attractive role for autonomous vehicles, such as USVs and Unmanned Aerial Vehicles (UAV) [6]. To operate effectively in these settings, USVs must integrate heterogeneous sensors such as GPS, IMUs, cameras, and sonar into a guidance [7], navigation, and control stack capable of reasoning under uncertainty and interacting safely with the surrounding environment [8]. However,maritime autonomy is complicated by vehicle-specific dynamics, actuator constraints, environmental disturbances, obstacle avoidance [9], and the need to comply with safety rules such as the International Regulations for Preventing Collisions at Sea (COLREGs). Safe navigation must therefore account not only for reaching a target path or goal, but also for collision avoidance, rule-compliant maneuvering, and robustness under uncertain and time-varying conditions. These challenges make control design and learning-based navigation particularly demanding in realistic maritime settings.

Classical navigation methods such as PID control and graph-based planning remain widely used, but they can struggle when vehicle dynamics, actuator limits, or mission conditions change. In particular, these methods often depend on careful tuning or explicit models, which can reduce robustness in uncertain maritime environments. Reinforcement Learning offers an attractive alternative because it can learn control behaviors directly from interaction [10]. However, many existing RL approaches for USVs learn monolithic policies that entangle task geometry with vehicle-specific dynamics, which motivates a factorized learning strategy that separates reusable naviga-

tion structure from platform-dependent actuation. This limitation motivates a factorized learning strategy in which transferable geometric structure is learned separately from vehicle-dependent actuation. This thesis addresses that problem by learning transferable navigation skills through dynamics-agnostic behavioral priors, using a geometry backbone and a vehicle-specific adapter to map that structure to control commands.

1.2 Objectives

The main objective of this thesis is to develop a transferable reinforcement learning framework for USV navigation based on dynamics-agnostic behavioral priors. The focus is separating reusable navigation structure from vehicle-dependent actuation to improve interpretability, reduce retraining effort, and support skill reuse across platforms. To this end, the thesis pursues the following specific objectives:

- Design and validate a two-stage reinforcement learning architecture for USV navigation, with a frozen geometry backbone and a vehicle-specific adapter, and develop a training pipeline for adapting the backbone from simplified kinematic simulation to a higher-fidelity vehicle model.
- Formulate and implement two transferable navigation skills, OrbitXY and FollowPath, which represent two relevant classes of maritime motion: orbit-keeping and path-following.
- Evaluate the proposed framework through simulation metrics, ablation studies, zero-shot backbone analysis, and transfer experiments in Gazebo, assessing both performance and generalization within the studied settings.
- Demonstrate that dynamics-agnostic behavioral priors provide a practical basis for reusable navigation skill learning across different vehicle embodiments.

The main contributions of this thesis are:

- the design of a two-stage reinforcement learning architecture for USV navigation, composed of a geometry backbone and a vehicle-specific adapter, enabling a separation between navigation geometry and platform-dependent actuation;
- the formulation and implementation of two transferable navigation skills, OrbitXY and FollowPath, representing two relevant classes of maritime motion: orbit-keeping and path-following;
- the development of a training pipeline in which the backbone is first learned in simplified kinematic simulation and is then adapted to a higher-fidelity vehicle model with thrust saturation, inertia, and hydrodynamic effects;

- the empirical evaluation of the proposed framework through simulation metrics, ablation studies, zero-shot backbone analysis, and transfer experiments in Gazebo;
- the demonstration that dynamics-agnostic behavioral priors provide a practical basis for reusable navigation skill learning across different vehicle embodiments within the studied settings.

1.3 Document Structure

The remainder of this thesis is organized as follows. Chapter 2 reviews the state of the art in reinforcement learning for autonomous navigation, with emphasis on RL method taxonomies, maritime USV applications, and modular policy-learning approaches relevant to transferable control. Chapter 3 presents the proposed methodology, including the transferable navigation problem formulation, the two navigation skills studied in this work—OrbitXY and FollowPath—the backbone–adapter policy architecture, and the corresponding training procedures. Chapter 4 reports the experimental results, covering backbone and adapter performance in simulation, ablation studies assessing the results of the proposed architecture. Finally, Chapter 5 summarizes the main conclusions, discusses the principal limitations of the approach, and outlines directions for future work.

Chapter 2

State of the art

This chapter surveys the state of the art in reinforcement learning (RL), emphasizing its application to navigation and task execution for uncrewed surface vehicles (USVs). It progresses from core RL concepts to a taxonomy of modern algorithms and domain-specific implementations, identifying key limitations in maritime environments such as stochastic dynamics and sparse rewards. Section 2.1 introduces RL fundamentals, including Markov Decision Processes (MDPs), agent-environment interactions, policies, and value functions. Section 2.2 categorizes modern RL methods (adapted from OpenAI Spinning Up [11]), covering model-free approaches – policy optimization (PPO, A3C), Q-Learning (DQN, QR-DQN), and hybrid actor-critic methods (DDPG, TD3, SAC) – while excluding model-based RL for reasons discussed in Section 2.3. Section 2.4 reviews RL applied to USV navigation, organized by this same taxonomy, noting recurring trends such as COLREG compliance and disturbance robustness, with particular emphasis on Proximal Policy Optimization (PPO) given its role as the training algorithm adopted in this thesis. Section 2.5 reviews modular policy learning in the literature and motivates the proposed framework adopted in this thesis. The chapter closes with a critical review that motivates the gap addressed by this work.

2.1 Reinforcement Learning: Introduction

In reinforcement learning (RL), the Markov Decision Process (MDP) models a basic reinforcement learning problem. An agent interacts with an environment over discrete time steps, as illustrated in Figure 2.1. At time t, the agent observes the current state $s_t \in S$, selects an action $a_t \in A$, and then receives a reward r_{t+1} and the next state s_{t+1} from the environment.

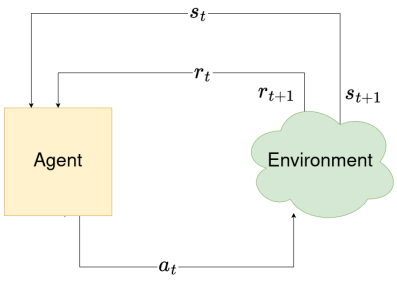


Figure 2.1: Illustration of agent-environment interaction in an MDP.

By interacting repeatedly with the environment, the agent learns a policy $\pi(a \mid s)$, mapping states in the state space S to a distribution over actions in the action space A, with the objective of maximizing the expected long-term cumulative reward. An MDP is called episodic if states are reset after a finite horizon T, and an episode is then defined by a sequence of states, actions, and rewards. Along an episode, the (discounted) return is defined as $R = \sum_{t=0}^{T-1} \gamma r_{t+1}$, where $\gamma \in [0,1]$ is the discount factor, which also assures convergence of the sum when the MDP is nonepisodic. The goal of RL is to find the optimal policy π^* that returns the maximum expected return over time:

$$\pi^* = argmax_\pi \mathbb{E}[R|\pi] \tag{2.1}$$

To characterize the performance of a policy it is important to introduce the *State-Value Function*, $V^\pi(s) = \mathbb{E}[R|s,\pi]$, which represents the expected return when the agent starts on a state s and follows a policy π, and the *Action-Value Function* $Q^\pi(s,a) = \mathbb{E}[R|s,a,\pi]$, which represents the expected return when starting from a state s, taking an action a, and then following a policy π. An optimal policy maximizes both values under $\pi = \pi^*$:

$$v^*(s) = max_\pi v_\pi(s) \tag{2.2}$$

$$q^*(s,a) = max_\pi q_\pi(s,a) \tag{2.3}$$

When state and action spaces are small and discrete, tabular methods can compute these functions as literal lookup tables. These methods are foundational, introducing core concepts such as Bellman Equations [12], Dynamic Programming [13], Monte Carlo methods, Temporal Difference (TD) learning [14], and Q-Learning [15] in their purest forms [16].

This basis is particularly relevant for USV applications, where complex maritime environmental conditions add constraints that exceed the capacity of traditional Q-tables, motivating a broader classification of modern RL methods better suited for high-dimensional tasks.

2.2 Taxonomy of RL Methods

Figure 2.2 illustrates a taxonomy of modern RL algorithms, with the major methods organized from left to right along each branch. Each branch is discussed individually below, with its principal methods explained in more detail. These categories are conceptual, and many modern reinforcement learning algorithms combine ideas from multiple branches.

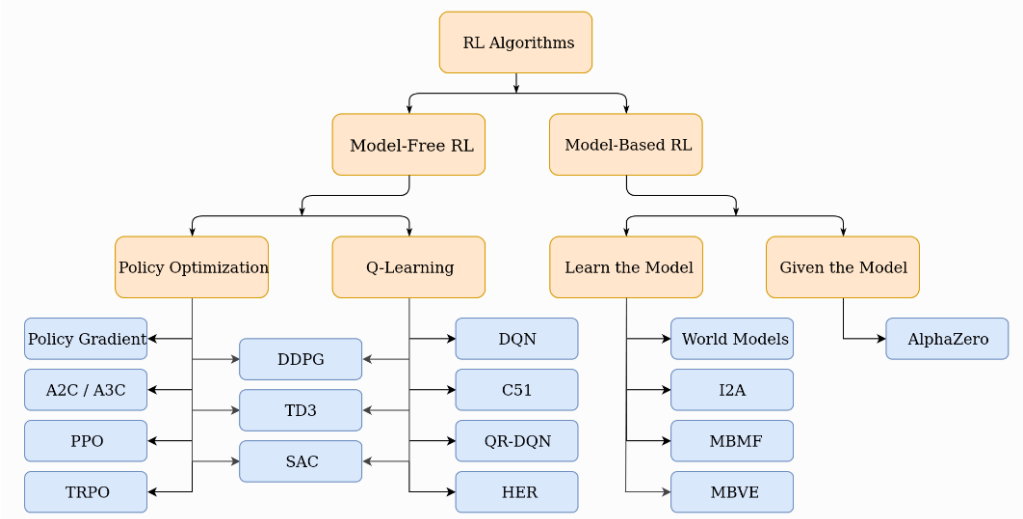


Figure 2.2: Taxonomy of modern RL algorithms (adapted from OpenAI Spinning Up [11]).

In model-free methods, the agent learns how to act directly from experience, without learning an explicit model of the environment's dynamics such as transition probabilities or the reward function. Instead of constructing or using $p(s'|s,a)$ for planning, the agent improves a value function and/or policy solely through interaction with the environment.

In contrast, model-based algorithms learn or are provided with a given model of the environment's dynamics and use this model for planning and policy improvement, attempting to approximate the distribution $p(s',r|s,a)$, which allows the agent to evaluate possible future trajectories and select actions based on predicted outcomes rather than relying only on direct trial-and-error experience [17].

2.2.1 Policy Optimization

Within model-free reinforcement learning, policy optimization methods directly optimize a parameterized policy, often using gradient-based techniques. Many modern approaches combine this idea with value-function learning, leading to hybrid actor–critic algorithms. The policy is typically represented by a differentiable function, such as a neural network, mapping states to actions or action probabilities, with the objective of maximizing expected cumulative reward via policy gradient estimators. Unlike value-based methods (Section 2.2.2), policy optimization enables direct learning of stochastic policies and is particularly well suited for continuous action spaces.

Starting from the policy gradient methods such as REINFORCE, subsequent approaches introduced variance reduction and constrained updates, leading to algorithms such as A3C, TRPO, and PPO.

REINFORCE, introduced by Williams [18] in 1992, established the policy gradient framework by demonstrating how a policy can be directly optimized using gradient ascent on the expected return. Its high-variance Monte Carlo returns and sequential on-policy sampling inefficiency motivated further developments.

Mnih et al. [19] proposed the Asynchronous Advantage Actor-Critic (A3C) method, an on-policy actor–critic approach that replaces experience replay with parallel environment interactions. Multiple worker agents run in parallel, each collecting n-step returns from independent environment instances and computing advantage estimates $A_t = R_t - V(s_t)$ using a shared critic network, with gradients asynchronously updating a central parameter server via RMSProp. This enables lock-free scaling across CPU cores while the stochastic policy naturally embeds exploration. A3C succeeded across a variety of continuous motor control problems on the MuJoCo Physics Simulator [20], including locomotion, balancing, and manipulation tasks [21].

While A3C accelerates training through parallelism, its unconstrained policy gradients can still produce destructive large updates. Trust Region Policy Optimization (TRPO), introduced by Schulman et al. [22], addresses this by solving a constrained optimization problem: maximize a surrogate advantage $L(\theta)$ subject to $KL(\pi_{OLD}||\pi_{NEW}) \leq \sigma$, using conjugate gradient for the trust region. This guarantees monotonic improvement but requires expensive Hessian-vector products.

Proximal Policy Optimization (PPO) [23] simplifies TRPO by alternating between collecting on-policy rollouts and optimizing a surrogate objective with clipped probability ratios, bounding the change in policy likelihood $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta OLD}(a_t|s_t)}$ and thus preventing destructive updates. This simple clipping scheme approximates the trust-region constraint of TRPO while using only first-order Stochastic Gradient Descent (SGD), allowing PPO to perform several epochs of minibatch updates on the same rollout data. This gives PPO a strong balance of stability, sample efficiency, and ease of implementation across continuous-control tasks – properties that motivate its adoption in this thesis, as detailed in Chapter 3. A second variant present in the original paper uses a KL-penalized objective with an adaptive coefficient L^{KLPPEN}, but since it performed worse in the authors' test environments, the clipped surrogate objective became the PPO algorithm of choice and is the variant considered throughout this thesis.

2.2.2 Q-Learning

An alternative model-free approach operates by learning the *value* of taking actions in given states. Q-Learning methods estimate an action-value function $Q(s,a)$ – the expected cumulative reward of taking action a in state s and following the optimal policy thereafter – and derive a policy implicitly by acting greedily with respect to Q. Starting from the tabular formulation of Watkins and Dayan [15], deep neural network function approximators gave rise to the Deep Q-Network (DQN), with subsequent refinements addressing its instabilities and leading to distributional variants such as C51 and QR-DQN, and extensions such as HER.

Q-Learning is a model-free, off-policy algorithm that updates its Q-values iteratively using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)] \tag{2.4}$$

where α is the learning rate and γ the discount factor. Being off-policy, Q-Learning can learn from transitions generated by any behavior policy, typically an ε-greedy policy. In tabular form, however, it does not scale to high-dimensional or continuous state spaces, as maintaining a separate Q-value estimate per state-action pair becomes intractable.

This limitation motivated Mnih et al. [24] to propose the Deep Q-Network (DQN), which replaces the tabular Q-function with a deep convolutional neural network approximating $Q(s,a:\theta)$ directly from raw pixel observations. DQN introduced two key stabilising mechanisms: an experience replay buffer, which breaks temporal correlations by sampling random minibatches of past transitions, and a target network – a periodically frozen copy of the Q-network used to compute Bellman targets, preventing oscillations from simultaneous shifts in predicted and target values. Together these enabled DQN to achieve human-level performance across a suite of Atari 2600 games from raw pixel input alone.

Building on DQN, Bellemare, Dabney, and Munos [25] proposed the Distributional DQN (C51), which learns the full *distribution* of returns rather than their scalar expectation, modeling $Z(s,a)$ as a discrete distribution over 51 fixed atoms and training via KL divergence against a distributional Bellman target. Dabney et al. [26] then proposed Quantile Regression DQN (QR-DQN), which instead represents the return distribution through a fixed set of quantiles with *learned locations*, removing the need to pre-specify a value range and offering a cleaner theoretical grounding through quantile regression.

A distinct challenge arises when reward signals are sparse. Andrychowicz et al. [27] addressed this with Hindsight Experience Replay (HER): after a failed episode, HER relabels the experienced transitions with a *hindsight goal* – typically the state actually reached – and replays them as if that outcome had been the original objective, turning failures into informative training data. HER is goal-conditioned and algorithm-agnostic, and was originally demonstrated with DDPG on sparse-reward robotic manipulation tasks.

2.3 Hybrid Methods

A class of algorithms draws from both paradigms above, combining the stability and continuous-action compatibility of policy gradient methods with the sample efficiency of off-policy Q-Learning. These hybrid actor-critic algorithms maintain an *actor* representing the policy and a *critic* estimating value, leveraging off-policy experience replay while policy gradient updates ensure stable learning in continuous action spaces. This subsection covers the three principal algorithms of this family: DDPG, TD3, and SAC.

Deep Deterministic Policy Gradient (DDPG), proposed by Lillicrap et al. [28], is based on the Deterministic Policy Gradient framework [29] and enables learning in continuous action spaces. The actor outputs continuous control actions and the critic approximates the action-value function; both are trained with minibatches from an experience replay buffer, with slowly-updated target networks improving stability and Ornstein-Uhlenbeck noise added externally for exploration, since the policy itself is deterministic.

Fujimoto, Hoof, and Meger [30] identified failure modes in DDPG stemming from value overestimation and premature policy updates, motivating the Twin Delayed DDPG (TD3), which introduces clipped double Q-Learning across two critic networks, delayed policy updates, and target policy smoothing, together reducing function approximation errors and improving stability.

Haarnoja et al. [31] proposed the Soft Actor-Critic (SAC), which instead learns a *stochastic* policy under a maximum-entropy objective: the training objective is augmented with an entropy term $H(\pi(\cdot|s))$ that encourages the policy to remain as random as possible while still maximising returns. SAC inherits TD3's clipped double Q-Learning, remains off-policy with experience replay, and automatically tunes its temperature parameter α through dual optimisation, demonstrating competitive or superior performance to other model-free methods on the MuJoCo benchmark suite.

Beyond model-free methods, Model-Based RL learns or exploits an explicit model of the environment's transition dynamics to enable sample-efficient planning through simulated rollouts (e.g. World Models, I2A, AlphaZero). However, the sea surface is a highly stochastic, non-stationary environment governed by the complex interplay of waves, wind, and currents, making the construction of a sufficiently accurate dynamics model an open research problem largely orthogonal to the navigation objectives of this work. Introducing model learning would risk compounding approximation errors and undermining policy reliability under real deployment conditions. For these reasons, Model-Based approaches are considered out of scope, and all subsequent developments in this thesis are grounded exclusively in Model-Free reinforcement learning.

2.4 Reinforcement Learning for USV Navigation

This section surveys RL applied to USV navigation, organized according to the taxonomy introduced above. The density of USV-specific literature varies considerably across families – value-based methods, and DQN in particular, have accumulated a substantial body of marine applications, while actor-critic and hybrid methods remain comparatively underrepresented in the USV domain specifically. The survey begins with value-based methods before progressing through policy optimization, with particular attention to PPO, and hybrid actor-critic methods.

USV navigation presents a challenging control problem, characterized by nonlinear dynamics, environmental disturbances such as currents and waves, partial observability [32], sensor noise, dynamic obstacles, and COLREG compliance requirements [33], alongside sim-to-real transfer difficulties [34]. Traditional approaches such as dynamic window [35] and artificial potential field methods [36] have been shown to underperform compared to RL-based methods in these

conditions ([37], [38]), motivating the adoption of data-driven policies that adapt to environmental complexity without explicit dynamic modelling.

Among the early deep RL approaches applied to USV navigation, Li et al. [39] proposed a COLREG-compliant collision avoidance framework combining DQN with an Artificial Potential Field (APF) formulation, which introduces virtual attraction and repulsion fields toward waypoints and away from obstacles, structuring the action space into a safe zone, a collision avoidance zone, and a mandatory collision avoidance zone. This zoning directly shapes the reward function, enabling the agent to learn multi-ship collision avoidance strategies consistent with COLREG conventions. The resulting APF-DQN achieved a 100% success rate and 20–30% shorter paths than DQN, DDPG, RRT, A*, and APF in obstacle-rich scenarios (Figure 2.3), while maintaining COLREGS compliance across multi-ship encounters. Validation, however, remained simulation-only, without real-world testing or sim-to-real analysis.

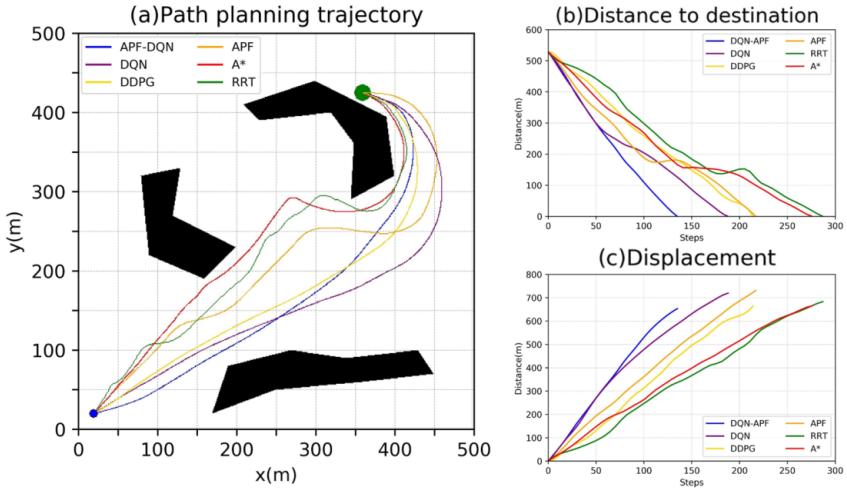


Figure 2.3: APF-DQN path planning trajectories, distances to destination, and displacement comparison with other methods, extracted from Li et al. [39].

This approach was extended by Mou et al. [40] into APF-RADQN, which introduces reward averaging – reducing reward-signal variance via a running-average baseline, conceptually related to the baseline subtraction of REINFORCE [16] but applied directly to the reward – and a Bézier-curve path-smoothing strategy offering lower computational overhead than spline-based alternatives [41].

This work is representative of a broader trend in the USV literature, where DQN and its variants appear frequently ([42], [43], [44]), sharing a common architectural foundation of online/target networks, experience replay, and ε-greedy exploration, with contributions primarily in problem formulation, reward design, and exploration strategy [45]. Despite their prevalence, however, DQN-based approaches present inherent limitations for continuous control systems such as USVs: acting on discretized action spaces restricts control resolution and can produce oscillatory trajectories, while ε-greedy and value-based learning can be inefficient and unstable in high-dimensional continuous domains, often requiring extensive reward shaping.

Building on DQN's value-based foundation, distributional RL models the full return distribution rather than its expectation, providing richer uncertainty estimates under environmental variability. Following QR-DQN [26], Implicit Quantile Networks (IQN – Dabney et al. [46]) generalize this to continuous quantile functions via neural embeddings. Lin, McConnell, and Englot [47] applied IQN-based distributional RL to robust USV navigation amid currents and moving obstacles, with a LiDAR-based planner in simulated Rankine vortex flows surpassing DQN and classical methods (APF, Bug) with 81–95% success rates, lower energy consumption, and smoother risk-aware trajectories.

While distributional value-based methods represent the frontier of Q-Learning refinements, a parallel line of development pursued improved training stability through actor-critic architectures, with A3C as one of the earliest deep RL methods of this family applied to USV navigation. Direct USV applications remain relatively limited: Zhou et al. [48] proposed a GPU-accelerated A3C variant for real-time navigation validated in Google's DeepMind Lab environment [49], using a five-action discrete space and CNN-based depth estimation from RGB input, reducing training time by over 59% compared to CPU-only A3C. Zeng et al. [50] proposed MK-A3C, augmenting A3C with a GRU belief state [51] to address partial observability and a curriculum-learning schedule for non-holonomic robot navigation, formulated as a POMDP [16] and evaluated across 1000 navigation tasks (Figure 2.4).

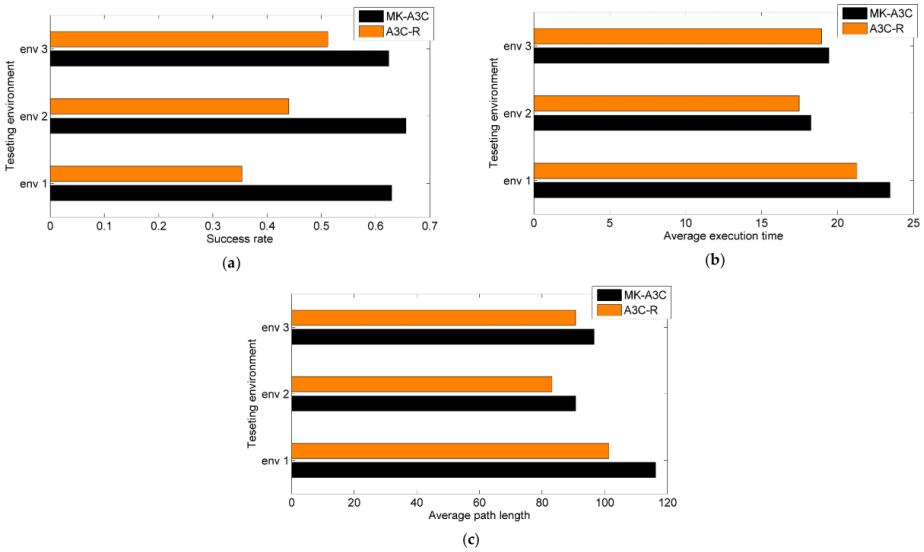


Figure 2.4: Performance of the MK-A3C and A3C-R methods in 1000 navigation tasks depicting success rate, path length, and execution time, extracted from Zeng et al. [50].

A3C established the viability of asynchronous actor-critic training from high-dimensional observations, and its on-policy nature places it at a natural transition point toward the off-policy actor-critic methods that followed. The on-policy framework it established was further developed through PPO, whose clipped surrogate objective provides a more stable and accessible optimization scheme – and, being the algorithm adopted in this thesis, PPO's USV applications are treated here in greater depth than the other families surveyed.

In the USV navigation literature, PPO has been applied to trajectory tracking control by Wu et al. [52], who propose an Intrinsic Curiosity Module augmented PPO scheme (ICM-PPO) for a single-propeller, single-rudder vessel under wind and wave disturbances, with propeller RPM and rudder angle as control inputs (Figure 2.5).

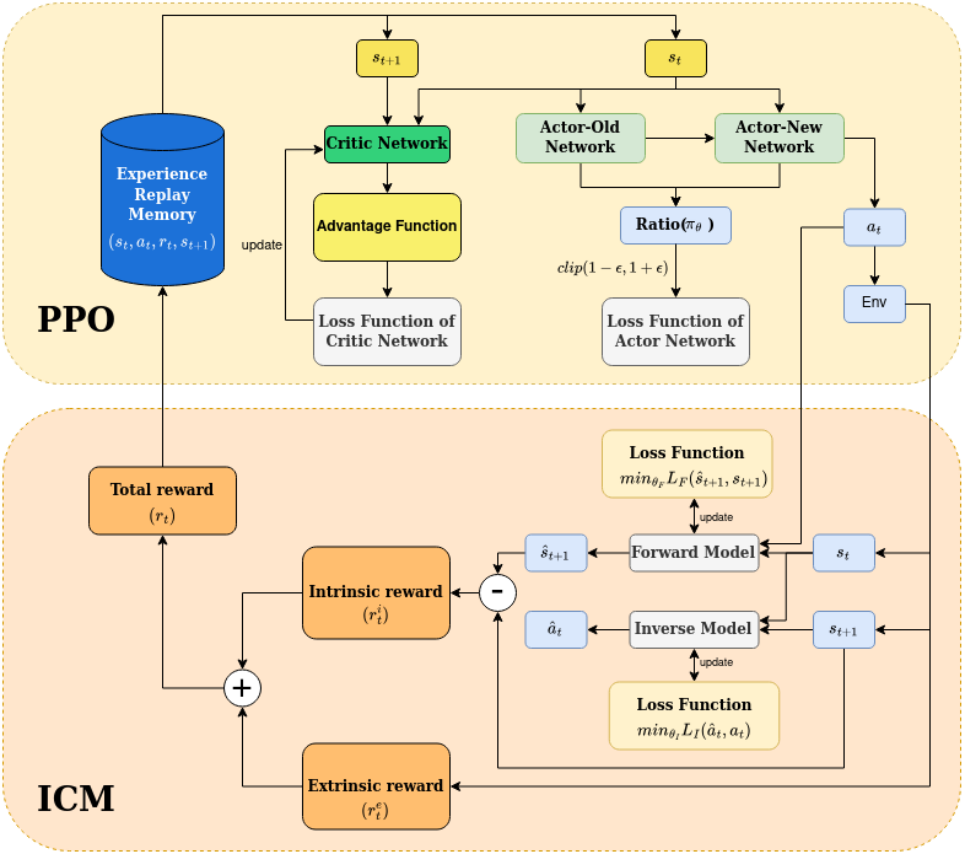


Figure 2.5: PPO-ICM algorithm illustrated, adapted from Wu et al. [52].

The base PPO follows a standard actor-critic architecture, where the policy network $\pi_\theta(a_t, s, t)$ is updated via the clipped surrogate objective, constraining the log-probability ratio $r_t(\theta)$ to $[1-\varepsilon, 1+\varepsilon]$, and the value network $V_\theta(s_t)$ is trained with squared-error loss to support Generalised Advantage Estimation (GAE). Two contributions extend this foundation. First, an improved guidance law limits the commanded surge speed when tracking errors are large, preventing infeasible control inputs and divergence during early exploration. Second, an Intrinsic Curiosity Module (ICM) supplements the extrinsic task reward with an intrinsic signal derived from the prediction error of a learned dynamics model, encouraging the agent to seek state transitions that are not yet well understood.

The ICM operates through two sub-models: an inverse model $\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$ predicting the action responsible for an observed transition, trained with a classification loss $L_I(\hat{a}_t, a_t)$, and a forward model $\hat{s}_t = f(s_t, a_t; \theta_F)$ predicting the next state, trained with squared error $L_F = \frac{1}{2} \| \hat{s}_{t+1} - s_{t+1} \|^2$. The intrinsic reward $r_t^i = \frac{\eta}{2} \| \hat{s}_{t+1} - s_{t+1} \|^2$ thus rewards poorly-understood transitions, encouraging structured exploration and reducing premature convergence. The total reward

combines this intrinsic component with the extrinsic task reward.

The extrinsic reward is a weighted combination of five exponential terms penalising heading error, speed error, along-track error, cross-track error, and control oscillation (measured as the standard deviation of the last 20 control inputs). ICM-PPO was evaluated against PID control and standard PPO across straight, circular, and complex trajectories under varying wind and current conditions, achieving substantially lower tracking error than both baselines (Figure 2.6).

Error analysis of complex trajectory tracking experiments.

Method	Mean value of e_x after 50 s	Mean value of e_y after 50 s	Mean value of e after 50 s
PID	1.945 m	5.989 m	6.734 m
PPO	1.012 m	1.983 m	2.432 m
ICM-PPO	0.277 m	0.566 m	0.677 m

Figure 2.6: PPO-ICM error analysis for a complex trajectory, extracted from Wu et al. [52].

A complementary application to map-based navigation is presented by Sun et al. [53], who replace the fully connected layers of a standard PPO network with convolutional layers operating directly on a USV-centred local radar map, demonstrating strong generalization across varied environment configurations in an end-to-end path planning and dynamic obstacle avoidance framework. Xia et al. [54] apply a standard PPO agent to COLREG-aware collision avoidance along a prescribed path in a dynamic obstacle field, with a reward function penalizing close-quarters situations and COLREG rule violations. Xue et al. [55] extend PPO to multi-USV scenarios, combining it with a Reciprocal Velocity Obstacles model [56] and a GRU network [51] encoding variable-length neighbour state information for distributed multi-vessel collision avoidance.

Across these works, PPO emerges as a frequently adopted algorithm in the USV navigation literature, appearing in applications from low-level trajectory tracking to map-based path planning and multi-agent collision avoidance. Its on-policy actor-critic formulation, clipped update objective, and stochastic policy make it well suited to maritime environments characterized by disturbances and modeling uncertainty, while its relative simplicity and training stability have contributed to its broad adoption – properties that directly motivate its selection as the training algorithm for this thesis.

Turning to the off-policy hybrid actor-critic family, the deterministic policy gradient framework of DDPG represents the earliest and most extensively documented method in the USV navigation literature. Woo, Yu, and Kim [57] propose a DDPG-based steering controller for path following, demonstrating sim-to-real transfer to a full-scale WAM-V platform. A 3-DOF dynamic model with a vector-field guidance law [58] decouples the steering objective from guidance and speed regulation, with a continuous steering command allowing DDPG to learn directly in actuator space. In simulation, the controller matched a tuned PID in tracking accuracy while producing smoother continuous outputs than discrete-action DQN controllers; policies trained purely in simulation were subsequently deployed to the physical WAM-V and evaluated across several path geometries in open water (Figure 2.7), demonstrating consistent tracking and robustness to unmodelled dynamics.

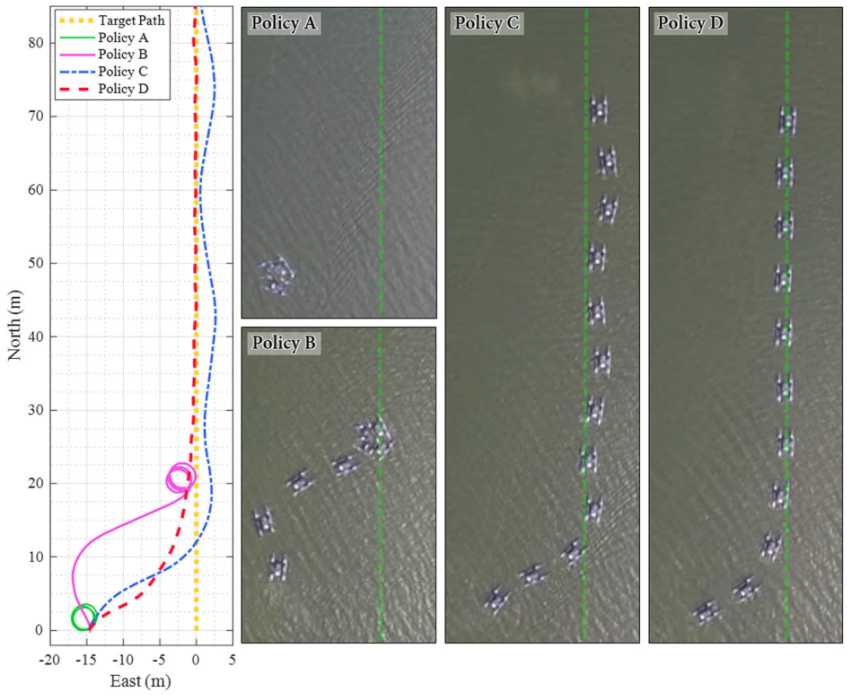


Figure 2.7: Trajectories from linear path following for successive policies and a video snapshot of the WAM-V during the experiments, extracted from Woo, Yu, and Kim [57].

DDPG has also been applied to joint path planning and COLREGs-compliant collision avoidance by Xu et al. [59], and to path planning by Zhao et al. [60], demonstrating improved performance over classical A* methods.

Building on DDPG's deterministic actor-critic foundation, TD3 variants have also been applied to USV navigation. A notable contribution is the Autonomous Collision Avoidance Decision-making (ACAD) model proposed by Cui et al. [61], which augments TD3 with LSTM and multi-head self-attention to exploit historical encounter states, outputting continuous rudder and thrust commands guided by a structured reward incorporating ship domain, arena boundary, yaw regulation, and immediate danger terms. The resulting ATL-TD3 architecture was evaluated on classical Imazu encounter scenarios and Suez Canal traffic data, outperforming TD3, DDPG, and AC baselines (Figure 2.8).

TD3 has also been applied to trajectory tracking under realistic ocean disturbances by Zhou, Gong, and Chen [62], and to static multi-obstacle avoidance by Jiang et al. [63].

While TD3 refinements demonstrated the potential of deterministic off-policy control, the entropy-regularized stochastic policy of SAC represents a further development within the hybrid actor-critic family. Zheng et al. [64] propose a hybrid control architecture combining a Linear Active Disturbance Rejection Controller (LADRC) with a SAC agent for adaptive gain scheduling under wind and wave disturbances, using SAC to adapt observer and control bandwidths online where fixed-gain LADRC only provides reliable tracking within a narrow regime. Across straight, circular, and mixed path geometries with injected disturbances, the adaptive scheme demonstrated

reduced overshoot, faster convergence at turning points, and lower rudder oscillation than fixed-gain LADRC (Figure 2.9).

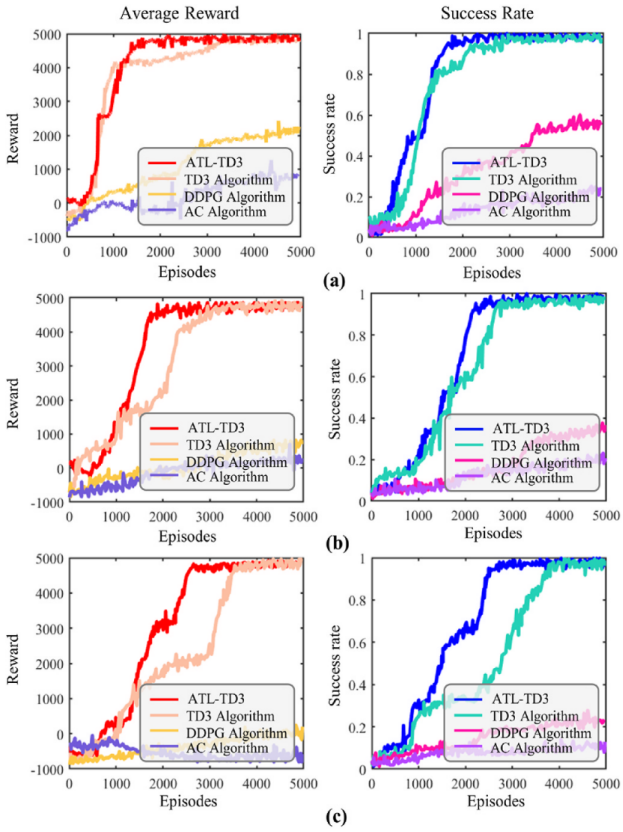


Figure 2.8: Comparison of training results based on ATL-TD3, TD3, DDPG, and AC algorithms for the ACAD model with 2, 3, and 4 USVs, depicted in (a), (b), (c). Extracted from Cui et al. [61].

Beyond low-level control, SAC has also been applied to higher-level USV decision-making: Zhang et al. [65] propose a modified SAC for evasive trajectory planning against a dynamic defensive perimeter, and Cui et al. [66] propose an improved SAC for COLREG-compliant navigation acting directly on course and speed commands.

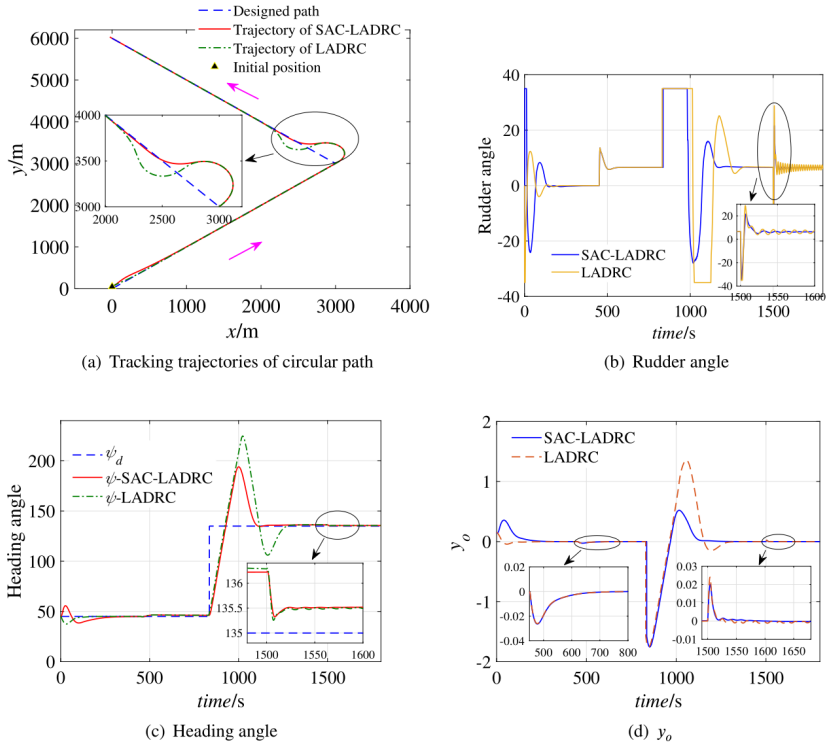
2.5 Modular Policy Learning

Recent work in robotic reinforcement learning suggests that policies can be made more transferable by separating task structure from robot-specific control. A key example is modular policy learning, as shown by Devin et al. [67], where the policy is decomposed into task-specific and robot-specific components so that knowledge can be reused across different robot-task combinations. This idea is very close to a two-stage backbone-adapter architecture, since the first stage can learn reusable task geometry while the second stage adapts that representation to the dynamics and actuation limits of a particular vehicle.

A related line of research uses hardware-conditioned policies, as shown by Chen, Murali, and Gupta [68], where a universal policy is conditioned on a representation of the robot hardware.

These methods show that conditioning on embodiment can improve zero-shot transfer and reduce the retraining needed when moving to new robots or dynamics – supporting the idea of freezing a geometry backbone and training only a small adapter to map abstract navigation commands to vehicle-specific thrust and steering commands.

Symmetry-aware reinforcement learning is also relevant, since the proposed navigation skills exploit geometric invariances such as clockwise/counterclockwise symmetry. Nguyen et al. [69] show that incorporating symmetry into learning can improve sample efficiency and generalization by reducing redundant parts of the state-action space, motivating invariant observations and symmetry losses in the backbone, while leaving the adapter to handle the remaining embodiment-specific asymmetries.



(a) Tracking trajectories of circular path (b) Rudder angle

(c) Heading angle (d) y_o

Figure 2.9: Tracking results of a straight path by SAC-LADRC with wave and wind disturbances, extracted from Zheng et al. [64].

A closely related paradigm has emerged in legged robotics, where a policy trained under idealized simulated conditions is later specialized to real embodiment constraints. Lee et al. [70] introduced this pattern as *privileged learning*: a teacher policy is trained in simulation with full access to ground-truth terrain and contact information, and its behavior is then distilled into a student policy that must operate from realistic onboard sensing alone, enabling robust quadrupedal locomotion over previously unseen rough terrain. Kumar et al. [71] formalized a similar two-stage decomposition in Rapid Motor Adaptation (RMA), pairing a base policy trained under simulated dynamics with a lightweight adaptation module that estimates an online latent representation of the current embodiment, allowing a single base policy to generalize zero-shot to unseen terrains and payloads on a physical quadruped without fine-tuning. Both works target legged locomotion

rather than maritime navigation, but their staged procedure – learn an idealized representation first, then adapt it to real embodiment constraints while keeping the first stage fixed – closely mirrors the training pipeline adopted in this thesis (Section 3.5.2). There, the geometry backbone is trained under simplified, frictionless dynamics and frozen before the vehicle-specific adapter is trained separately under realistic thrust, inertia, and hydrodynamic constraints. The backbone-adapter factorization explored here can thus be read as an instance of this broader pattern, adapted to a symmetry-aware, geometry-centred formulation not previously applied to USV navigation.

2.6 Critical Review

Although prior reinforcement learning work for USV navigation has demonstrated strong results in collision avoidance, trajectory tracking, and path following, most approaches remain tightly coupled to a specific vehicle model, action interface, or reward design. As a result, the learned policy captures both navigation geometry and vehicle dynamics in a single function, which limits transfer to new platforms. Modular and hardware-conditioned methods indicate that separating task structure from embodiment can improve reusability, and the privileged-learning and Rapid Motor Adaptation paradigms further show, in legged robotics, that a policy trained under idealized simulated conditions can be specialized to real embodiment constraints through a dedicated adaptation stage. However, neither direction has been developed for maritime navigation with an explicit, symmetry-aware geometry backbone and a vehicle-specific adapter. The main gap addressed by this thesis, therefore, is not the absence of RL methods for USVs. It is the absence of a transferable, symmetry-aware, dynamics-separated architecture – of the kind established in legged locomotion – that preserves navigation geometry while adapting efficiently to different maritime vehicles.

Chapter 3

Two-Stage Transferable Navigation Framework

This chapter presents the methodology for learning transferable navigation skills for USVs. The proposed approach decomposes the control problem into two stages: a geometry backbone that learns vehicle-agnostic navigation structure in simulation, and a vehicle-specific adapter that maps the resulting geometric representations to the control actions required by a given vessel. This factorization offers several advantages: it improves the interpretability of the task by separating geometric navigation from vehicle dynamics, reduces the amount of retraining required when transferring to a new platform, and cleanly decouples navigation reasoning from execution. To demonstrate this framework, two navigation skills are developed: orbit following - OrbitXY, for persistent area coverage around a fixed point, and curvilinear path tracking - FollowPath, for waypoint-to-waypoint transit along open trajectories.

The remainder of this chapter is organized as follows: Section 3.1 formalizes the transferable navigation problem, Sections 3.2 and 3.3 introduce the two navigation skills, Section 3.4 describes the PPO-based policy architecture, and Section 3.5 details the training procedure and evaluation protocol.

3.1 Problem Formulation

Transferring skills across vehicles can be difficult because vehicles differ substantially in their dynamics and control constraints. For instance, a holonomically actuated aerial vehicle and a non-holonomic unmanned surface vehicle have substantial differences in their motion dynamics and control limitations. The resulting policy, which directly maps observations to actions, inherently entangles task geometry, vehicle dynamics and reward shaping. This makes skills harder to reuse across platforms and increases training effort whenever a new skill, an existing skill, or a vehicle is introduced. Additionally, the reward structure may vary across tasks, which further complicates the design of a monolithic policy architecture. To address this, it is proposed a clear separation between geometric navigation and vehicle-dependent control through a two-stage architecture: a

backbone that learns transferable geometric structure, and an adapter that maps this representation to the target vehicle's control actions.

The formulation of the transferable navigation is a sequential decision-making problem in which the agent receives a task-dependent observation and produces a control action for the vessel. Rather than learning a single end-to-end policy, the proposed framework separates geometric navigation from vehicle-dependent control through a two-stage architecture: a backbone that learns transferable geometric structure and an adapter that maps this representation to the control actions of the target vehicle. This allows the shared backbone to represent the task's navigation structure, while the adapter accounts for platform-dependent actuation constraints.

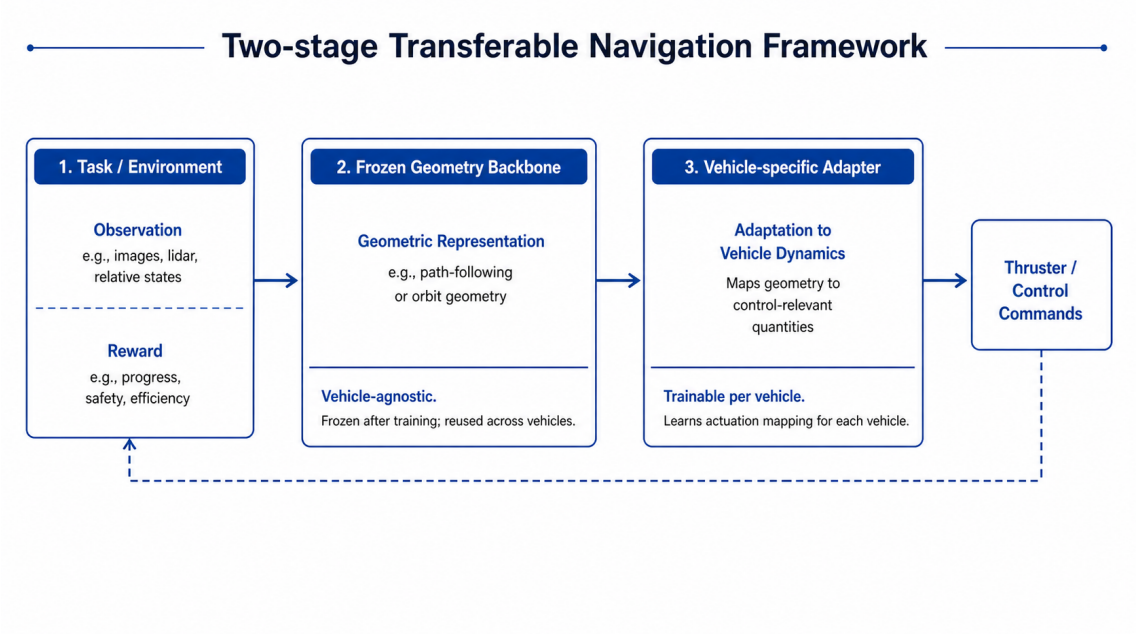


Figure 3.1: Two-stage transferable navigation framework showing a frozen geometry backbone and a vehicle-specific adapter.

Figure 3.1 presents the adopted architecture. The observation extracted from the environment, which may include positional, velocity, and orientation information, is first processed by a geometry backbone that is specific to each skill but agnostic to the vehicle type. This backbone captures the task geometry and outputs a compact representation of the intended navigation behavior. The resulting geometric output is then passed to a vehicle-specific adapter, which maps it to platform-dependent control actions, such as thrust and rudder commands for the USV or velocity setpoints for the drone.

3.2 Orbit Following

The first skill developed is Orbit Following, or as it will be thereafter called - OrbitXY, a circular orbit-keeping task designed to learn transferable geometric navigation under clockwise (CW)

and counterclockwise (CCW) symmetry, while also accommodating variable orbit radii and target speeds. This skill serves as the first testbed for the proposed two-stage framework, since it combines direction invariance with continuous control and vehicle-specific actuation.

The following subsections describe the observation and action spaces, the reward design, the termination conditions, and the metrics used to assess performance.

3.2.1 Observation space

The backbone agent operates on a structured observation that encodes the vessel's relative position, velocity, and heading with respect to the orbit center, while symmetry constraints ensure that a single policy can generalize across both directions. Before additional details on the observation space are provided, Figure 3.2 depicts the information flow from the environment to the backbone and adapter.

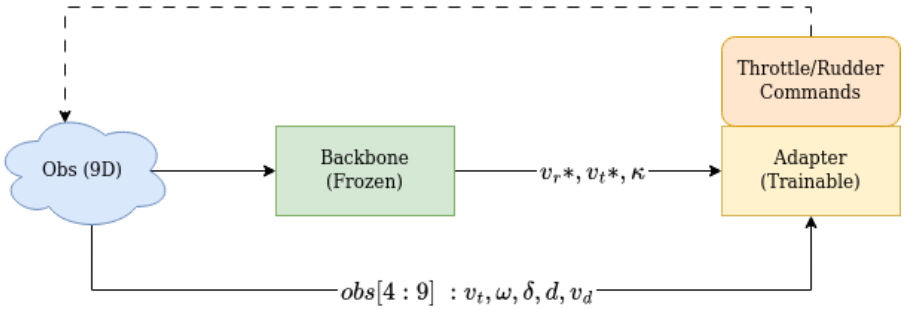


Figure 3.2: Observation flow through the backbone-adapter architecture. The environment provides the full 9D invariant observation to the backbone and a 5D slice to the adapter. The backbone outputs 3D geometry commands $[v_r^*, v_t^*, \kappa^*]$, which the adapter combines with the raw state to produce throttle and rudder commands, closing the control loop.

.

The backbone operates on a structured observation that captures the vessel's relative position, velocity, and heading with respect to the orbit center, while symmetry constraints ensure that a single policy generalizes across both orbit directions. The following subsections then detail the observation and action spaces, reward design, termination conditions, and evaluation metrics.

Although some of these variables are already processed by the backbone, they are provided again to the adapter because the backbone output is a compact geometry intent, whereas the adapter still requires actual vehicle-state context to translate that representation into effective thruster commands.

The backbone maps this observation into a compact 3D geometry representation—radial velocity, tangential velocity, and curvature, and the adapter converts this representation, together with vehicle-state inputs, into throttle and rudder commands for the target vehicle. Here, in polar geometry, v_r is the radial velocity , v_t is the tangential velocity and κ is the curvature of path as outputs. This also makes the observation invariant to orbit direction.

The backbone observation is specifically a 9D vector :

$$obs_{backbone} = \begin{bmatrix} e_r & \sin\phi & \cos\phi & v_r & v_t & \omega & \delta & d & v_d \end{bmatrix}^\top.$$

As illustrated in Figure 3.3, the observation vector encodes the agent's geometric relation to the orbit through six components. The radial error $e_r = r - r_d$ measures how far the agent is from the desired orbit radius, being positive when outside and negative when inside. The bearing $\phi = \text{wrap}(\text{atan2}(-y_{rel}, -x_{rel}) - \theta)$ expresses the direction to the orbit center in the agent's body frame, providing a heading-invariant signal that tells the agent which way to turn to face the center. The tangent angle $\theta_t = \text{atan2}(y_{rel}, x_{rel}) + d\frac{\pi}{2}$ defines the ideal heading direction at the agent's current position, from which the heading error $\delta = \text{wrap}(\theta - \theta_t) \cdot d$ is derived — a value of zero indicating perfect alignment with the orbit tangent. Finally, the radial and tangential velocity components $v_r = \mathbf{v} \cdot \hat{\mathbf{r}}$ and $v_t = \mathbf{v} \cdot \hat{\mathbf{t}}$ decompose the agent's velocity into the outward and tangential directions respectively, allowing the policy to regulate both orbital tracking and speed simultaneously.

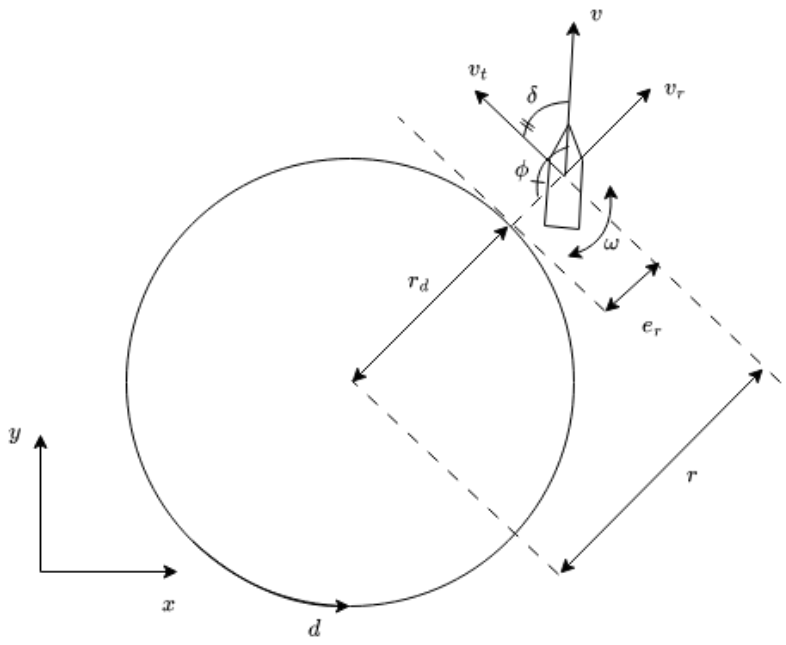


Figure 3.3: OrbitXY invariant observation — geometric quantities expressed in the body frame of the agent. Bearing ϕ and heading error δ are $\pm 90^\circ$ apart, with the sign set by orbit direction $d \in \{+1, -1\}$.

Geometrically, the bearing and heading error are related by a $\pm 90^\circ$ offset:

$$\phi = \delta - d \cdot \frac{\pi}{2} \pmod{\pi}$$

Here $\text{wrap}(\alpha)$ denotes angle normalization to $(-\pi, \pi]$, and all observation components are expressed with standard physical units to make the geometry and dynamics explicit.

For completeness, the radial and tangential unit vectors used to project the vessel velocity onto the orbit frame are defined below.

$$\hat{\mathbf{r}} = \frac{\mathbf{p}_{\text{rel}}}{\|\mathbf{p}_{\text{rel}}\|}, \qquad \hat{\mathbf{t}}_0 = \begin{bmatrix} -\hat{r}_y \\ \hat{r}_x \end{bmatrix}, \qquad \hat{\mathbf{t}} = d\,\hat{\mathbf{t}}_0,$$

As so for the backbone :

$$obs_{backbone} \in \mathbb{R}^9 \xrightarrow{\text{MLP}} a_{geom} = \begin{bmatrix} v_r^* & v_t^* & \kappa^* \end{bmatrix}^\top$$

The backbone maps the 9D invariant observation to 3D geometry commands: desired radial velocity, tangential velocity, and path curvature.

For the second stage, the adapter receives:

$$obs_{adapter} = \begin{bmatrix} a_{geom} & v_t & \omega & \delta & d & v_d \end{bmatrix}^\top \in \mathbb{R}^8 \xrightarrow{\text{MLP}} a_{ctrl} = \begin{bmatrix} \tau & \rho \end{bmatrix}^\top$$

The adapter uses the geometric commands from the backbone together with five raw state measurements : measured tangential velocity, yaw rate, heading error orbit direction, desired speed. With proper training it learns to translate these commands into real thruster commands [throttle,rudder] under diff-drive physics , as will be further explored in later sections

Because the backbone extracts the essential geometric abstraction, the adapter does not need the full observation. These inputs provide the tracking-error and dynamic context required to translate geometric commands into vehicle-specific thruster actions.

The adapter concatenates the frozen backbone's geometric output with a subset of the raw observation containing dynamic state variables (velocity, angular rate, heading error, and target setpoints). This design reflects a classical control decomposition: the backbone provides a kinematic reference trajectory, while the adapter acts as a low-level tracking controller that requires both the reference and the current state to generate appropriate actuator commands.

3.2.2 Action Space

With the observation defined, action spaces are now specified. The backbone and adapter use different action spaces, reflecting their different roles and vehicle-specific actuation. For the backbone, the geometric policy outputs:

$$a_{geom} = \begin{bmatrix} v_r^* \text{ [m/s]} & v_t^* \text{ [m/s]} & \kappa^* \text{ [m}^{-1}\text{]} \end{bmatrix}^\top$$

- v_r^*: radial velocity (toward/away from orbit center, bounded ± 2 m/s)
- v_t^*: tangential velocity along orbit direction (bounded ± 2 m/s)
- κ^*: path curvature, with $\omega = d \cdot \kappa^* \cdot v_t^*$ (bounded ± 0.2 m^{-1})

The backbone operates in a polar geometry frame: v_r^* and v_t^* are converted to world-frame velocities via the radial and tangential unit vectors, and κ^* determines the yaw rate through the curvature relation.

The backbone operates in a polar geometry frame and outputs geometric commands rather than physical actuator commands.

- τ: throttle command
- ρ: rudder command

The adapter maps these outputs to a diff-drive thruster pair: left = $\tau - \rho$, right = $\tau + \rho$, with each thruster saturated at ± 250 N. The rudder is multiplied by d (orbit direction) externally to handle the CW/CCW sign — the MLP only learns the unsigned magnitude.

Table 3.1: Comparison of Backbone and Adapter policies.

	Backbone	**Adapter**
Action	$[v_r, v_t, \kappa]$	$[\tau, \rho]$
Physics	Kinematic (ideal)	Inertial + thruster
Yaw rate	$\omega = d \cdot \kappa \cdot v_t$	$\omega = f(\tau, \rho)$ via diff-drive
Transfer	Any vehicle (geometry)	Specific vehicle (actuator)

Additionally, an exponential moving average (EMA) is also applied on the rudder output with a smoothing factor α, which smooths the rudder command over time, [$\rho_t = \alpha \cdot \rho_t^{\text{raw}} + (1-\alpha) \cdot \rho_{t-1}$.]This models the response lag of a real ASV thruster and helps prevent the adapter from learning jittery, high-frequency control policies in the first stage of simulation that would not transfer well to physical hardware.

3.2.3 Termination Condition

After defining the observation space, the termination conditions are specified. An episode is successful when the agent completes at least one full orbit while remaining within the allowed angular and radial tolerances. All other termination cases are failures and include leaving the operational bounds, drifting too far from the target, getting too close to the orbit center, reaching the maximum episode length, or triggering the stuck and no-progress grace conditions. As shown in Table 3.2, success requires both angular completion and radial accuracy, while the failure conditions mainly prevent stagnation, divergence or boundary violations.

Table 3.2: Episode termination conditions ($n_{\text{orbits}} = 1$ for all experiments). Success requires both angular completion and radial tolerance; all other terminations are considered failures.

Condition	Formula	Backbone	Adapter
Success	$\sum \omega \cdot \Delta t \geq 2\pi n_{\text{orbits}} - \varepsilon$ and $\lvert r - r_d \rvert < \tau$	$\varepsilon = 0.1$ $\tau = 0.8$ m	$\varepsilon = 0.1$ $\tau = 0.5$ m
Out of bounds	$\lvert x \rvert \geq x_{\text{lim}}$ or $\lvert y \rvert \geq y_{\text{lim}}$	20 m	25 m
Far from center	$\lVert p_{\text{rel}} \rVert > r_d + 8.0$ m	13 m +	13 m +
Center too close	$\lVert p_{\text{rel}} \rVert < 1.0$ m	1.0 m	1.0 m
Max steps	step $\geq N_{\max}$	1600	1600
Grace: stuck	$\lVert v \rVert < 0.05$ m/s for N consecutive	$N = 300$	$N = 400$
Grace: no progress	$\Delta n_{\text{orbits}} < \delta$ over N steps	$\delta = 0.02$, $N = 500$	$\delta = 0.03$, $N = 400$

Table 3.3: Grace termination parameters for backbone and adapter policies.

Grace Type	Backbone	Adapter	Trigger
Stuck	300 steps	400 steps	$\lVert v \rVert < 0.05$ m/s
No-progress	500 steps	400 steps	$\Delta n_{\text{orbits}} < 0.02$ (bb) / 0.03 (adpt)
Radial	disabled	disabled	$e_r > 3.0$ (bb) / 5.0 (adpt)

3.2.4 Reward Function

With the observation and action spaces defined, the reward function is the next component of the RL formulation. The reward system is designed to encourage correct radial progression, tangent alignment, and target-speed tracking. The following terms implement these objectives through penalties on radial deviation, heading error, speed mismatch, and directional inconsistency.

Starting with the backbone, the reward is specified in terms of per-step terms, post-sum additions, completion bonuses, and penalties as follows.

The core of the reward signal consists of four penalty terms. The radial penalty $r_{\text{radial}} = -w \cdot e_r^2$ discourages deviation from the desired orbit radius, with the backbone using a higher weight of 12.0 reduced to 8.0 in the adapter to allow slightly looser tracking while the actuator dynamics are learned. The heading penalty $r_{\text{heading}} = -w \cdot \delta^2$ and speed penalty $r_{\text{speed}} = -w \cdot \lvert v_t - v_d \rvert$ remain unchanged across both stages, penalizing misalignment with the orbit tangent and deviation from the desired speed respectively. The direction penalty $r_{\text{direction}} = -w \cdot \max(0, -v_t)$ discourages reverse motion along the orbit, with its weight reduced from 5.0 in the backbone to 2.0 in the adapter. The backbone additionally includes a smoothness penalty $r_{\text{smooth}} = -w \cdot \lVert a_t - a_{t-1} \rVert^2$ with weight 0.12, which is disabled in the adapter as the rudder EMA filter handles smoothing at the dynamics level. The adapter introduces an on-orbit bonus $r_{\text{on_orbit}} = +2.0$ when $\lvert e_r \rvert < 0.5\,\text{m}$, rewarding the agent for staying close to the orbit, a term absent in the backbone. The on-orbit bonus is absent in the backbone because GeometryDrive operates under idealised kinematics: the agent cannot drift from the commanded orbit, so radial and heading penalties alone suffice. The adapter requires this bonus because the Thruster-based differential drive introduces momentum, thrust saturation, and drag — physical effects that push the agent off the orbit and demand a dense

shaping signal to encourage active recovery. A grace termination penalty of -50 is applied when a grace condition is triggered in both policies, while the failure penalty is set to 0 in the adapter compared to -10 in the backbone. A reward clamp is used to restrict the final return to a fixed interval, preventing extreme penalties or bonuses from dominating the optimization signal. The clamp is set wide (± 1000) to preserve the large radial penalty gradient ($e_r = 10$ m $\rightarrow -1200$ raw, clamped to -1000). The completion bonus is added after clamp so it is never absorbed. The total reward is then clamped and augmented with a lap completion bonus:

$$R_{\text{total}} = \text{clamp}\big(r_{\text{radial}} + r_{\text{heading}} + r_{\text{speed}} + r_{\text{direction}} + r_{\text{on_orbit}} + r_{\text{grace}},\ -1000,\ 1000\big) + r_{\text{completion}}$$

where $r_{\text{completion}} = 50$ per completed lap in the adapter, increased from 10 in the backbone. The full breakdown of weights is summarized in Table 3.4.

Table 3.4: Reward components for the OrbitXY backbone and adapter policies. Dashes indicate the component is not applicable.

Component	Formula	Backbone	Adapter
r_{radial}	$-w \cdot e_r^2$	12.0	8.0
r_{heading}	$-w \cdot \delta^2$	1.5	1.5
r_{speed}	$-w \cdot \lvert v_t - v_d \rvert$	4.0	4.0
$r_{\text{direction}}$	$-w \cdot \max(0, -v_t)$	5.0	2.0
r_{smooth}	$-w \cdot \lVert a_t - a_{t-1} \rVert^2$	0.12	–
$r_{\text{on_orbit}}$	$+2.0$ if $\lvert e_r \rvert < 0.5$ m, else 0	–	2.0
r_{grace}	-50 if grace termination, else 0	-50	-50
r_{failure}	penalty on failure termination	-10	0
$r_{\text{completion}}$	$+w$ per completed lap	10.0	50.0

Grace penalties are also used which result from episode termination due to violation of these grace periods defined to avoid having the agent getting stuck. Further elaboration on these is laid out in the next section.

3.3 Path tracking

As noted in the previous section, the next natural step is to extend the framework to the more challenging Path tracking, or from now on called FollowPath, which is formulated in the Frenet-Serret frame over spline trajectories interpolating a set of waypoints. These waypoints can be provided by a mission controller or hierarchical system through a sequence of messages. The structure of this section mirrors the previous one, covering the observation and action spaces, reward design, termination conditions, and metrics used to evaluate the agent's performance.

3.3.1 Observation and Action Spaces

The same section structure is followed for FollowPath. We first present the information flow from the environment to the backbone and adapter. As in OrbitXY, Figure 3.4 shows the observation flow between the backbone and adapter, now extended with the additional carrot-based lookahead inputs.

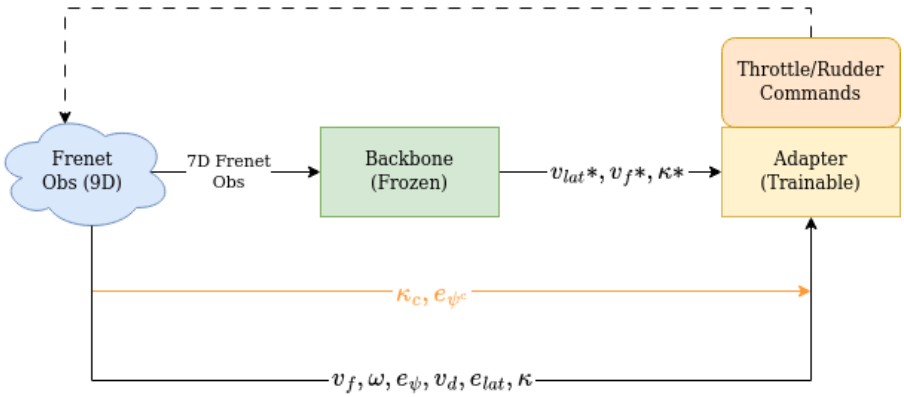


Figure 3.4: FollowPath observation flow through the frozen backbone and trainable adapter, augmented with the additional carrot lookahead observations used to anticipate upcoming turns.

.

The lookahead point, or carrot, (colored in orange) is used as an auxiliary observation to support anticipatory control. This information is less critical for the geometry-only backbone, since it does not model inertia, friction, or drag, but it is valuable for the USV adapter, where limited thrust, rudder authority, and vehicle inertia make early anticipation of sharp turns beneficial. In this case, the carrot helps the policy adjust its behavior with sufficient foresight rather than reacting only after the error has already grown. Furthermore, the lookahead distance is adapted per step by shrinking on tight turns or large heading errors, and expanding on straighter,smoother segments.

Now that the information flow has been established, the next step is defining the observation space, a brief introduction on the Frenet-Serret frame. The Frenet-Serret frame, hereafter referred to as the Frenet frame, is a moving coordinate frame used to describe the local geometry of a differentiable curve in three-dimensional Euclidean space. It is defined by the tangent, normal, and binormal unit vectors, and by how these vectors evolve along the curve. In the following, this frame is used to express the path-relative quantities required by the FollowPath observation.

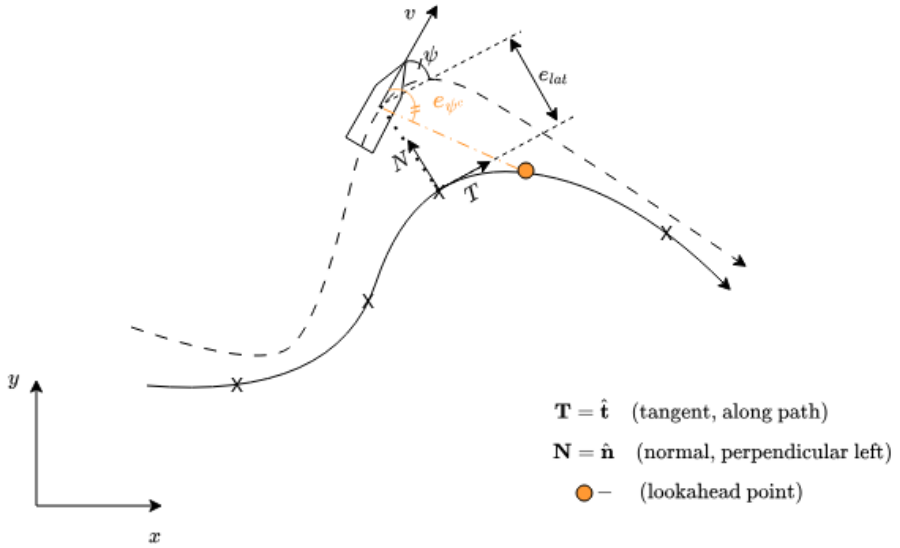


Figure 3.5: Frenet-frame geometry for the FollowPath task, showing the closest point on the spline, tangent and normal directions, lateral deviation e_{lat}, heading error ψ, and the carrot lookahead point used to compute anticipation terms such as carrot heading error and lookahead curvature.

The workflow for Frenet projection requires an additional interpolation step, so a proper discrete evaluation sequence must first be defined.

As mentioned above, the higher-level controller, such as the mission planner, provides a list of waypoints. These waypoints are then interpolated using a smooth Catmull-Rom spline, which is C1-continuous and local, yielding smooth tangents while still passing through the waypoints, exactly as required by the Frenet projection. This formulation assumes non-intersecting paths, since closest-point Frenet projection becomes ambiguous on self-intersecting trajectories. The waypoint sequence is then extended with mirrored endpoints to enforce smooth boundary conditions. The Catmull-Rom spline is evaluated densely and re-sampled by arc length, yielding a uniformly spaced representation suitable for Frenet projection.

The Frenet observation vector captures the vessel's state relative to the path at the closest point. The lateral deviation e_{lat} measures the signed cross-track error, while the heading error ψ quantifies the angular mismatch between the vessel heading and the local path tangent. The curvature κ describes the local path geometry, and the forward velocity v_f is the projection of the vessel velocity onto the tangent direction. The angular velocity ω and desired speed v_d provide dynamic and task-level context, while the path-direction flag p remains constant in the forward-only setting. The carrot terms κ_c and e_{ψ^c} provide lookahead guidance by encoding the curvature and heading error at a point ahead on the path, helping the adapter anticipate upcoming turns.

Full observation for the backbone comes as :

$$obs_{backbone} = \begin{bmatrix} e_{\text{lat}} & \psi & \kappa & v_f & \omega & \delta & v_d & p \end{bmatrix}^\top$$

$$obs_{backbone} \in \mathbb{R}^7 \xrightarrow{\text{MLP}} a_{geom} = \begin{bmatrix} v^*_{\text{lat}} & v^*_f & \kappa^* \end{bmatrix}^\top$$

The backbone maps the 7D Frenet observation — lateral error, heading error, curvature, forward velocity, angular velocity, path direction, target speed — to 3D geometry commands. It operates in a frictionless kinematic world and is trained in simulation . For the adapter, the observation concatenates backbone commands, raw state, and carrot signals:

$$obs_{adapter} = \begin{bmatrix} a_{geom} & s_{raw} & c_{carrot} \end{bmatrix}^\top \in \mathbb{R}^{11} \xrightarrow{\text{MLP}} a_{ctrl} = \begin{bmatrix} \tau & \rho \end{bmatrix}^\top$$

where $s_{raw} = \begin{bmatrix} v_f & \omega & \varepsilon_\psi & v_d & e_{\text{lat}} & \kappa \end{bmatrix}^\top \in \mathbb{R}^6$ are raw measurements and $c_{carrot} = \begin{bmatrix} \kappa_{\text{carrot}} & \varepsilon_{\text{carrot}} \end{bmatrix}^\top \in \mathbb{R}^2$ encode lookahead information — curvature ahead and the pure-pursuit heading error to the carrot point.

With proper training the adapter learns to translate geometric commands and carrot signals into real thruster commands under diff-drive physics, which we will explore in later sections. As shown previously in Section 3.2, some observations are ingested by the backbone in order to produce geometric intents, that the adapter together with additional raw observations will translate into throttle and rudder commands.

The action space structure follows the same backbone/adapter patters as OrbitXY, except that FollowPath backbone outputs geometry commands :

$$a_{geom} = \begin{bmatrix} v^*_{lat}\,[\text{m/s}] & v^*_f\,[\text{m/s}] & \kappa^*\,[\text{m}^{-1}] \end{bmatrix}^\top$$

Having defined the observation and action spaces the Reward Function follows. In the next section full termination conditions are presented.

3.3.2 Termination Conditions

Similarly to Section 3.2.3, FollowPath episodes terminate on either success or failure. Success requires both sufficient path progress and bounded lateral error, while failure occurs when the agent leaves the workspace, deviates too far from the path, experiences a Frenet jump, or reaches the maximum episode length.

Unlike OrbitXY, FollowPath uses only a stuck grace condition; no-progress and radial grace conditions are not applicable because progress is measured directly through Frenet projection and lateral deviation is enforced as a hard termination criterion.

Table 3.5: Episode termination conditions. Success requires both path completion and lateral tolerance; all other terminations are considered failures.

Condition	Formula	Backbone	Adapter
Success	$s \geq 1 - \varepsilon_s$ and $\lvert e_{\text{lat}} \rvert < \tau_{\text{lat}}$	$\varepsilon_s = 0.05$, $\tau_{\text{lat}} = 1.5$ m	$\varepsilon_s = 0.05$, $\tau_{\text{lat}} = 1.5$ m
Out of bounds	$\lvert x \rvert \geq x_{\text{lim}}$ or $\lvert y \rvert \geq y_{\text{lim}}$	15 m	25 m
Lateral violation	$\lvert e_{\text{lat}} \rvert \geq L_{\text{bound}}$	5.0 m	5.0 m
Frenet jump	$\Delta s > f \cdot \Delta s_{\text{step}}$	$f = 4.0$	$f = 4.0$
Max steps	step $\geq N_{\text{max}}$	1600	1600
Stuck grace	$\lVert v \rVert < 0.05$ m/s for N consecutive steps	$N = 300$	$N = 300$

3.3.3 Reward Function

As suggested by the observation design, the reward combines lateral tracking, heading alignment, turn anticipation, motion encouragement, and failure penalties to promote stable path following on open spline trajectories.

The reward function for the FollowPath adapter is redesigned relative to the backbone to account for the inertial dynamics of the differential-drive platform and to prioritize smooth, forward-progressing behavior along the path. It can be understood in two groups: tracking penalties and motion incentives.

The tracking penalties penalize geometric deviation from the path. The lateral error penalty $r_{\text{lat}} = -w \cdot e_{\text{lat}}^2$ is the primary tracking signal, with its weight reduced from 8.0 in the backbone to 3.5 in the adapter, reflecting the looser tracking tolerance expected from a physically constrained vehicle. Two heading penalties are included: $r_{\text{heading}} = -w \cdot e_{\psi}^2$ penalizes misalignment with the tangent at the closest point on the path, while $r_{\text{carrot}} = -w \cdot (e_{\psi}^c)^2$ penalises misalignment with a lookahead carrot point, implementing a pure-pursuit-style objective with weight 3.0. Together these two terms encourage the agent to both align locally and steer proactively toward the upcoming path geometry.

The motion incentives encourage the agent to make forward progress at the desired speed. The progress reward $r_{\text{progress}} = +w \cdot v_f^+ \cdot \Delta t$ accumulates reward proportional to forward velocity along the path, with its weight substantially increased from 1.5 to 5.0 to counteract the higher inertia of the platform. The motion reward $r_{\text{motion}} = +w \cdot \lvert v_f \rvert$ provides a continuous incentive to keep moving, also increased from 0.01 to 1.0. Three penalties discourage undesirable motion patterns: the reverse penalty $r_{\text{reverse}} = -w \cdot [\min(0, -v_f)]^2$ quadratically penalizes backward motion, introduced in the adapter with weight 5.0; the stall penalty r_{stall} applies a fixed penalty when the agent is near the path but moving below 0.1 m/s, added with weight 2.0; and the speed tracking term r_{speed}, active in the backbone with weight 3.0, is disabled in the adapter as explicit speed regulation was found to conflict with the progress objective under inertial constraints. Finally, a completion

bonus of $+10$ is awarded upon successfully traversing the full path, and a failure penalty of -50 is applied on any termination failure, both unchanged from the backbone.

The full weight comparison is summarized in Table 3.6.

Table 3.6: Reward components for the FollowPath adapter policy.

Component	Formula	Weight	Note
r_{lat}	$-w \cdot e_{\text{lat}}^2$	3.5	backbone: 8.0
$r_{heading}$	$-w \cdot e_{\psi}^2$	0.5	closest-point tangent alignment
r_{carrot}	$-w \cdot (e_{\psi}^c)^2$	3.0	pure-pursuit to carrot point
$r_{progress}$	$+w \cdot v_f^+ \cdot \Delta t$	5.0	backbone: 1.5
r_{motion}	$+w \cdot \lvert v_f \rvert$	1.0	backbone: 0.01
$r_{reverse}$	$-w \cdot [\min(0, -v_f)]^2$	5.0	backbone: 0.0
r_{stall}	$-\mathbf{1}_{\text{near}} \cdot \mathbf{1}_{v_f<0.1}$	2.0	backbone: 0.0
r_{speed}	$-w \cdot e^{-0.3e_{\text{lat}}^2} \left(v_f^+ - v_d\right)^2$	0.0	backbone: 3.0 (gated)
$r_{completion}$	$+10$ if path completed, else 0	10.0	unchanged
$r_{failure}$	-50 if failure condition triggers	–	unchanged

Compared with OrbitXY reward design (Section 3.2.4), two changes are particularly important in FollowPath. First, the carrot heading loss is weighted more strongly than the tangent-based heading term, reflecting the fact that turn anticipation becomes more important in the later curriculum stages than precise alignment with the closest spline segment. Second, the speed term is disabled, since the experiments showed that, for this task, prioritizing lateral and heading accuracy is more effective than enforcing a target forward speed.

For the total clamped reward:

$$R_{\text{total}} = \text{clamp}\big(r_{lat} + r_{heading} + r_{carrot} + r_{progress} + r_{motion} + r_{reverse} + r_{stall},\ -100,\ 100\big) + r_{completion} + r_{failure}$$

The total reward is clamped to $[-100, 100]$, which is tighter than in OrbitXY and reflects the smaller scale of the path-following reward terms. Completion and failure bonuses are applied after clamping, while termination is handled separately in the next section.

3.4 Reinforcement Learning Architecture

With the theoretical background for both skills now established, the implementation and training details of the reinforcement learning policiesfollows . The next section present the RL architecture and its hyperparameter configuration.

3.4.1 Backbone and Adapter Policies

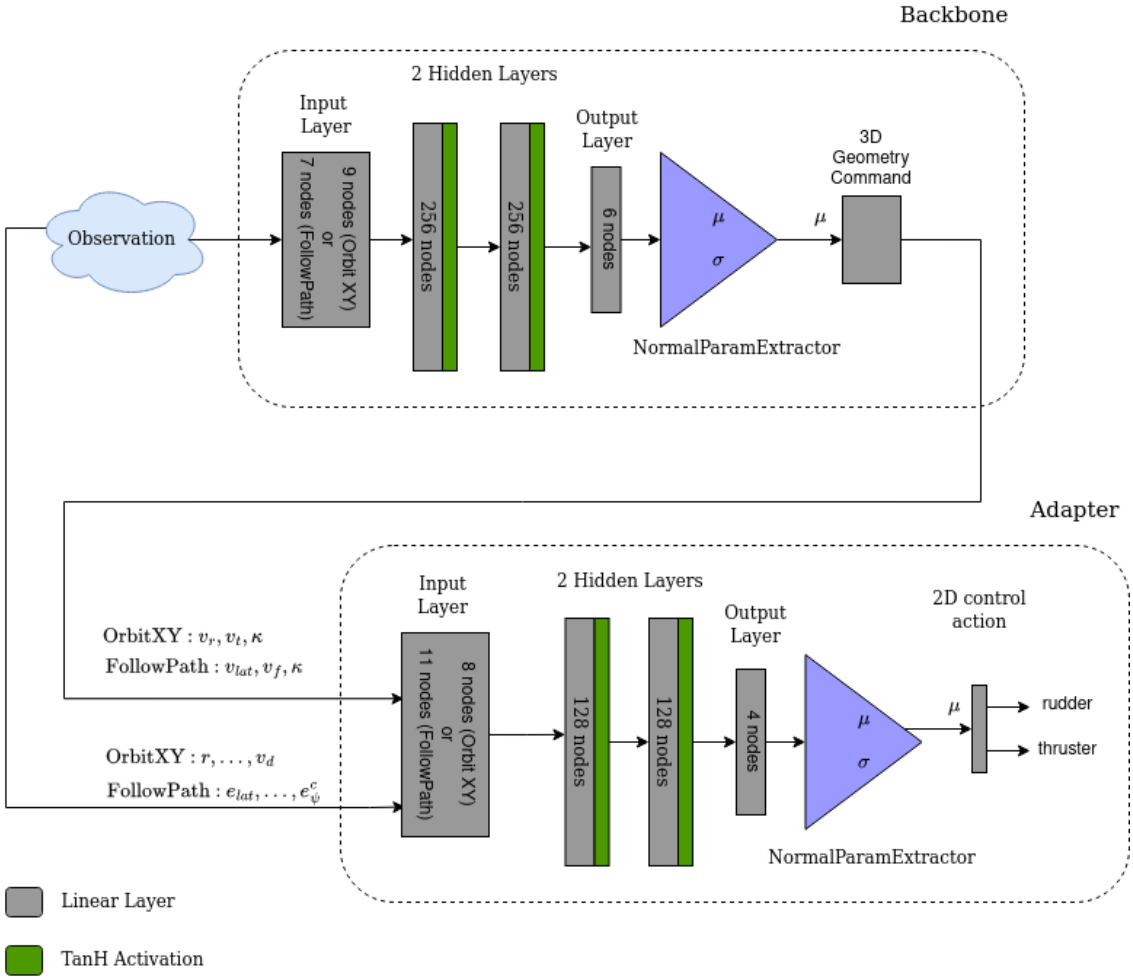


Figure 3.6: Two-stage reinforcement learning architecture used in this work. The backbone maps task observations to a 3D geometric command, while the adapter combines this command with task-specific state information to produce low-level control actions

.

As reviewed in Chapter 2 (Sections 2.2.1 and 2.3), two algorithms stand out as strong candidates for continuous-control Reinforcement Learning tasks of this nature: PPO and SAC . PPO was selected over SAC because this methodology demands stable optimization and architectural clarity over sample efficiency. PPO also adapts cleanly to curriculum learning, as each rollout reflects the current task distribution, whereas SAC's replay buffer can contain stale data from earlier curriculum stages, potentially destabilizing learning.

The backbone is an MLP policy network trained in a Vectorized Multi Agent Simulator (VMAS), and saved once performance is satisfactory, under the criteria defined in Chapter 4. These include, success rate, distance and speed errors and reward obtained. The adapter is trained separately, also in VMAS, receiving the backbone's geometric output together with vehicle-state observations and mapping them to platform-specific control actions. This separation reduces training cost and facilitates transfer across vehicles. Figure 3.6 illustrates the full architecture, including network sizes,

outputs, and observation flow.

The backbone uses a two-layer MLP with 256 hidden units, while the adapter uses a smaller two-layer MLP with 128 units. The backbone is wider because it must compress richer state information into a compact vehicle-agnostic command, whereas the adapter has the simpler role of mapping that command and a small set of vehicle-specific observations to low-level controls. Both networks use Tanh activations, a smooth zero-centered nonlinearity that supports stable optimization and smooth outputs in continuous-control settings, and are parameterized through a NormalParamExtractor that splits the output vector into a mean and a positive scale for the stochastic action distribution. During inference only the mean is retained, while the scale drives exploration during training.

Both policies are trained under the actor-critic PPO framework. In the backbone phase, actor and critic operate on the geometric observation space. In the adapter phase, the frozen backbone supplies the geometric command to a second actor-critic pair, which learns the vehicle-specific mapping. The critic estimates $V(s_t)$ to compute the advantage, and both actor and critic are updated through the clipped PPO objective using the transition tuple $(s_t, a_t, r_{t+1}, s_{t+1})$.

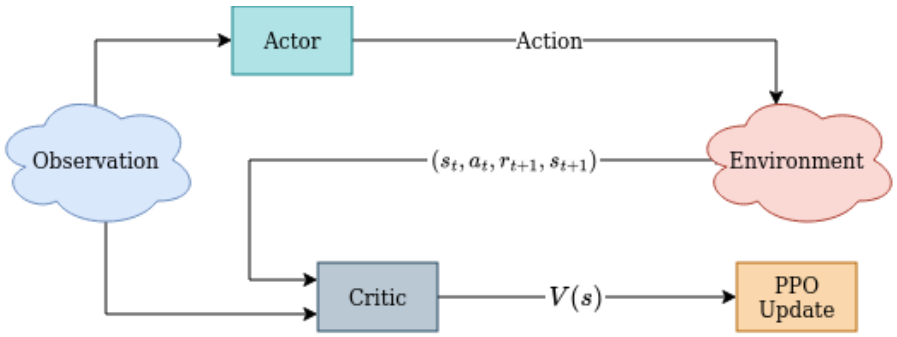


Figure 3.7: Actor-Critic training loop used with PPO. The actor maps the observation to an action, which is applied to the environment; the environment then returns the transition tuple $(s_t, a_t, r_{t+1}, s_{t+1})$. The critic estimates the state Value $V(s_t)$, which is combined with the collected transitions to compute PPO update, consisting of clipped policy loss, value loss and entropy bonus

.

3.4.2 PPO Parameters

The PPO parameters can be summarized below, in Tables 3.7 and 3.8 :

Some differences between the backbone and adapter hyperparameters are intentional. The backbone is trained with a smaller learning rate so that updates remain conservative while learning the reusable geometric representation, which helps preserve the shared navigation structure. By contrast, the adapter learns a simpler mapping from structured geometric features to actuation, where larger steps are stable because the frozen backbone provides a consistent, bounded input distribution. The clipping range follows a different logic. The backbone uses a looser clip (0.2) because it operates in an idealized kinematic setting, where moderate policy divergence does not destabilise the agent — the physics are forgiving and the state space is well-behaved. The

adapter has a tighter clip (0.1) ,because it models a thruster based differential drive which introduces inertia, thrust saturation, and momentum: a policy that diverges too far from its previous version can produce destructive thruster oscillation or saturation, so the surrogate objective must be constrained more aggressively.

Table 3.7: Shared PPO parameters. These parameters are shared across all 4 settings.

Parameter	Value
Discount factor γ	0.99
GAE λ	0.9
Value-function coefficient	1.0 (default)
Max gradient norm	0.5
Advantage normalization	Yes

Table 3.8: PPO hyperparameters for all four policy variants. Backbones share one regimen; adapters share another.

Parameter	OrbitXY Backbone	FollowPath Backbone	OrbitXY Adapter	FollowPath Adapter
Learning rate α	3.0×10^{-5}	3.0×10^{-5}	2.5×10^{-4}	2.5×10^{-4}
Clipping range ε	0.2	0.2	0.1	0.1
Entropy coefficient	0.03	0.03	0.08	0.08
PPO epochs	10	10	8	8
Minibatch size	400	400	200	200
Steps per update	3200	3200	6400	6400
Value-function coefficient	1.0 (default)	1.0 (default)	1.0 (default)	1.0 (default)
Vectorised envs	2	2	4	4

The adapter also uses a higher entropy coefficient to encourage stronger exploration while learning the control mapping under more complex dynamics. Likewise, the backbone uses more epochs than the adapter because the former benefits from more repeated optimization over cleaner rollouts, whereas the adapter is trained with larger rollout batches and fewer epochs to stabilize learning under noisier vehicle-specific dynamics and reduce the risk of overfitting to a single batch of trajectories.

The training configuration for backbone and adapter policies is presented, for both skills, in Table 3.9. There are also evident differences in the steps per batch, iteration number, epochs and minibatch sizes. As stated previously hyperparameters were selected based on the distinct learning objectives of each policy. Backbone policies, which map invariant observations to geometric commands via idealized dynamics, used smaller batch sizes (3,200–9,600) with more epochs (10) and a lower learning rate (3e-5) for fine-grained geometry learning. Adapter policies, which map frozen backbone outputs to real thruster commands via asv specific dynamics, used larger batch sizes (9,600–12,800) with fewer epochs (8) and a higher learning rate (2.5e-4) to handle the increased stochasticity of physical actuation. All policies trained for 600,000 environment steps, sufficient for convergence based on preliminary experiments.

Table 3.9: Training configuration for all backbone and adapter policies.

Policy	Total Steps	Steps/Batch	Iterations	Epochs/Batch	Minibatch
OrbitXY Backbone	600 000	3 200	188	10	400
OrbitXY Adapter	600 000	9 600	63	8	200
FollowPath Backbone	600 000	9 600	63	10	400
FollowPath Adapter	600 000	12 800	47	8	200

3.5 Implementations and Training Details

With the skills and reinforcement learning architectures defined, this section details the environment and vehicle dynamics, training pipeline, and curriculum schedules used in each phase.

3.5.1 Environment and Vehicle Dynamics

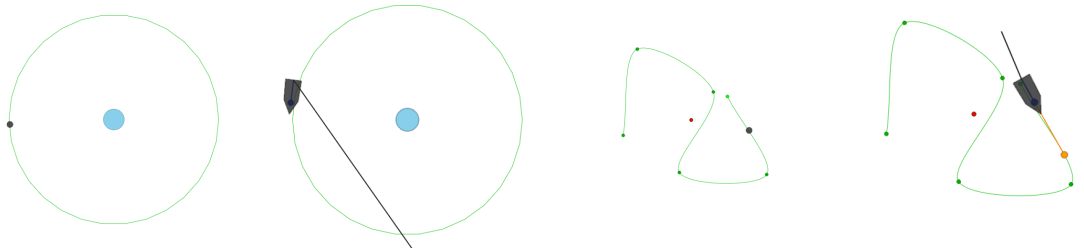
Both backbones are trained in a frictionless kinematic setting with no inertia or drag and an agent mass of 10 kg, with episodes lasting up to 1600 steps at $\Delta t = 0.2$ s. This setup represents a 6DOF generic agent, in our setup limited to planar action, which simplifies and accelerates the backbone training.

The adapters are trained on a differential vehicle model that maps policy outputs to physical thrust commands. The network produces stochastic throttle and rudder commands, scaled and split into left and right thruster inputs, bridging the backbone's geometric representation with the real actuation constraints of the ASV. Both adapters share the same 130 kg platform, summarized in Table 3.10. The same command structure is used at deployment time in Gazebo, which is a physics-based robot simulator widely used in maritime and aerial robotics research,and was used in conjunction with ROS for communication and control, and RViz for real-time visualization of the vehicle state and trajectory. The backbone trains with a 10 kg agent under an idealised velocity command, while the adapter and Gazebo use a 130 kg vessel under force-based dynamics. This mass difference is immaterial to the backbone because it outputs kinematic setpoints (v_{fwd}, κ) rather than force commands — the adapter learns the thrust mapping required for the heavier platform.

Table 3.10: Main vehicle parameters used in adapter training and deployment.

Parameter	Value	Notes
Mass	130 kg	Adapter training vehicle
Length	3.0 m	ASV hull length
Beam	1.4 m	ASV width
Thruster baseline	0.7 m	Distance between thrusters
Max thrust per thruster	$\pm$250 N	Saturation limit

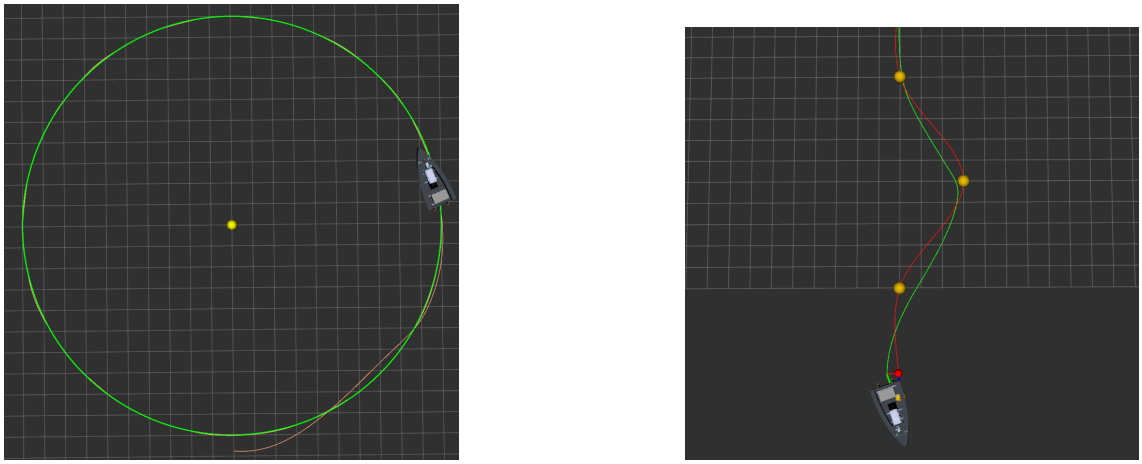
The visual inspection is made first in a VMAS inference rendering, which is depicted in Figure 3.8, then in a Gazebo simulation, here also depicted in Figure 3.9.



(a) OrbitXY backbone. (b) OrbitXY adapter. (c) FollowPath backbone. (d) FollowPath adapter.

Figure 3.8: VMAS visual inspection of trained policies for both skills and stages.

Below the RViz visualization is presented, where it is possible trace the ROS topics that are published. An ideal orbit is traced, with green color and the actual ASVs path with an orange path.



(a) Counter-clockwise orbit, $r = 10$ m, $v_d = 1.0$ m/s.

(b) FollowPath, example of a randomly generated path.

Figure 3.9: Skill evaluation in Gazebo. This is the final simulation environment for evaluation.

3.5.2 Training Pipeline

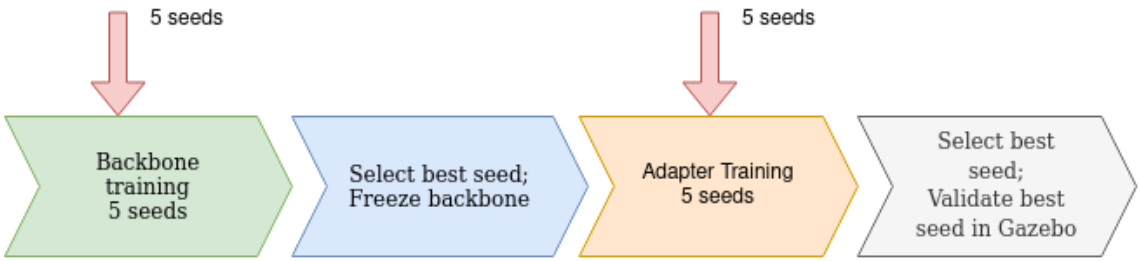


Figure 3.10: Training pipeline. Five seeds are trained per backbone stage; the best checkpoint is frozen and used as the fixed foundation for adapter training, also with five seeds. The final model is validated in Gazebo.

The training procedure is staged. Five independent seeds are trained per skill, and the best checkpoint is selected by success rate, then evaluation reward, with visual rendering in VMAS as a qualitative check. Once validated, the backbone is frozen and the adapter is trained using the same multi-seed strategy. The best adapter seed is then validated in Gazebo; if performance is unsatisfactory, the next-best seed is evaluated instead.

3.5.3 Curriculum Learning Schedules

Both backbone skills use curriculum learning to stabilize training. For OrbitXY, the scenario parameters are updated at fixed step boundaries as shown in Table 3.11. For FollowPath, path complexity increases through turn sharpness and waypoint count, with a blending mechanism that linearly ramps over a 60 k-step window before each transition, allowing environments to gradually sample from the next stage rather than switching abruptly (Table 3.12).

For the adapters, a similar staged approach is used. The OrbitXY adapter enforces a 50/50 CCW/CW split at the batch level and introduces radius randomization earlier than the backbone, keeping the Stage 1 range narrower before expanding to the full $[5, 15]$ m range at Stage 2 (Table 3.13). The FollowPath adapter skips Stage 0 entirely, beginning directly at gentle curves and progressing to sharp turns over 300 k steps with the same blending mechanism as the backbone (Table 3.14).

Table 3.11: Three-stage curriculum schedule for the OrbitXY backbone.

Stage	Steps	Radius	Speed	Direction
0	$0 - 200$k	Fixed at 5 m	Fixed at 1.0 m/s	Random CW/CCW
1	200k $- 400$k	Random $[5, 15]$ m	Fixed at 1.0 m/s	Random CW/CCW
2	400k$+$	Random $[5, 15]$ m	Random $[0.3, 1.5]$ m/s	Random CW/CCW

Table 3.12: Four-stage curriculum schedule for the FollowPath backbone.

Stage	Steps	Min Turn	Max Turn	Min Waypts	Max Waypts	Description
0	$0 - 50$k	0°	10°	4	5	Nearly straight
1	50k $- 150$k	15°	30°	5	7	Gentle turns
2	150k $- 300$k	30°	60°	6	8	Moderate turns
3	300k$+$	60°	120°	6	10	Sharp turns

Table 3.13: Three-stage curriculum schedule for the OrbitXY adapter.

Stage	Steps	Radius	Speed	Direction
0	$0 - 200$k	Fixed at 5 m	Random $[0.5, 1.5]$ m/s	Random CW/CCW
1	200k $- 280$k	Random $[5, 9]$ m	Random $[0.5, 1.5]$ m/s	Random CW/CCW
2	280k+	Random $[5, 15]$ m	Random $[0.5, 1.5]$ m/s	Random CW/CCW

Table 3.14: Curriculum schedule for the FollowPath adapter. Stage 0 is skipped; training begins at Stage 1.

Stage	Steps	Min Turn	Max Turn	Min Waypts	Max Waypts	Description
0	skipped	0°	10°	4	5	Nearly straight (never used)
1 ← start	0	15°	30°	5	7	Gentle curves
2	150k	30°	60°	6	8	Moderate turns
3	300k	60°	120°	6	10	Sharp turns

3.5.4 Evaluation Protocol

For OrbitXY training, evaluation spans a grid of radii (5–20 m), speeds (0.5–2.0 m/s), and orbit directions (CW/CCW), requiring multiple episodes to assess convergence across the full operating envelope. While the orbit geometry remains parametrically simple, the policy must generalise across radius and speed combinations. FollowPath adapter training is more involved. Since a new random Catmull-Rom spline is generated in every episode, with varying turn angles, segment lengths, and curvature profiles, the adapter must be evaluated on geometries that have not been seen before, rather than on a fixed parametric family.

To account for this setting, three evaluation streams are defined in Table 3.15. For FollowPath, the randomized deterministic stream is used as the primary evaluation, since action stochasticity is removed and failures can therefore be attributed to path geometry rather than sampling noise. The fixed deterministic stream is retained to measure peak performance on a stable benchmark, while the randomized stochastic stream is kept as a robustness check under exploration noise. This split is not used for OrbitXY, since its topology remains circular and only parametric variation is introduced through radius, speed, and direction.

The three-stream setup is introduced for the adapter because its end-to-end success rate is markedly noisier than the backbone's, and it is more strongly affected by vehicle dynamics, action stochasticity, and rollout variability than by the geometric structure itself.

Table 3.15: FollowPath adapter evaluation streams.

Stream	**Meaning**	**Purpose**
Randomized deterministic	Random path per episode, no action noise	Measures generalisation to unseen path geometries
Fixed deterministic	Fixed curriculum path, no action noise	Measures peak performance on a stable benchmark
Randomized stochastic	Random path per episode, with action noise	Measures robustness under exploration noise

This chapter presented the methodology adopted in this dissertation, from problem formulation and skill definitions to the reinforcement learning architecture and training procedures, establishing the proposed two-stage framework for learning transferable navigation skills in unmanned surface vehicles.

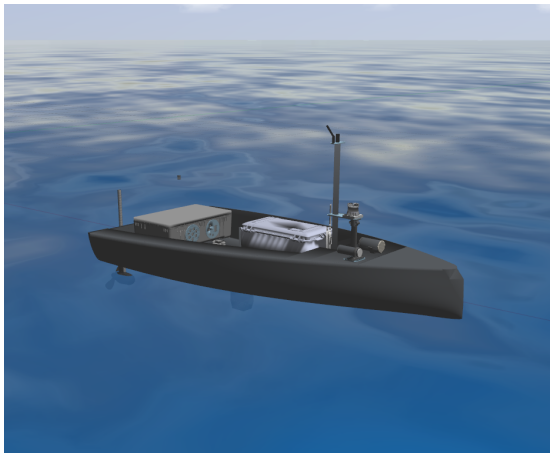
Chapter 4

Results

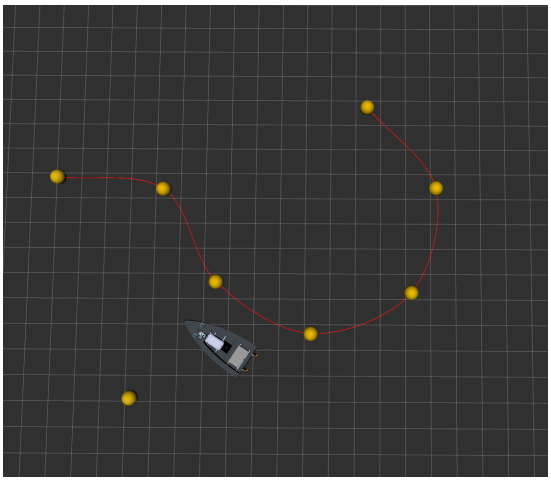
In the present chapter, the experimental results obtained from the previously presented methodology are reported. The reinforcement learning policies (skills) introduced in Chapter 3 are implemented in Python using the TorchRL reinforcement learning library from the open-source PyTorch machine learning framework. VMAS — Vectorized Multi-Agent Simulator — is used in the first stage of training, enabling fast parallel iteration and best seed selectionOnce the best seed is selected for each skill, a higher-fidelity simulator, Gazebo, is employed for validation. Robot Operating System (ROS) provides standardized inter-process communication for robotics applications via a publish–subscribe message architecture. In this work, ROS bridges the trained policies with Gazebo: policies publish thruster commands and receive state feedback. RViz, ROS's 3D visualization tool, renders robot trajectories, reference paths, and navigation markers in real time. It is used throughout this chapter to produce qualitative screenshots of tracking performance — showing the agent's trail, intended path, and geometric relationships in the simulation environment. While VMAS provides a fast, vectorized training environment with idealized dynamics, Gazebo evaluation tests transferability to a realistic physics engine (ODE) with proper rigid-body dynamics, ROS integration, and real-world noise artifacts. Success in Gazebo validates that the frozen geometric backbone extracts transferable structure, not simulator-specific dynamics. In Gazebo, the USV platform is modeled as the Nautilus[2] — a 130 kg differential-drive surface vessel with length 3.0 m, width 1.4 m, and a thruster baseline of 0.68 m. The vehicle is actuated by two independent thrusters mounted aft (left and right), each producing a maximum force of ±250 N. Hydrodynamic drag is modeled with linear (xU = 51.3) and quadratic (xUU = 72.4) damping coefficients, operating in water at 997.8 kg/m³ density. All training and evaluation were performed on a workstation running Ubuntu 22.04, equipped with an Intel Core i9-13950HX (24 cores, 32 threads) and an NVIDIA RTX 3500 Ada Generation Laptop GPU (12 GB VRAM). VMAS training utilized CPU parallelization (6 environments per batch), for faster policy improvement iteration cycles.

Before the training and transfer results are shown, the relevant metrics and testing scenarios limits are explained for both skills. Then backbone is evaluated for each skill, followed by the adapter results, with OrbitXY addressed before FollowPath. The discussion focuses on task performance, learning stability, curriculum progression, and transfer behavior in Gazebo simulation.

Ending the chapter, some conclusions are drawn from the main achieved results. The analysis covers training convergence and generalization in simulation, grid evaluation under realistic physics, cross-platform transfer to an unmanned aerial vehicle, and an ablation study isolating the contribution of the frozen geometric backbone.

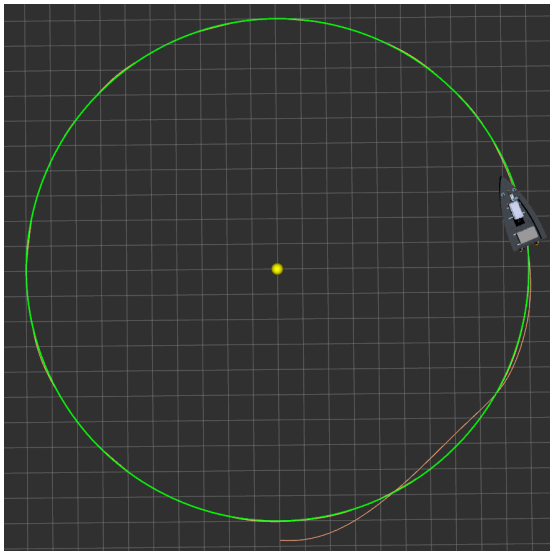


(a) Nautilus, the USV used for the final validation step in Gazebo.

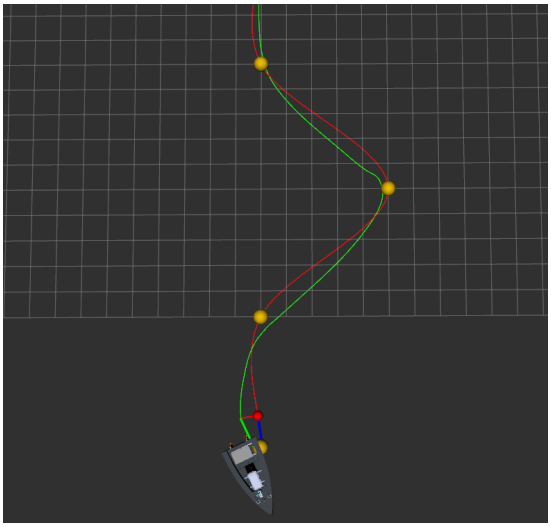


(b) RViz, the visualization tool of ROS, used for visual inspection and validation of the Nautilus skills.

Figure 4.1: Gazebo and RViz, the final stage for test and validation.Gazebo provides a realistic simulation scenario, while RViz allows the visualization of trails, markers and sensor information.



(a) RViz visualization of the OrbitXY skill testing in Gazebo.



(b) RViz visualization of the FollowPath skill testing in Gazebo.

Figure 4.2: RViz visualizations of both skills. After the selection of the best performing seed, a visual inspection of the inference was performed, alongside quantitative log analysis.

4.1 Metrics and Test Scenario

To evaluate training and transfer performance, metrics are grouped into three categories: task completion, tracking quality, and control behavior. For OrbitXY, task completion is measured by success rate (completion of $\geq$1 orbit within radial tolerance) and the number of orbits traversed. Tracking quality is captured by radial distance error and speed error . Control behavior is assessed via action smoothness, which penalizes thruster oscillation. For FollowPath, task completion is measured by success rate (reaching $\geq$95% path progress) and episode length. Tracking quality uses lateral error, supplemented by path progress and control smoothness is again reported to assess action jitter. Validation metrics (eval success rate, eval episode reward) are logged alongside training metrics to monitor generalization and detect overfitting. Taken together, these metrics separate objective completion, geometric tracking, and low-level control quality — offering a more informative assessment than any single scalar.

Table 4.1: Training and evaluation metrics logged during policy training for OrbitXY.

Metric	What it measures	What it catches
Success rate	% of training episodes where agent completed $n_{\text{orbits}} = 1$ within radial tolerance	Primary task performance — is the agent actually solving the orbit?
Eval success rate	Same metric computed on validation episodes.	Overfitting check — if train success is high but eval is low, the policy memorised, not generalised
Episode reward	Mean total reward over done episodes	Training progress — upward trend means the policy is improving
Eval episode reward	Same on validation episodes	Generalisation of reward — should track training reward; divergence signals overfitting
Distance error	Mean $\lvert r - r_d \rvert$ at episode end	Geometric accuracy — how close to the desired orbit radius the agent settles
n_{orbits} completed	$\sum \dot{\theta} \cdot \Delta t \,/\, (2\pi)$ at episode end	How far the agent travelled around the orbit — <1 means it never finished, >1 means it lapped
Speed error	Mean $\lvert v_t - v_d \rvert$ over done episodes	Velocity tracking — how well the agent maintains the desired orbit speed
Smoothness	Mean $\lVert a_t - a_{t-1} \rVert^2$ over done episodes	Action jitter — high values mean the policy oscillates, which would damage physical thrusters or cause motion sickness

To contextualize the metrics reported above, the table below summarizes the Gazebo evaluation grid and termination criteria for the OrbitXY skill. These limits define the operational envelope within which the success rates and tracking errors were measured.

Table 4.2: Scenario parameters and evaluation limits for the OrbitXY skill.

Parameter	Value	Description
Orbit radii evaluated	5, 7, 10, 15 m	Grid conditions spanning the training distribution
Target speeds	0.5, 1.0, 1.5, 2.0 m/s	Linear velocity setpoints tested in Gazebo
Orbit directions	CW (-1), CCW ($+1$)	Both directions evaluated for symmetry validation
World bounds	$\pm$50 m (x, y)	Expanded bounds to accommodate 15–20 m radii
Max episode steps	1200	Hard limit at 5 Hz inference ($\approx$60 s real-time)
Radial tolerance band	1.5 m	Allowed deviation from desired orbit radius
Completion criterion	$\geq$1.0 orbit $\pm$0.1 tolerance	Success requires completing the angular distance
Far tolerance (termination)	50.0 m from center	Out-of-bounds threshold

Table 4.3: Training and evaluation metrics logged during FollowPath policy training.

Metric	What it measures	What it catches
Success rate	% of training episodes where agent reached path end ($s \geq 0.95$) within lateral tolerance $\lvert e_{\text{lat}} \rvert < 1.5$m	Primary task performance — is the agent actually tracking the path end-to-end?
Eval success rate	Same metric computed on validation episodes.	Overfitting check — if train success is high but eval is low, the policy memorised, not generalised
Path progress	Mean s at episode end (0 = start, 1 = completion)	How far the agent travelled along the path — < 1 means it never finished; > 1 (rare) means overshoot past termination
Lateral error	Mean $\lvert e_{\text{lat}} \rvert$ over done episodes	Tracking accuracy — how close the agent stays to the path centerline
Step count	Mean steps per episode (max 1600)	Efficiency — fewer steps at constant speed means straighter tracking; more steps means hesitation or detours
Heading error (carrot)	Mean $\lvert e^c_\psi \rvert$ over done episodes	Anticipatory steering — how well the agent aligns with the carrot point ahead rather than reacting to the closest point
Smoothness	Mean $\lVert a_t - a_{t-1} \rVert^2$ over done episodes	Action jitter — high values mean oscillatory thrust commands that waste energy and stress actuators

Table 4.4: Scenario parameters and evaluation limits for the FollowPath skill.

Parameter	Value	Description
Curvature stages	0 (0–10°), 1 (15–30°), 2 (30–60°), 3 (60–120°)	Progressive difficulty from straight to sharp turns
Path lengths	20 m, 30 m	Short and medium paths evaluated
Target speeds	0.5, 1.0, 1.5 m/s	Linear velocity setpoints tested in Gazebo
World bounds	$\pm$100 m (x, y)	Expanded bounds for long paths
Max episode steps	1600	Hard limit at 5 Hz inference ($\approx$80 s real-time)
Completion criterion	$\geq$95% path progress	Success requires reaching near the path end
Lateral tolerance band	1.0 m	Allowed deviation from reference path at completion
Lateral hard bound	5.0 m	Out-of-bounds threshold for termination

4.2 Backbone Results

Starting with OrbitXY, the learning curves are presented in order of geometric abstraction. Distance and speed errors are shown first, as they directly measure the backbone's geometric output — how accurately the policy estimates radial deviation, tangential velocity, and orbit curvature. These are the foundational motion primitives. Completed orbits and validation success rate follow, as they represent the task-level consequence of accurate geometric tracking. This ordering reflects the causal chain in the architecture: the backbone must first learn correct geometry before the downstream adapter (or the environment's done condition) can succeed. Only the main metrics used for seed selection and Gazebo transfer are shown: distance error, speed error, completed orbits, and validation success rate.

The distance-error curves in Figure 4.3 provided the first evaluated metric for the backbone. The training distance error dropped quickly to about 0.2 m, while the validation error remained consistently below 0.07 m. This behavior was expected, since the backbone was trained in a frictionless environment and learned the circle geometry through the curvature term κ, which was retained across several radii. For speed, Figure 4.4 showed a sharper early decrease followed by a noisier plateau, with the solid lines reflecting exploration noise and the dashed lines indicating validation error. After the initial drop, speed error settled in a moderate range, but its residual variance and occasional spikes indicated that speed tracking was less consistent than geometric tracking. Although OrbitXY backbone training reaches near-saturated success very early in training, validated by resemblance in the evaluation episodes, the curves show downward spikes that likely correspond to isolated failed rollouts or harder evaluation batches caused by the curriculum stages. This curriculum progressively expands the training distribution: beginning with a fixed 5 m radius and constant speed, then introducing radius randomization (5–15 m), and finally adding speed randomization (0.3–1.5 m/s) while maintaining the 50/50 CW/CCW split throughout as defined in Table 3.11.

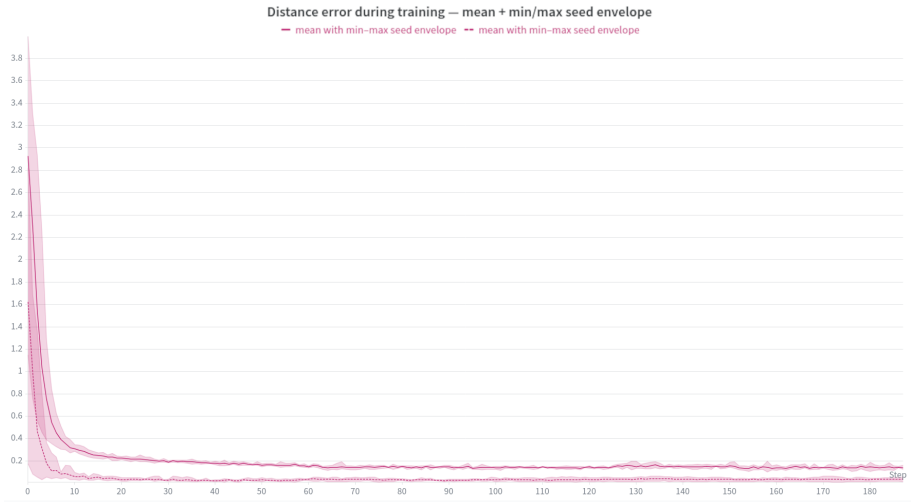


Figure 4.3: **OrbitXY Backbone - Radial distance error (m)** : The mean absolute deviation from the desired orbit radius, averaged across all 5 seeds with min/max envelope shown. The backbone converges rapidly from an initial error of 3.8 m to a steady-state value below 0.2 m within the first 20 training steps, indicating that the geometric prior for radial tracking is learned almost immediately.

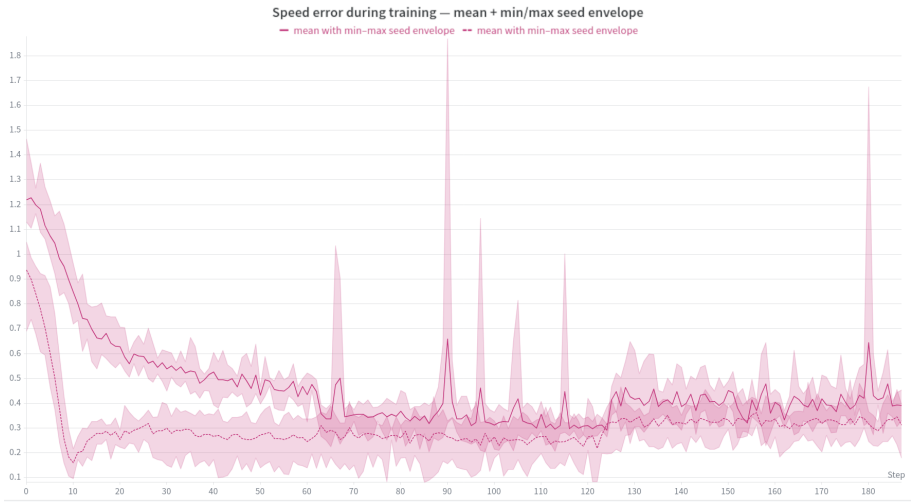


Figure 4.4: **OrbitXY Backbone - Tangential speed error (m/s)**: The mean absolute deviation from the target orbit speed, averaged across all 5 seeds with min/max envelope shown. The backbone reduces speed error from 1.2 m/s to a baseline of 0.3–0.4 m/s within the first 30 steps. Transient spikes (notably at steps 90, 100, and 120) correspond to curriculum transitions where speed randomization is introduced, temporarily increasing variance until the policy adapts to the expanded velocity distribution.

Even so, success was considered robust: all five policies achieved 100% validation success and completed, on average, 0.985–0.986 orbits. This near-unity value reflects the success criterion in VMAS: completion of $\geq$0.9 orbits ($n_{\text{orbits}} = 1.0$ with a tolerance of -0.1), meaning policies terminate precisely upon reaching the angular threshold without significant overshoot. Inference test of the backbone policy showed the intended behavior, as the agent orbited continually under the specified conditions. Accordingly, success rate was used mainly as a pass/fail gate, while

the distance-error curves in Figure 4.3 provided the meaningful comparison. Table 4.5 shows the metrics per seed adding to the previous statements.

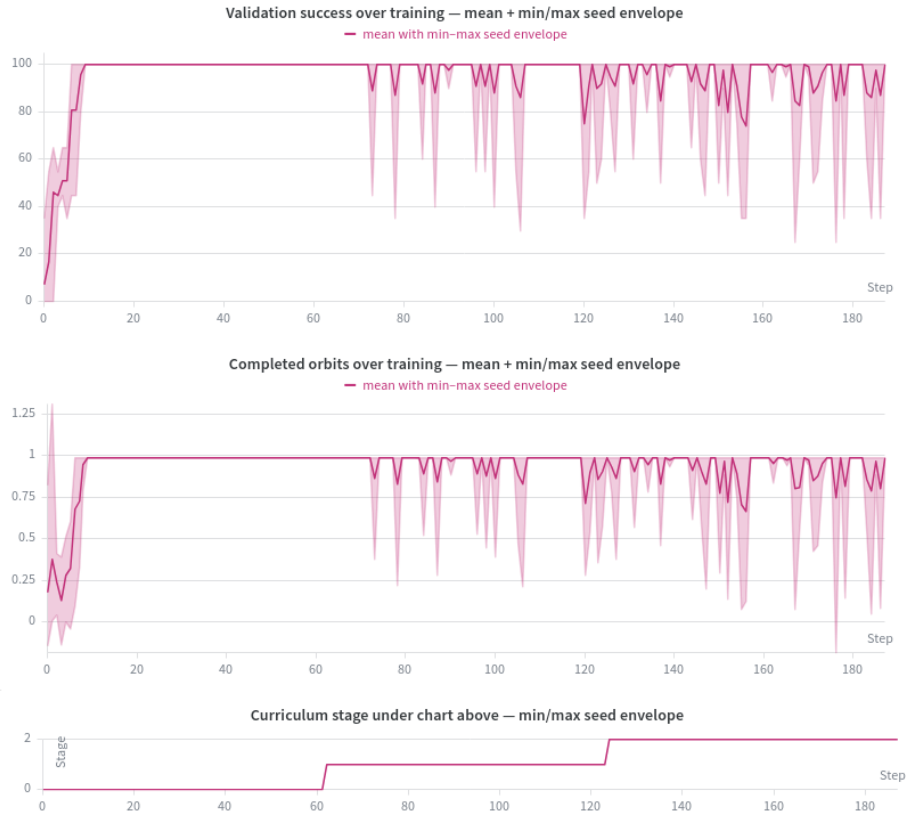


Figure 4.5: **OrbitXY Backbone - Validation Success over training in % and cumulative angular progress normalized by** 2π, averaged across 5 seeds. The backbone converges to 1.0 orbit within 10 steps, with periodic dips at curriculum transitions (stages visible on the lower panel).

The negative reward values in the backbone training log reflect the chosen reward shaping: the backbone is penalized heavily for radial deviation at every timestep ($r_{radial} = 12.0$), with only a modest completion bonus (+10.0) at episode end. In contrast, the adapter uses a higher completion bonus (+50.0) and per-step on-orbit rewards, yielding positive totals. The absolute reward sign is not meaningful across architectures — only the relative comparison between seeds and the convergence trend are relevant for selection. Seed 5 was selected as the best seed, and was used to train the adapter next, because it optimizes the complete trajectory: highest reward (-1429.1) and fewest steps (181.25). While Seed 2 achieves tighter radial tracking (0.0149 m vs 0.0238 m), Seed 5 superior speed error and action smoothness produce a cleaner, more efficient orbit.

Table 4.5: Summary of OrbitXY backbone experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better). Best value per column in **bold**.

Run	Eval success ↑	Orbits ↑	Eval reward ↑	Dist. error ↓	Speed error ↓	Best reward ↑	Best steps ↓
seed1	100%	0.9858	-4037.8	0.0410	0.3953	-1594.6	190.25
seed2	100%	0.9860	-4612.6	**0.0149**	0.3567	-1549.4	195.85
seed3	100%	**0.9862**	-4087.6	0.0568	0.3055	-1650.4	200.30
seed4	100%	0.9854	**-2824.1**	0.0410	**0.1800**	-1823.2	222.65
seed5	100%	0.9860	-4031.1	0.0238	0.3214	**-1429.1**	**181.25**
Mean ± std	100% ± 0.0	0.9859 ± 0.0003	−3918.6 ± 639.8	0.0355 ± 0.0152	0.3118 ± 0.0773	−1609.3 ± 140.7	198.06 ± 14.73



Figure 4.6: **FollowPath Backbone - Lateral error (m) and speed error (m/s)**: Mean tracking deviations across all 5 seeds with min/max envelope shown. Lateral error converges rapidly from 2 m to near-zero within the first 5 steps, indicating the backbone quickly learns to stay close to the reference path. Speed error decreases from 0.8 m/s to a steady-state of 0.2–0.4 m/s over the first 20 steps, reflecting progressive velocity tuning. Transient increases in both errors coincide with curriculum transitions (as defined in Table 3.12) to tighter turns (stages visible on the lower panel), but the policy recovers and maintains sub-metre lateral tracking throughout training.

For the FollowPath skill, the backbone must track a time-varying reference curve rather than

a fixed geometric primitive. The learning curves therefore begin with lateral error and heading error — these measure how accurately the frozen network estimates deviation from the Catmull-Rom spline tangent and alignment with the local path direction. Unlike the circular orbit, where curvature is constant, the path curvature varies continuously; the backbone must generalize across straight segments, gentle bends, and sharp turns. Path progress and success rate follow as the task-level outcome: does the agent advance along the spline and reach the terminal region? Only the metrics used for seed selection and Gazebo transfer are shown: lateral error, heading error, path progress, and validation success rate.



Figure 4.7: **FollowPath Backbone - Validation success rate (%) and path progress (dimensionless)**: Mean across 5 seeds with min/max envelope. Success climbs from 20% to 100% by step 15 under stage-1 curriculum, while progress plateaus near 0.95. Both metrics degrade transiently at stages 2 and 3 (steps 30 and 45) where tighter turns challenge tracking before partial recovery.

Before analyzing the results it is important to state that FollowPath curriculum advances

through four stages of increasing turn difficulty: straight-line and gentle curves (0–10°), moderate turns (15–30°), sharp bends (30–60°), and hairpin turns (60–120°), with the agent facing progressively tighter path geometry as training proceeds, as defined in Table 3.12. The curves in Figure 4.6 expose a convergence pattern and small late-training oscillations, providing a more faithful view of run stability. These figures show that lateral error is driven rapidly toward zero after the first training stages and then remains tightly bounded, with only minor residual oscillations. Speed error decreases gradually and remains visibly noisier for longer, which indicates that speed tracking is the harder part of the task and converges less cleanly than lateral positioning.

Figure 4.7 summarizes the backbone learning outcome by showing that validation success and path progress both increase rapidly during the early part of training and then reach a stable plateau across seeds.

The drop in path progress and success rate despite stable lateral and speed errors reflects a local minimum in the reward landscape that is only made evident for the hardest phase of the learning curriculum. Lateral error measures instantaneous proximity to the reference path — the agent remains close to the curve even when oscillating at a turn entry or moving slowly. This satisfies the per-step reward (small lateral deviation) but traps the policy in a globally suboptimal behavior: it tracks the tangent accurately yet lacks the turning authority or speed modulation to execute the corner, causing timeout before reaching the terminal region. Thus, the geometric prior is locally correct but fails to compose into full-path traversal under tight curvature constraints.

Although the existence of these late-training fluctuations remains in the filtered envelopes, the final performance stays consistently high around the 80% mark.

Table 4.6 summarized the per-seed values, which shows that the best candidate is seed 2, since only one seed reaches 100% validation success while also achieving the strongest path progression, making it the best candidate under the metric-gate analysis.

Table 4.6: Summary of FollowPath backbone experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better, ≈ 0 closer to zero is better). Best value per column in **bold**.

Run	Eval success ↑	Path progress ↑	Eval reward ↑	Lat. error ≈ 0	Speed error ↓	Smoothness ↓	Best steps ↓
seed1	85.0%	0.8198	-2.584	0.0084	0.2003	0.2213	67.35
seed2	**100.0%**	**0.9550**	**-0.295**	0.0086	**0.1673**	**0.0082**	71.30
seed3	55.0%	0.6417	-10.474	-0.0584	0.2647	0.2559	76.70
seed4	85.0%	0.8175	-7.786	-0.0069	0.1802	0.3871	69.55
seed5	90.0%	0.8789	-4.088	**0.0397**	0.1818	**0.0101**	**67.45**

From Table 4.6, the selected seed to train the adapter is seed 2, because it's the only seed to reach 100% success and tops on path progress, reward speed error and smoothness.

4.3 Adapter Results

The adapter training results are presented next, having been trained on the best seeds, which had previously been identified for each skill. As with the backbone analysis, first skill to be analyzed is OrbitXY. Based on the summary in Table 4.5, experiment seed 5 is selected as the strongest overall candidate because it combines the lowest distance error with the best reward and shortest episode length, while experiment seed 4 provides the best speed-tracking performance. As in the backbone analysis, validation success over training is considered first, followed by the evolution of distance error over training.

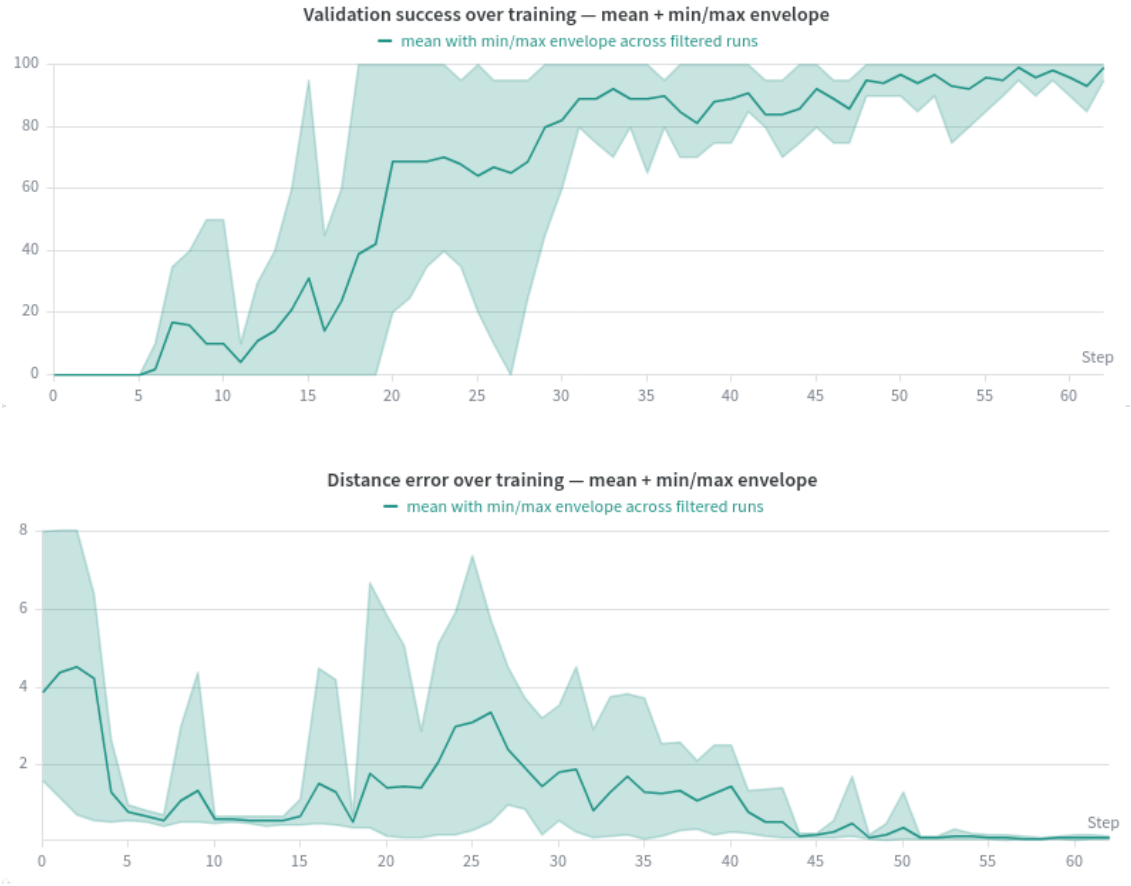


Figure 4.8: **OrbitXY Adapter - Validation success (%) and distance error (m)**: Mean across 5 seeds with min/max envelope. Success climbs from 0% to 70% by step 20, reaching 90% by step 60 as the adapter learns the dynamics mapping. Distance error drops from 8 m to 1 m within the first 10 steps but exhibits high variance (envelope up to 7 m) throughout training, reflecting the exploration noise inherent in learning thruster dynamics from geometric commands.An additional contribution is the varying difficulty across the evaluation grid: extreme combinations such as 15 m radius at 1.5 m/s or 5 m radius at 0.5 m/s challenge the policy more than nominal conditions, widening the seed envelope.

It is possible to observe in Figure 4.8 that the adapter gets the learned geometry from the backbone, and after a predominantly noisy training, the adapter converges for all 5 seeds. Both

evaluation success rate and completed orbits converge to their intended final values.

This figure reveals a fundamental difference between backbone and adapter training. While the backbone converges to near-perfect geometry within 10 steps (Figure 4.3), the adapter requires 60 steps to reach comparable success rates. This gap reflects the adapter's harder learning problem: it must map the frozen geometric output $[v_r, v_t, \kappa]$ to platform-specific actuation under non-linear thruster dynamics, rather than learning idealized kinematics. The persistent variance in distance error — with seed envelopes spanning 1–7 m throughout training — is the signature of this exploration process: each policy update perturbs both the geometric tracking and the low-level control simultaneously, creating compound instability that the backbone never experiences. That the envelope eventually narrows to $\sim$1 m by step 60 confirms the adapter ultimately learns a robust dynamics bridge, but the slower, noisier convergence underscores why the two-stage architecture (frozen geometry + trainable dynamics) is necessary: decoupling these learning problems allows each to converge cleanly. Although the distance-error curves exhibit this substantial variability across seeds, all runs converge to a low-error regime.

Next, the speed error in Figure 4.9 is analyzed. Here, the trend of the previous metric is reversed, indicating that speed tracking follows a different learning dynamic.

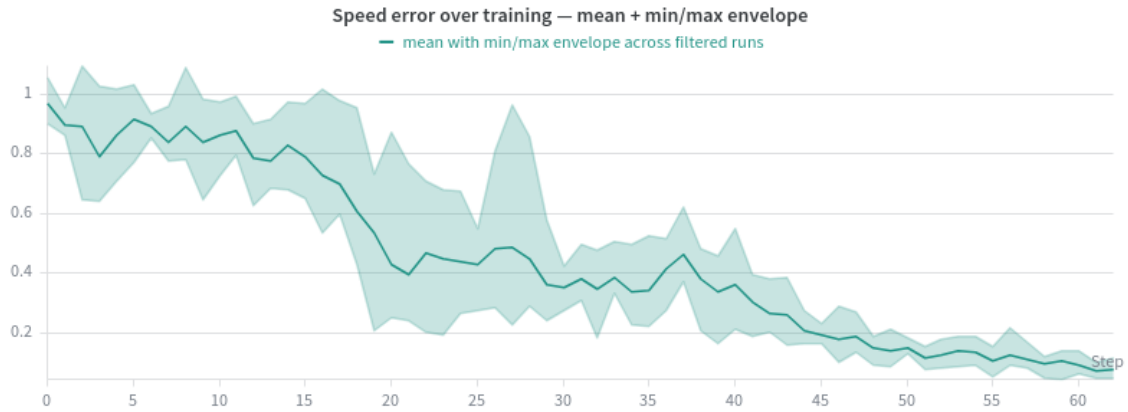


Figure 4.9: **OrbitXY - Speed error (m/s)**: Mean across 5 seeds with min/max envelope. The adapter reduces speed error from 1.0 m/s to 0.1 m/s over 60 steps, showing gradual convergence as the policy learns to modulate throttle under differential drive dynamics. The initial plateau (steps 0–15) reflects exploration noise before the dynamics mapping stabilizes; subsequent monotonic decay confirms the adapter successfully calibrates forward velocity from the frozen geometric reference.

The speed-error curve shows the same converging pattern as distance error, but with a different shape : improvement is slower, more seed-dependent, and more gradually sustained across training. This is expected because speed tracking depends not only on geometric correction but also on the policy's ability to settle the target orbit speed under exploration noise. A final upper limit on the evaluation speed error is for all five seeds less than 0.65 m/s .

A summary of the main metrics across the 5 seeds is given in Table 4.7.

Table 4.7: Summary of OrbitXY adapter experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better). Best value per column in **bold**.

Run	Eval success ↑	Orbits ↑	Eval reward ↑	Dist. error ↓	Speed error ↓	Best reward ↑	Best steps ↓
Seed 1	**100%**	0.9856	574.6	0.1153	0.0684	**677.8**	418.55
Seed 2	**100%**	0.9855	**655.1**	0.1087	0.0571	655.1	402.80
Seed 3	**100%**	0.9855	622.4	**0.0864**	**0.0497**	635.0	387.50
Seed 4	95%	**1.3196**	168.7	0.1531	0.1180	622.0	432.95
Seed 5	**100%**	0.9860	380.7	0.1573	0.1061	562.2	**385.70**
Mean ± std	99.0% ± 2.2	1.0524 ± 0.1493	480.3 ± 204.1	0.1241 ± 0.0303	0.0798 ± 0.0304	630.4 ± 43.6	405.50 ± 20.29

The positive reward values in the adapter training log reflect a different reward shaping than the backbone: the adapter receives a large completion bonus (+50.0) upon finishing the orbit, which dominates the accumulated step penalties, while the backbone used a smaller completion bonus (+10.0) with heavier per-step radial penalties. The absolute sign is not meaningful across the two training stages — only the relative comparison between adapter seeds and the convergence trend are relevant. From Table 4.7, Seed 3 was selected as the OrbitXY Adapter's best seed because it achieved the lowest distance and speed errors, although being inferior in the validation reward. Although Seed 2 attained the highest evaluation reward, Seed 3 provided the most convincing overall training outcome under the adopted selection criteria.

FollowPath, Seed 2 was selected as the basis policy. The adapter training results are presented next for the corresponding backbone. In line with the backbone analysis, the relevant metrics are shown in Figure 4.10, Figure 4.11 and Figure 4.12.

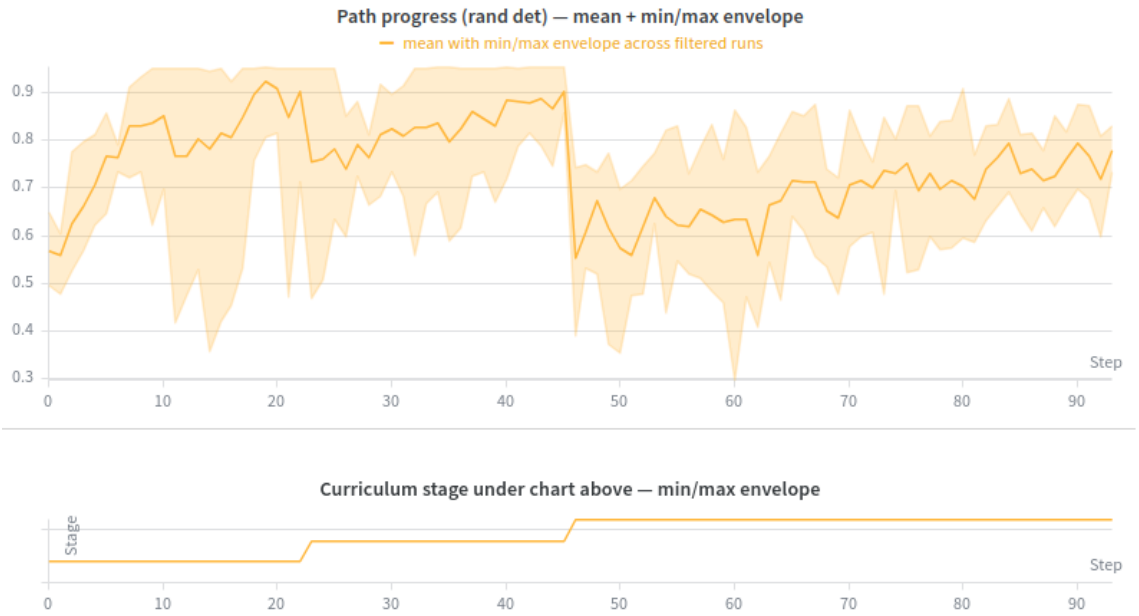


Figure 4.10: **FollowPath Adapter - Randomized deterministic path progress (dimensionless)** : Mean arc-length fraction across 5 seeds. Deterministic evaluation on randomized paths shows the policy's true tracking capability without exploration artifacts. Progress peaks at 0.95 under stage 1, then drops to 0.6–0.7 at stages 2–3 where tight turns stall the agent before path completion.

Table 4.8: Summary of FollowPath adapter experiments. Arrows indicate optimization direction (↑ higher is better, ↓ lower is better, ≈ 0 closer to zero is better). Best value per column in **bold**.

Run	Eval success ↑	Path progress ↑	Eval reward ↑	Lat. error ≈ 0	Speed error ↓	Smoothness ↓	Best steps ↓
Seed 1	85.0%	0.8198	-2.584	0.0084	0.2003	0.2213	**67.35**
Seed 2	**100.0%**	**0.9550**	**-0.295**	0.0086	**0.1673**	**0.0082**	71.30
Seed 3	55.0%	0.6417	-10.474	-0.0584	0.2647	0.2559	76.70
Seed 4	85.0%	0.8175	-7.786	**-0.0069**	0.1802	0.3871	69.55
Seed 5	90.0%	0.8789	-4.088	0.0397	0.1818	0.0101	67.45

Validation success reaches 100%, while mean path progress peaks at about 85% and stays noisy because the randomized evaluation mixes different path lengths, curvatures, and speeds; the step-45 drop matches the stage-3 transition to tighter turns. It is also important to note that the FollowPath adapter curriculum advances from stage 1 (15–30° turns) through progressively tighter path geometry: stage 2 (30–60° bends) at 150k steps and stage 3 (60–120° hairpins) at 300k steps. The lateral error is reduced quickly to near zero and remains tightly bounded thereafter, showing that the adapter learns effective path alignment. Minor fluctuations appear around curriculum changes, but the long-term trend indicates clear convergence and stable tracking.

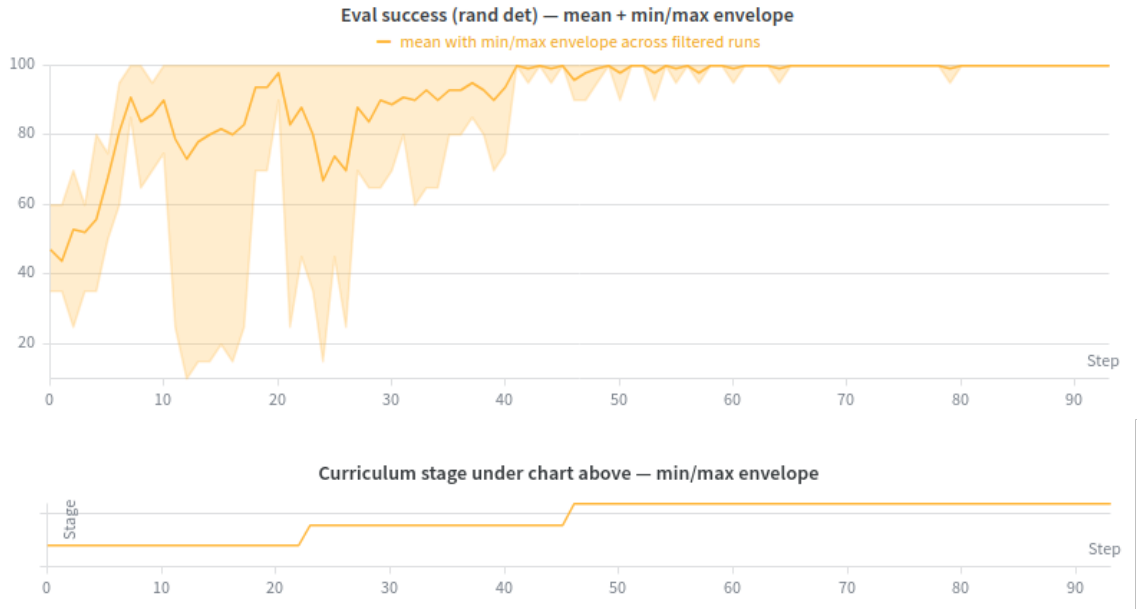


Figure 4.11: **FollowPath Adapter - Randomized deterministic validation success (%)**: Mean across 5 seeds with min/max envelope. The deterministic policy (no exploration noise) is evaluated on randomized paths every N training iterations. Success climbs from 40% to near 100% by step 45 under stage-1 curriculum. Transient drops at curriculum transitions (stages 2 and 3) reflect the harder test conditions, not exploration noise.

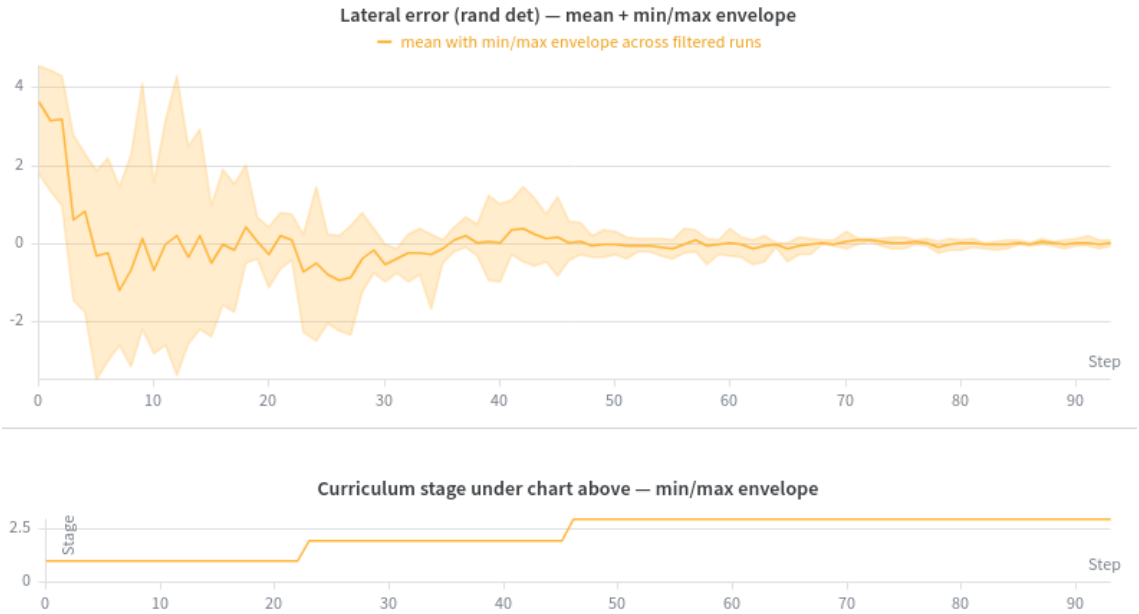


Figure 4.12: **FollowPath - Randomized deterministic lateral error (m)**: Mean signed deviation across 5 seeds. Deterministic evaluation reveals the policy's geometric tracking without exploration noise. Convergence to near-zero error under stage 1 confirms the backbone extracts a clean Frenet prior; residual variance at later stages reflects the intrinsic difficulty of sharp turns, not policy instability.

Table 4.8 presents the information in a fashion that is easier to select the best seed used for the Gazebo transfer. Here Seed 2 is a clear best , and it will be used to proceed with the Gazebo transfer.

4.4 Transfer Results

Once the best seeds for the backbone and adapter had been identified, transfer from VMAS to Gazebo was carried out to obtain a more realistic representation of USV simulation. The resulting policy was then evaluated over a grid of directions, radii, and speeds in order to measure generalization beyond the training conditions.

The results in Table 4.9 show that the OrbitXY adapter transfers robustly to Gazebo across most of the tested operating conditions. Success remains at 100 % for the smaller radii and moderate speeds, and the tracking errors stay low in the central region of the grid, which indicates that the backbone-adapter decomposition generalizes well beyond the training setup. The adapter struggles at low speed and large radius (not high speed), where the agent cannot complete the full orbit within the 1200-step episode limit. At 0.5 m/s, the 20 m orbit circumference (125 m) requires 250 s, exceeding the maximum episode duration. This is a timeout issue, not a tracking failure — distance error at termination remains low (0.3 m). Higher speeds (1.0–2.0 m/s) achieve 100% success at all radii.

Table 4.9: OrbitXY adapter Gazebo evaluation across radius and speed conditions. Success rate averaged over CW and CCW directions. Metrics reported as mean ± std across episodes. Larger radius combined with the lowest speed produce 0% success in two cells on the grid, but this is due to timeout.

Radius (m)	Speed (m/s)	Success (%)	Dist. error (m)	Head. error (rad)	Orbits completed
5	0.5	100	0.465 ± 0.077	0.109 ± 0.009	0.901 ± 0.001
5	1.0	100	0.738 ± 0.027	0.213 ± 0.003	0.902 ± 0.001
5	1.5	100	0.994 ± 0.001	0.243 ± 0.011	0.902 ± 0.001
5	2.0	100	1.178 ± 0.042	0.260 ± 0.009	0.902 ± 0.000
7	0.5	100	0.413 ± 0.057	0.085 ± 0.011	0.900 ± 0.000
7	1.0	100	0.631 ± 0.038	0.188 ± 0.008	0.901 ± 0.001
7	1.5	100	0.847 ± 0.008	0.235 ± 0.003	0.901 ± 0.001
7	2.0	100	0.982 ± 0.022	0.260 ± 0.002	0.902 ± 0.001
10	0.5	50	0.368 ± 0.065	0.066 ± 0.005	0.450 ± 0.636
10	1.0	100	0.512 ± 0.042	0.167 ± 0.000	0.901 ± 0.001
10	1.5	100	0.668 ± 0.006	0.214 ± 0.005	0.901 ± 0.001
10	2.0	100	0.749 ± 0.001	0.247 ± 0.001	0.902 ± 0.001
15	0.5	0	0.328 ± 0.056	0.056 ± 0.001	0.000
15	1.0	100	0.369 ± 0.040	0.137 ± 0.006	0.901 ± 0.000
15	1.5	100	0.457 ± 0.008	0.178 ± 0.003	0.901 ± 0.001
15	2.0	100	0.477 ± 0.015	0.209 ± 0.001	0.901 ± 0.001
20	0.5	0	0.297 ± 0.051	0.041 ± 0.001	0.000
20	1.0	100	0.275 ± 0.022	0.112 ± 0.002	0.901 ± 0.001
20	1.5	100	0.309 ± 0.002	0.149 ± 0.007	0.901 ± 0.000
20	2.0	100	0.300 ± 0.019	0.171 ± 0.002	0.901 ± 0.000
Overall	–	**80%**	$\mathbf{0.568 \pm 0.242}$	$\mathbf{0.170 \pm 0.071}$	$\mathbf{0.769 \pm 0.291}$

Although the adapter was trained and extensively evaluated in VMAS, transferability is ultimately assessed in the higher-fidelity Gazebo simulator. Accordingly, a grid evaluation is conducted for the best selected seed with 3 episodes per condition over 5 radii $\times$ 4 speeds $\times$ 2 directions, yielding 40 conditions and 120 total episodes. The resulting mean and variance estimates are reported in Table 4.9 and Figure 4.14, which is used to characterize transfer quality and identify failure modes without duplicating the replication already performed during VMAS training.

Overall, the Gazebo results confirm that the learned backbone-adapter pipeline is not confined to VMAS and can maintain stable orbit-following behavior under a more realistic USV model. The strongest performance is observed in the region where the training distribution and vehicle dynamics are most compatible, whereas the low-speed, large-radius cases form the main failure boundary.

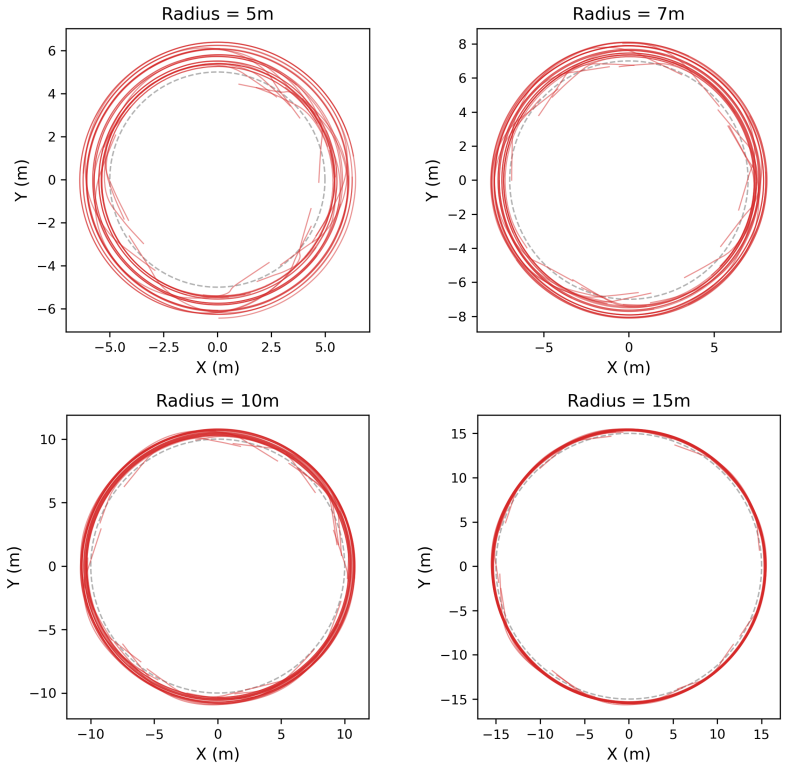


Figure 4.13: Reference orbits (dashed circles) and actual agent trajectories (solid lines) for radii of 5 m, 7 m, 10 m, and 15 m. Each solid line represents one episode executed by the policy in Gazebo.

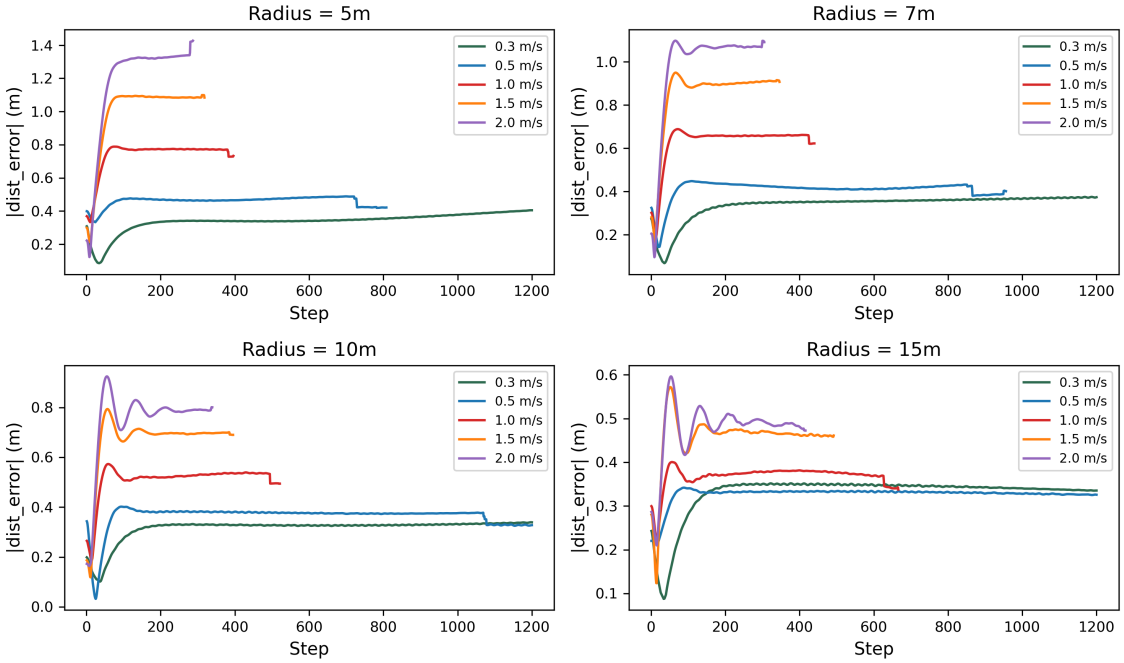


Figure 4.14: **OrbitXY Adapter on Gazebo - Distance error (m) over episode steps** : Per-step radial deviation for OrbitXY adapter in Gazebo across four radii and five speeds. Curves terminate early at higher speeds (notably 2.0 m/s) because the agent completes the orbit faster.

Figure 4.14 indicates that the adapter drives the distance error down rapidly in all cases, but the final residual error depends on both speed and radius. The smaller-radius cases show larger overshoot and fluctuation, while the 15 m orbits settle more smoothly and remain closer to the target, suggesting improved transfer in less curvature-intensive conditions. The gaps at 2.0 m/s (where curves terminate before step 1200) are not missing data or failures — they reflect faster task completion. At 2.0 m/s, the agent traverses the required angular distance (2π radians) in significantly fewer steps than at 0.3 m/s. The episode terminates upon completing one full orbit, which the policy achieves in roughly 150–350 steps at 2.0 m/s versus 800–1200 steps at 0.3 m/s. This early termination is the desired behavior: the agent solves the task faster, not worse. The flat line at the terminal step indicates the final steady-state tracking error at the moment of success. Additionally for completeness, a trajectory map is presented in Figure 4.13 from the generalization test and an inference test performed on Figure4.15.

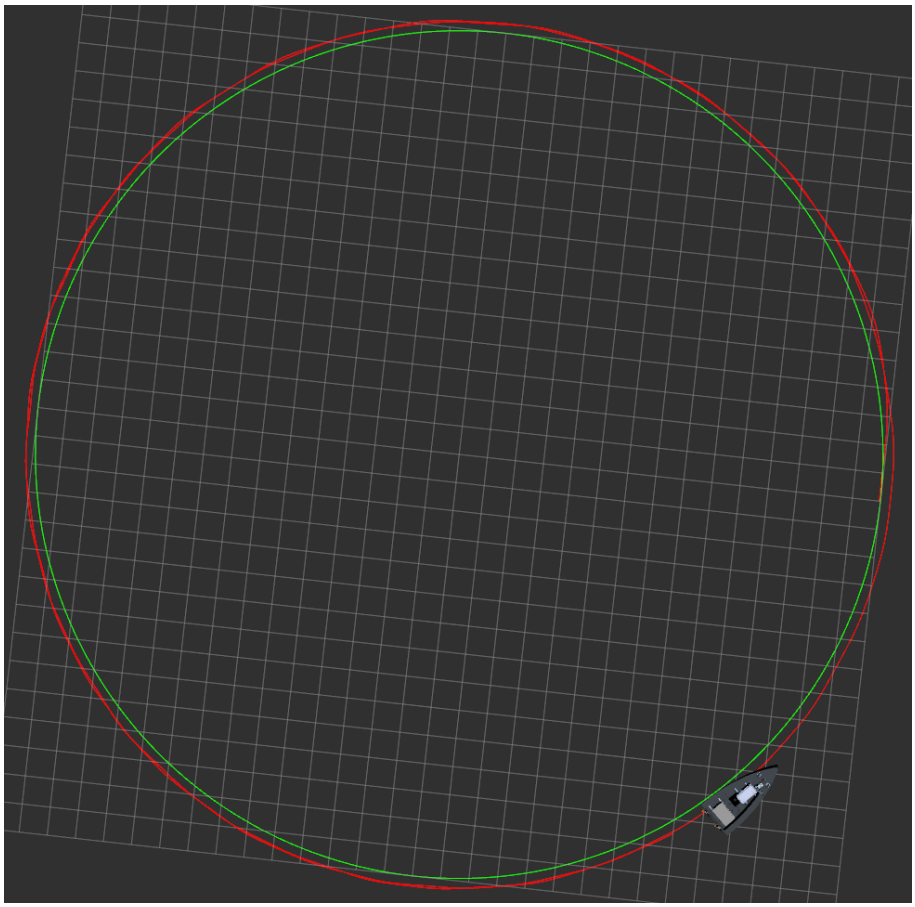
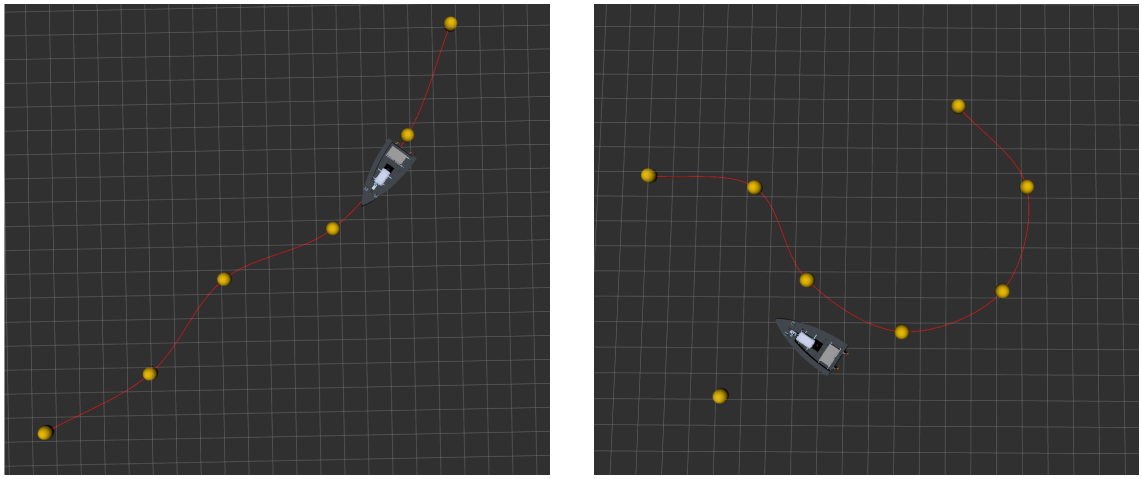


Figure 4.15: RViz visualization of an inference test for OrbitXY.It is possible to observe the constant orbit done colored red, and the target orbit in green.

For FollowPath, the Gazebo evaluation grid spans 4 curvature stages (0–10°, 15–30°, 30–60°, 60–120°) × 3 target speeds (0.5–1.5 m/s) × 2 path lengths (20–30 m), yielding 24 conditions. Each condition was tested with 3 episodes to capture mean performance and variance under Gazebo's simulation. This design balances statistical representativeness against practical runtime constraints: three episodes per condition provide confidence bounds without requiring the

extensive replication already performed in VMAS training. Metrics recorded per episode include success rate, mean lateral error, heading error, and path completion progress.



(a) Randomly generated path 1.

(b) Randomly generated path 2.

Figure 4.16: Two examples of randomly generated paths for the Gazebo simulation for two different curriculum stages.

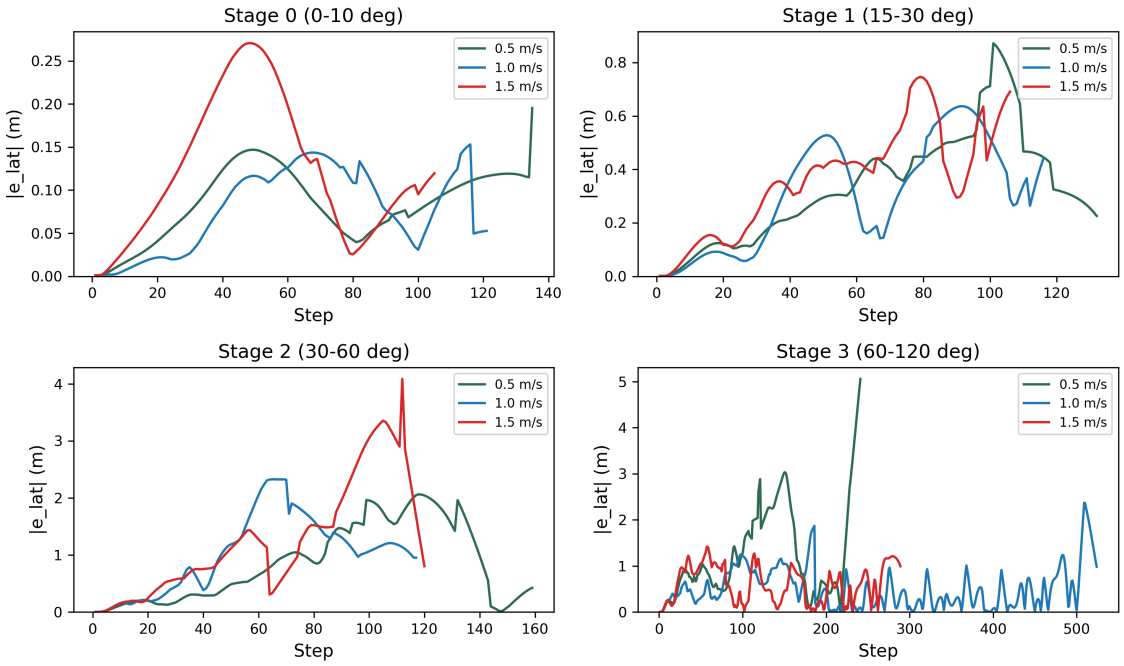


Figure 4.17: FollowPath lateral error for the grid evaluation conditions.

During Gazebo transfer evaluation, the trained backbone and adapter are exposed to a realistic environment across a systematic grid of task conditions. Figure 4.17 shows the mean error per step. The truncated curves in Figure 4.17 are not plotting artifacts — they reflect variable episode length across the evaluation grid. Faster target speeds (1.5 m/s, red curves) complete the path in fewer steps than slower speeds (0.5 m/s, green), producing shorter traces. At Stage 3 (bottom right), some curves terminate abruptly around step 100–200; this indicates episode termination due

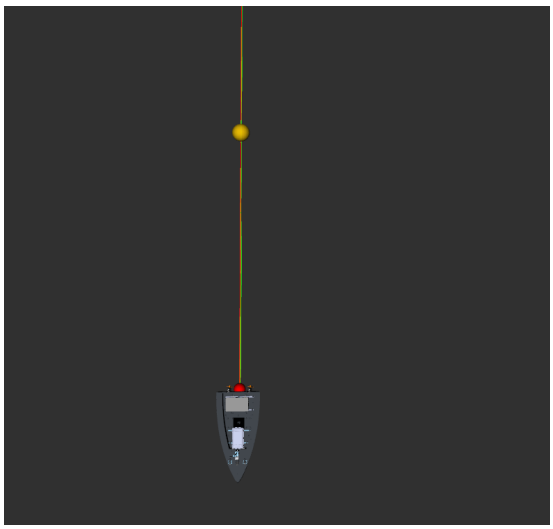
to exceeding the 5.0 m lateral bound or Frenet jump detection on the sharpest turns, not missing data. The x-axis extent differs per subplot because episode duration scales with path length, speed, and curvature.

Table 4.10: FollowPath adapter Gazebo evaluation across curriculum stages, speeds and path lengths. Metrics reported as mean ± std across 3 episodes. Partial success conditions shown with exact rate.

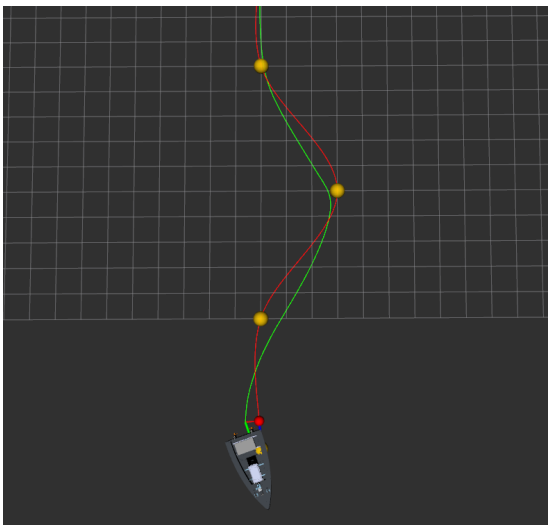
Stage	Speed (m/s)	Path (m)	Success	Dist. error (m)	Head. error (rad)	Speed error (m/s)	Progress
0	0.5	20	100%	0.083 ± 0.008	0.023 ± 0.006	0.295	0.954
0	0.5	30	100%	0.079 ± 0.027	0.032 ± 0.004	0.295	0.954
0	1.0	20	100%	0.065 ± 0.016	0.026 ± 0.008	0.063	0.952
0	1.0	30	100%	0.082 ± 0.032	0.037 ± 0.012	0.066	0.952
0	1.5	20	100%	0.145 ± 0.036	0.025 ± 0.009	0.437	0.956
0	1.5	30	100%	0.125 ± 0.035	0.031 ± 0.006	0.425	0.951
Stage 0 mean	–	–	100%	0.097 ± 0.031	0.029 ± 0.005	0.264 ± 0.170	0.953 ± 0.002
1	0.5	20	100%	0.260 ± 0.023	0.125 ± 0.019	0.213	0.958
1	0.5	30	100%	0.348 ± 0.147	0.180 ± 0.067	0.435	0.971
1	1.0	20	100%	0.231 ± 0.022	0.122 ± 0.035	0.190	0.956
1	1.0	30	100%	0.308 ± 0.034	0.125 ± 0.019	0.175	0.954
1	1.5	20	100%	0.324 ± 0.142	0.164 ± 0.046	0.666	0.972
1	1.5	30	100%	0.321 ± 0.064	0.110 ± 0.022	0.538	0.954
Stage 1 mean	–	–	100%	0.299 ± 0.042	0.138 ± 0.028	0.370 ± 0.198	0.961 ± 0.008
2	0.5	20	100%	0.338 ± 0.034	0.294 ± 0.088	0.253	0.965
2	0.5	30	100%	1.022 ± 0.945	0.309 ± 0.118	0.437	0.970
2	1.0	20	100%	1.078 ± 0.268	0.556 ± 0.128	1.050	1.000
2	1.0	30	67%	0.929 ± 0.422	0.293 ± 0.065	0.661	0.838
2	1.5	20	67%	0.915 ± 0.684	0.247 ± 0.063	1.274	0.874
2	1.5	30	67%	1.106 ± 0.355	0.342 ± 0.031	1.230	0.937
Stage 2 mean	–	–	83%	0.898 ± 0.285	0.340 ± 0.109	0.818 ± 0.394	0.931 ± 0.058
3	0.5	20	67%	0.877 ± 0.363	0.766 ± 0.205	0.622	0.841
3	0.5	30	67%	1.177 ± 0.058	0.632 ± 0.038	0.884	0.940
3	1.0	20	100%	0.462 ± 0.100	0.972 ± 0.307	0.996	0.987
3	1.0	30	67%	0.849 ± 0.414	0.661 ± 0.049	0.974	0.984
3	1.5	20	100%	0.556 ± 0.183	0.719 ± 0.205	1.358	0.963
3	1.5	30	100%	1.027 ± 0.673	0.635 ± 0.135	1.236	0.991
Stage 3 mean	–	–	83%	0.825 ± 0.249	0.731 ± 0.131	1.012 ± 0.259	0.951 ± 0.058
Overall	–	–	**92%**	$\mathbf{0.530 \pm 0.372}$	$\mathbf{0.310 \pm 0.262}$	$\mathbf{0.616 \pm 0.441}$	$\mathbf{0.949 \pm 0.042}$

The combined backbone-adapter policy remains effective on the easier curriculum stages, where lateral error is relatively small and bounded, but performance degrades as curvature and speed increase, particularly in the later stages. This suggests that the basic Frenet geometry of FollowPath is captured, while sharper turns still reveal limitations in the current representation and controller coupling. For the FollowPath adapter, the curriculum advances from stage 1 (15–30° turns) through progressively tighter path geometry: stage 2 (30–60° bends) is introduced at 150,000 training steps, followed by stage 3 (60–120° hairpins) at 300,000 steps. In particular, the first two stages suggest stable tracking on straight and gently curved paths, whereas Stage 2 already shows large transient errors at higher speeds, and Stage 3 exhibits the most pronounced instability with sustained oscillations and sharp error spikes. This supports the interpretation that the curriculum successfully increases task difficulty, but also that the learned backbone is not yet uniformly robust across all curvature regimes, which is precisely where the adapter and the carrot-based turn-anticipation mechanism become important. The data for each evaluation run

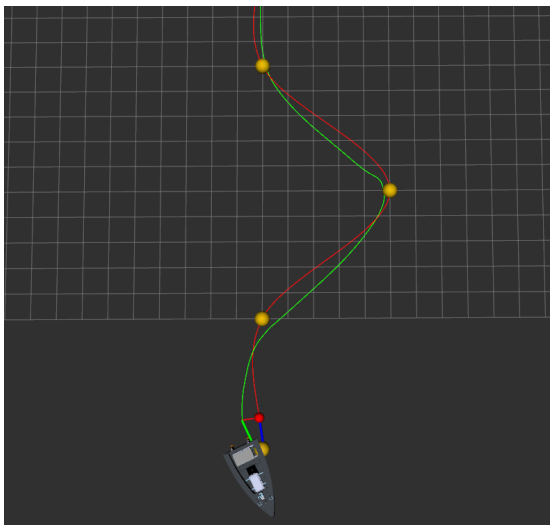
are also presented. Each grid cell corresponds to a unique combination of curriculum difficulty, target speed, and path geometry, providing a qualitative sanity check of the transfer to a higher fidelity simulation generalization beyond the statistical robustness already established in VMAS. The evaluation results are shown in Table 4.10.



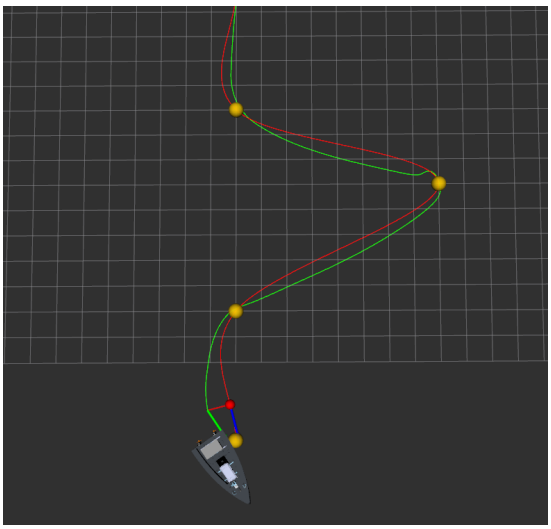
(a) Randomly generated path 1.



(b) Randomly generated path 2.



(c) Randomly generated path 3.



(d) Randomly generated path 4.

Figure 4.18: Four additional examples of randomly generated paths for the Gazebo simulation. The red line represents the intended path and the green line represents the actual path driven by the agent

The results indicate robust performance across most grid conditions, with 100% success maintained throughout curriculum stages 0 and 1 and in several stage-3 settings as well. Tracking error remains comparatively bounded at the lower and mid-level stages, but it increases noticeably once stage 2 is reached, especially for the higher-speed and longer-path combinations. The most demanding stage-2 cases are the least reliable, with one condition dropping to 67% success and others exhibiting large lateral, heading, and speed errors. Overall, the table suggests that the frozen

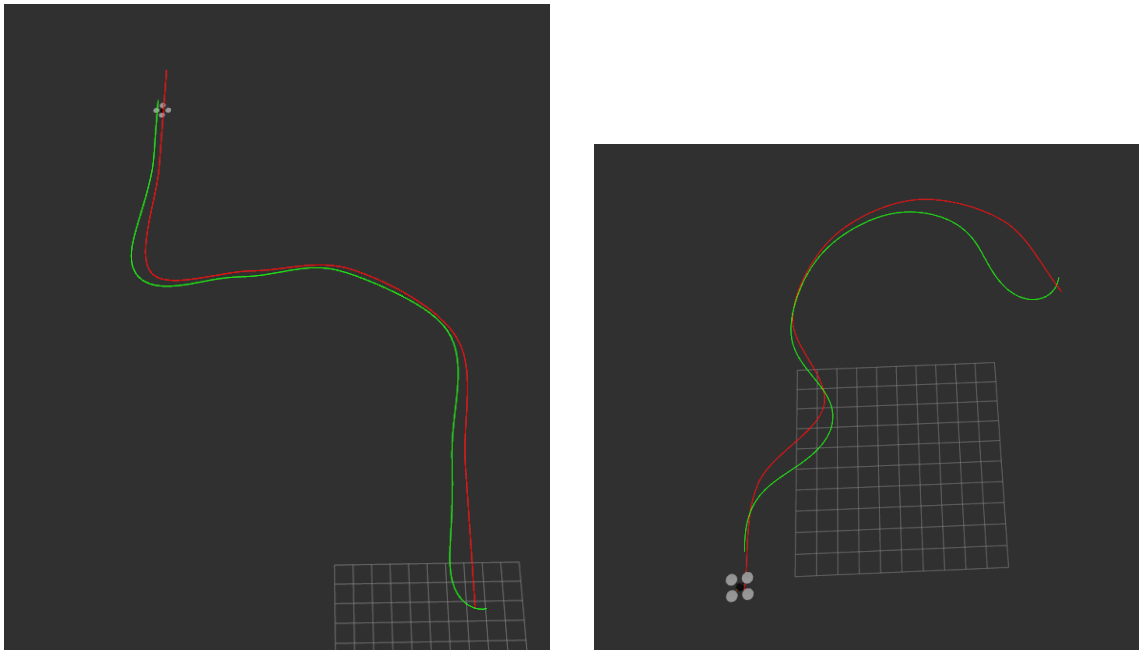
backbone transfers effectively through the adapter to Gazebo dynamics, but robustness is reduced in the hardest path and speed regimes.

For a qualitative assessment, an image for each path representing each curriculum stage is shown in Figure 4.18

4.5 Backbone zero-shot

As shown previously, the backbone provides a useful geometric prior, even if a monolithic policy can outperform it on a single platform. To test whether this abstraction transfers zero-shot to a different holonomic vehicle, the backbone outputs were analytically mapped to `cmd_vel` commands and evaluated on the Hector quadrotor, a lightweight ROS/Gazebo testbed for demonstrating cross-platform reuse. For OrbitXY, the polar backbone output $[v_r, v_t, \kappa]$ — where v_r is radial correction velocity, v_t is tangential (orbit) velocity, and κ is path curvature (1/m) — is converted into planar velocity commands and yaw rate ω (rad/s) via $\omega = d \cdot \kappa \cdot v_t$, with $d = \pm 1$ denoting orbit direction (counter-clockwise or clockwise). For FollowPath, the path-aligned output $[v_{\text{lat}}, v_{\text{fwd}}, \kappa]$ — where v_{lat} is lateral correction velocity and v_{fwd} is forward velocity — is reduced to v_{fwd} and $\omega = \kappa \cdot v_{\text{fwd}}$, with v_{lat} discarded because the drone's `Twist` interface is forward/yaw based. This makes OrbitXY more directly transferable to the quadrotor than FollowPath, showing that geometric abstraction should match the target actuator interface as well as the training dynamics.

Drone zero-shot evaluation was performed only on FollowPath, since it is the harder transfer case and is sufficient as a proof of concept. The same evaluation grid was run for the performance evaluation, and inference started at a fixed height after a takeoff procedure, thus executing planar movements. Two randomly generated trajectories for the drone can be seen in Figure 4.19



(a) Randomly generated path 1.

(b) Randomly generated path 1.

Figure 4.19: Two examples of randomly generated paths for the Gazebo simulation for the Hector drone. Red line is the intended path , green line is the actual drone trajectory.

The same evaluation grid was executed for the Hector drone. The lateral error per stage is presented in Figure 4.20

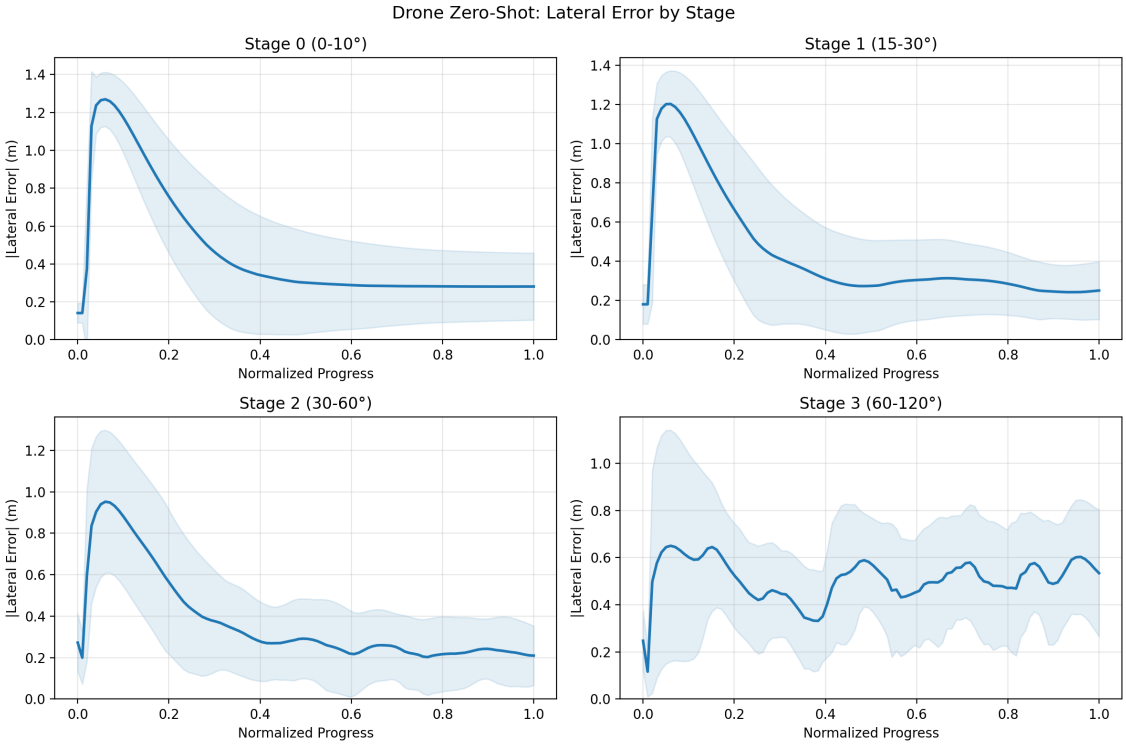


Figure 4.20: FollowPath lateral error for the grid evaluation conditions for Hector drone.

Table 4.11: FollowPath backbone zero-shot transfer to Hector drone, aggregated by curriculum stage. Metrics reported as mean ± std across all speed and path length conditions within each stage.

Stage	Success	Dist. error (m)	Head. error (rad)	Speed error (m/s)	Progress
0	100%	0.488 ± 0.112	0.205 ± 0.032	0.553 ± 0.402	0.953 ± 0.001
1	100%	0.453 ± 0.097	0.189 ± 0.030	0.609 ± 0.444	0.953 ± 0.001
2	100%	0.376 ± 0.099	0.167 ± 0.042	0.818 ± 0.529	0.953 ± 0.001
3	97%	0.481 ± 0.046	0.299 ± 0.124	0.907 ± 0.447	0.938 ± 0.110
Overall	**97%**	$\mathbf{0.449 \pm 0.098}$	$\mathbf{0.219 \pm 0.082}$	$\mathbf{0.749 \pm 0.440}$	$\mathbf{0.945 \pm 0.060}$

The FollowPath backbone training begins at stage 0 with straight-line and gentle-curve paths (0–10° turns), then advances through four progressively tighter curvature stages: stage 0 (15–30°) at 50,000 steps, stage 1 (30–60°) at 150,000 steps, and stage 2 (60–120° hairpins) at 300,000 steps, with stage 3 (fully randomized) at 450,000 steps, as specified in Table 3.12.

The FollowPath backbone was also evaluated in a zero-shot transfer scenario on the Hector drone platform, without any adapter training or fine-tuning. The backbone transfers to the Hector drone because its geometric output (v_{fwd}, κ) is kinematic and vehicle-agnostic — it makes no assumptions about USV hydrodynamics or drone aerodynamics. The drone's onboard velocity controller handles the low-level attitude mapping, allowing the backbone's path curvature commands to be realized through standard Twist messages without platform-specific retraining. The results confirm this expectation, with 97% overall success across all tested conditions.

4.6 Ablation Results

In this section, a brief ablation study is reported in which the backbone observation is removed for both skills. In the no-backbone setting, the policy receives the full raw observation directly: 9D invariant features for OrbitXY (deviation, bearing, velocities, heading error, orbit direction, and speed) and 9D Frenet features for FollowPath (lateral error, heading error, curvature, velocities, path direction, target speed, and carrot state). Without the geometric backbone, both task geometry and vehicle-specific dynamics must be learned simultaneously from raw state. The full observation specification is given in Sections 3.2.1 and 3.3.1.

For OrbitXY, this variant performed very poorly, which suggests that the backbone provides a crucial geometric prior rather than being just an auxiliary feature. Only a seed was trained, which shows convergence of the reward both in training and in the evaluation settings. Using the exact same hyperparameters used to train the adapter on our regular architecture, this adapter was without the commands from the backbone.

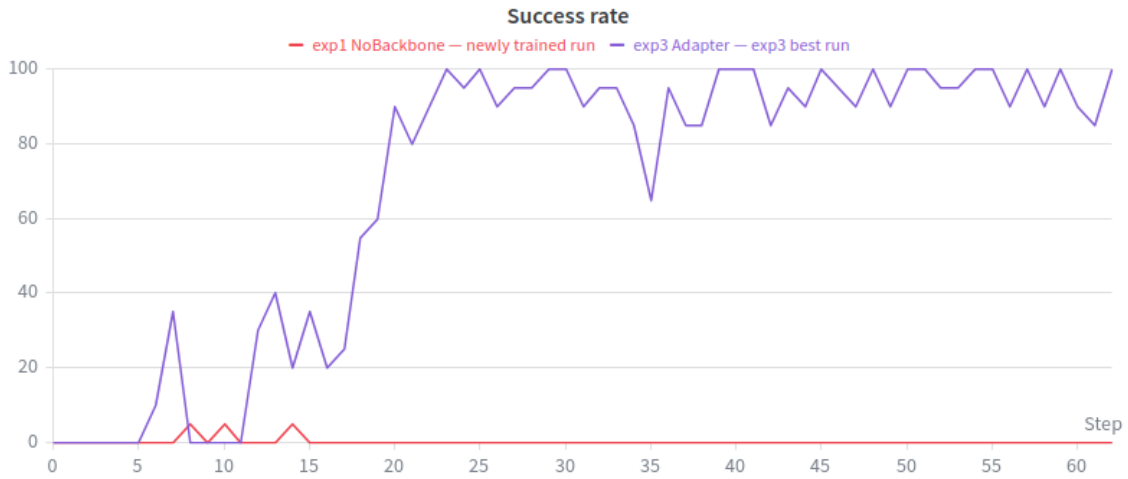


Figure 4.21: Validation metrics for OrbitXY no backbone variant training ablation.

It is evident that the gap is already clear; this variant peaks at 5% success rate at early curriculum stages. The no-backbone variant is consistently weaker, which is enough to show that the backbone is an essential geometric prior.

On the FollowPath skill it is also evident that the backbone's prior training benefits the agent's performance. Training logs in Figure 4.22 make this evident. The no-backbone ablation reaches high success early on, but its performance becomes much less stable over training, with noticeable fluctuations in lateral error and a clear drop in path progress after the curriculum advances. In contrast, the backbone+adapter run is more consistent across the full training horizon: it maintains better path progress while keeping lateral error closer to zero, which suggests that the backbone is a valuable geometric prior that helps the adapter preserve tracking quality as the task becomes harder.

Figure 4.23 complements the training logs by showing that the backbone+adapter model generally completes episodes in fewer steps than the no-backbone ablation, especially on the more difficult stages. In the gentler stages the difference is present but modest, whereas in Stage 2 and Stage 3 the no-backbone model often requires substantially more steps and becomes noticeably less efficient, reflecting its weaker ability to maintain progress while correcting deviation. Overall, the figure indicates that the backbone is not only improving tracking accuracy, but also helping the policy move more directly along the path and complete episodes more efficiently under harder conditions.

Figure 4.24 shows that the backbone+adapter variant consistently produces lower lateral error than the no-backbone model across the easier curriculum stages, particularly in Stage 0 and Stage 1.As path curvature increases, both models deteriorate, but the no-backbone ablation degrades faster and reaches substantially higher error levels, with larger oscillations and weaker stability near the end of the trajectory. This suggests that the backbone contributes a useful geometric prior that improves tracking consistency, especially when the path remains within the regime the backbone has learned to represent well.

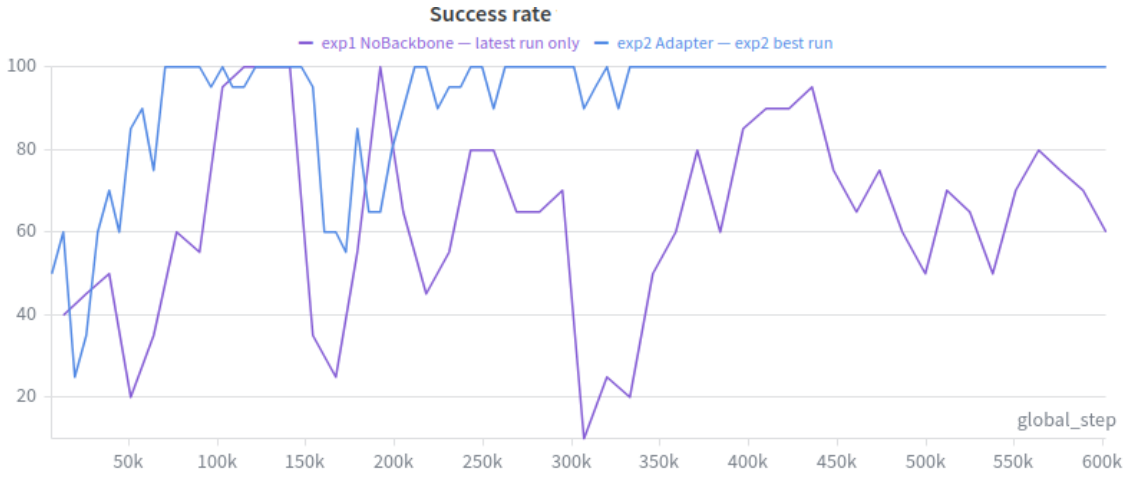


Figure 4.22: Validation metrics for no backbone training ablation, specifically success rate and lateral error.

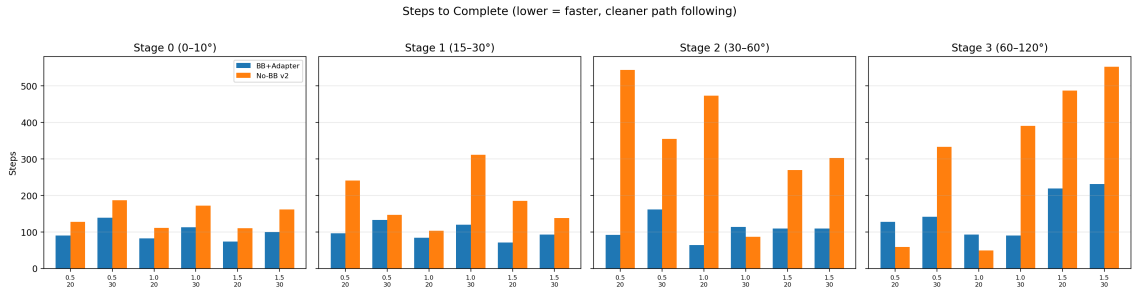


Figure 4.23: Steps to complete per stage.

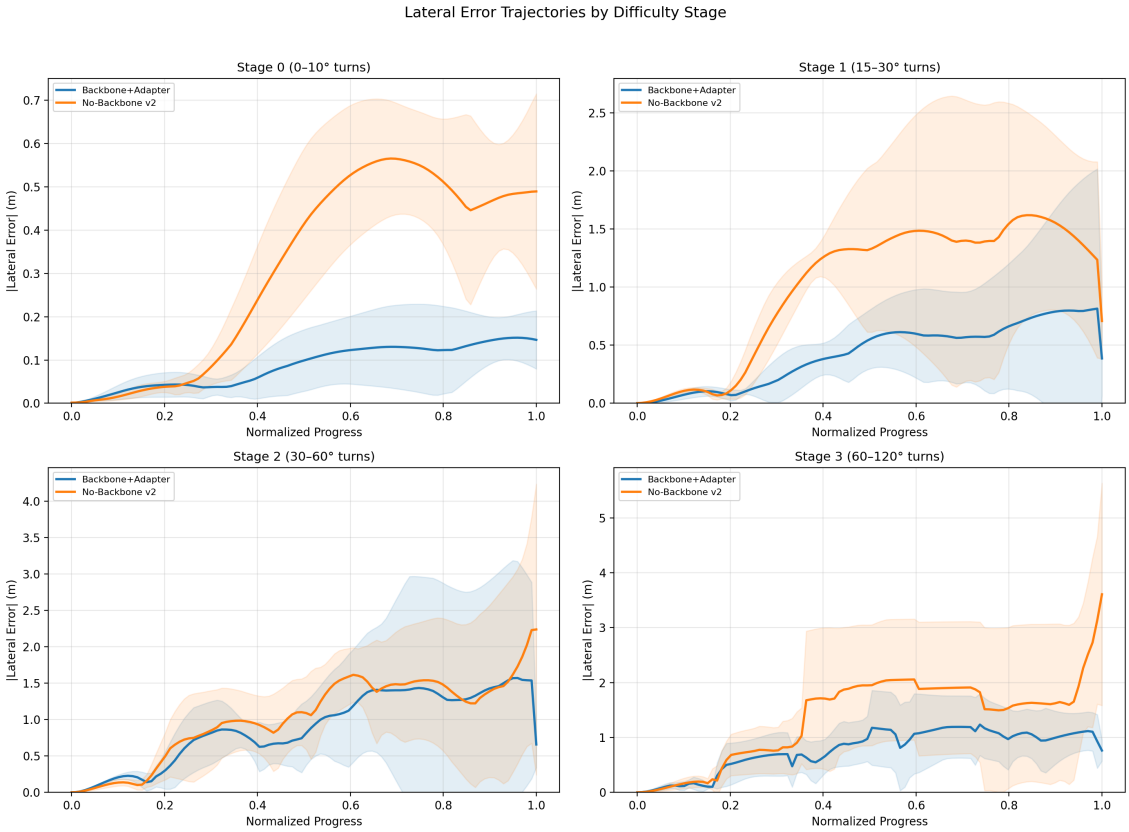


Figure 4.24: Lateral error trajectories by difficulty stage.

Chapter 5

Conclusions and Future Work

This thesis proposed a transferable reinforcement learning framework for USV navigation based on a two-stage decomposition into a frozen geometry backbone and a vehicle-specific adapter.

The results show that this factorization is not merely a structural convenience, but a useful way of separating reusable navigation geometry from platform-dependent control, which improves interpretability and reduces the amount of retraining required when moving across vehicle models.

Across the OrbitXY and FollowPath skills, the proposed architecture learned stable navigation behaviors in simplified simulation and transferred them successfully to higher-fidelity settings. In OrbitXY, the symmetry-aware backbone learned direction-invariant orbit geometry, while the adapter mapped that representation to thruster commands; in FollowPath, the backbone captured the path geometry and the adapter handled the actuation required for realistic vehicle dynamics. In Gazebo, the OrbitXY backbone-adapter policy achieved an overall success rate of 80% across the evaluated radius-speed grid, with mean distance error of 0.568 m, mean heading error of 0.170 rad, and an average of 0.769 completed orbits, while the best OrbitXY adapter seed achieved 100% validation success in simulation with low distance and speed errors. For FollowPath, the Gazebo transfer results showed 100% success in the easier curriculum stages, with low lateral and heading error, but performance degraded in the most demanding conditions, where success dropped to 67% in some stage-2 and stage-3 cases and lateral error increased noticeably, confirming that transfer remained effective but less uniform under sharper curvature and higher-speed regimes.

The experimental results also indicate that the backbone contributes a meaningful geometric prior rather than acting as a redundant intermediate layer. This is supported by the ablation studies, where the no-backbone variants were consistently less stable, and by the Gazebo transfer results, where the backbone+adapter architecture outperformed the harder conditions of the transfer grid, especially in maintaining path structure and reducing tracking error compared with direct raw-observation learning. Furthermore, the zero-shot transfer of the FollowPath backbone to the Hector drone platform — without any adapter training or fine-tuning — achieved 97% overall success across all tested conditions, providing direct evidence that the geometric representation learned by the backbone is vehicle-agnostic and transfers across platforms with fundamentally different dynamics.

At the same time, the thesis highlights a clear boundary to the method. The FollowPath formulation assumes non-intersecting trajectories because closest-point Frenet projection becomes ambiguous on self-intersections, and the transfer experiments would benefit from broader validation across more episodes and more demanding environmental conditions.

Future contributions can extend this architecture in three directions. First, platform diversity: testing transfer to other vehicle classes — such as AUVs with 6-DOF dynamics, Ackermann-steered ground robots, or omnidirectional rovers — would stress-test whether the geometric abstraction generalizes across fundamentally different actuation constraints. Second, skillset extension: beyond orbit following and path tracking, additional primitives such as station-keeping, formation control, or dynamic obstacle avoidance could be developed, with composition through a shared backbone demonstrating true behavioral modularity. Third, real-world deployment: closing the sim-to-real gap on the physical Nautilus platform requires addressing sensor noise, communication latency, and environmental disturbances (wind, current, waves) absent in Gazebo, representing the definitive validation of transferable geometric navigation.

Overall, the main contribution of this work is to show that dynamics-agnostic behavioral priors can be learned in a compact, reusable form and then adapted to a specific USV through a lightweight vehicle interface. This makes the approach a promising basis for future transfer learning systems in maritime autonomy, especially when the goal is to reuse navigation skill across platforms rather than relearn the full policy from scratch.

References

[1] Andry Maykol Pinto et al. “Promoting the use of robotics in the inspection and maintenance of offshore wind”. In: *Robotics and Automation Solutions for Inspection and Maintenance in Critical Infrastructures*. 2024. DOI: 10.1561/9781638282839.ch9.

[2] Daniel F Campos et al. “Nautilus: An autonomous surface vehicle with a multilayer software architecture for offshore inspection”. In: (2024). DOI: 10.1002/rob.22304. URL: https://onlinelibrary.wiley.com/doi/10.1002/rob.22304.

[3] Huarong Zheng and Chenguang Liu. “An overview of Unmanned Surface Vehicles: Methods, practices, and applications”. In: *Control Engineering Practice* 164 (Nov. 2025), p. 106479. ISSN: 0967-0661.

[4] Renato Jorge Silva, Pedro Nuno Leite, and Andry Maykol Pinto. “Multi-agent optimization for offshore wind farm inspection using an improved population-based metaheuristic”. In: *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*. 2020. DOI: 10.1109/ICARSC49921.2020.9096107.

[5] Daniel Filipe Campos, Aníbal Matos, and Andry Maykol Pinto. “Multi-domain inspection of offshore wind farms using an autonomous surface vehicle”. In: *SN Applied Sciences* 3 (4 2021). ISSN: 25233971. DOI: 10.1007/s42452-021-04451-5.

[6] Rafael M. Claro et al. “Energy Efficient Path Planning for 3D Aerial Inspections”. In: *IEEE Access* 11 (2023). ISSN: 21693536. DOI: 10.1109/ACCESS.2023.3262837.

[7] Maria Ines Pereira et al. “Advancing Autonomous Surface Vehicles: A 3D Perception System for the Recognition and Assessment of Docking-Based Structures”. In: *IEEE Access* 9 (2021). ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3070694.

[8] Pedro Nuno Leite and Andry Maykol Pinto. “Multimodal information fusion using pyramidal attention-based convolutions for underwater tri-dimensional scene reconstruction”. In: *Information Fusion* 124 (2025). ISSN: 15662535. DOI: 10.1016/j.inffus.2025.103339.

[9] Daniel Filipe Campos, Anibal Matos, and Andry Maykol Pinto. “An Adaptive Velocity Obstacle Avoidance Algorithm for Autonomous Surface Vehicles”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2019. DOI: 10.1109/IROS40897.2019.8968156.

[10] Maria Inês Pereira and Andry Maykol Pinto. “Reinforcement learning based robot navigation using illegal actions for autonomous docking of surface vehicles in unknown environments”. In: *Engineering Applications of Artificial Intelligence* 133 (2024). ISSN: 09521976. DOI: 10.1016/j.engappai.2024.108506.

[11] OpenAI. *Part 2: Kinds of RL Algorithms*. Accessed: 2026-03-02. 2021. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[12] Richard Bellman. *DYNAMIC PROGRAMMING*. 2021. DOI: 10.2307/j.ctv1nxcw0f.

[13] Richard Bellman. "The Theory of Dynamic Programming". In: *Bulletin of the American Mathematical Society* 60 (6 1954). ISSN: 02730979. DOI: 10.1090/S0002-9904-1954-09848-8.

[14] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: *Machine Learning* 3 (1 1988). ISSN: 15730565. DOI: 10.1023/A:1022633531479.

[15] Christopher J.C.H. Watkins and Peter Dayan. "Technical Note: Q-Learning". In: *Machine Learning* 8 (3 1992). ISSN: 15730565. DOI: 10.1023/A:1022676722315.

[16] Richard S. Sutton and Andrew G. Barto. "Reinforcement learning: An Introduction Second edition". In: *Learning* 3 (9 2012). ISSN: 18726240.

[17] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *International Journal of Robotics Research* 32 (11 2013). ISSN: 02783649. DOI: 10.1177/0278364913495721.

[18] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8 (3 1992). ISSN: 15730565. DOI: 10.1023/A:1022672621406.

[19] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *33rd International Conference on Machine Learning, ICML 2016*. Vol. 4. 2016.

[20] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *IEEE International Conference on Intelligent Robots and Systems*. 2012. DOI: 10.1109/IROS.2012.6386109.

[21] Shixiang Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2017. DOI: 10.1109/ICRA.2017.7989385.

[22] John Schulman et al. *Trust Region Policy Optimization*. 2017. arXiv: 1502.05477 [cs.LG]. URL: https://arxiv.org/abs/1502.05477.

[23] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[24] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (7540 2015). ISSN: 14764687. DOI: 10.1038/nature14236.

[25] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning". In: *34th International Conference on Machine Learning, ICML 2017*. Vol. 1. 2017.

[26] Will Dabney et al. "Distributional reinforcement learning with quantile regression". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. 2018. DOI: 10.1609/aaai.v32i1.11791.

[27] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in Neural Information Processing Systems*. Vol. 2017-December. 2017.

[28] Timothy P Lillicrap et al. *Continuous Control with Deep Reinforcement Learning*. Tech. rep. DOI: https://doi.org/10.48550/arXiv.1509.02971. URL: https://arxiv.org/abs/1509.02971.

[29] David Silver et al. "Deterministic policy gradient algorithms". In: *31st International Conference on Machine Learning, ICML 2014*. Vol. 1. 2014.

[30] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *35th International Conference on Machine Learning, ICML 2018*. Vol. 4. 2018.

[31] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *35th International Conference on Machine Learning, ICML 2018*. Vol. 5. 2018.

[32] Nan Yan, Subin Huang, and Chao Kong. "Reinforcement Learning-Based Autonomous Navigation and Obstacle Avoidance for USVs under Partially Observable Conditions". In: *Mathematical Problems in Engineering* 2021 (2021). ISSN: 15635147. DOI: 10.1155/2021/5519033.

[33] Valentin Sonntag et al. "A COLREGs compliance reinforcement learning approach for USV manoeuvring in track-following and collision avoidance problems". In: *Ocean Engineering* 316 (2025). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2024.119907.

[34] Ning Wang et al. "Sim-to-Real: Mapless Navigation for USVs Using Deep Reinforcement Learning". In: *Journal of Marine Science and Engineering* 10 (7 2022). ISSN: 20771312. DOI: 10.3390/jmse10070895.

[35] Zhang Yangyang et al. "Dynamic obstacle avoidance for USV based on velocity obstacle and dynamic window method". In: *Journal of Shanghai University* 23.1, 1 (2017), p. 1. DOI: 10.3969/j.issn.1007-2861.2016.07.021. URL: https://www.journal.shu.edu.cn/EN/abstract/article_6060.shtml.

[36] Shaorong Xie et al. "The obstacle avoidance planning of USV based on improved artificial potential field". In: *2014 IEEE International Conference on Information and Automation, ICIA 2014*. 2014. DOI: 10.1109/ICInfA.2014.6932751.

[37] Rui Song, Yuanchang Liu, and Richard Bucknall. "Smoothed A* algorithm for practical unmanned surface vehicle path planning". In: *Applied Ocean Research* 83 (2019). Cited by: 289, pp. 9–20. DOI: 10.1016/j.apor.2018.12.001. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058232041&doi=10.1016%2fj.apor.2018.12.001&partnerID=40&md5=64be811d3ca36dbb99764868086dca89.

[38] Yuanchang Liu, Richard Bucknall, and Xinyu Zhang. "The fast marching method based intelligent navigation of an unmanned surface vehicle". In: *Ocean Engineering* 142 (2017). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2017.07.021.

[39] Lingyu Li et al. "A path planning strategy unified with a COLREGS collision avoidance function based on deep reinforcement learning and artificial potential field". In: *Applied Ocean Research* 113 (2021). ISSN: 01411187. DOI: 10.1016/j.apor.2021.102759.

[40] Jianhui Mou et al. "A novel reinforcement learning framework-based path planning algorithm for unmanned surface vehicle". In: *Frontiers in Marine Science* 12 (2025). ISSN: 22967745. DOI: 10.3389/fmars.2025.1641093.

[41] Baoye Song, Zidong Wang, and Lei Zou. "An improved PSO algorithm for smooth path planning of mobile robots using continuous high-degree Bezier curve". In: *Applied Soft Computing* 100 (2021). ISSN: 15684946. DOI: 10.1016/j.asoc.2020.106960.

[42] Yunsheng Fan, Zhe Sun, and Guofeng Wang. "A Novel Reinforcement Learning Collision Avoidance Algorithm for USVs Based on Maneuvering Characteristics and COLREGs". In: *Sensors* 22 (6 Mar. 2022). ISSN: 14248220. DOI: 10.3390/s22062099.

[43] Bowen Xing et al. “An Algorithm of Complete Coverage Path Planning for Unmanned Surface Vehicle Based on Reinforcement Learning”. In: *Journal of Marine Science and Engineering* 11 (3 2023). ISSN: 20771312. DOI: 10.3390/jmse11030645.

[44] Zhengwei Zhu et al. “An improved dueling deep double-q network based on prioritized experience replay for path planning of unmanned surface vehicles”. In: *Journal of Marine Science and Engineering* 9 (11 2021). ISSN: 20771312. DOI: 10.3390/jmse9111267.

[45] Zhuo Chen et al. “Deep Reinforcement Learning Methods for USV Control: A Review”. In: *Proceedings - 2023 China Automation Congress, CAC 2023*. 2023. DOI: 10.1109/CAC59555.2023.10450528.

[46] Will Dabney et al. “Implicit Quantile Networks for Distributional Reinforcement Learning”. In: *Proceedings of Machine Learning Research*. Vol. 80. 2018.

[47] Xi Lin, John McConnell, and Brendan Englot. “Robust Unmanned Surface Vehicle Navigation with Distributional Reinforcement Learning”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2023. DOI: 10.1109/IROS55552.2023.10342389.

[48] Zhiguo Zhou et al. “A Real-time Algorithm for USV Navigation Based on Deep Reinforcement Learning”. In: Cited by: 8. 2019. DOI: 10.1109/ICSIDP47821.2019.9173280. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85091891243&doi=10.1109%2fICSIDP47821.2019.9173280&partnerID=40&md5=094e305125048d0712a76c33d4fc58f1.

[49] Charles Beattie et al. *DeepMind Lab*. 2016. arXiv: 1612.03801 [cs.AI]. URL: https://arxiv.org/abs/1612.03801.

[50] Junjie Zeng et al. “Navigation in unknown dynamic environments based on deep reinforcement learning”. In: *Sensors (Switzerland)* 19 (18 2019). ISSN: 14248220. DOI: 10.3390/s19183837.

[51] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. 2014. DOI: 10.3115/v1/d14-1179.

[52] Chuanbo Wu et al. “Deep reinforcement learning with intrinsic curiosity module based trajectory tracking control for USV”. In: *Ocean Engineering* 308 (2024). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2024.118342.

[53] Pengqi Sun et al. “Path Planning for Unmanned Surface Vehicles with Strong Generalization Ability Based on Improved Proximal Policy Optimization”. In: *Sensors (Basel, Switzerland)* 23 (21 2023). ISSN: 14248220. DOI: 10.3390/s23218864.

[54] Jiawei Xia et al. “Research on Collision Avoidance Algorithm of Unmanned Surface Vehicle Based on Deep Reinforcement Learning”. In: *IEEE Sensors Journal* 23 (11 2023). ISSN: 15581748. DOI: 10.1109/JSEN.2022.3222575.

[55] Delai Xue et al. “Proximal policy optimization with reciprocal velocity obstacle based collision avoidance path planning for multi-unmanned surface vehicles”. In: *Ocean Engineering* 273 (2023). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2023.114005.

[56] Jur Den Van Berg, Ming Lin, and Dinesh Manocha. “Reciprocal velocity obstacles for real-time multi-agent navigation”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2008. DOI: 10.1109/ROBOT.2008.4543489.

[57] Joohyun Woo, Chanwoo Yu, and Nakwan Kim. "Deep reinforcement learning-based controller for path following of an unmanned surface vehicle". In: *Ocean Engineering* 183 (2019). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2019.04.099.

[58] Derek R. Nelson et al. "Vector field path following for miniature air vehicles". In: *IEEE Transactions on Robotics* 23 (3 2007). ISSN: 15523098. DOI: 10.1109/TRO.2007.898976.

[59] Xinli Xu et al. "Path planning and dynamic collision avoidance algorithm under COLREGs via deep reinforcement learning". In: *Neurocomputing* 468 (2022). ISSN: 18728286. DOI: 10.1016/j.neucom.2021.09.071.

[60] Jian Zhao et al. "A DDPG-Based USV Path-Planning Algorithm". In: *Applied Sciences (Switzerland)* 13 (19 2023). ISSN: 20763417. DOI: 10.3390/app131910567.

[61] Zhewen Cui et al. "Autonomous collision avoidance decision-making method for USV based on ATL-TD3 algorithm". In: *Ocean Engineering* 312 (2024). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2024.119297.

[62] Yuan Zhou, Chongwei Gong, and Keran Chen. "Adaptive Control Scheme for USV Trajectory Tracking Under Complex Environmental Disturbances via Deep Reinforcement Learning". In: *IEEE Internet of Things Journal* 12 (11 2025). ISSN: 23274662. DOI: 10.1109/JIOT.2025.3528049.

[63] Dengyao Jiang et al. "Obstacle avoidance USV in multi-static obstacle environments based on a deep reinforcement learning approach". In: *Measurement and Control (United Kingdom)* 57 (4 2024). ISSN: 00202940. DOI: 10.1177/00202940231195937.

[64] Yuemin Zheng et al. "Soft Actor–Critic based active disturbance rejection path following control for unmanned surface vessel under wind and wave disturbances". In: *Ocean Engineering* 247 (2022). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2022.110631.

[65] Zhao Zhang et al. "Defense penetration strategy for unmanned surface vehicle based on modified soft actor–critic". In: *Ocean Engineering* 304 (2024). ISSN: 00298018. DOI: 10.1016/j.oceaneng.2024.117840.

[66] Zhewen Cui et al. "Autonomous Navigation Decision-Making Method for a Smart Marine Surface Vessel Based on an Improved Soft Actor–Critic Algorithm". In: *Journal of Marine Science and Engineering* 11 (8 2023). ISSN: 20771312. DOI: 10.3390/jmse11081554.

[67] Coline Devin et al. "Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer". In: *arXiv preprint arXiv:1609.07088* (2016). URL: https://arxiv.org/abs/1609.07088.

[68] Tao Chen, Adithyavairavan Murali, and Abhinav Gupta. "Hardware Conditioned Policies for Multi-Robot Transfer Learning". In: *arXiv preprint arXiv:1811.09864* (2018). URL: https://arxiv.org/abs/1811.09864.

[69] Hai Nguyen et al. "Symmetry-aware Reinforcement Learning for Robotic Assembly under Partial Observability with a Soft Wrist". In: *arXiv preprint arXiv:2402.18002* (2024). URL: https://arxiv.org/abs/2402.18002.

[70] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". In: *Science Robotics* 5.47 (2020), eabc5986.

[71] Ashish Kumar et al. *RMA: Rapid Motor Adaptation for Legged Robots*. 2021. arXiv: 2107.04034.