# Learning to Dock: A Simulation-based Study on Quantifying and Reducing the Sim2Real Gap in Autonomous Underwater Docking

By
Kevin Chang

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 27, 2025
Commencement June 2025

# AN ABSTRACT OF THE THESIS OF

Kevin Chang for the degree of Honors Baccalaureate of Science in Computer Science presented on May 27, 2025.
Title: Learning to Dock: A Simulation-based Study on Quantifying and Reducing the Sim2Real Gap in Autonomous Underwater Docking

Abstract approved:

---

Geoffrey Hollinger

Underwater robotic vehicles offer a promising pathway towards safer and more efficient completion of underwater tasks such as exploring unsafe environments, inspecting infrastructure, and collecting scientific data. Yet the efficacy of these robots is severely limited by their low battery life. Docking stations offer a viable solution towards long-term autonomy by allowing underwater robots to land on them, recharge and deliver data, and then continue on with their missions. However, complex non-linear hydrodynamics, unexpected disturbances like payloads or waves and currents, and robot failures such as jammed thrusters require docking controllers to be incredibly robust. In this work, we seek to utilize deep reinforcement learning coupled with a highly-parallelized simulation environment to develop docking controllers that are robust to these problems. Additionally, in our simulation, we perform controlled studies with modeled disturbances to seek to quantify the sim2real gap, and then test and evaluate existing methods for lowering that gap. Our findings provide insight into which real-world factors contribute most to the sim2real gap, and how existing methods are able to address those issues. Our work has valuable implications towards developing robust docking controllers that maintain their performance against unexpected disturbances and long deployment times.

Key Words: Reinforcement Learning, Robotic Control, Autonomous Underwater Vehicles

Corresponding e-mail address: changk2@oregonstate.edu

# Learning to Dock: A Simulation-based Study on Quantifying and Reducing the Sim2Real Gap in Autonomous Underwater Docking

By
Kevin Chang

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 27, 2025
Commencement June 2025

Honors Baccalaureate of Science in Computer Science project of Kevin Chang presented on May 27, 2025.

APPROVED:

Geoffrey Hollinger, Mentor, representing MIME

Alan Fern, Committee Member, representing EECS

Stefan Lee, Committee Member, representing EECS

Toni Doolen, Dean, Oregon State University Honors College

Kevin Chang, Author

# Contents

## 1 Introduction

Throughout human history, the ocean, covering 70% of the planet and containing 97% of the planet's water, has played a crucial role in supporting life on Earth through regulating the climate, providing a multitude of valuable resources, and more. Much of society's technological development has been terrestrial, yet in recent years, there has been an increasing need to complete marine-based tasks. Most notably, the urgency of the climate crisis has led to an increased public effort to study the damage being done to marine ecosystems and find ways to mitigate it [1]. For example, many oceanographers are searching for non-invasive ways to monitor biodiversity or evaluate the health of coral reefs [2, 3]. Furthermore, there has been a recent rise in underwater infrastructure being developed by companies and governments such as for underwater oil drilling operations [1]. Another notable example of this is the recent investment towards floating off-shore wind farms which seek to harness renewable energy from wind while minimizing the negative anthropogenic effects of constructing such large-scale energy systems [4].

With the increasing focus on the ocean and the rapid development of cutting-edge technology in the last century, many are looking to utilize advanced technology to support marine efforts. Most prominently, across academia and industry there has been a push towards developing underwater robotic vehicles that are capable of supporting many marine-related tasks. For example, a team at Woods Hole Oceanographic Institution recently developed an autonomous underwater vehicle (AUV) with hydrophones to study the acoustic landscape of a coral reef and identify different habitats, demonstrated through a field trial where they successfully identified areas where snapping shrimp tended to [2]. The same group used object detection algorithms trained in a semi-supervised manner to successfully track and follow various marine animals such as a barracuda and a jellyfish [5]. Additionally, glider robots have been used to monitor oceanographic data by following paths that are designed to maximize information value [6]. Towards ocean conservation efforts, underwater robots have also been designed to monitor dissolved substances around offshore oil or gas production regions in order to support early detection of ecologically disastrous spills and leaks [7].

Yet underwater robots face many challenges that significantly limit their utility in the real-world. For example, short battery lives restrict the duration that they can continue to operate, and often require a human crew to come and recharge the battery. Furthermore, underwater robots can suffer many electrical and mechanical setbacks such as seaweed and fishing lines causing thrusters to jam, hindering their capacity for long-term autonomy. A recent effort towards addressing these challenges has been the development of docking stations, or physical structures where these robots can land during missions in order to autonomously recharge or unload crucial data [8, 9]. Yet even with these docking stations in place, underwater robots need controllers that enable them to safely land on these structures even under the unpredictable and highly non-linear and dynamic forces present throughout the ocean. In addition to this, there is a need for controllers to be able to adapt to the many technical issues that may arise during longer-term deployments such as thruster failures.

In this thesis, we design, run, and evaluate simulation-based experiments in order to both quantify the sim2real gap, through evaluating different policies on different evaluation environments and then reducing it through using various robustness techniques. Specifically, we present an evaluation framework where we are able to train policies in a baseline simulation environment and then transfer them into other simulation environments containing challenging disturbances which will then determine how well a policy adapts to novel circumstances. We now provide a summary of our chapters. In Chapter 1, we present an introduction to our work. In Chapter 2, we describe our project's background including information on underwater robotic vehicles and reinforcement learning. In Chapter 3, we talk about related works including work on robotics simulators, learning-based control, and underwater docking control. In chapter 4, we discuss simulating the underwater docking problem, including information on our docking environment and custom dynamics model. In chapter 5, we write about our MDP and learning setup. In chapter 6 we discuss using simulation to study sim2real transfer including modeling real-world disturbances and defining our policy evaluation schema. In chapter 7, we discuss our naive learning-based docking strategy where we discuss the experimental process, naive policy results, and performance analysis. In chapter 8 we talk about improving controller robustness including techniques for robust learning, our experimental process. In chapter 9 we talk about our conclusions and future work where we perform a simulation-based analysis of the sim2real gap, an evaluation of robust training techniques, and our future work.

In this thesis, we seek to address the problem of developing controllers that are robust to the many technical challenges and issues that arise during real-world deployments in order to enable longer-term autonomy and improve the efficacy of docking stations. In particular, we seek to create learning-based controllers that utilize extensive data to develop robustness to changes in hydrodynamics and robot failures. To support this, we also present a highly-parallelized aquatic simulator designed for the underwater docking task. This simulator is demonstrated in Figure 1.

# 2 Background

## 2.1 Underwater Robotic Vehicles

Over the last couple of decades, significant effort has been made towards developing underwater robotic vehicles that can enable a variety of ocean-related efforts such as underwater exploration [10, 2], scientific sampling [11], and infrastructure inspection [12]. These robots are typically driven by propellers or thrusters [13, 10, 14], though other mechanisms exist as well such as flippers [15] or buoyancy shifters [16]. One robot that is commonly used in marine robotics research is called BlueROV2,

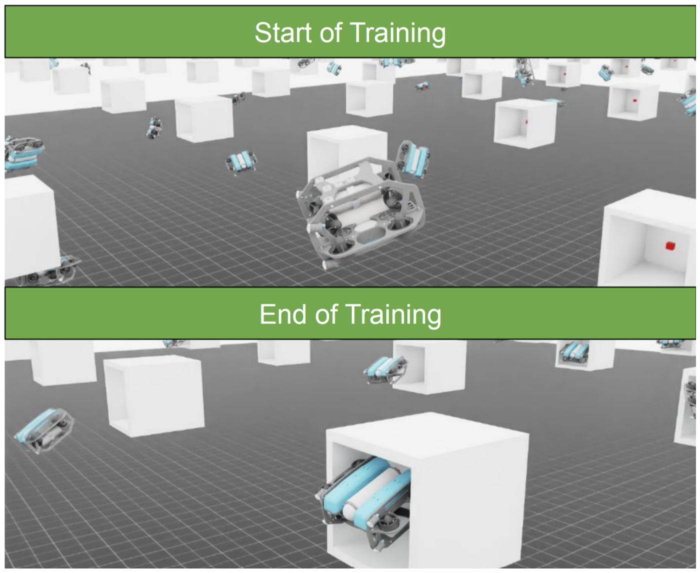

Figure 1: A demonstration of the capabilities of our highly-parallelized aquatic simulator. We are able simulate thousands of robots in parallel which enables our learning algorithm to quickly explore the state space and converge to an effective policy in minutes.

which was developed by Blue Robotics [17, 18]. An important aspect of developing underwater robotic vehicles is the control system, which must be designed to be highly robust against the varying disturbances found in the ocean including waves and currents. Additionally, various payloads are often attached to these vehicles, so a controller must be capable of adapting to that as well. Typically, algorithmic controllers are used for underwater robots like proportional-integral-derivative (PID), sliding-mode (SM) control [19], and model-predictive control (MPC) [8]. Yet more recently, learning-based methods for control have begun to be explored.

An existing barrier to using underwater robotic vehicles for long-term missions is the limited battery life and data storage or transfer capability. A promising solution to this issue is using docking stations to automatically recharge or collect data from the robots [20]. Yet, for docking stations to be useful, the robot's controller must be capable of accurately, safely, and robustly landing itself onto the docking station. Throughout this work, our focus is on developing controllers for autonomous docking of underwater robotic vehicles.

## 2.2 Reinforcement Learning

Reinforcement learning (RL) is a subset of artificial intelligence (AI) in which the goal is to develop intelligent agents that can learn to accomplish a task through experience and trial and error. Mathematically, the task of intelligent decision-making is formalized using a type of stochastic process called a Markov decision process (MDP). An MDP is represented as a tuple $< S, A, T, R >$ where $S$ represents the state space, or the possible states the agent can be in. $A$ represents the set of actions available to the agent. $T(s, a, s')$, represents the probability of transitioning into a new state $s'$ if at the original state $s$, the agent takes action $a$, or $P(s'|s, a)$. And lastly $R(s, a)$ represents the reward given to the agent for taking action $a$ at state $s$. Alternatively, $R$ can be formulated as $R(s)$ or $R(s, a, s')$. From this, we can derive a mathematical objective corresponding to the intelligent decision-making task.

The goal of reinforcement learning is to find a policy, or a stochastic mapping from states to actions that when used by the agent, maximizes the expected cumulative reward. More formally, we want to find a policy $\pi(a|s) \in \Pi$ which gives us a probability distribution of actions conditioned on a given state, which maximizes the expected cumulative reward achieved by the agent. This objective is formalized below:

$$\underset{\pi \in \Pi}{\text{argmax}}\, \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)] \tag{1}$$

,
where $\tau$ is a sampled trajectory, $T$ is the horizon length, and $\gamma \in [0, 1)$ is a discount factor which shapes the objective to prioritize earlier rewards.

In the last two decades, the development of deep learning (DL), or the use of densely parametrized models, large datasets, and heavy compute power to train deeply complex models, has naturally made its way into RL. In particular, what has enabled DL to rapidly accelerate progress in AI in recent years has been the development of neural networks (NNs) and the rapid improvements and expansions made within the space of NNs. An NN is itself a computational model inspired by how in nature, brains consist of a densely connected set of neurons, each sending and receiving signals to and from other neurons. One of the first and most widely-used NNs is called the multi-layer perceptron (MLP). In an MLP, the network is built with layers of artificial neurons, starting with an input layer from which input data is passed, hidden layers which process the input data, then an output layer which produces the output of the model. The learning is typically done by formulating some loss or objective function and then iteratively minimizing or maximizing using gradient descent.

In addition to these tasks, DL has also been applied to RL in what is called deep reinforcement learning (DRL). Much of the work here comes from the idea that a policy can be modeled and parameterized using a deep NN, and then using millions of samples, complex policies can be learned that can successfully operate in continuous state and action spaces. One of the earliest DRL algorithms is called REINFORCE, in which various trajectories are sampled in a model-free manner, then those trajectories are used to get an unbiased estimate of the gradient of the objective, then a step of gradient ascent is applied [21]. Then, much more recently, an algorithm called trust region policy optimization (TRPO) was developed which builds on standard policy gradients but enforces policy updates to be within a trust region by adding a constraint to the optimization problem which restricts the Kullback-Leibler (KL) divergence between the previous policy and the new, updated one [22]. After that, the same researchers developed proximal policy optimization (PPO) which slightly modifies TRPO by adding the KL divergence constraint directly into the objective, allowing standard gradient descent methods to be used rather than constrained optimization methods [23]. At this point it is important to introduce another distinction between different RL algorithms. So far, the DRL methods described above have involved updating the same policy that is being used to dictate the agent's exploration throughout its environment, they are thus known as on-policy methods. However, many methods also exist in which the policy being updated differs from the policy used to generate trajectories, collectively known as off-policy methods. One popular off-policy method is called soft actor-critic (SAC), in which a buffer is used to store previous experiences from which are than randomly used to estimate the gradient. In addition, a core novelty of SAC is that it not only seeks to maximize the expected cumulative reward, but also the policy's entropy, essentially seeking to act as randomly as possible [24].

An important development that came from DRL was the application of RL to real-world robotic control. Classical methods could not handle the continuous nature of real-world state and action spaces or the deep complexities of well-performing policies, but the use of deep NNs and powerful training algorithms in DRL has enabled the technology to be used to control robots in the real-world. Yet even with the ability to model complex continuous policies and iteratively optimize them, DRL algorithms still needed access to copious amounts of data to learn reasonable policies. Naturally, if a robot is to be deployed in the real-world, it is ideal to train its policy using real-world data, but for many safety and efficiency reasons this is largely impractical. Thus, researchers have resorted to training policies in simulations in which robots can train safely and data can be collected faster. However, despite efforts to improve the fidelity of training simulations, there commonly exists a gap between the dynamics seen in simulation versus those experienced in the real-world, leading to a significant deterioration in performance between training and evaluation. This is known as the sim2real gap, and is a crucial barrier to using DRL for real-world robotic control.

# 3 Related Work

## 3.1 Robotics Simulators

Across many robotic domains and tasks, there has always been a need for more powerful and realistic robotics simulators. Besides being crucial to learning-based control methods, simulators have been used extensively as a way to evaluate the performance of a robotic system or algorithm without having to perform a real-world deployment. For example, many researchers have used simulation to evaluate information gathering, path planning and simultaneous localization and mapping (SLAM) algorithms.

One of the most extensive and widely-used robotics simulators is called Gazebo. At the time of its release, most existing simulators were limited to 2-dimensional environments, so Gazebo was truly groundbreaking in how it supported 3-dimensional systems. A primary focus in the design of Gazebo was its ability to be easily extended through open-source development to support a wide variety of sensors, actuators, environments, and control systems. Originally, Gazebo used the Open Dynamics Engine (ODE) as its physics engine, though they had mechanisms in place to easily switch to other engines if opportunity presented itself. In addition to this, OpenGL is used to create visually complex and diverse scenes. Furthermore, Gazebo supports complex control systems through interfacing with the Robot Operating System (ROS). Gazebo has formed the basis for a significant breadth of work on extending the framework and utilizing it to model real-world tasks [25, 26, 27, 28].

Many simulators are also based on a physics engine called MuJoCo which is specifically designed to efficiently handle complex dynamical problems [29]. MuJoCo has shown impressive success as a back-end for simulators used in DRL in many robotic domains and tasks, including bipedal locomotion [30, 31] and humanoid locomotion [32]. Yet, one major issue in existing simulators used for DRL is their lack of parallelization and thus slowness in training. NVIDIA's Isaac Labs framework, formerly known as Orbit and Isaac Gym and built on their Isaac Sim platform, enables high-fidelity, highly-parallelized robotic simulation designed specifically for solving learning problems across many robotic domains. Its highly-parallelized design enables training of complex policies for robotic control in minutes. Another important benefit of the framework is its simple API which is highly robust yet flexible enough to be easily extendable to a wide variety of custom training environments. Additionally, its use of the Universal Scene Description (USD) API makes creating complex, high-fidelity, 3-dimensional scenes fairly trivial [33, 34]. This framework has shown success when used for DRL across many problems including controlling underwater robotic vehicles [35] and quadrupeds [36].

## 3.2 Learning-based Robotic Control

In recent years, extensive work has been done towards learning-based robotic control across many domains. In particular, the rise of DRL has enabled the development of robust controllers for complex high-dimensional problems that improve with experience. Furthermore, significant progress has been made towards reducing the sim2real gap. With quadrupeds, or four-legged robots, DRL-based controllers have demonstrated impressive locomotion performance even under complex terrain. Tan et al. demonstrated using DRL to train controllers for a variety of quadruped gaits, and also conducted a survey into how effective existing techniques for reducing the sim2real gap are such as domain randomization and compact observation spaces [37]. Then, Lee et al. was able to show effective DRL-based controls for locomotion over complex terrain, particularly through using privileged learning techniques and a recurrent policy architecture. Miki et al. performed a similar task but with a focus on effectively consolidating proprioceptive and exteroceptive inputs [38]. Towards a more direct way of reducing the sim2real gap, Peng et al. proposed an online fine-tuning process where during training the policy is conditioned on a latent embedding of the environmental parameters, and then during real-world deployment, a search is performed over the latent space to find an embedding that leads to the best real-world performance [39]. Their work illustrates how carefully conditioning policies can lead to improvements in performance through more specified behaviors.

Furthermore, significant progress has been made towards learning-based controls in the bipedal domain as well. Siekmann et al. successfully used recurrent policy architectures and domain randomization to train a bipedal robot to climb a variety of stairs without using any external perception modules [40]. Then, Pandit et al. demonstrated the use of decentralized DRL-based control to enable multiple bipeds to cooperatively carry various large payloads [41]. Towards robust and diverse learning-based controls for bipeds, Li et al. presented a general framework for training policies for bipedal locomotion with demonstrated effectiveness across a wide variety of locomotion tasks. Furthermore, they design and validate a novel method for integrating both short-term and long-term state history into a controller. Additionally, they demonstrate that beyond domain and dynamics randomization, task randomization also leads to more robust and general controllers [42].

Beyond ground vehicles, DRL-based controllers have shown promising performance in aerial tasks. Many existing works follow similar strategies as other domains including using privileged learning and online adaptive fine-tuning processes [43, 44, 45]. Notably, the highly agile nature of aerial vehicles such as quadcopters requires perception modules to be able to perceive important information as efficiently as possible, so researchers have experimented with developing DRL-based controllers that are perception-aware, or seek to simultaneously perform a task while maximizing the rate at which an on-board perception module gathers information. This often involves adding a term to the reward function that encourages orienting the perception module in the direction of the vehicle's motion [45].

Lastly, in recent years, learning-based control has entered the underwater robotics domain. Lu et al. utilized DRL for control of an underwater robot and proposed a novel method of using real-world data to improve the robustness of policies trained entirely in simulation [46]. Later, Cai et al. utilized NVIDIA's Isaac Labs framework to train policies for 6-DOF control of a thruster-driven underwater robot and successfully demonstrated fully zero-shot transfer into the real-world with similar performance to PID control [35].

### 3.3 Underwater Docking Control

Significant progress has been made towards developing control systems for successfully docking underwater robotic vehicles. Typically, the docking station itself is located relative to the robot through visible light sources [47], fiducial markers [8], or acoustics [48]. Many existing docking controllers are based on MPC [9, 8], yet recently, there has been interest in learning-based docking control. This first began with a work by Anderlini et. al, where the authors performed docking of a torpedo-shaped underwater robotic vehicle in simulation using the popular DRL algorithms Deep Q-Networks (DQN) and Deep Deterministic Policy Gradients (DDPG) [19]. Then, Patil et. al built off of this work, further refining their proposed reward function and testing on other state-of-the-art DRL algorithms like PPO, Twin Delayed Deep Deterministic (TD3), and SAC [49]. Later, Zhang et al. demonstrated using DRL for full 3-dimensional docking control and also considered external disturbances including wave and current forces, making efforts towards studying and reducing the sim2real gap [50]. Then, Palomeras et al. similarly demonstrated DRL-based docking control in a 3-dimensional simulation environment, but also proposed a novel method of augmenting the observation space using an extended Kalman filter (EKF) to improve performance against noisy observations [51]. Yet crucially, these previous works fail to consider the full extent of possible disturbances that a docking controller might experience in the real world such as thruster failures and attached payloads.

## 4 Simulating the Underwater Docking Problem

To support our work on learning-based autonomous underwater docking, we extend the highly-parallelized aquatic simulator based on NVIDIA's Isaac Sim and originally developed by Cai et al. [35] to support the underwater docking task. We describe their contributions as well as our own. First, we present the implementation details of our docking environment.

### 4.1 Docking Environment

We construct an environment that accurately models the docking problem. To model the docking station itself, we import an asset of a $0.7\ m \times 0.7\ m \times 0.7\ m$ hollow cube with an open face for the robot to enter through. We choose to use the BlueROV2 from Blue Robotics due to its widespread use in marine robotics research. Additionally, we use the Heavy configuration which adds 2 additional thrusters, making the total number of thrusters on the robot 8. Additionally, at the start of each training episode, we uniformly sample the robot's starting position from a $2\ m \times 2\ m \times 2\ m$ cubic space. We assume that similarly to the underwater docking work done by Vivekanandan et al. [8], the docking station will be localized with a fiducial marker pointing outwards in the direction of the open face, and so we do not consider the challenging case where

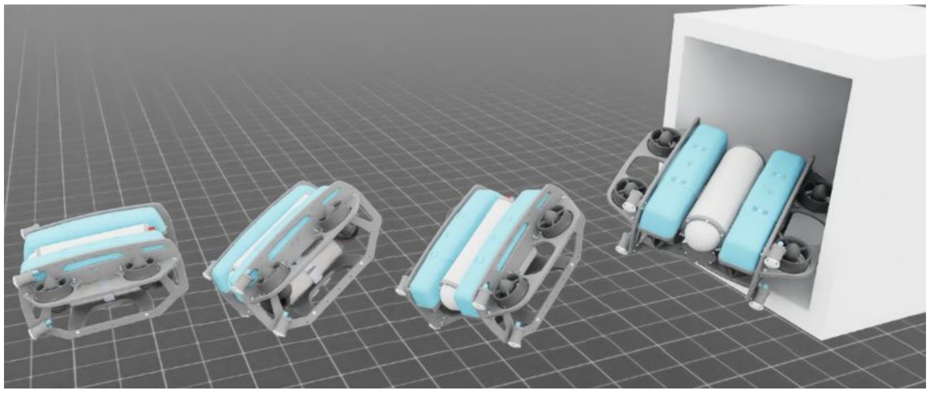
Figure 2: An example trajectory in our docking environment. The robot randomly spawns somewhere a few meters in front of the docking station and is initially set to point in the forward direction, and then must navigate into the docking station while maintaining its forward orientation in order to maximize reward.

the robot is behind the docking station. Our environment along with an example trajectory generated by a trained policy is shown in Figure 2.

## 4.2 Custom Dynamics Model

To support the complex dynamics involved in underwater robotics, Cai et al. [35] utilize the flexibility of Isaac Labs' API to implement a custom dynamics model which supports both hydrodynamic forces such as drag and buoyancy and thruster or propeller forces. We make use of their dynamics model for our work, and describe it here for completeness.

### 4.2.1 Hydrodynamic Forces

The hydrodynamic forces used in our simulation environment are ported from a simple inertia model found in the physics engine MuJoCo [29]. We describe the model here.

The shape of the robot is modeled through an inertia box with the following half-dimensions:

$$r_x = \sqrt{\frac{3}{2\mathcal{M}}(\mathcal{I}_{yy} + \mathcal{I}_{zz} - \mathcal{I}_{xx})} \tag{2}$$

$$r_y = \sqrt{\frac{3}{2\mathcal{M}}(\mathcal{I}_{zz} + \mathcal{I}_{xx} - \mathcal{I}_{yy})} \tag{3}$$

$$r_z = \sqrt{\frac{3}{2\mathcal{M}}(\mathcal{I}_{xx} + \mathcal{I}_{yy} - \mathcal{I}_{zz})} \tag{4}$$

,

$\mathcal{M}$ is a mass matrix
$\mathcal{I}$ is an inertia matrix

Then, the fluid forces exerted onto the AUV are separated into quadratic drag forces, denoted with $D$, and viscous forces, denoted with $V$. Then the forces and torques can be represented as follows:

$$\boldsymbol{f}_{inertia} = \boldsymbol{f}_D + \boldsymbol{f}_V \tag{5}$$
$$\boldsymbol{g}_{inertia} = \boldsymbol{g}_D + \boldsymbol{g}_V \tag{6}$$

,
Then, where $\rho$ is the density of the fluid, and $\boldsymbol{v}, \boldsymbol{\omega}$ are respectively the local linear and angular velocities of the robot, the components of the quadratic drag term are as follows:

$$f_{D,i} = -2\rho r_j r_k |v_i| v_i \tag{7}$$
$$g_{D,i} = -\frac{1}{2}\rho r_i (r_j^4 + r_k^4)|\omega_i|\omega_i \tag{8}$$

,
Then, where $\beta$ is the fluid viscosity and $r_{eq} = (r_x + r_y + r_z)/3$ is the radius of the equivalent sphere,

$$f_{V,i} = -6\beta\pi r_{eq} v_i \tag{9}$$
$$g_{V,i} = -8\beta\pi r_{eq}^3 \omega_i \tag{10}$$

,
These forces are calculated at each environment step and directly applied to the robot in simulation.

### 4.2.2 Thruster Forces

To compute thruster forces, we begin by assuming that our policy directly outputs pulse width modulation (PWM) commands and that there is no signal latency. We then convert these commands to angular velocities in radians per second using a quadratic model fit to data from the thrusters' manufacturer. This function $V(a)$ where $a$ is the PWM value sent to an individual thruster is defined as:

$$V(a) = \begin{cases} 0 & \text{if } -0.08 < a < 0.08 \\ -139a^2 + 500a + 8.28 & \text{if } a \geq 0.08 \\ 161a^2 + 517.86a - 5.72 & \text{if } a \leq -0.08 \end{cases} \tag{11}$$

,
We then model the physical latency involved in utilizing a thruster using the following first-order dynamics model represented by a function $D$:

$$\omega_{t_{x+1}} = D(\omega_{t_x}, t_x, t_{x+1}) = (e^{(t_{x+1}-t_x)/\tau})(\omega_{t_x}) + (1 - (e^{(t_{x+1}-t_x)/\tau}))(v) \tag{12}$$

,
$\tau$ is a dynamic time constant
$t_x, t_{x+1}$ are subsequent simulation timesteps
$\omega_{t_x}, \omega_{t_{x+1}}$ are agent states at corresponding time steps
$v$ is angular velocity command
where $\tau$ is a dynamic time constant, $t_x$ and $t_{x+1}$ are two successive time stamps in simulation, $\omega_{t_{x+1}}$ and $\omega_{t_x}$ are the agent states at those respective time stamps, and $v$ is the angular velocity command sent to the thruster. The effect $\tau$ has on the thruster latency is demonstrated in Figure 3.

Then, we use the model proposed by Yoerger et al. [52] to convert thruster angular velocities to an actual thrust. The model is shown below:

$$T(\omega) = C(|\omega|)(\omega) \tag{13}$$

,
$\omega$ is the thruster's angular velocity
$C$ is a tunable rotor constant T where $\omega$ is the angular velocity of the thruster's propeller at some arbitrary time.

Then, our entire model to convert from an individual thruster action output by a policy to a thrust within a single timestep is as so:

$$Thrust_{t_{x+1}} = T(D(V(a_{t_x}), t_x, t_{x+1})) \tag{14}$$

,
In software, the way this is implemented is by sampling an action from the policy, passing it through the above model, applying the resulting force to the robot, then repeating that process in each iteration.

# 5 MDP and Learning Setup

## 5.1 MDP Formulation

### 5.1.1 States

We take inspiration from the work by Cai et al. [35] when defining our state space but extend it to support the docking task as well as memory-based policies. We make an important distinction here between a robot state $\vec{s^*}$, which represents the actual physical state of the robot in simulation, and the agent state, which is the state passed

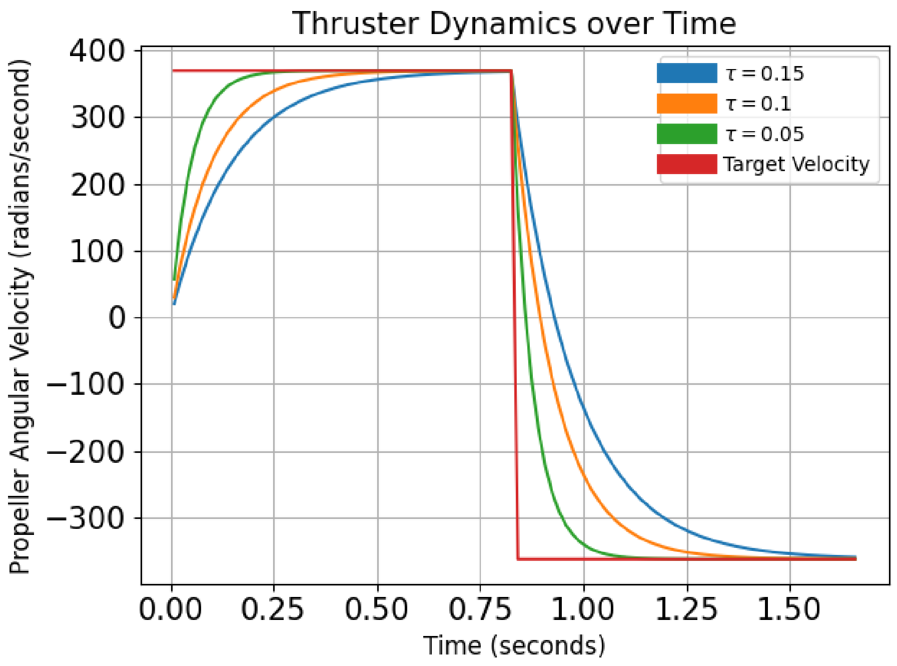

Figure 3: A demonstration of the effect $\tau$ has on the first-order dynamics model for the thruster motion. The thruster is first commanded to spin at the maximum speed in the positive direction, and then immediately commanded to switch directions. From the plot, it is evident that increasing $\tau$ leads to greater latency.

into a policy during training and evaluation $\vec{s}$. Specifically, we define a robot state at time $t$ as follows:

$$\vec{s_t} = (x_{dock t}, q_t, \dot{x}_t, \dot{q}_t, a_{t-1}) \tag{15}$$

,
where $x_{dock t} \in \mathbb{R}^3$ is the offset of the docking station from the robot, $q_t \in \mathbb{R}^4$ is a quaternion representing the robot's orientation, $\dot{x}_t \in \mathbb{R}^3$ is the robot's linear velocities, $\dot{q}_t \in \mathbb{R}^3$ is the robot's angular velocities in the form of Euler angles, and $a_{t-1} \in \mathbb{R}^8$ is the action taken in the previous step, or the action taken that resulted in the present state. Then, to support memory-based policies, each agent state can potentially consist of a sequence of subsequent robot states. We model this as follows:

$$\vec{s^*}_t = (\vec{s}_{t-h+1}, \vec{s}_{t-h+2}, \ldots, \vec{s}_{t-1}, \vec{s_t}) \tag{16}$$

,
where $h$ is the history length or the number of subsequent robot states included in the agent's state.

### 5.1.2 Actions

$$a \in \mathbb{R}^8 \tag{17}$$

,
We again build off of the work by Cai et al. [35] when defining our action space, but modify it slightly to support the BlueROV2 with the Heavy configuration. In particular, our action space is now in $\mathbb{R}^8$ since we are working with 8 thrusters instead of 6. Similarly to Cai et al., our agent's actions represent PWM values sent to each thruster.

### 5.1.3 Rewards

To formulate our reward function, we are again inspired by the work by Cai et al. [35] due to the similarity between their position and orientation holding task and our docking task. Specifically, we define our reward function as follows:

$$R_t = \lambda_1 R_{\text{dist}} + \lambda_2 R_{\text{orient}} \tag{18}$$

,
where $R_{dist}$ and $R_{orient}$ are defined as:

$$R_{\text{dist}} = \exp(-||p_t - p_{\text{dock}}||_2) \tag{19}$$
$$R_{\text{orient}} = \exp(-\theta_t) \tag{20}$$

,
where $p_t, p_{dock} \in \mathbb{R}^3$ are the position of the robot at time $t$ and the fixed position of the docking station respectively, and $\theta_t \in \mathbb{R}$ is the angular distance from the robot's orientation at time $t$ to the forward direction which points straight into the docking station. The angular distance is measured as the angle component of the axis-angle representation. We tune the weighting coefficients $\lambda_1, \lambda_2$ through first setting $\lambda_1 = 2$ and then experimenting with setting $\lambda_2$ to each value in the set $\{0.0, 0.15, 0.3, 0.45, 0.6\}$, and then selecting the value that enables the desired tradeoff between speed and stability.

## 5.2 Learning Setup

To learn how to control the AUV to land in the docking station, we again follow Cai et al. [35] and choose to use PPO [23]. Besides seeing its successful use by Cai et al. [35], the decision to use PPO is also motivated by the vast amount of learning-based robotic control literature that has demonstrate its effectiveness across many robotic domains [35, 39, 41, 38, 37]. Additionally, we hypothesize that given the large scale, highly-parallelized nature of our training environment, PPO, as an on-policy learning algorithm, will be able to make the best use of the extensive available training data to estimate highly stable and accurate gradient updates. The overall training setup is described in Figure 4.

# 6 Using Simulation to Study Sim2Real Transfer

As we do not utilize real-world deployments throughout this work, we develop a method of evaluating the real-world robustness of a trained docking policy entirely in simulation. To do this, we identify potential contributors to the sim2real gap and explicitly model them in our simulation.

## 6.1 Modeling Real-world Disturbances

### 6.1.1 Mass Shift

Often times, the specific mass distribution of a robot can not be accurately modeled in simulation. Marine robotics simulations often simplify this by modeling the robot as a point mass [35]. This simplification may potentially lead to degradations in performance when a policy is deployed in the real-world as the true mass distribution

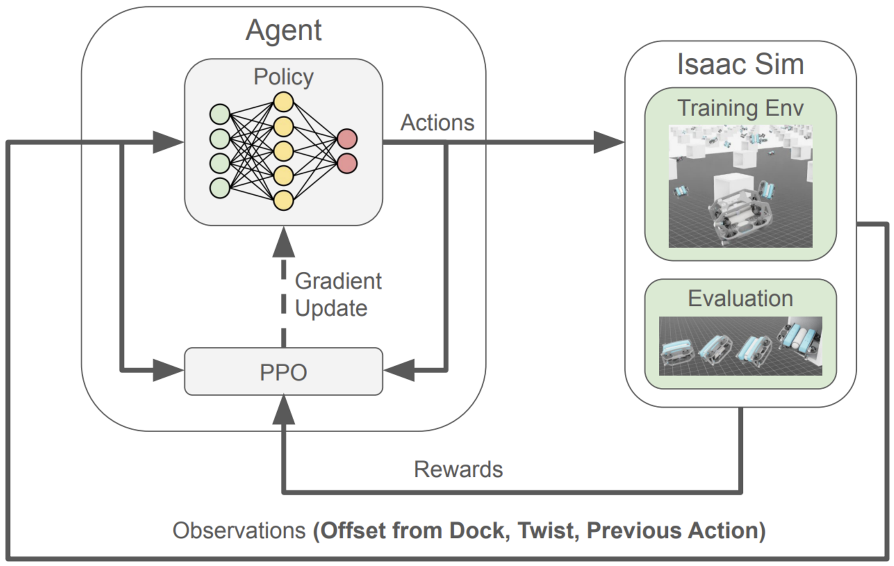

Figure 4: The training setup for our robot. During training, our agent is sent observations or states from the simulation, and then outputs actions which then affect the simulation and subsequently the next state. The learning algorithm PPO continually utilizes the states, actions, and rewards to compute gradient updates in order to improve the agent's performance. During evaluation, the agent interacts with a simulation environment designed for evaluation.

will affect the robot's stability and its reactiveness to control inputs. Furthermore, in many real-world scenarios, researchers or practitioners may attach various payloads to underwater robots such as cameras or hydrophones in order to collect scientific data [2]. These payloads may significantly shift the vehicle's mass distribution, and their varying, on-the-fly nature make it difficult to model them in simulation without severely restricting the robot's real-world versatility. We represent this disturbance by modeling the effect of having an arbitrary payload, represented as a point mass, at some location on the vehicle. Specifically, we calculate the force and torque caused by the mass and then apply that to the robot in simulation. For evaluation purposes, we decided to determine a policy's ability to adapt to real-world payloads by simulating placing a weight of 5 kilograms onto the robot, 0.3 meters along the x-axis, which corresponds to the back of the robot.

### 6.1.2 Center of Buoyancy Shift

The real-world position of the center of buoyancy (CoB) may also be difficult to model in simulation. While computer-aided design may provide a rigorous way of identifying a robot's CoB, similar to before, payloads attached during real-world deployments may shift the CoB on-the-fly. For example, if a set of sensors within an air-sealed container must be attached to the vehicle, then that will cause the CoB to shift. We model this change by parameterizing the relative offset of the CoB from the center of mass (CoM) and then taking it into account when computing the buoyant forces and torques. Initially, the CoB is estimated to be 0.01 meters directly above the CoM, but to evaluate this real-world disturbance we then test policies with the CoB shifted forward along the x-axis by 0.15 meters. This decision was motivated by how as described above we chose to shift the vehicle mass closer to the back during evaluation, and so we believed that shifting the CoB in the opposite direction would lead to the most challenging yet realistic dynamical problem.

### 6.1.3 Wave and Current Forces

In the real-world, wave and current forces are highly unpredictable and may change depending on the season and time of day, making precisely modeling them in simulation rather challenging. Furthermore, many researchers lack the resources to evaluate underwater robot performance in the real-world under wave and current disturbances in a controlled manner, and must test in the highly unstructured and unpredictable ocean. Thus, it a method of evaluating a docking policy's performance under these disturbances entirely in simulation is highly valuable. We model wave forces by periodically applying a force to the robot along the global Z-axis which is dictated by a sinusoidal function. We model current forces by applying a constant force onto the robot in a constant direction. In our evaluation schema, when testing a policy under wave and current disturbances we simulate a current force of 15 Newtons uniformly pointing to the positive y-direction, and a wavelength and amplitude of 75 and 100 respectively. In this case, the units represented by the amplitude are Newtons and the units represented by the wavelength are environment steps which each correspond to $\frac{1}{120}$. Thus a wavelength of 100 environment steps corresponds to $0.8\overline{333}$ seconds. We note that this seems exceptionally fast, but we did not use a more physically realistic value because we observed qualitatively in our simulation that given the speed of the robot, it made the most sense. The function for the sinusoidal wave force is as thus:

$$F_{waves}(s) = A(\sin(\frac{2\pi}{\lambda}s)) \tag{21}$$

,
where $s$ is the number of environment steps, $A$ is the amplitude in Newtons, and $\lambda$ is the wavelength in environment steps.

### 6.1.4 Thruster Latency

Across all real-world robotic domains, actuator latency is a significant contributor to the sim2real gap [37]. In this case, we are specifically speaking of the time it takes for an actuator to achieve a physical motion after the command for it is sent. In many cases, it is possible to model actuator latency somewhat accurately in simulation by fitting a dynamics model to real-world data. However, the BlueROV2 does not have encoders on its thrusters, so creating an informed model for its thruster latency is nontrivial. Thus, it is crucial to develop controllers that are robust under varying amounts of latency. Furthermore, in the real-world, as with most thruster-driven underwater robots, the BlueROV2's thrusters may accumulate debris which may increase their latency. Although this may lead to asymmetric latency across different thrusters, for simplicity, we assume that all 8 thrusters on the robot experience the exact same amounts of latency. As a baseline, we assume that the dynamic time constant $\tau$ which controls latency in our dynamics model is initially set to $\tau = 0.05$, but then to evaluate a policy under a disturbance scenario we set $\tau = 0.15$. The latency under these two $\tau$ settings are visualized in Figure 3.

### 6.1.5 Disabled Thrusters

A common issue that arises in real-world deployments is that of thrusters becoming disabled. This can have a variety of causes, from mechanical issues such as jams caused by seaweed or fishing line to electrical issues. This represents an important barrier to long-term autonomous missions because a thruster is more likely to fail as missions become longer as there is a higher chance of both mechanical and electrical failures taking place. Thus, towards developing learning-based controllers capable of supporting long-term docking-based autonomous missions, we evaluate trained policies based on how well they can continue to perform if a random thruster is disabled.

## 6.2 Policy Evaluation Schema

To evaluate a policy in simulation, we utilize Isaac Labs' evaluation feature, in which agents can be loaded into an arbitrary environment along with a pretrained policy and then episodes can be performed without any gradients being calculated and applied to the policy. Our primary methodology is to create multiple environments specifically designed for evaluation, each set with different parameters to test a policy's robustness to various disturbances. The evaluation environments and their parameters are shown in Table 1.

$$\tau = 0.1 \tag{22}$$

,

| **Name** | $\tau$ | **CoM-CoB Offset** | **Payload** | | **Waves** | | **Current Force** | **Disable Thruster** |
|---|---|---|---|---|---|---|---|---|
| | | | Mass | Offset | Wavelength | Amplitude | | |
| Base | 0.05 | $[0.0m, 0.0m, 0.01m]$ | 0 kg | 0 m | n/a | n/a | 0 N | No |
| Easy | 0.1 | $[-0.075m, 0.0m, 0.01]$ | 2.5 kg | 0.15 m | 150 | 50 | 7.5 N | Yes |
| Medium | 0.15 | $[-0.15, 0.0, 0.01]$ | 5 kg | 0.3 m | 150 | 100 | 15 N | Yes |
| Payload | 0.05 | $[0.0m, 0.0m, 0.01m]$ | **5 kg** | **0.3 m** | n/a | n/a | 0 N | No |
| CoB Shift | 0.05 | $\boldsymbol{[-0.15m, 0.0m, 0.01m]}$ | 0 kg | 0 m | n/a | n/a | 0 N | No |
| Latency | **0.15** | $[0.0m, 0.0m, 0.01m]$ | 0 kg | 0 m | n/a | n/a | 0 N | No |
| Disabled Thruster | 0.05 | $[0.0m, 0.0m, 0.01m]$ | 0 kg | 0 m | n/a | n/a | 0 N | **Yes** |
| Waves/Currents | 0.05 | $[0.0m, 0.0m, 0.01m]$ | 0 kg | 0 m | **150** | **100** | **15 N** | No |

Table 1: The evaluation environments and their respective parameters.

# 7 Naive Learning-based Docking

In order to understand and attempt to solve the issue of the sim2real gap, it is crucial to first identify a baseline performance. We identify the baseline method as a training conducted under the naive assumption that there is no such sim2real gap, thus the policy may overfit to the training environment because the evaluation environment will be identical. Theoretically, this should result in the best possible performance in that particular evaluation environment.

## 7.1 Naive Policy Experimental Process

Our process revolves around naively training policies and then evaluating them under various disturbances to comparatively study the degrees to which disturbances cause degradation in the performance of a policy. Following sim2real, this can be thought of as simulating a sim2real transfer to gain insights into the existing gap. In particular, we train policies under the **Base** evaluation environment, and then evaluate those policies under each of the other evaluation environments, including **Base**. We train naive policies across 5 different seeds for reproducibility. We train for around 1000

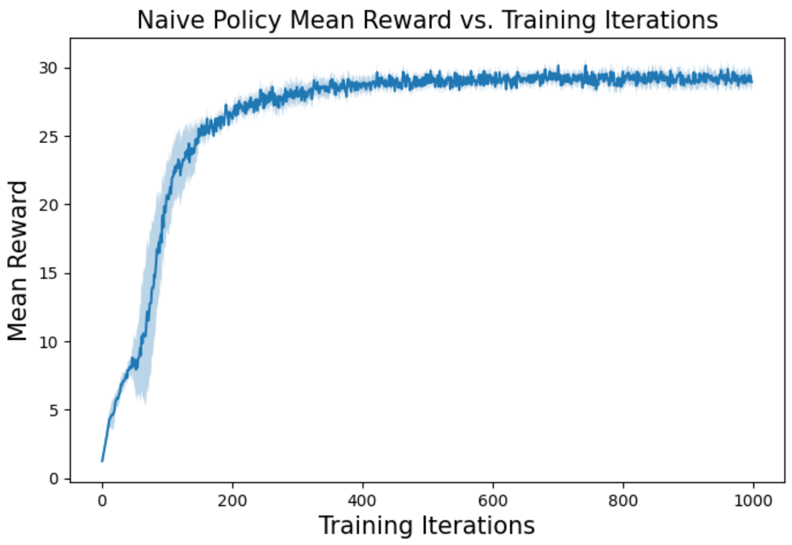

Figure 5: The training curves of the naive policies. These policies are trained in a fixed environment with no disturbances, following the naive assumption that there are no disturbances in the real world or differences between simulation and reality.

iterations since we find that of all of our planned training configurations, that is the maximum amount of time required for a training to converge.

## 7.2 Naive Policy Results

### 7.2.1 Training Results

The training curve for our naive policies is shown in Figure 5. We observe that the training is very stable and has low variance across the several seeds. We also observe that it converges very quickly, with most of the training after iteration 300 appearing to be redundant.

### 7.2.2 Evaluating Against Real-world Disturbances

We utilize the naively trained policies to evaluate how different disturbances have varying effects on a generic policy's performance. The positional and angular errors collected during the evaluation of naive policies against various disturbances are shown in Figures 6 and 7 respectively.

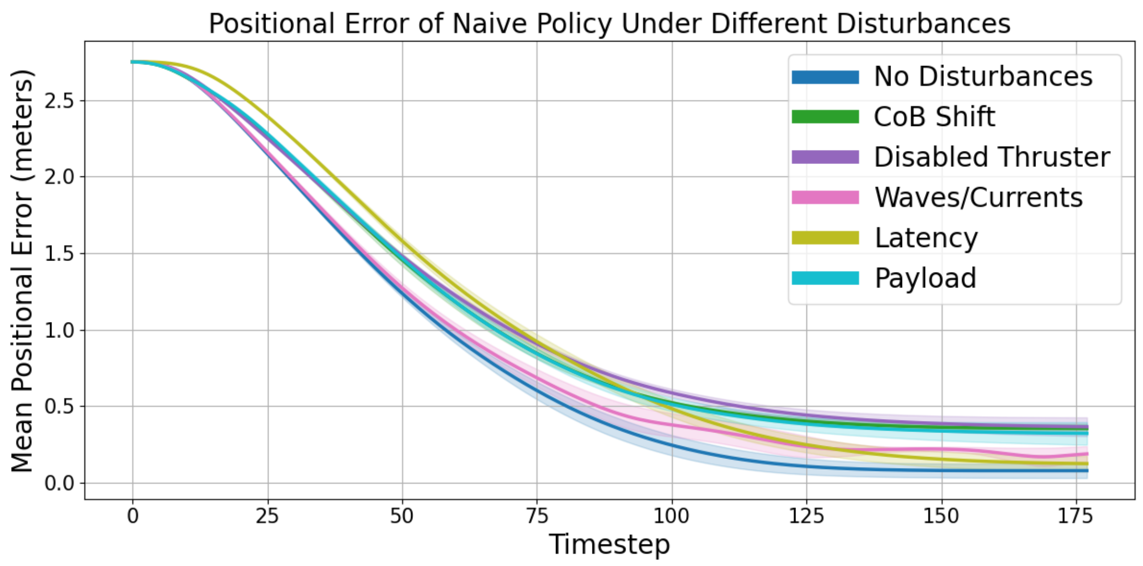

Figure 6: Positional error of naively trained docking policies evaluated under various individual disturbances.

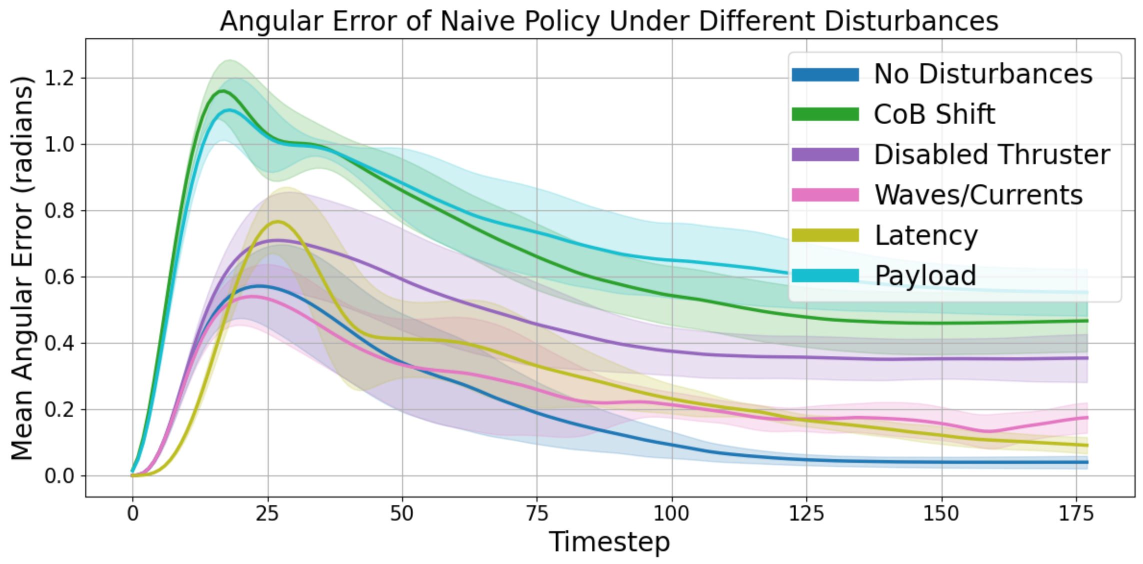

Figure 7: Angular error of naively trained docking policies evaluated under various individual disturbances.

## 7.3 Analyzing Naive Performance

In analyzing both positional and angular error, we are able to confirm that the naively trained policy indeed performs best under no disturbances. We will now analyze both error metrics for each disturbance.

**CoB Shift**: Shifting the CoB leads to one of the highest increases in positional error, missing the target by ∼0.4 m. This is a very significant error in real-world docking, and should not be ignored when considering attaching a buoyant payload to an underwater robot. Additionally, we see that this disturbance leads to one of the highest angular errors, resulting in a misalignment of ∼0.5 radians. Depending on the type of docking station, this may or may not be a significant issue, especially if some sort of auto-alignment mechanism is in place such as a funnel.

**Disabled Thruster**: Randomly disabling a thruster seems to cause the most positional error, similarly missing the target by ∼0.4 m. Given how thruster failure is a common issue in longer term underwater robot deployments, this suggests that a naively trained docking policy likely wouldn't be effective in the long-term, even if it's power needs were supported by the docking station. We also see that compared to other disturbances, randomly disabling a thruster leads to a fairly significant increase in angular error.

**Waves/Currents**: Under the presence of wave and current forces, we observe very minimal differences in performance, suggesting that naively training a policy is sufficient to handle these conditions. We hypothesize this is due to how in our model, waves and currents do not apply any torques to the robot, and so the robot simply needs to adapt to different linear forces. It is important to note however that while we do model oscillatory wave forces being applied to the robot, we do not model the docking station's motion, which may make the control problem more challenging.

**Latency**: Increasing the physical latency of our thruster model seems to also have a fairly negligible impact with regards to both error metrics. This makes theoretical sense, as the amount of latency should not significantly change the actions an optimal policy would take under different situations. However it is important to consider that we are only considering increasing latency on its own, and it is possible that combining an increase in latency with another disturbance like a payload or CoB shift would exacerbate the degradation caused by that disturbance alone.

**Payload**: Attaching a payload to the robot seems to have the worst overall effect on the controller's performance. In terms of positional error, it leads to an increase in error by ∼0.4 m and it boasts the worst angular error, increasing it by ∼0.6 radians. It makes sense that the angular error would be very poor, since attaching a payload shifts the robot's mass distribution and thus applies significant torques. This implies that when deploying a learning-based docking controller in the real-world, care should be taken to attach minimal payloads that do not shift the center of mass too significantly. Additionally, an interesting question is that of how much of the positional error is caused by the torques applied from the shifted center of mass. For example, would the positional error be significantly reduced of the payload were attached directly at the center of mass? Depending on the mass of the payload, this is analogous to a wave or current disturbance, so it would likely perform similarly.

Through these experiments, we gain insights into the varying effects that different disturbances have on a naively trained policy. We observe that generally, shifting the CoB, randomly disabling a thruster, and attaching a payload seem to cause the most degradation in performance. We do again note that this is only considering the individual effects of each disturbance, and that more work can be done in exploring the effects of different pairs or triplets of disturbances. Still, this provides important information on what contributes the most to the sim2real gap in learning-based docking, and thus what still needs to be addressed.

## 8 Improving Controller Robustness

After exploring the performance of a naively trained policy, a natural direction for further research is how can the policy be improved to become more robust to real-world disturbances? Various techniques have been explored in literature, and so we implement these methods and evaluate their performance against naively trained policies.

### 8.1 Techniques for Robust Learning

One of the most commonly used techniques for improving the robustness of a policy involves randomizing dynamics parameters during training like mass, latency, or damping, in a process called dynamics randomization. Additionally, environmental parameters like fluid density or terrain difficulty can be randomized as well in a method known as domain randomization. From now on we do not distinguish between dynamics randomization and domain randomization and instead refer to both collectively through the acronym DR. Through DR, a policy must learn to be robust to a wider range of conditions, so then when it is deployed in a new environment for evaluation it has the following benefits. Firstly, it is more likely that the evaluation conditions were seen during training if a wider range of conditions were used in the training environment. Secondly, even if the evaluation conditions are outside of the distribution of training conditions, DR forces the policy to make less assumptions about the environment, preventing overfitting and thus increasing its ability to generalize. The effectiveness of DR has been validated across many robotic domains [35, 40, 38, 30].

$$h = 3 \tag{23}$$

,

Another common approach is that of memory-based architectures. With purely reactive policies, such as those that are represented through MLPs and only take in the robot's latest state as input, the policy often lacks sufficient information to adapt to unexpected disturbances. In particular, if for example a payload is attached to a naively trained policy during evaluation, then the policy may understand that something has changed due to observing new states, but other than that it must react to it as if there were no payload. This issue worsens when DR is used, as the policy is mostly unaware of what specific environmental variation it is currently learning in, and thus must learn to take actions that maximize the average performance over the range of possible conditions, often leading to overly conservative behaviors. One possibility is to directly condition the policy on the specific environmental settings seen during DR so that the policy can learn to adapt its behavior to different environments. However, in the real-world, the agent does not have access to this information, so it must use alternate methods or find a way to discover that information. One method of attaining that information implicitly is through memory-based architectures. Hypothetically, information on dynamics and environmental disturbances should be captured in a robot's state history and so the policy can almost perform a type of implicit system identification with a similar effect as parameterizing the policy on environmental settings directly. In our work, we perform a fairly naive form of this where we feed a sequence of previous states alongside the latest state to our policy in order to condition it on a state history. Other methods include using recurrent policy architectures or transformers. Memory-based policies have also shown success on a variety of robotic tasks, especially in the biped domain [40, 53, 42].

## 8.2 Robust Training Experimental Process

We design various training configurations that capture the usage of both DR and memory-based architectures. In particular, for DR, we consider our intended evaluation settings when selecting our parameter distributions such that we are able to test both in-distribution and out-of-distribution performance. Furthermore, for each evaluation environment described in Table 1, we compare these robust policies with two naively trained policies, one trained under no disturbances, and another trained for the specific evaluation settings currently being tested. The goal is to determine how our efforts to directly mitigate the sim2real gap compare with methods that follow the naive assumption that there is no such sim2real gap under the case where that assumption is valid and where it is not. Our methods for performing DR on each environmental parameter are shown below:

- $\boldsymbol{\tau}$: the dynamic time constant is sampled uniformly within some specified interval. It is applied symmetrically across all thrusters.
- **CoM-CoB Offset**: the CoB shift relative to the CoM is sampled uniformly within a sphere of a specified radius.
- **Payload**: the mass of the payload is sampled uniformly from specified interval. The offset from the center of the robot is sampled uniformly from a sphere of a specified radius.

- **Waves**: the wavelength is fixed but the amplitude is sampled uniformly from 0 to a specified upper bound.
- **Current**: the direction of the current is sampled uniformly from all possible directions and then the force is sampled uniformly from 0 to a specified upper bound.
- **Disable Thruster**: if enabled, in each episode a random thruster is chosen uniformly to be disabled for the duration of the episode.

The specific training parameters including parameter intervals for DR are shown in Table 2.

| **Parameter** | **Base** | **Small DR** | **Medium DR** | **Medium DR w/ History** |
|---|---|---|---|---|
| $\tau$ | 0.05 | 0.1 | 0.15 | 0.15 |
| CoM-CoB Radius | 0 m | 0.075 m | 0.15 m | 0.15 m |
| Payload Mass | 0 kg | $[0, 2.5]$ kg | $[0, 5]$ kg | $[0, 5]$ kg |
| Payload Spawn Radius | 0 m | 0.15 m | 0.3 m | 0.3 m |
| Wavelength | n/a | 150 | 150 | 150 |
| Max Amplitude | n/a | 50 | 100 | 100 |
| Max Current Force | 0 N | 7.5 N | 15 N | 15 N |
| Disable Thruster | No | Yes | Yes | Yes |
| History Length | 1 | 1 | 1 | 3 |

Table 2: The training environments and their respective parameters.

Similarly to before, each configuration is trained across 5 seeds for reproducibility.

## 8.3 Results and Analysis of Robust Policy Experiments

### 8.3.1 Training Results

The training curves of the robust policies along with the naive policies as a baseline are shown in Figure 8. We observe that as the amount of DR applied during training increases, the reward the policy converges to decreases, which makes sense as when parameters are randomized, the agent must sacrifice performance for robustness. Notably, we see a marginal improvement in reward from incorporating memory into our policy. We expected the improvement to be higher, though it is possible that the policy architecture is not large and complex enough to condition its behavior on the limited available memory.

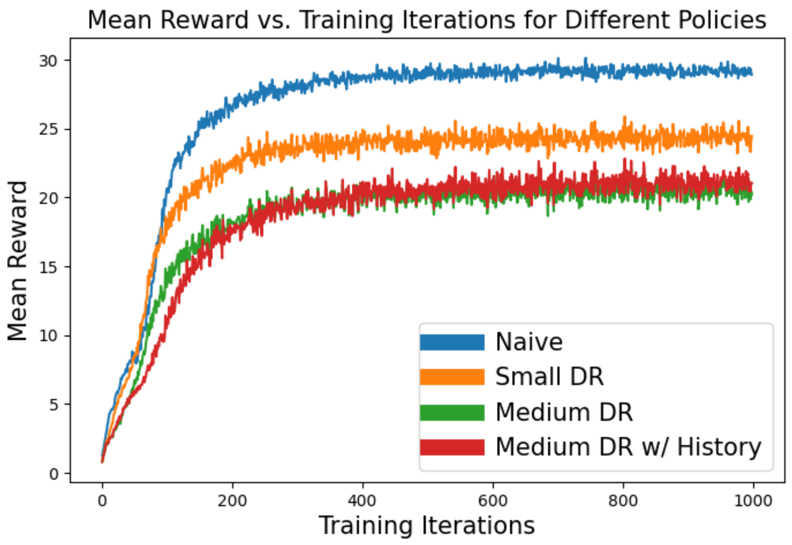

Figure 8: The training curves of the naive and robust policies. Robust policies are trained using DR and state histories. Naive policies are shown again as a baseline reward curve.

#### 8.3.2 Evaluation Under No Disturbances

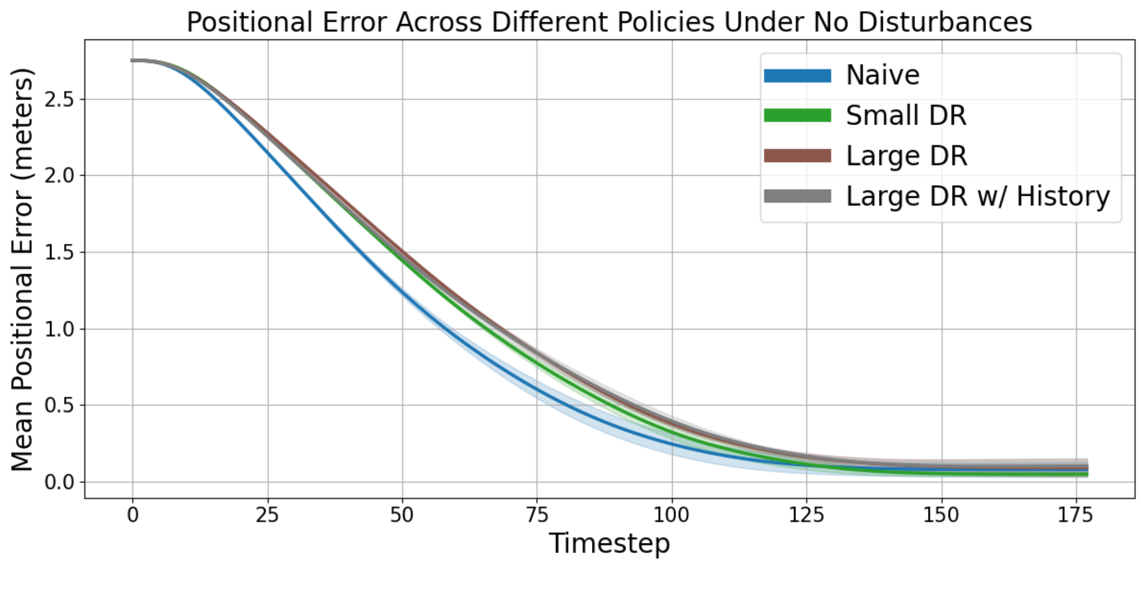

Figure 9: Positional error of different training configurations evaluated under no disturbances or **Base** evaluation configuration.

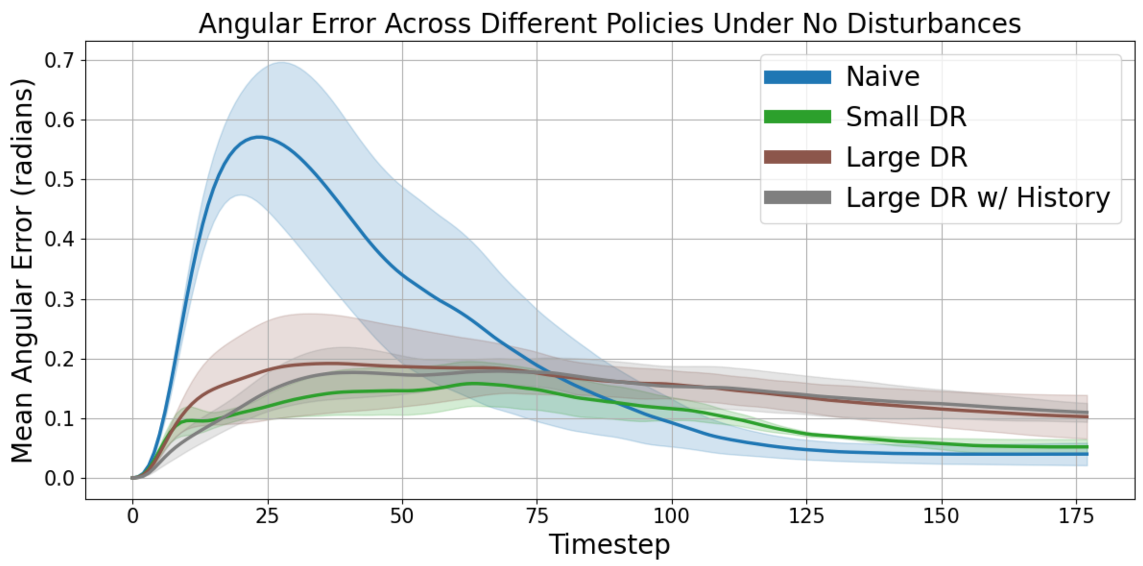

Figure 10: Angular error of different training configurations evaluated under no disturbances or **Base** evaluation configuration.

We observe in our plots shown in Figures 9 and 10 that when our different policies are evaluated under no disturbances, then the robust policies perform roughly as well as the naively trained policies. However, we note that while the naive and robust policies converge to the same positional error, the robust policies move towards the target at a slower, steadier pace. This is likely caused by DR producing more conservative behaviors. Regarding the angular error, we notice that within our limited horizon, some of the robust policies do not reach the minimal error attained by the naive policies, though given the downward trend of the robust policies' error, it is possible that given a longer horizon, they would all converge to the same error. We note here that the error produced by the naive policies tend to spike at the beginning and then quickly diminish. This can most likely be attributed to the naively trained agent learning to orient itself in order to utilize its thruster more efficiently. We also note that on the other hand, the steadiness of the angular error produced by the robust policies could possibly explain why their positional error decreases so steadily by indicating that the agent is not attempting to orient its thrusters more efficiently.

### 8.3.3 Evaluation Under Easy Conditions

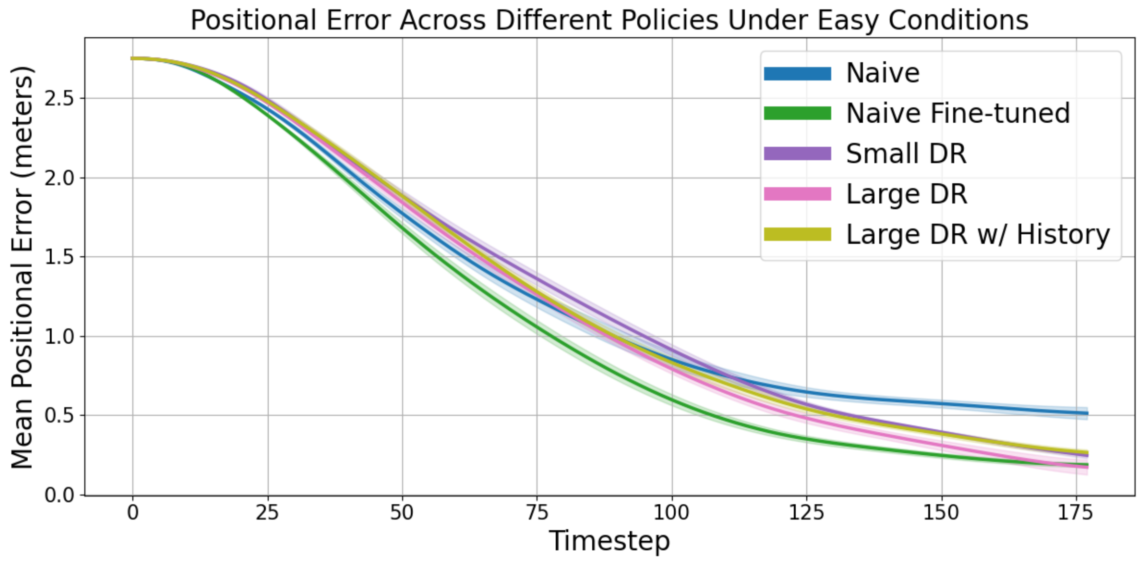

Figure 11: Positional error of different training configurations evaluated under easy conditions or **Easy** evaluation configuration.

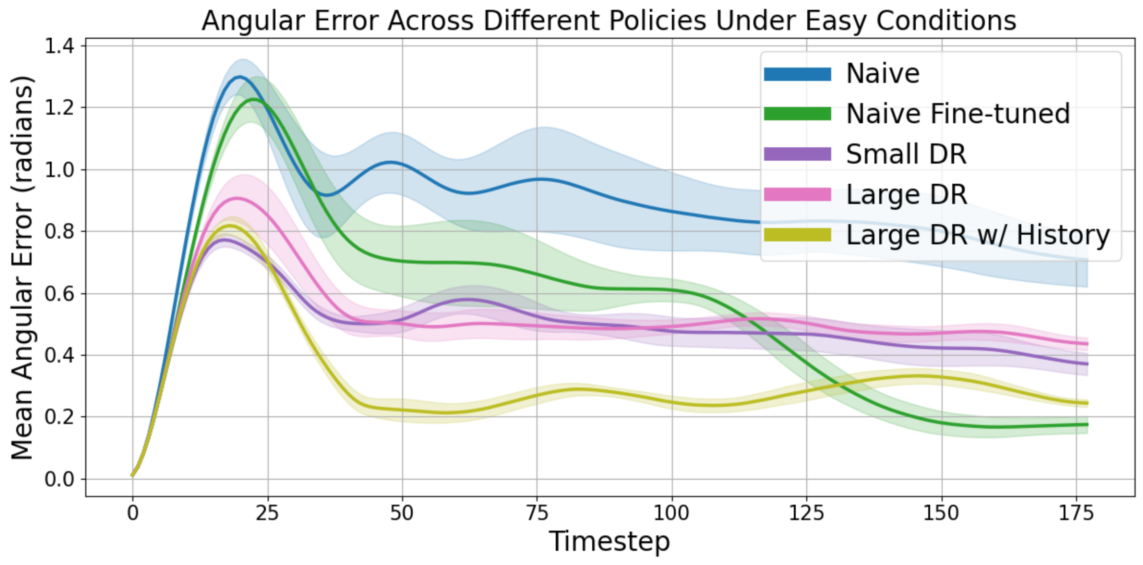

Figure 12: Angular error of different training configurations evaluated under easy conditions or **Easy** evaluation configuration.

Analyzing our results from evaluating our various training configurations under easy conditions, shown in Figures 11 and 12, we observe that our robust training techniques produce superior performance to policies naively trained under no disturbances. In particular, we see in 11 that the robust policies are able to exceed the performance of the standard naive policies and converge to the same error as the fine-tuned naive policies. This indicates that under relatively easy disturbances, even without knowing the exact evaluation configuration, our robust training techniques enable performance that matches naively trained policies that had access to the evaluation environment. With regards to the angular error, shown in 12, we again observe that our robust policies exceed the performance of the standard naive policies, though they do not quite reach the performance of the naively fine-tuned policies. Notably, we see our memory-based policies performing slightly better than our purely reactive DR policies, suggesting that conditioning on a history of states does indeed lead to improved performance.

### 8.3.4 Evaluation Under Medium Conditions

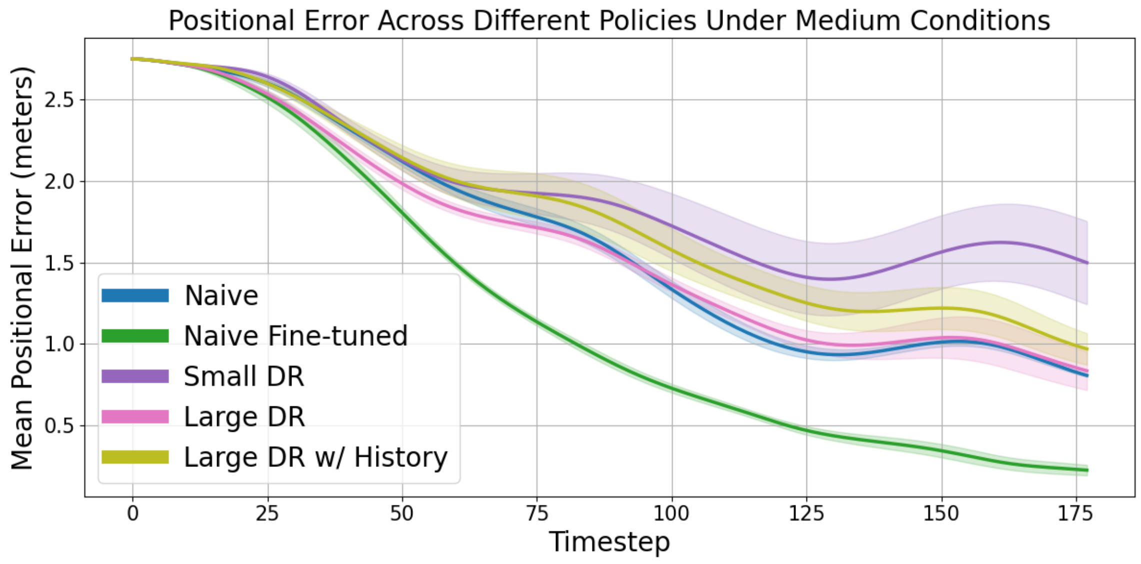

Figure 13: Positional error of different training configurations evaluated under medium conditions or **Medium** evaluation configuration.

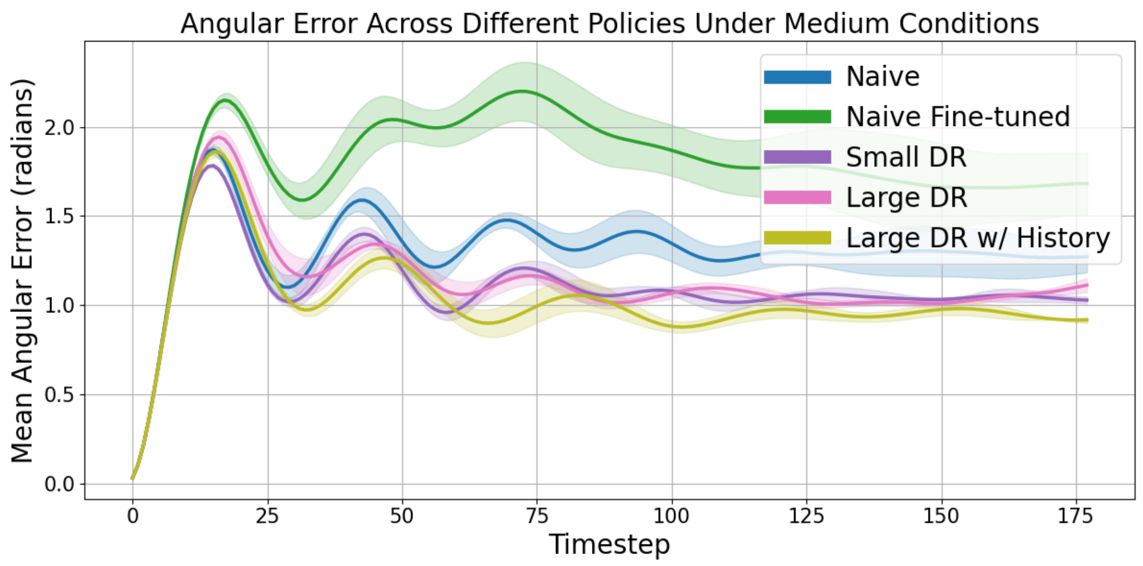

Figure 14: Angular error of different training configurations evaluated under medium conditions or **Medium** evaluation configuration.

When evaluating our policies on harder, medium conditions, results shown in Figures 13 and 14, we observe drastically different results than those from the evaluations performed under easy conditions. Crucially, we see that with regards to positional error shown in Figure 13, none of the robust policies are able to exceed the performance of the standard naive policies. However, we do note that in the angular case, shown in Figure 14, the robust policies outperform the standard naive policies. Furthermore, across all policies, we observe highly oscillatory angular error, which can potentially be attributed to the policies' inability to maintain rotational stability against various disturbances due to the high thruster latency. Surprisingly, we observe that the fine-tuned naive policy greatly outperforms all other policies in terms of positional error, but converges to the highest angular error. Given that the evaluation environment is within the DR distribution used to train the **Large DR** and **Large DR w/ History** policies, this result suggests that training on the exact environmental instance used during evaluation leads to poorer performance than training on a distribution containing that same environmental instance, which is challenging to understand or explain.

### 8.3.5 Evaluation Under Shifted CoB

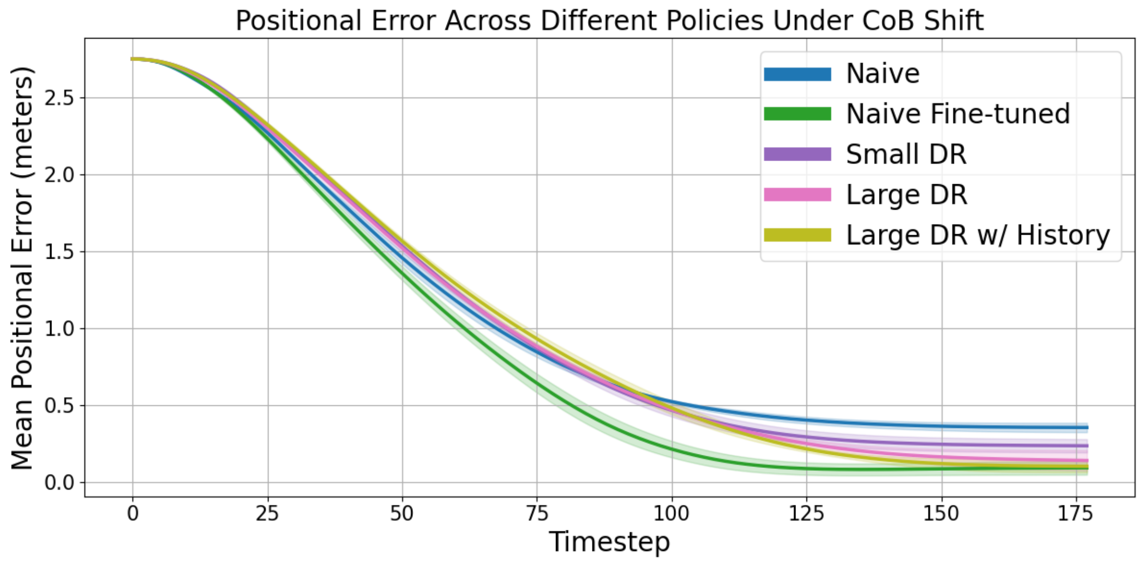

Figure 15: Positional error of different training configurations evaluated under CoB shift or **CoB Shift** evaluation configuration.

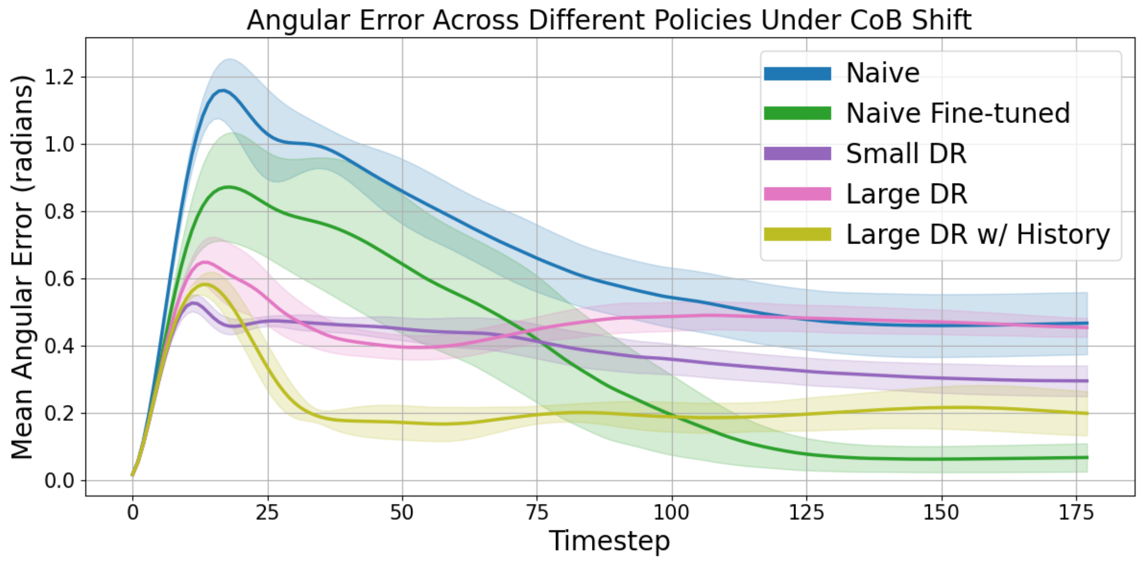

Figure 16: Angular error of different training configurations evaluated under CoB shift or **CoB Shift** evaluation configuration.

Studying the effect that shifting the CoB has on our policies shown in Figures 15 and 16, we observe similar results to previous disturbances where our robust policies exceed the performance of the standard naive policies but do not quite reach the error achieved by the fine-tuned naive policies. Notably, when looking at the angular error in Figure 16, we see that the robust policies achieve lower variance than their naive counterparts. This can likely be explained by how DR has a regularization effect on the policy where it becomes less likely to overfit to the training distribution. Yet it is interesting that despite being trained on the exact evaluation environment, the fine-tuned naive policies exhibit the highest variance. Additionally, a crucial detail here is that using memory-based policies shows a large improvement over standard reactive policies, even when trained with the same DR settings.

### 8.3.6 Evaluation Under Increased Latency

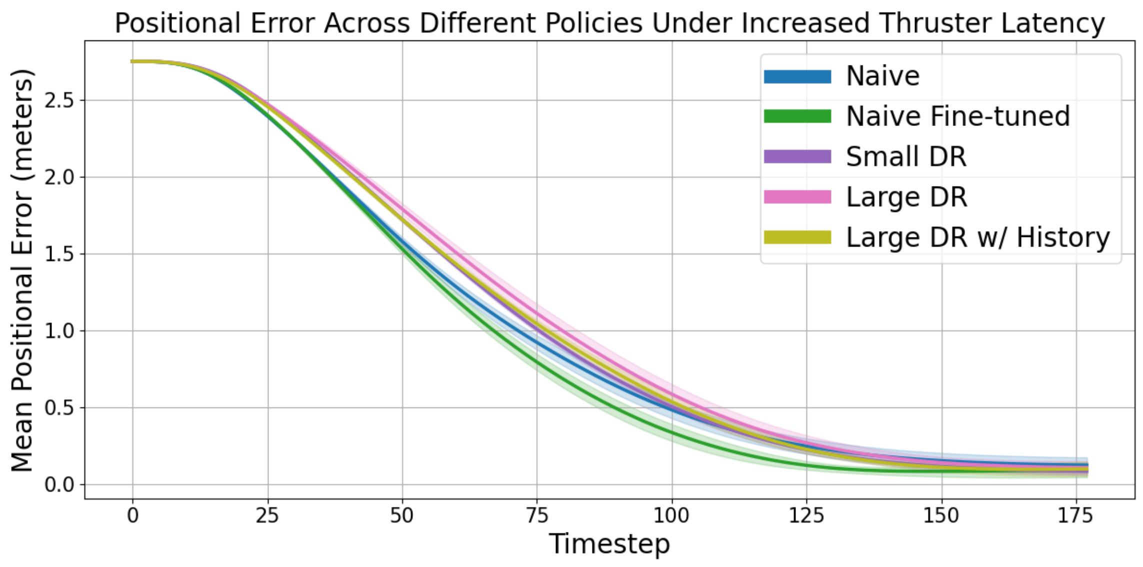

Figure 17: Positional error of different training configurations evaluated under increased thruster latency or **Latency** evaluation configuration.

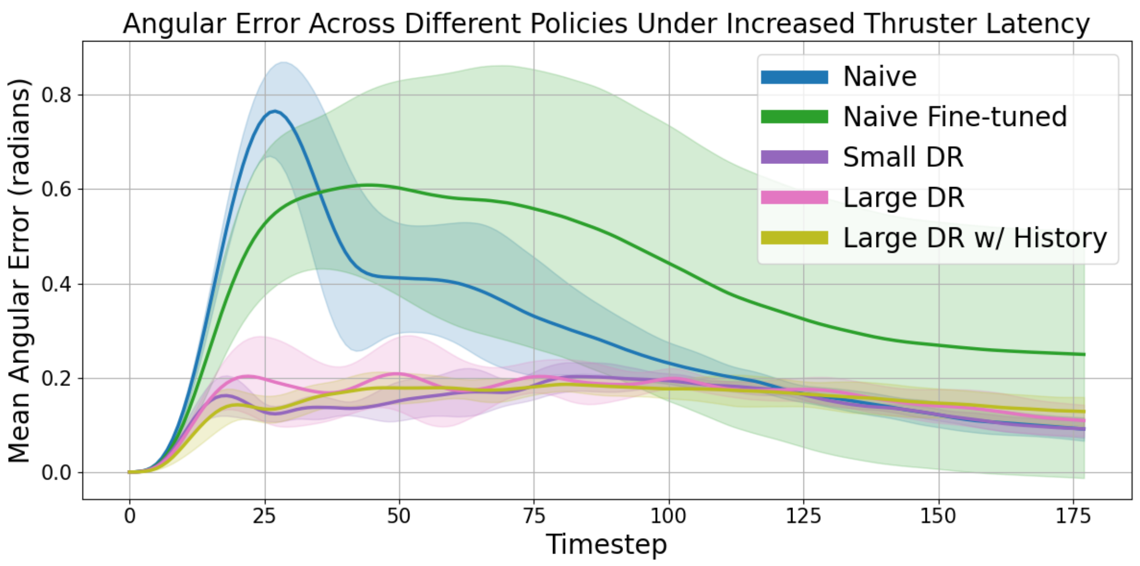

Figure 18: Angular error of different training configurations evaluated under increased thruster latency or **Latency** evaluation configuration.

When evaluating the positional errors produced by our various policies under increased thruster latency, shown in Figures 17, we observe that our naive and robust policies all roughly converge to the same positional error. However, we see unexpected results in the angular error shown in Figure 18 where the policies naively trained on the evaluation environment show the worst error and the highest variance. This is surprising, as it suggests overfitting, even though the training and evaluation environments are identical. We note that the robust policies achieve roughly the same angular error as the standard naively trained policy. Both the positional and angular error results suggest that using robust training techniques does not lead to any significant improvement when evaluated under increased latency conditions. Interestingly, we do not observe any improvements from using memory-based policies, possibly because memory does not offer a significant advantage against increased thruster latency.

### 8.3.7 Evaluation Under Randomly Disabled Thrusters

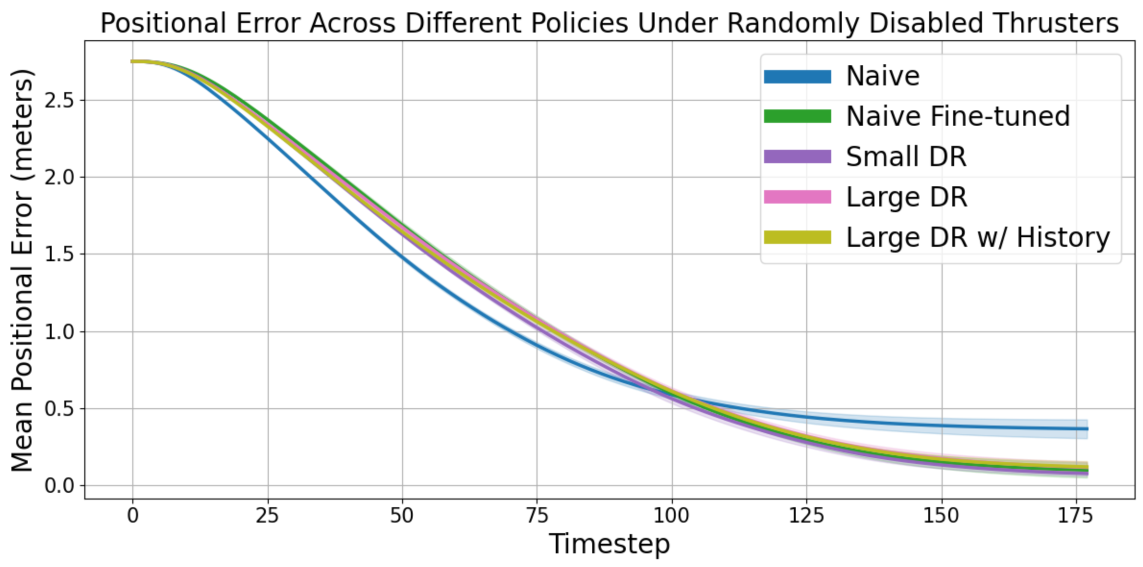

Figure 19: Positional error of different training configurations evaluated under randomly disabled thrusters or **Disable Thruster** evaluation configuration.

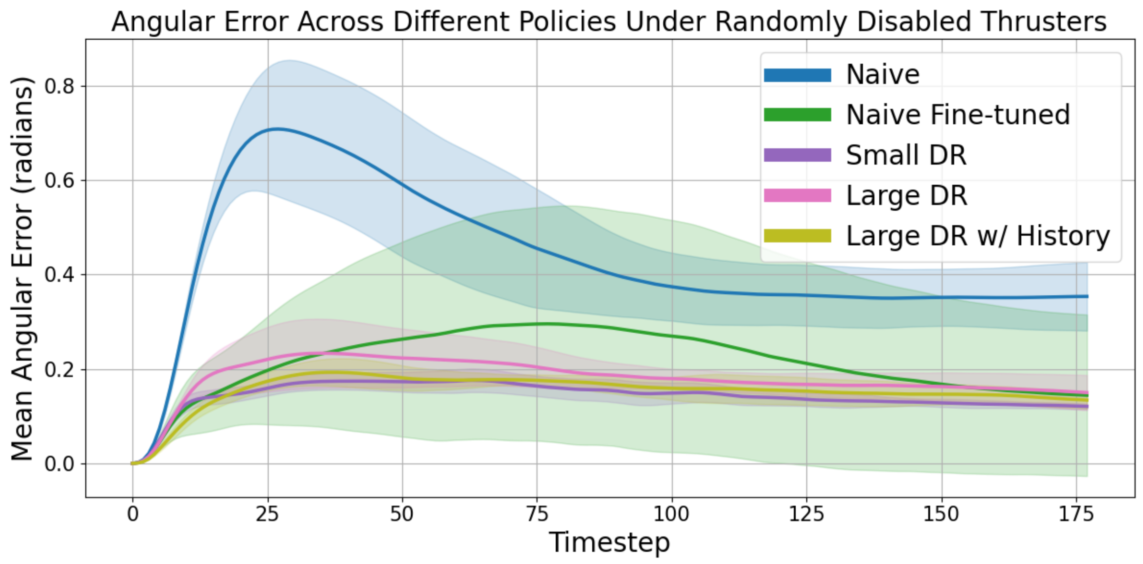

Figure 20: Angular error of different training configurations evaluated under randomly disabled thrusters or **Disabled Thruster** evaluation configuration.

Across the positional and angular error results produced by evaluating our different policies while randomly disabling thrusters shown in Figures 19 and 20 respectively, we see consistent improvements from our robust training techniques. In particular, we observe that our robust policies converge to the same positional and angular error as the fine-tuned naive policies, while achieving lower variance, especially in the angular case. This is a crucial result as it demonstrates that DR is sufficient to counteract the issue of random thruster failures. We suspect that using DR leads to a regularization effect similar to dropout in deep learning where our policy learns to not rely on any single thruster too heavily, but rather to distribute importance to each thruster relatively equally in order to minimize potential losses if it is to lose a thruster. We again observe that memory-based policies do not show any improvement, suggesting that either the state history does not sufficiently implicitly identify the thruster conditions or knowing that information does not provide the policy with a significant advantage. An interesting question is if disabling more thrusters during training and evaluation would lead to larger improvements from memory-based architectures since theoretically the purely reactive policies trained using DR would become overly conservative and would require a greater ability to condition their behavior.

### 8.3.8 Evaluation Under Waves and Currents

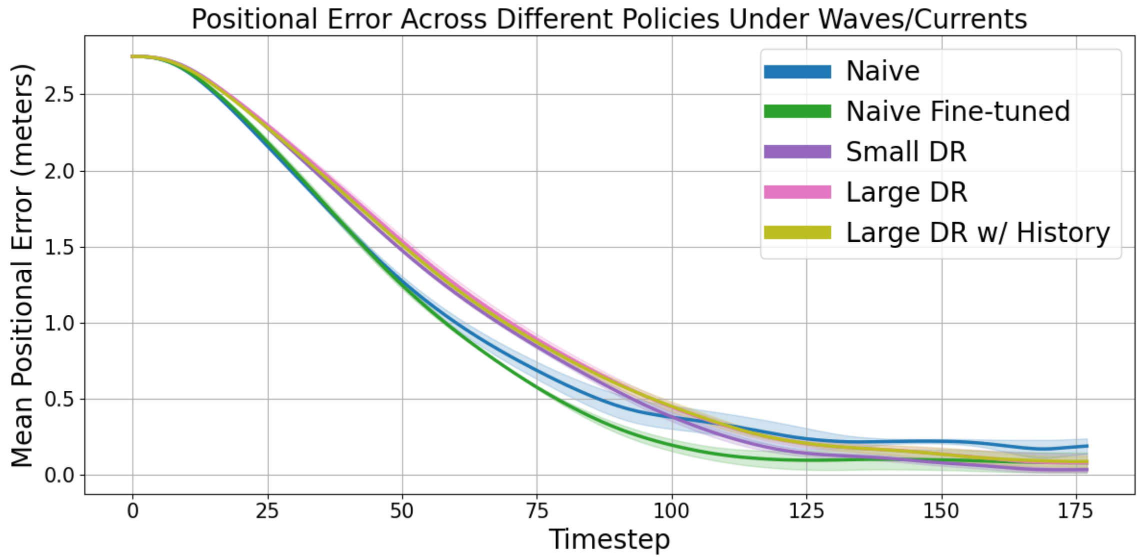

Figure 21: Positional error of different training configurations evaluated under waves and currents or **Waves/Currents** evaluation configuration.

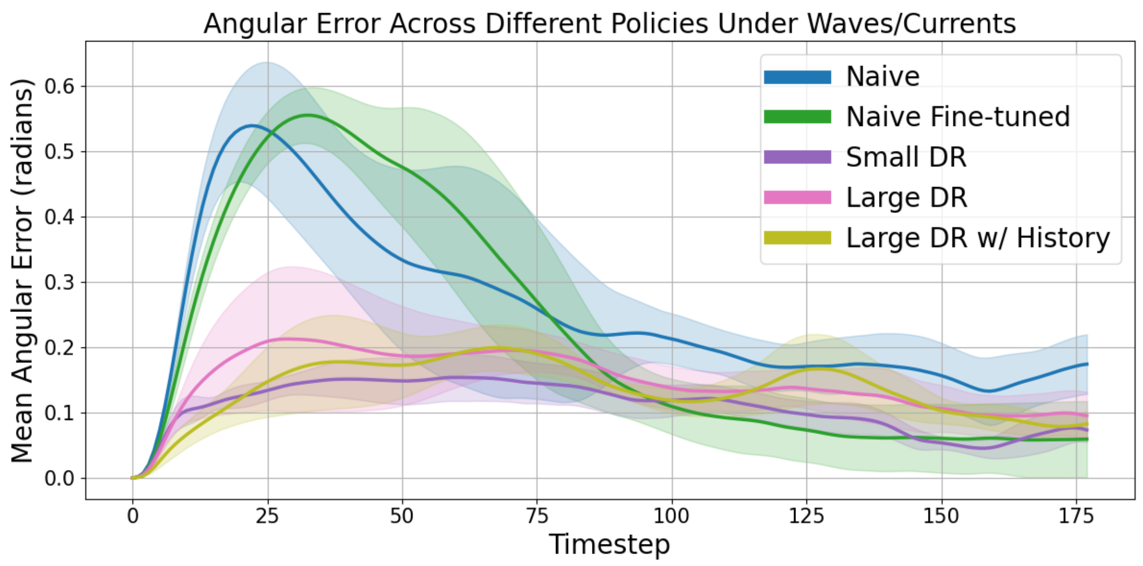

Figure 22: Angular error of different training configurations evaluated under waves and currents or **Waves/Currents** evaluation configuration.

Evaluating our different policies under wave and current disturbances, with error plots shown in Figures 21 and 22, we observe marginal improvements from our robust training techniques. Specifically, we again notice the pattern where the standard naive policies perform the worst, the fine-tuned naive policies perform the best, and then the robust policies attain an error slightly worse than the best fine-tuned policies. We again observe that our robust policies achieve lower variance. Overall, this again suggests that our robust training techniques lead to improvements in performance under real-world disturbances. We again observe no major difference between purely reactive policies and memory-based policies, potentially indicating that waves and currents are not a challenging enough disturbance to require further conditioning of a policy. It is also possible that the purely reactive policies can already roughly identify the wave and current conditions since different settings enable the agent to see relatively disjoint parts of the state space.

### 8.3.9 Evaluation Under Attached Payloads

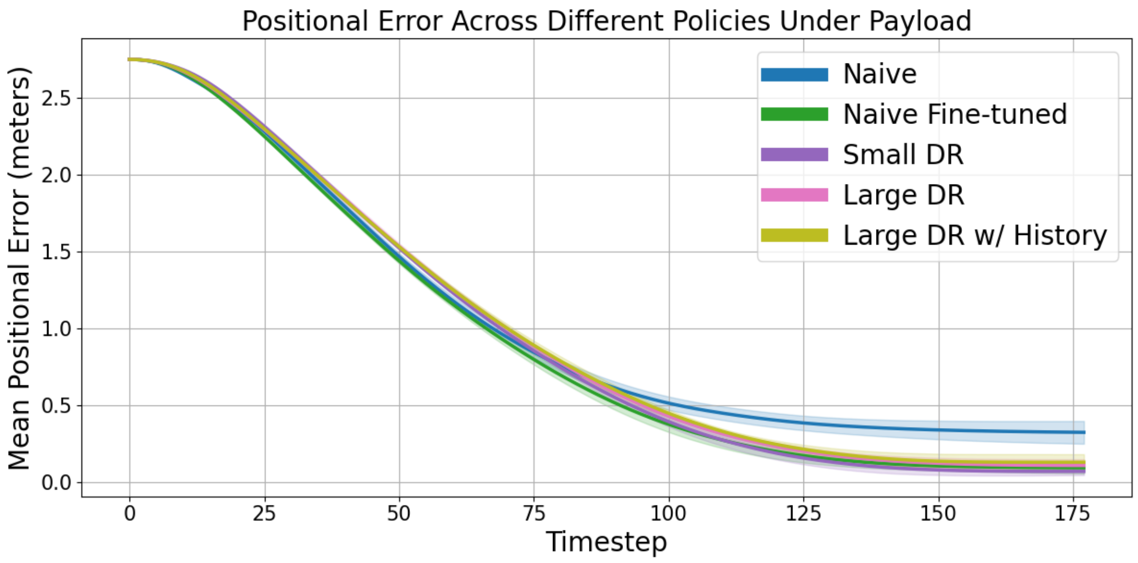

Figure 23: Positional error of different training configurations evaluated under payloads or **Payload** evaluation configuration.

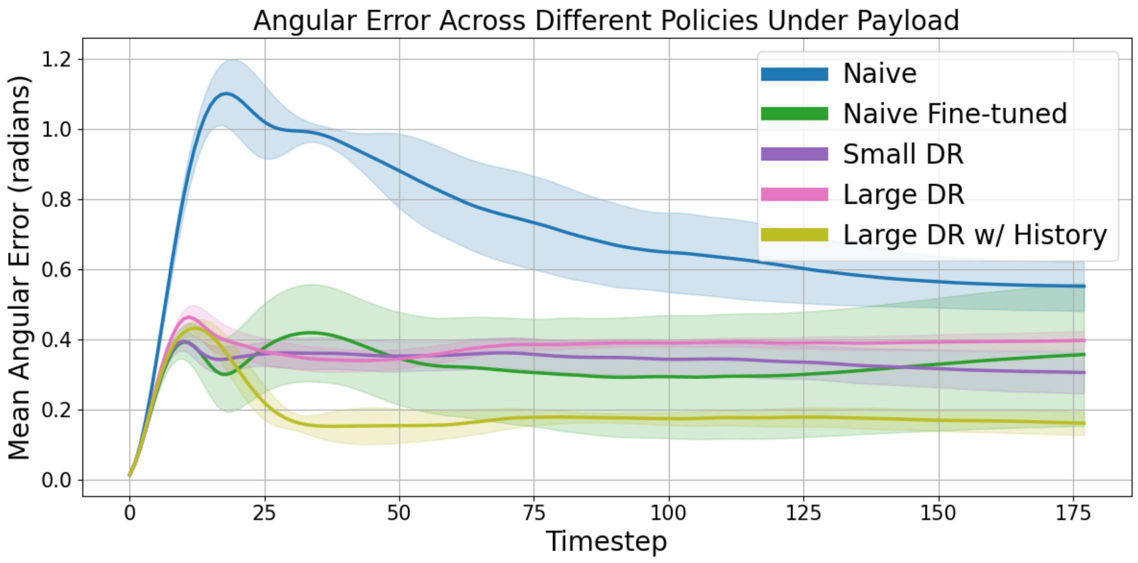

Figure 24: Angular error of different training configurations evaluated under payloads or **Payload** evaluation configuration.

When payloads are applied, we observe in the positional and angular error plots shown in Figures 23 and 24, that we see incredible performance improvements from using robust training techniques. In particular, we observe that with regards to positional error, our robust policies exceed the performance of the standard naive policies and converge to the same error as the fine-tuned naive policies. In terms of angular error, we see similar performance but interestingly we observe that incorporating memory improves performance beyond that of the fine-tuned naive policies. This is surprising, as training with history should only enable performance that is at most as high as a policy trained directly on the evaluation environment. More research will have to be done to further explore what is causing this.

## 9 Conclusions and Future Work

Through modeling different real-world disturbances and evaluating different policies under each one individually, as well as all of them combined, we have gained insights into the comparative impacts of each disturbance, developing an understanding of what factors contribute to the sim2real gap. In particular, we observe that for a naively trained policy, which we believe provides the best baseline analysis of effects of each disturbance, randomly disabling a thruster, shifting the CoB, and attaching a payload lead to the most significant degradations. On the other hand, waves and currents, and increased thruster latency only have marginal effects on the performance of a naively trained policy. These results are shown in Figures 6 and 7. Another important result is that different disturbances lead to varying amounts of improvements from incorporating memory into a policy. Overall, these results suggest that when seeking to minimize the sim2real gap in autonomous docking, the most important real-world disturbances to consider are thruster failures, CoB shifts, and payloads. On the other hand, thruster latency and waves and currents are potentially less crucial.

Throughout our robust training results, we continually see DR and incorporating memory leading to significant improvements in both positional and angular performance. In particular, we observe that these techniques improve performance under CoB shifts, thruster failures, waves and currents, and payloads. This suggests that when developing a learning-based controller for docking in the real-world, these techniques are appropriate methods for improving robustness, especially when the disturbances listed above are expected to be experienced during deployment. Additionally, we observe that incorporating memory often leads to performance improvements. Specifically, we notice that under a shifted CoB, an attached payload, and under easy and medium conditions where all disturbances are enabled to varying degrees, we see various improvements in angular error from incorporating memory. This could suggest that shifting the CoB and attaching payload is either a significant enough disturbance that memory becomes useful, or possibly that compared to other disturbances, it is particularly easy to identify a shift in the CoB or an attached payload from a history of states. Additionally, the fact that incorporating memory led to improvements in environments where all disturbances were present suggests that the interplay between specific disturbances when combined together creates a control problem where memory becomes necessary. Further research could potentially study this issue deeper through possibly attempting to train a decoder to identify the disturbance parameters from the state history in order to determine if an implicit system identification is even possible in certain cases. In general, we validate that DR is an appropriate technique for reducing the sim2real gap. Additionally, we find that incorporating memory into policies can lead to improved performance, although a potential open issue is that of overfitting to simulation, which we do not explore our work.

### 9.0.1 Future Work

In the future, our most important next step is to deploy our controllers in the real-world to validate our simulated findings. Importantly, we would want to first show that some learned policy can successfully perform docking in the real-world, as to the best of our knowledge this has yet to be demonstrated in literature. Furthermore, we would like to validate our findings regarding improvements in performance through robust training techniques. This may involve physically or electronically modifying an AUV by for example attaching different payloads or manually disabling thrusters. Additionally, we would like to compare our learning-based controllers against existing state-of-the-art docking controllers such as those that are based on MPC [8]. A challenging question we will have to answer will be how to design an experiment that most fairly and comprehensively compares the two methods. We will likely focus on analyzing each method's robustness to various real-world disturbances.

## References

[1] Apr 2025. [Online]. Available: https://education.nationalgeographic.org/resource/all-about-the-ocean/

[2] S. McCammon, S. Jamieson, T. A. Mooney, and Y. Girdhar, "Discovering biological hotspots with a passively listening auv," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 3789–3795.

[3] M. Estes, C. Anderson, W. Appeltans, N. Bax, N. Bednaršek, G. Canonico, S. Djavidnia, E. Escobar, P. Fietzek, M. Gregoire, E. Hazen, M. Kavanaugh, F. Lejzerowicz, F. Lombard, P. Miloslavich, K. O. Möller, J. Monk, E. Montes, H. Moustahfid, M. M. Muelbert, F. Muller-Karger, L. E. Peavey Reeves, E. V. Satterthwaite, J. O. Schmidt, A. M. Sequeira, W. Turner, and L. V. Weatherdon, "Enhanced monitoring of life in the sea is a critical component of conservation management and sustainable economic growth," *Marine Policy*, vol. 132, p. 104699, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308597X21003109

[4] S. Hong, J. McMorland, H. Zhang, M. Collu, and K. H. Halse, "Floating offshore wind farm installation, challenges and opportunities: A comprehensive survey," *Ocean Engineering*, vol. 304, p. 117793, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801824011314

[5] L. Cai, N. E. McGuire, R. Hanlon, T. A. Mooney, and Y. Girdhar, "Semi-supervised visual tracking of marine animals using autonomous underwater vehicles," *International Journal of Computer Vision*, vol. 131, no. 6, p. 1406–1427, Mar. 2023. [Online]. Available: http://dx.doi.org/10.1007/s11263-023-01762-5

[6] R. N. Smith, M. Schwager, S. L. Smith, D. Rus, and G. S. Sukhatme, "Persistent ocean monitoring with underwater gliders: Towards accurate reconstruction of dynamic ocean processes," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1517–1524.

[7] N. Kato, M. Choyekh, R. Dewantara, H. Senga, H. Chiba, E. Kobayashi, M. Yoshie, T. Tanaka, and T. Short, "An autonomous underwater robot for tracking and monitoring of subsea plumes after oil spills and gas leaks from seafloor," *Journal of Loss Prevention in the Process Industries*, vol. 50, pp. 386–396, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950423017302383

[8] R. Vivekanandan, D. Chang, and G. A. Hollinger, "Autonomous underwater docking using flow state estimation and model predictive control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1062–1068.

[9] J. Wallen, N. Ulm, and Z. Song, "Underwater docking system for a wave energy converter based mobile station," in *OCEANS 2019 MTS/IEEE SEATTLE*, 2019, pp. 1–8.

[10] Y. Girdhar, N. McGuire, L. Cai, S. Jamieson, S. McCammon, B. Claus, J. E. S. Soucie, J. E. Todd, and T. A. Mooney, "Curee: A curious underwater robot

for ecosystem exploration," in *2023 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, May 2023, p. 11411–11417. [Online]. Available: http://dx.doi.org/10.1109/ICRA48891.2023.10161282

[11] R. D. Patmore, D. Ferreira, D. P. Marshall, M. D. du Plessis, J. A. Brearley, and S. Swart, "Evaluating existing ocean glider sampling strategies for submesoscale dynamics," *Journal of Atmospheric and Oceanic Technology*, vol. 41, no. 7, pp. 647 – 663, 2024. [Online]. Available: https://journals.ametsoc.org/view/journals/atot/41/7/JTECH-D-23-0055.1.xml

[12] H. Hirai and K. Ishii, "Development of dam inspection underwater robot," *Journal of Robotics, Networking and Artificial Life*, vol. 6, p. 18, 01 2019.

[13] S. Hotta, Y. Mitsui, M. Suka, N. Sakagami, and S. Kawamura, "Lightweight underwater robot developed for archaeological surveys and excavations," *ROBOMECH J.*, vol. 10, no. 1, Jan. 2023.

[14] M. S. Mohd Aras, H. Kasdirin, M. Jamaluddin, and M. Basar, "Design and development of an autonomous underwater vehicle (auv-fkeutem)," *Proceedings of MUCEET2009 Malaysian Technical Universities Conference on Engineering and Technology, MUCEET2009, MS Garden, Kuantan, Pahang, Malaysia*, 01 2009.

[15] G. Dudek, P. Giguere, C. Prahacs, S. Saunderson, J. Sattar, L.-a. Torres-Mendez, M. Jenkin, A. German, A. Hogue, A. Ripsman, J. Zacher, E. Milios, H. Liu, P. Zhang, M. Buehler, and C. Georgiades, "Aqua: An amphibious autonomous robot," *Computer*, vol. 40, no. 1, pp. 46–53, 2007.

[16] K. Macauley, L. Cai, P. Adamczyk, and Y. Girdhar, "Reefglider: A highly maneuverable vectored buoyancy engine based underwater robot," 2024. [Online]. Available: https://arxiv.org/abs/2405.06033

[17] T. Alinei-Poiană, D. Reţe, D. Martinovici, V.-M. Maer, and L. Buşoniu, "A bluerov2-based platform for underwater mapping experiments," *IFAC-PapersOnLine*, vol. 58, no. 20, pp. 470–475, 2024, 15th IFAC Conference on Control Applications in Marine Systems, Robotics and Vehicles CAMS 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896324018536

[18] J. S. Willners, I. Carlucho, T. Luczynski, S. Katagiri, C. Lemoine, J. Roe, D. Stephens, S. Xu, Y. Carreno, È. Pairet, C. Barbalata, Y. R. Petillot, and S. Wang, "From market-ready rovs to low-cost auvs," *CoRR*, vol. abs/2108.05792, 2021. [Online]. Available: https://arxiv.org/abs/2108.05792

[19] E. Anderlini, G. G. Parker, and G. Thomas, "Docking control of an autonomous underwater vehicle using reinforcement learning," *Applied Sciences*, vol. 9, no. 17, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/17/3456

[20] M. Lin, R. Lin, C. Yang, D. Li, Z. Zhang, Y. Zhao, and W. Ding, "Docking to an underwater suspended charging station: Systematic design and

experimental tests," *Ocean Engineering*, vol. 249, p. 110766, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801822002141

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3–4, p. 229–256, May 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00992696

[22] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017. [Online]. Available: https://arxiv.org/abs/1502.05477

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[25] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ROS and gazebo," *CoRR*, vol. abs/1608.05742, 2016. [Online]. Available: http://arxiv.org/abs/1608.05742

[26] J. Deng, S. Marri, J. Klein, W. Pałubicki, S. Pirk, G. Chowdhary, and D. L. Michels, "Gazebo plants: Simulating plant-robot interaction with cosserat rods," 2024. [Online]. Available: https://arxiv.org/abs/2402.02570

[27] H. R. M. Sardinha, M. Dragone, and P. A. Vargas, "Closing the gap in swarm robotics simulations: An extended ardupilot/gazebo plugin," *CoRR*, vol. abs/1811.06948, 2018. [Online]. Available: http://arxiv.org/abs/1811.06948

[28] T. R. Player, A. Chakravarty, M. M. Zhang, B. Y. Raanan, B. Kieft, Y. Zhang, and B. Hobson, "From concept to field tests: Accelerated development of multi-auv missions using a high-fidelity faster-than-real-time simulator," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 3102–3108.

[29] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[30] H. Duan, B. Pandit, M. S. Gadde, B. van Marum, J. Dao, C. Kim, and A. Fern, "Learning vision-based bipedal locomotion for challenging terrain," 2024. [Online]. Available: https://arxiv.org/abs/2309.14594

[31] F. Yu, R. Batke, J. Dao, J. Hurst, K. Green, and A. Fern, "Dynamic bipedal maneuvers through sim-to-real reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2207.07835

[32] J. Dao, H. Duan, and A. Fern, "Sim-to-real learning for humanoid box loco-manipulation," 2023. [Online]. Available: https://arxiv.org/abs/2310.03191

[33] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, “Isaac gym: High performance gpu-based physics simulation for robot learning,” 2021. [Online]. Available: https://arxiv.org/abs/2108.10470

[34] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, “Orbit: A unified simulation framework for interactive robot learning environments,” *IEEE Robotics and Automation Letters*, vol. 8, no. 6, p. 3740–3747, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1109/LRA.2023.3270034

[35] L. Cai, K. Chang, and Y. Girdhar, “Learning to swim: Reinforcement learning for 6-dof control of thruster-driven autonomous underwater vehicles,” 2025. [Online]. Available: https://arxiv.org/abs/2410.00120

[36] M. Shafiee, G. Bellegarda, and A. Ijspeert, “Manyquadrupeds: Learning a single locomotion policy for diverse quadruped robots,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 3471–3477.

[37] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, “Sim-to-real: Learning agile locomotion for quadruped robots,” *CoRR*, vol. abs/1804.10332, 2018. [Online]. Available: http://arxiv.org/abs/1804.10332

[38] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning robust perceptive locomotion for quadrupedal robots in the wild,” *CoRR*, vol. abs/2201.08117, 2022. [Online]. Available: https://arxiv.org/abs/2201.08117

[39] X. B. Peng, E. Coumans, T. Zhang, T. E. Lee, J. Tan, and S. Levine, “Learning agile robotic locomotion skills by imitating animals,” *CoRR*, vol. abs/2004.00784, 2020. [Online]. Available: https://arxiv.org/abs/2004.00784

[40] J. Siekmann, K. Green, J. Warila, A. Fern, and J. W. Hurst, “Blind bipedal stair traversal via sim-to-real reinforcement learning,” *CoRR*, vol. abs/2105.08328, 2021. [Online]. Available: https://arxiv.org/abs/2105.08328

[41] B. Pandit, A. Gupta, M. S. Gadde, A. Johnson, A. K. Shrestha, H. Duan, J. Dao, and A. Fern, “Learning decentralized multi-biped control for payload transport,” 2024. [Online]. Available: https://arxiv.org/abs/2406.17279

[42] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, “Reinforcement learning for versatile, dynamic, and robust bipedal locomotion control,” 2024. [Online]. Available: https://arxiv.org/abs/2401.16889

[43] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, “Learning high-speed flight in the wild,” *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abg5810

[44] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza, "Bootstrapping reinforcement learning with imitation for vision-based agile flight," 2024. [Online]. Available: https://arxiv.org/abs/2403.12203

[45] Y. Song, K. Shi, R. Penicka, and D. Scaramuzza, "Learning perception-aware agile flight in cluttered environments," 2023. [Online]. Available: https://arxiv.org/abs/2210.01841

[46] W. Lu, K. Cheng, and M. Hu, "Reinforcement learning for autonomous underwater vehicles via data-informed domain randomization," *Applied Sciences*, vol. 13, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/3/1723

[47] S. Cowen, S. Briest, and J. Dombrowski, "Underwater docking of autonomous undersea vehicles using optical terminal guidance," in *Oceans '97. MTS/IEEE Conference Proceedings*, vol. 2, 1997, pp. 1143–1147 vol.2.

[48] R. Stokey, M. Purcell, N. Forrester, T. Austin, R. Goldsborough, B. Allen, and C. von Alt, "A docking system for remus, an autonomous underwater vehicle," in *Oceans '97. MTS/IEEE Conference Proceedings*, vol. 2, 1997, pp. 1132–1136 vol.2.

[49] M. Patil, B. Wehbe, and M. Valdenegro-Toro, "Deep reinforcement learning for continuous docking control of autonomous underwater vehicles: A benchmarking study," *CoRR*, vol. abs/2108.02665, 2021. [Online]. Available: https://arxiv.org/abs/2108.02665

[50] T. Zhang, X. Miao, Y. Li, L. Jia, Z. Wei, Q. Gong, and T. Wen, "Auv 3d docking control using deep reinforcement learning," *Ocean Engineering*, vol. 283, p. 115021, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801823014051

[51] N. Palomeras and P. Ridao, "Autonomous underwater vehicle docking under realistic assumptions using deep reinforcement learning," *Drones*, vol. 8, no. 11, 2024. [Online]. Available: https://www.mdpi.com/2504-446X/8/11/673

[52] D. Yoerger, J. Cooke, and J.-J. Slotine, "The influence of thruster dynamics on underwater vehicle behavior and their incorporation into control system design," *IEEE Journal of Oceanic Engineering*, vol. 15, no. 3, pp. 167–178, 1990.

[53] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1512.04455