Università degli studi di Verona

· **DIPARTIMENTO DI INFORMATICA** ·

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Tesi di Laurea Magistrale

Deep Reinforcement Learning for Autonomous Boat Navigation

Development of a realistic simulation environment.

github.com/cesare-montresor/barchette

Author

Cesare Montresor

Matricola VR481252

cesare.montresor@studenti.univr.it

Advisor

Alessandro Farinelli

Anno Accademico 2022-2023

Contents

List of Figures

List of Tables

List of Abbreviations

AI: Artificial Intelligence

NN: Neural Network

ML: Machine Learning

DL: Deep Learning

DNN: Deep Neural Networks

RL: Reinforcement Learning

DRL: Deep Reinforcement Learning

SDRL: Safe Deep Reinforcement Learning

Abstract

Autonomous boat navigation has demonstrated significant potential for applications in maritime transportation, water and environmental analysis, search and rescue operations, and scientific exploration. The primary objective of this project is to develop a versatile and robust simulation environment that can serve as a Digital Twin for testing safely and efficiently autonomous navigation systems for boats.

Deep Reinforcement Learning (DRL) techniques have emerged as highly effective approaches for training autonomous agents in complex and unpredictable environments, which involve precise, otherwise potentially critical, machine-human, machine-environment, or machine-machine interactions, such as self-driving cars, kitchens, agriculture, and production lines.

Simulated environments and digital twins have become crucial in developing autonomous robotics solutions for real-life critical applications. They significantly reduce training costs and time, making the process more affordable, efficient, and reproducible. This project aims to provide a comprehensive framework for developing and verifying DRL and Safe DRL techniques. It includes a ready-to-use physical simulation of water, waves, and obstacles, specifically designed for the port of the city of Garda. Additionally, the project offers Python code for training the autonomous agent, both with and without safety constraints. It also provides all the necessary assets, tools, and instructions for easily customizing the simulator to different real-world locations.

By focusing on the development of a realistic physical interaction between water, boat geometries, and sensor responses in various weather conditions, this project lays the foundations for training boats to perform complex tasks. Although the experiments in this study are limited to simple way-point navigation using GPS coordinates, they serve as a starting point for future endeavours involving multi-agent environments, pollutant source tracing, and intricate obstacle avoidance scenarios.

1 Introduction

1.1 Motivations and Objectives

In recent years, the field of Machine Learning and Artificial Intelligence has experienced remarkable growth and achieved significant results, largely attributed to the advancements in Deep Neural Networks and back-propagation. One particularly powerful approach that has emerged is Deep Reinforcement Learning (DRL), which has shown versatility and effectiveness across various domains, promising to optimize and automate tasks.

DRL algorithms enable agents to explore and develop sophisticated strategies by establishing long-term correlations between actions and rewards. However, applying these techniques in real-life environments remains a challenge due to the complex nature of real-world data. High dimensionality, unstructured formats, and noise in the data pose difficulties for agents to make sense of the world and extract relevant information for the task at hand.

This thesis focuses on the field of maritime navigation, specifically training autonomous boats to navigate safely using inputs solely from sensors typically found on boats, such as GPS, IMU, compass, and sonar.

Given the cost, time, and complexities associated with real-world boat operations, utilizing simulated environments becomes an attractive choice. Simulated environments provide a cost-effective, efficient, and safe means for the agent to learn the basics of navigation before transitioning to real-world scenarios. This approach parallels the training of aspiring human pilots using flight simulators.

Significant effort has been devoted to creating a realistic physical interaction between the boat, water, and the boat's geometries and masses, as well as simulating sensor responses under various weather conditions. While this project offers the potential to train boats for various complex tasks, such as tracing pollutant sources in multi-agent environments with obstacles, the current experiments are limited to simple way-point navigation using GPS

coordinates. These experiments serve as a starting point to evaluate and compare the performance of several DRL for learning this initial task, laying the foundations for future advancements.
The application of Safe Deep Reinforcement Learning (SDRL) algorithms, a fast-growing branch of DRL, is probably the most interesting evolution for this kind of task, probably the latest step before being able to deploy trained agents in real-life, with confidence.

1.2 Thesis Outline

This section provides an overview of the main topics covered in each chapter:

- **Chapter 1: Introduction**

  This chapter introduces the project and its objectives, emphasizing the importance of Deep Reinforcement Learning (DRL) in the domain of autonomous boat navigation. It presents the motivation behind the project and outlines the structure of the thesis.

- **Chapter 2: Background**

  This chapter presents the core concepts of RL and DRL. It explains the foundational concepts behind the mathematical formulation of RL and provides a theoretical background for the subsequent chapters.

- **Chapter 3: Simulator Development**

  This chapter analyzed all individual components used in the realization of this project, mentioning each of the tools used, and why they were chosen.

- **Chapter 4: Methodology**

  This chapter focuses on clarifying the goal of the testing. Explains in detail the task used to train the agent. It also gives details on the methodology used to run and access the experiments.

- **Chapter 5: Experiments and Results**

  This chapter focuses on the experimental scenarios, tasks, and evaluation

procedures used in the project. It presents the details of the training and evaluation process, including the performance metrics and measures employed.

- **Chapter 6: Discussion and Conclusions**

  This chapter summarizes and discusses the findings and outcomes of the project. It offers a concise summary of the project's main contributions, implications and limitations of the project and suggests potential future research directions.

1.3 State-of-the-art: DRL Algorithms

The current advancements in open-water navigation involve the development of techniques to enable autonomous systems to navigate effectively in unstructured marine environments. This field faces various challenges, such as unpredictable weather conditions, collision avoidance, and path planning. To address these challenges, state-of-the-art approaches combine deep reinforcement learning with perception modules that process data from sensors such as sonar, lidar, and cameras. These approaches leverage advanced neural network architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to handle the complexity of marine environments and learn efficient navigation policies. Additionally, advancements in Simultaneous Localization and Mapping (SLAM) techniques contribute to accurate positioning and mapping, further enhancing the navigation capabilities of autonomous systems.
When it comes to autonomous boat navigation, the focus lies on training boats to navigate safely and efficiently without human intervention. Advanced perception systems, such as GPS, IMU, compass, and sonar, provide crucial information for navigation. Deep reinforcement learning algorithms learn policies that enable boats to make optimal decisions based on the current state and desired objectives while considering safety constraints and environmental factors.
By integrating these advancements in perception, deep reinforcement

learning, and navigation algorithms, autonomous boats can navigate open water environments autonomously and safely. Ongoing research continues to refine and improve these techniques, paving the way for widespread applications of autonomous boats in transportation, surveillance, and environmental monitoring. [18][14][8][48][19]

Two prominent algorithms in these domains are Proximal Policy Optimization (PPO)[41] and Soft Actor-Critic (SAC)[19]. PPO, introduced in 2017, is an on-policy algorithm that has garnered attention for its impressive performance across various domains. It has become a benchmark for comparison in many DRL tasks. SAC, on the other hand, was introduced in 2018 as an off-policy algorithm that combines the benefits of maximum entropy reinforcement learning and off-policy updates. This combination leads to improved stability and sample efficiency.

For this work, I choose PPO as it natively supports discrete actions. In addition to PPO I also made use of REINFORCE as it provides a faster convergence. A more detailed analysis between SAC and PPO and between REINFORCE and PPO can be found in the background chapter.

1.4 State-of-the-art: DRL Simulators

Simulators are an integral component of a DRL setup. The use of simulators allows for cheap, safe, fast and repeatable experiments, allowing researchers to efficiently test and iterate on their DRL solutions.

Furthermore, simulators offer unique opportunities. For example, that can be parallelized and the time inside the simulator can be "compressed" allowing for otherwise impossible to achieve results. An exemplary case of this advantage is the numbers reported by the DeepMind team during the training of AlphaStar. The whole training process took 14 days but allowed the agent to gather experience equivalent to 200 years of gameplay.[9]

Simulators commonly used as training environments for reinforcement learning algorithms include among others AI Habitat, DeepMind Control Suite, DeepMind Lab, MuJoCo, ROS & Gazebo, and Unity3D with

MLAgents and several others.[24]

- **AI Habitat**

  AI Habitat is a virtual embodiment simulator that provides an efficient photo-realistic 3D environment for training RL agents. It aims to enable research in embodied AI by providing realistic environments with various visual and physical properties.[40][45][2]

- **DeepMind Control Suite**

  DeepMind Control Suite focuses on continuous control tasks in physics-based simulation environments. It provides a set of benchmark tasks that involve controlling robotic systems to achieve specific goals, such as object manipulation or locomotion.[10]

- **DeepMind Lab**

  DeepMind Lab is a 3D navigation and puzzle-solving simulator developed by DeepMind. It offers a platform for training RL agents in complex environments that require exploration, navigation, and solving cognitive tasks.[11]

- **MuJoCo**

  MuJoCo (Multi-Joint dynamics with Contact) is a physics engine that provides a full-featured simulator for modelling complex mechanisms. It is designed for model-based optimization and supports simulation in generalized coordinates with accurate contact dynamics. MuJoCo is known for its speed, accuracy, and modelling power, making it suitable for applications in robotics, bio-mechanics, graphics, animation, and more. It was acquired by Deep-Mind and made freely available as an open-source project.[35].

- **ROS and Gazebo**

  ROS (Robot Operating System) is a flexible framework for writing robot software. It includes simulation capabilities through packages such as Gazebo. Gazebo is a widely used 3D simulator that provides a realistic environment for simulating robots and their interactions with the

surroundings. It supports physics-based simulation, sensor simulation, and visualization. Gazebo is often used in conjunction with ROS for developing and testing robot control algorithms.[37][16]

- **Unity with MLAgents**

  Unity with MLAgents extension offers a framework that combines the Unity game engine with machine learning capabilities. It allows researchers and developers to train agents in Unity's 3D environments using reinforcement learning techniques. MLAgents provides tools for creating and customizing simulation environments, provides basic sensors and actuators, as well as some fully functioning scenes, integrating machine learning algorithms and collecting training data. It also provides ready-to-go SDK and code to make it compatible with other standards, such as ROS2 or Gym. The great advantage that only Unity3D offers depends on having been around for a very long time, the first user-friendly and free-to-use 3D game engines available today. Because of this, it has the best possible collections of pre-made plugins and assets.[47][32]

Most simulators used for reinforcement learning, including MuJoCo, ROS with Gazebo, and Unity with MLAgents, focus on simulating physics to some extent. However, it is true that many simulators do not fully simulate complex physical phenomena and often rely on rigid body simulation, especially when it comes to tasks such as robotic arm control. Simulating more complex phenomena such as soft-bodies dynamics (e.g.: water, textiles, smoke) can be challenging and computationally expensive, which is why they are not as commonly simulated in DRL environments.
In the context of maritime navigation, there is a need for a dedicated simulator because boat navigation is a niche field within reinforcement learning. The existing simulators often focus on classic problems such as robotic arms or self-driving cars, which are more widely studied and researched. Therefore, a specific maritime navigation simulator was developed to provide a comprehensive framework for training autonomous boats and ships to navigate safely in open waters. This simulator

incorporates realistic physical interaction between the boat, water, and sensor responses in various weather conditions, addressing the unique challenges of boat navigation. The development of such a simulator offers an easily accessible and customizable simulators for training and testing autonomous boat navigation algorithms.

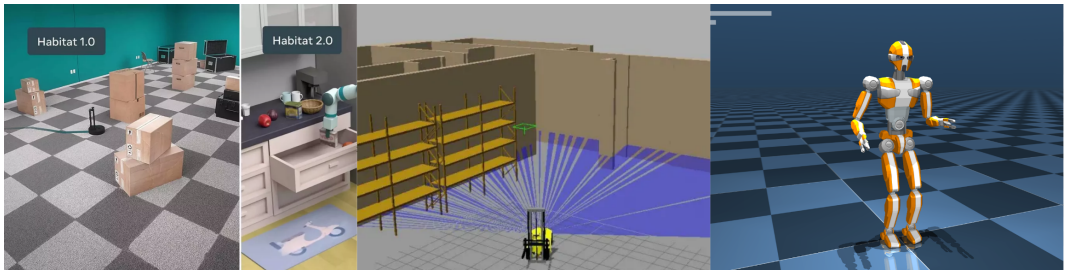

Figure 1.1: From left to right: AI Habitat, GazeboSim, MuJoCo

1.5 Digital Twin

A Digital Twin is a virtual replica of a physical object or system that spans its life-cycle. Digital twins are focused on specific physical entities or systems, such as machines, buildings, or processes. They aim to provide a real-time, data-driven replica that reflects the behaviour and performance of the physical counterpart. While a simulator can be a Digital Twin, not all simulators aim to model a physical reality. On the contrary, most simulators used today in the context of DLR research are rarely intended as Digital Twins of some physical entity.

Digital twin technology has gained significant traction in recent years and plays a crucial role in the development and testing of autonomous systems, including boats and ships. A digital twin is a virtual representation of a physical object, process, or system that replicates its behaviour and characteristics in a simulated environment. In autonomous boat navigation, a digital twin serves as a virtual counterpart of a real boat, allowing for comprehensive testing, training, and evaluation of navigation algorithms and control systems.

The development of a digital twin for autonomous boat navigation involves creating a realistic simulation of the boat, its physical properties, and the

surrounding environment. This includes modelling the boat's dynamics, propulsion systems, sensor inputs, and the behaviour of water, waves, and obstacles. By accurately representing these aspects, the digital twin provides a reliable and safe environment for training and testing autonomous navigation algorithms.

The use of a digital twin offers several advantages. Firstly, it allows for cost-effective and efficient training of autonomous agents, as reduces the need for more expensive and time-consuming real-world experiments. Secondly, it provides a controlled and reproducible environment, enabling researchers to evaluate and compare different algorithms and approaches systematically. Additionally, the digital twin can simulate various scenarios and conditions, including various weather conditions, sea states, and navigational challenges, ensuring that the trained agents are robust and capable of handling real-world situations.

In the context of this thesis, the development of a digital twin for autonomous boat navigation involves integrating accurate physical behaviour of water and waves into the simulator. By incorporating these aspects into the simulator, the digital twin can provide a realistic environment for training and evaluating the autonomous navigation system.

2 Background

The history of reinforcement learning can be traced back to various disciplines. In animal learning, Edward Thorndike introduced the concept of trial-and-error learning in 1911, describing how animals reinforce behaviours that lead to satisfaction and deter behaviours that produce discomfort.
The Law of Effect, proposed by Thorndike, became a foundational principle for understanding reinforcement in animal psychology[49]. The application of reinforcement learning principles in animal behaviour suggests that animals are capable of learning and optimizing rewards.
Over time, reinforcement learning has evolved and gained significance in various domains. It has been applied in game theory[36], control theory[7], operations research[21], information theory[39], simulation-based optimization[17] and multi-agent systems[52]. In recent years, deep reinforcement learning, which combines RL with deep neural networks, has gained attention for its ability to handle complex environments and achieve remarkable results in domains such as board games and computer games.[9]

2.1 Reinforcement Learning

Reinforcement learning (RL) in Computer Science is a machine learning paradigm concerned with how intelligent agents can make sequential decisions in an environment to maximize cumulative rewards. Unlike supervised learning that relies on labelled input/output pairs or unsupervised learning that seeks patterns in unlabeled data, RL focuses on finding a balance between exploration (discovering actions that can provide high rewards) and exploitation (leveraging existing knowledge) to optimize long-term rewards.[44][38]
The fundamental parts of a reinforcement learning setup include:

- **Agent**

  The agent is the learner or decision-maker in the reinforcement learning system. It interacts with the environment and learns through trial and

error to maximize the cumulative reward. The agent takes actions based on the current state and receives feedback in the form of rewards.

- **Environment**

  The environment is the external system with which the agent interacts. It provides the agent with observations (state information) and receives actions from the agent. The environment determines the next state based on the actions taken and provides rewards to the agent accordingly.

- **State**

  The state represents the current situation or configuration of the environment. It captures the relevant information that the agent needs to make decisions. The state can be fully observable or partially observable, depending on whether the agent has access to complete or incomplete information about the environment.

- **Action**

  Actions are the decisions made by the agent based on the observed state. The agent selects actions from a set of available actions according to its policy. The policy defines the agent's behaviour and determines how it maps states to actions.

- **Reward**

  The reward signal is the feedback provided to the agent after each action. It represents the desirability or quality of the agent's actions in a given state. The agent's objective is to maximize the cumulative reward over time. Rewards can be positive, negative, or zero, indicating good, bad, or neutral outcomes, respectively.

These fundamental components interact with each other in a cyclic manner. The agent perceives the state from the environment, selects an action based on its policy, and receives a reward from the environment. This process repeats over multiple iterations, allowing the agent to learn and improve its decision-making through trial and error.

It's important to note that reinforcement learning setups can also include additional elements such as a value function, which estimates the expected cumulative reward from a given state, and a model of the environment, which the agent can use for planning and prediction. However, the four components mentioned above (agent, environment, state, and action) form the core elements of a basic reinforcement learning setup.

Markov Decision Process

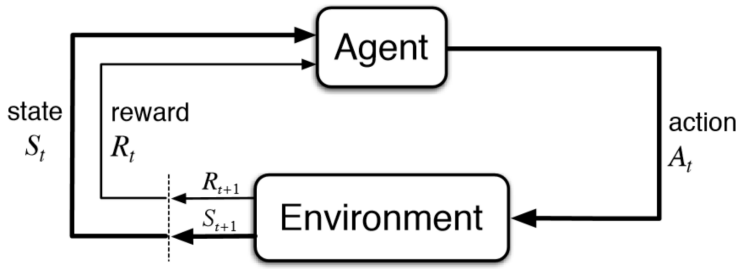

Figure 2.1: The agent–environment interaction in a Markov decision process.

Reinforcement Learning (RL) is formulated mathematically using the Markov Decision Process (MDP). The agent's goal is to find an optimal policy that maximizes the expected cumulative reward over time. MDPs provide the formal framework for modelling the decision-making problem faced by the agent.

The MDP assumes the Markov Property, which states that the future state and rewards depend only on the current state and action, and not on the past history. In other words, all the past history is now fully represented in the current state.

The objective in an MDP is to find a policy π that maps each state s to an action a (i.e., $\pi(s) = a$) in order to maximize the expected cumulative reward over time. To further explain, at each time step, the MDP is in a particular state $s \in S$, and the decision maker selects an action $a \in A$ based on the current state. The MDP then transitions to a new state s' according to the probability distribution $P(s' \mid s, a)$. Additionally, the decision maker receives an immediate reward of $R(s, a, s')$ for the transition.

The objective is to find the policy π that maximizes the expected cumulative reward. This can be achieved by using various algorithms and techniques,

such as value iteration or policy iteration, to iteratively update the value function or the policy until convergence is reached. Here is the mathematical formulation of an MDP [44][38]:
A Markov Decision Process is defined by 4 components:

$$\langle\ S, A, P, R\ \rangle$$

- S is the set of states, representing all possible configurations of the system.
- A is the set of actions, representing all possible actions the agent can take.
- P is the state transition function $P(s' \mid s, a)$, which specifies the probability of transitioning to state s' given that the system is in state s and the agent takes action a.
- R is the reward function $R(s, a, s')$, which determines the immediate reward received when transitioning from state s to state s' by taking action a.

While this is the basic formulation of an MDP, there are variations that extend the basic MDP framework to address specific characteristics or requirements of decision-making problems. Each variation introduces additional complexity and challenges in modelling and solving the corresponding problem.

Bellman Equation

The Bellman equation is a recursive equation that relates the value of a state or state-action pair to the expected rewards obtained from taking an action and transitioning to the next state. The Bellman equation plays a crucial role in solving MDPs, it's used to calculate the optimal value function, which in turn helps determine the optimal policy. The Bellman equation allows the agent to update its value estimates iteratively, taking into account the expected rewards and values of future states.
Here is the mathematical formulation of the Bellman Equation [44][38]:

$$Q(s,a) = E[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')]$$

- $Q(s,a)$ is the value of taking action a in state s.
- $R(s,a)$ is the immediate reward received when taking action a in state s.
- γ is the discount factor, a value between 0 and 1 that determines the importance of future rewards.
- $P(s' \mid s,a)$ is the probability of transitioning to state s' when taking action a in state s.
- $\max_{a'} Q(s',a')$ represents the maximum Q-value among all possible actions a' in the next state s'.

This equation captures the idea of optimizing the Q-values by considering the immediate reward and the expected future rewards. The goal is to find the optimal policy that maximizes the expected cumulative reward over time.
It's important to note that the specific formulation and notation of the Bellman equation may vary depending on the context and notation used in different resources. The equation presented here is a common representation in the field of reinforcement learning.

2.2 Deep Reinforcement Learning

Deep Reinforcement Learning combines RL with Deep Learning techniques, specifically Deep Neural Networks, to handle complex and high-dimensional state spaces. DRL leverages the representation learning capabilities of DNN to extract meaningful features from raw input data, enabling the agent to learn directly from high-dimensional sensory inputs such as images or audio. The DNN serve as a function approximator to estimate the value function or policy in RL.
DRL has achieved remarkable successes in various domains, including playing complex games such as AlphaGo[42] and Atari games[34], robotic control[28],

and autonomous driving[6]. DRL may require a large number of interactions with the environment to learn effectively, making sample efficiency an important consideration.[20][15][29]
The first significant result of DRL was the Deep Q-Network (DQN) that was introduced by researchers at DeepMind in a paper published in 2015 [34].
The first DQN was specifically designed and tested on the Arcade Learning Environment (ALE). Introduced in 2013 as a platform for evaluating general AI agents in a wide range of Atari 2600 games. The ALE provided a suite of challenging games with different dynamics and complexities, serving as a benchmark for testing RL algorithms.
The original DNQ implementation included the following features:

- **DQN Algorithm**

  The DQN algorithm combines Q-learning, which is a value-based RL algorithm, with deep neural networks to approximate the action-value function (Q-function) of an RL agent. Instead of using hand-engineered features as inputs. The network takes a state as input and outputs Q-values for each possible action. The agent selects actions based on an epsilon-greedy policy, balancing exploration and exploitation.

- **Deep Neural Networks**

  Thanks to the application of DNN is possible for DQN to directly processes such as pixels from game screens, using deep convolutional neural networks (CNNs)

- **Experience Replay**

  DQN introduced the concept of experience replay, which addresses the issues of data efficiency and sample correlation. Experience replay involves storing the agent's experiences, consisting of state, action, reward, and next-state transitions, in a replay buffer. During training, mini-batches of experiences are sampled randomly from the buffer, enabling the agent to learn from diverse and uncorrelated experiences, leading to more stable learning.

- **Target Network**

  To improve the stability of learning, DQN utilizes a separate target network, which is a copy of the main Q-network. The target network's parameters are updated less frequently, providing more consistent target values during training. This decouples the target estimation from the network's updates and helps mitigate issues caused by the non-stationarity of the targets.

2.3 DRL Algorithms: PPO

PPO (Proximal Policy Optimization) incorporates or inherits techniques from various works and methodologies in the field of reinforcement learning. Here is a list of some of the techniques that PPO includes or draws inspiration from

- **Actor-Critic**

  PPO's actor-critic architecture enhances the training stability and performance of the algorithm. By leveraging the strengths of both policy-based and value-based methods, PPO can overcome challenges such as slow progress and poor sample efficiency.

- **Policy Gradient Methods**

  PPO builds upon the foundations of policy gradient methods, which optimize the policy directly by estimating the gradient of the expected return.

- **Trust Region Policy Optimization (TRPO)**

  PPO improves upon TRPO by using a more conservative approach to policy updates, addressing some of the limitations of TRPO.

- **Clipped Surrogate Objective**

  PPO introduces a clipped surrogate objective function, which approximates the performance improvement and constrains the policy update to ensure more stable learning.

- **Proximal Policy Optimization**

  PPO utilizes a proximal policy optimization approach that restricts the policy update to a local region, preventing large policy divergences and ensuring more stable learning.

- **Adaptive Step Sizes**

  PPO dynamically adjusts the step sizes during policy optimization to prevent large policy updates that may disrupt learning and stability.

- **Value Function Approximation**

  PPO can incorporate value function approximation techniques to estimate the state-value function and improve policy performance.

- **Parallelization**

  PPO can leverage parallelization methods, such as running multiple agents or simulations concurrently, to collect more diverse and efficient data for policy updates.

These techniques, combined and tailored specifically for PPO, contribute to its effectiveness in reinforcement learning tasks, providing stability, improved sample efficiency, and enhanced performance.

2.4 DRL Algorithms: REINFORCE

The REINFORCE is a Monte Carlo variant of a policy gradient algorithm. The agent collects samples of an episode using its current policy and uses it to update the policy. Since one full trajectory must be completed to construct a sample space, it is updated as an off-policy algorithm.
Here are the main features of the REINFORCE algorithm:

- **Policy-based approach**

  The REINFORCE algorithm directly learns a policy function, which maps states to actions, without explicitly estimating value functions such as in value-based methods.

- **Stochastic policy**

  REINFORCE uses a stochastic policy that outputs a probability distribution over actions given a state. This allows for exploration and randomness in action selection.

- **Monte Carlo sampling**

  REINFORCE relies on Monte Carlo sampling to estimate the expected return for each state-action pair. It collects trajectories by interacting with the environment, samples actions according to the policy, and computes the gradients based on the obtained rewards.

- **Policy gradient optimization**

  The REINFORCE algorithm updates the policy parameters using gradient ascent on the expected return. It maximizes the expected return by adjusting the policy distribution to favour actions that lead to higher rewards.

- **Baseline function**

  To reduce the variance of the policy gradient estimates, REINFORCE often incorporates a learned baseline function. The baseline is subtracted from the estimated return to reduce the impact of high or low rewards that are not indicative of the policy's quality.

These features make the REINFORCE algorithm a flexible and effective method for training agents in various reinforcement learning tasks.

2.5 PPO & REINFORCE Trade-offs

While the goal is to use PPO (Proximal Policy Optimization), I initially decided to use REINFORCE in the beginning. Despite its shortcomings. Using REINFORCE allows for much faster convergence and enables the execution of a larger number of benchmarks, useful for hyper-parameter tuning. Here are some of the considerations taken into account when making the choice:

- **Faster Convergence**

  REINFORCE is a simple policy gradient algorithm that updates the policy based on the gradients of the expected return. It doesn't involve training a critic network, which is typically done in methods such as PPO. By omitting the training of a critic, the training process becomes computationally less expensive and converges faster. This enables quicker iterations and experimentation with different settings and parameters.

- **Simplicity**

  REINFORCE is a straightforward algorithm that is relatively easy to understand and implement. It doesn't involve complex mechanisms such as trust regions or value function approximation used in PPO. This simplicity allows for a more rapid development and iteration process. It also reduces the time spent on implementation and debugging.

- **Benchmarking**

  By using REINFORCE initially, it becomes feasible to run a larger number of benchmarks to evaluate the performance of the policy. Since REINFORCE converges faster, it allows for more iterations within a given time frame. This increased efficiency in benchmarking enables a better understanding of the strengths and weaknesses of the policy in different scenarios and facilitates the identification of areas for improvement.

- **Exploration-Exploitation Trade-off**

  REINFORCE has a straightforward exploration-exploitation trade-off. It explores the environment by sampling actions according to the current policy, allowing for a diverse range of actions to be tried. This exploration can be particularly useful in the early stages of training when the agent's policy is far from optimal. Once the policy starts converging, exploitation becomes more dominant, and the agent focuses on exploiting the learned policy.

- **Parallelism (lack of)**

  Collecting data from multiple trajectories at once reduces correlation in

the dataset. This improves convergence for online learning systems such as neural networks, which work best with. [33] Data collection is faster overall, which improves clock time to obtain the same result. This may make better use of other resources too.[33] Both the simulator and the learning algorithm are computationally demanding. Mostly due to the limitation imposed by the ZibraAI plugin, compiling only on Windows and OSX, making it unsuitable for the Linux cluster typically used by the University of Verona for this kind of project. Thus, at the time of writing, I didn't have access to the computational resources necessary for leveraging parallelism, which would have most likely reduced significantly the time and the stability of the PPO's training. Further details on the ZibraLiquid limitations and the hardware setup used, can be found respectively in sections 3.2 and 5.3 of this thesis.

It's important to note that while REINFORCE has advantages in terms of faster convergence and simplicity, it also has limitations such as higher sample complexity and variance. However, in the initial stages of the project, these drawbacks were deemed acceptable given the benefits mentioned above. As the project progresses and more data becomes available, transitioning to more advanced algorithms such as PPO can be considered to further refine the policy and improve performance.

3 Simulator Development

3.1 Unity3D

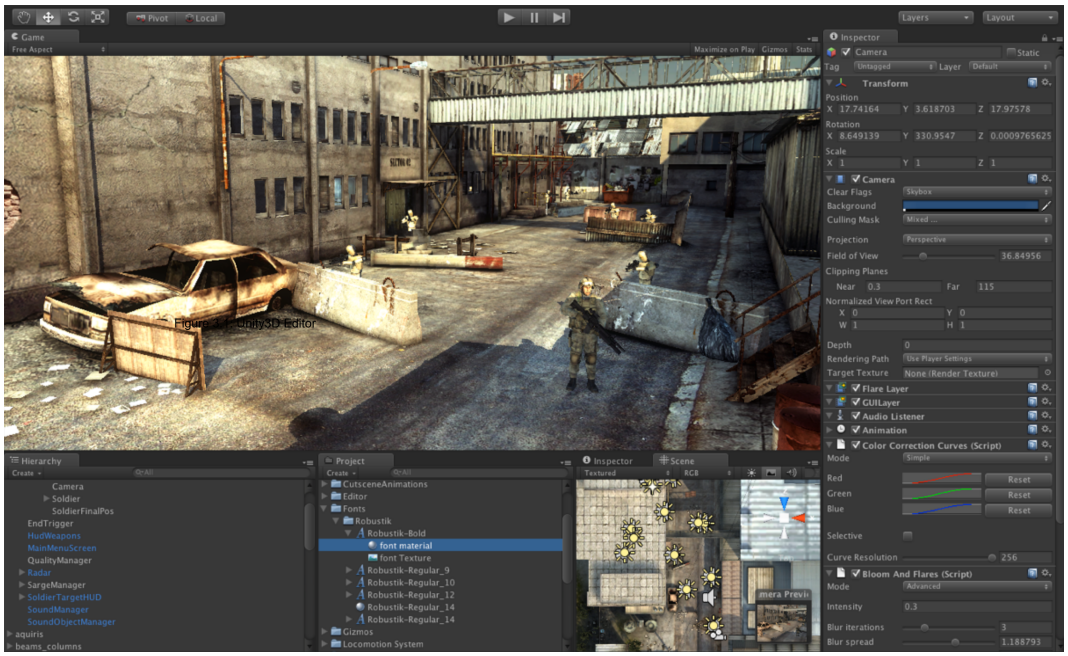

Figure 3.1: Unity3D Editor

Unity3D is a cross-platform game engine developed by Unity Technologies. It was first announced and released in June 2005 at the Apple Worldwide Developers Conference as a Mac OS X game engine. Over the years, Unity3D has been extended to support various platforms, including desktop, mobile, console, and virtual reality.[50]

Unity3D is particularly popular for iOS and Android mobile game development and is widely used by indie game developers due to its ease of use. It allows developers to create both three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences.[50]

The game engine's objective since its launch has been to "democratize" game development by making it accessible to a wider audience of developers. Unity3D has gained significant traction in the gaming industry and has also been adopted by other industries such as film, automotive, architecture, engineering, construction, and the United States Armed Forces.[50][46]

As for the current state of the Unity3D ecosystem, it continues to thrive with regular updates and releases. Unity Technologies maintains an active documentation website providing guides and resources for developers and has an extensive asset store where developers can access a wide range of ready-to-use assets, plugins, and tools to enhance their game development process.
Moreover, Unity3D offers a free version, allowing developers to get started without any upfront costs.
Overall, Unity3D remains a leading game engine with a robust ecosystem, extensive documentation, and a thriving community of developers. Its versatility, ease of use, and cross-platform capabilities contribute to its popularity among both novice and experienced game developers. The closest competitor is represented by Unreal Engine, which, however, just recently has picked up the challenges and opportunities offered by DRL. UE in general is considered less user-friendly and entry-level than Unity3D but offers state-of-the-art results both in terms of performance and effectivness.

3.2 ZibraLiquids

Figure 3.2: ZibraLiquids Editor

ZibraLiquids is a Unity3d plugin offered by ZibraAI, a deep-tech company that focuses on building a platform for creating AI-generated assets for

virtual worlds and experiences. The plugin offers several options including fire, smoke, explosions, force-fields and more.
For my purposes, I used the part of realistic liquids simulation that allows users to add interactive, real-time 3D liquid to the projects. The plugin offers a wide range of parameters to configure visual and physical properties to create different types of liquids.
The pipeline is very efficient due to the leveraging of DNN at various points, from generating assets to controlling collisions with complex geometries. The plugin allows you to run a live simulation inside the editor and save the result. Later when the simulation begins the saved particles will be loaded always in the same starting state, contributing to a higher reproducibility.
Despite the efficiency of the pipeline the team still discourages the use of the plugin for large bodies of water. Even with high-end hardware simulating liquids can be computationally demanding because it involves modelling and calculating the behaviour of a very large number of particles composing the liquid. The more particles or elements involved in the simulation, the more computational resources are required to perform the necessary calculations.
It may require several iterations of tine-tuning, in order to obtain a large enough body of water while maintaining the performance under control and the physical simulation reliable. The main approach to achieve this is scaling down the boat and adjusting the masses of the various parts accordingly while tuning the water parameters. However, a long and probably imperfect manual process can be improved, once the first agent is finally deployed in the real environment. By having an accurate digital twin and the sensor data coming from the real environment, it should be possible to automatically tune the parameters of the water to match the effect produced on the sensors over time. This approach, over time, should improve the overall training efficiency.
The ZibraLiquid plugin doesn't support for Linux. While the team is confident to release it in the future, at the time of writing it is not yet available. This is generally not a favourable factor as traditionally all the software stack for RL runs on Linux. Practically limiting access to

computational power.

On the upside, the ZibraAI team offers very rapid and effective support via their Discord channel. They also provide free educational licenses, which facilitate the development of the simulator.

3.3 Blender

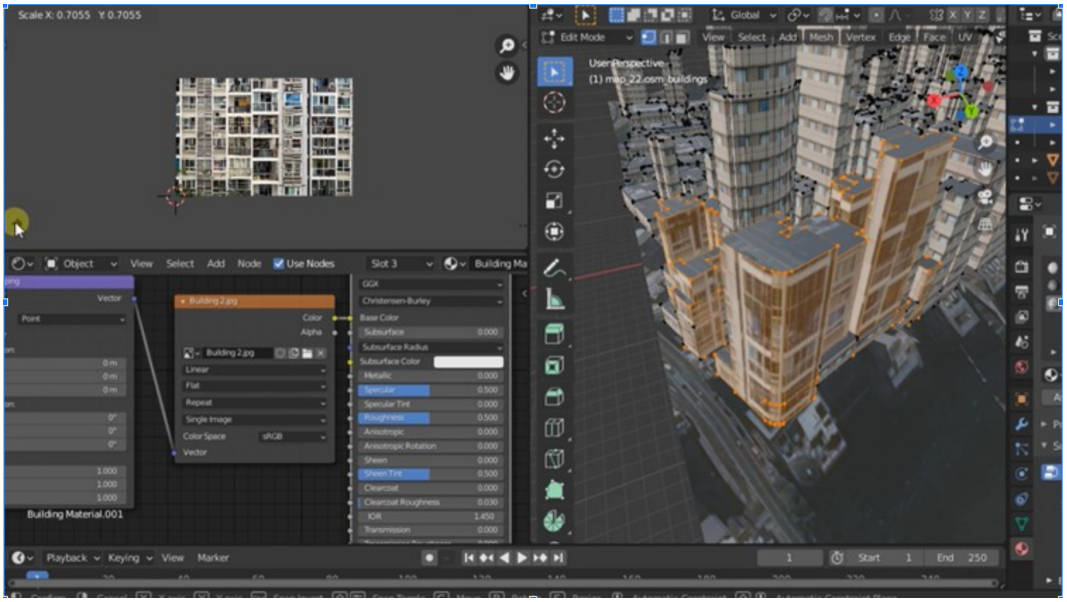

Figure 3.3: Blender Editor

Blender is a free and open-source 3D creation suite that offers a wide range of tools for modelling, rendering, animation, and more [3]. It is suitable for various media production purposes, including animations, game assets, motion graphics, TV shows, concept art, commercials, and feature films [3]. Blender is known for its extensive modelling toolset, which includes features such as sculpting, retopology, and curve modelling[4].

In the context of the simulator, Blender has been used to model all assets. The simulator imports models created in Blender. Currently, there are three models of boats available in the simulator. Two of them are modelled after a simple keel, fin and bulb design. These two boats come with two different engine types: a differential engine and a tilt engine.

The third is modelled after the real boat that will be deployed in the port of Garda by the end of the year and aims to become the Digital Twin of the

real boat. Features 2 differential engines and a mainly flat keel, with catamaran-like hulls on the side, to improve control.
The bulb keel design and the flat-catamaran hull design, in combination with the two engine types, provide a small starting set of parts that can be combined differently to test various configurations.

3.4 Geolocalization

Figure 3.4: Geolocalization and GPS Beacon

Using the GIS Blender plugin, a coarse but realistic reconstruction of the port of the City of Garda has been made in Blender. This reconstruction utilizes satellite data obtained via Google Maps APIs and includes various elements such as altitude, buildings, roads, and waterways. The plugin allows for the import of 3D models from Google Maps into Blender, providing a convenient way to incorporate real-world geographical data into Blender projects, and allowing for further customization and manipulation.[5].
In the case of the water part of the map, a manual process was employed to replace it with a real bathymetry of Lake Garda, ensuring an accurate representation of the lake's depths and contours. This manual intervention ensures that the water portion of the scene accurately reflects the characteristics of the actual lake.

By combining the GIS Blender plugin's capabilities with satellite data from Google Maps, the port of the Garda scene in Blender achieves a realistic representation of the geographical features and structures present in that location. This integration of real-world data enhances the visual fidelity and accuracy of the scene, providing a more immersive and authentic experience for users.

The final model, including terrain, altitudes, buildings, satellite images and bathymetry, has been imported inside the Unity3D and the whole scene has been geolocalized using manually placed "GPS Beacons" in various parts of the map, together with precise information about latitude, longitude and altitude. Thanks to the satellite image embedded in the terrain, geolocalizing the scene using this method becomes a trivial task.

The GPS Beacons are part of the newly developed GPS sensor and will be discussed in the next section.

3.5 MLAgents

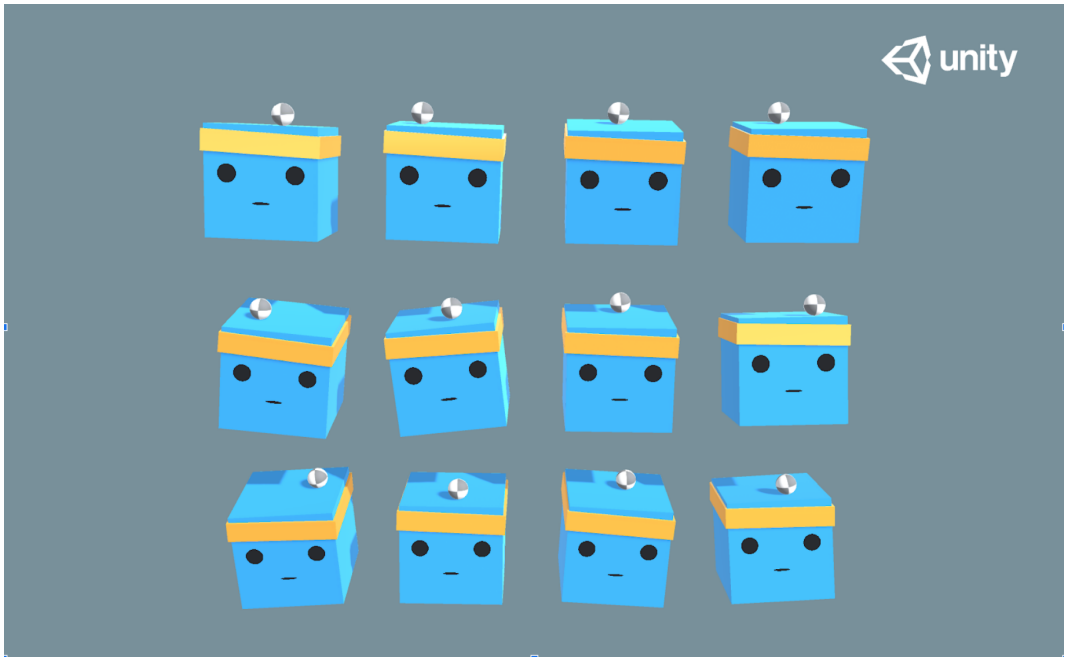

Figure 3.5: MLAgents example scene

In recent years, more and more people have adopted the use of MLAgents, an official Unity3D extension developed by former employee Arthur Juliani, for

developing reinforcement learning environments, now an official Unity3D plugin. MLAgents provides state-of-the-art machine learning capabilities that allow developers to create intelligent character behaviours in any Unity environment, including games, robotics, and film [30].

MLAgents offers several features out of the box that make it a valuable tool for developing reinforcement learning environments. It provides a C# SDK that can be integrated into Unity projects, allowing developers to convert any Unity scene into a learning environment for training intelligent agents. With MLAgents, developers can define agents, which are entities that generate observations, take actions, and receive rewards from the environment [31].

One of the key benefits of MLAgents is its compatibility with OpenAI Gym, which allows to leverage the extensive ecosystem of reinforcement learning algorithms and techniques available.[30]. This compatibility enables easy integration with existing reinforcement learning workflows.

3.6 Sensor & Actuators

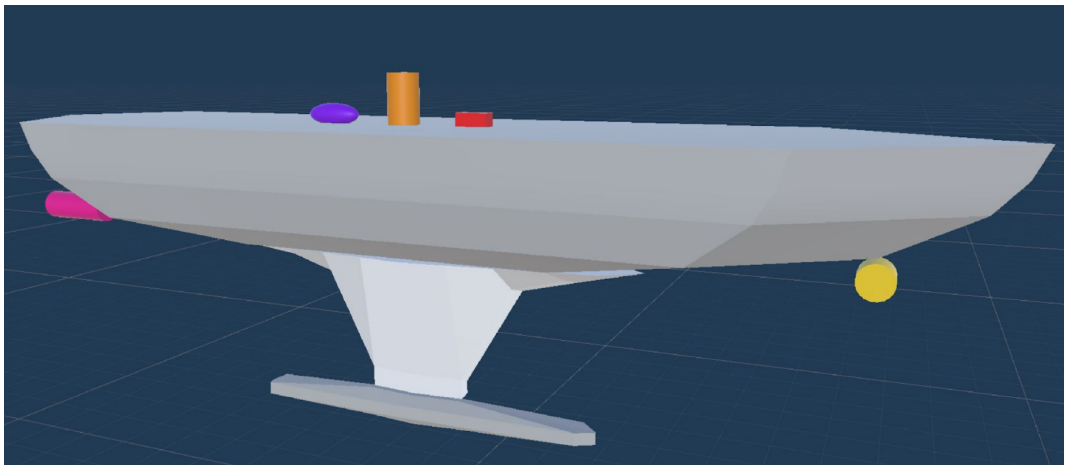

Figure 3.6: Sensor and actuators. From left to right: engine (pink), GPS(purple), lidar(orange), imu(red), sonar(yellow)

All the sensors that have been developed for this project are designed as standalone components, modular, and reusable. They follow the best practices for extending MLAgents, ensuring compatibility and ease of integration.

Having modular and reusable sensors is essential for assembling and

customizing new boats with ease. This modular design allows for flexibility in selecting and arranging sensors, adapting them to the unique characteristics and needs of different boats or environments. It simplifies the process of creating custom sensor configurations, without requiring extensive modifications or redesign.

Moreover, the modular and reusable nature of these sensors extends beyond the initial project. They can be utilized as drop-in components in other projects, allowing for efficient knowledge transfer and leveraging existing sensor implementations. This reusability not only saves development time but also promotes collaboration and knowledge sharing within the MLAgents community, enabling developers to build upon and enhance existing sensor functionality.

The list of sensors made available for this project includes:

- **GPS (Global Position System)**

  Provides accurate location information by utilizing a network of satellites. The GPS Sensor also provides the GPS Beacons objects that can be placed in the scene.

  Each beacon will read the position of the others and compute the proportion between latitude, longitude, and altitude of the real world and the x, y, and z of the simulator providing an easy way to convert from one to the other and vice-versa. Then GPS sensors simply query periodically the beacons and stream the data to the agent.

- **IMU (Inertial Measurement Unit)**

  Combines an accelerometer, gyroscope, and magnetometer sensors to measure acceleration, rotation, and orientation. The simulators provide the deltas of rotation and position of the x, y, and z-axis.

- **Compass**

  Measures the direction or bearing relative to the Earth's magnetic field. In the simulator, the north pole is extrapolated via GPS data and the angle between the orientation of the boat and the north point is calcu-

lated.

- **Sonar**

  Utilizes sound waves to measure distances underwater and detect objects. Inside the simulator is realized using ray casting to measure the distance from the seafloor. In this setup the sonar has only 1 ray in front at 30° angle, to detect immense obstacles, the shore and the docks. While the GPS sensor can give clues about the direction to take, the same makes sure that direction is navigable. The sensor is built upon the MLAgents standard sensor RayPerceptionSensor.

- **Lidar (Light Detection and Ranging)**

  Uses laser beams to measure distances and create detailed 3D maps of the surroundings. Inside the simulator, it works in the way of the sonar. The addition of a lidar would allow for collision avoidance which is an essential part of safe autonomous navigation. However, in the case of boats, which are constantly tilting, the lidar sensor can be not as effective.

- **Engine Immersion sensor**

  Designed to detect when the boat's engine is outside of the water, is built by using a particle detector provided by Zebra Liquid plugin which counts how many particles of water are in the area of the engine. During navigation, especially in the presence of waves, the engines may be exposed, outside of the water. This is an undesirable condition because leaves the boat without control, loses efficiency and the exposed propellers may be dangerous. The simulator correctly scales down the power of the engine proportional to its level of immersion.

4 Methodology

The simulator has been built thinking about a series of possible tasks and scenarios and therefore the project supports a variety of different sensor configurations. Also, environmental components, such as the docks and the shore, sensitive to collisions, or the waver generators, can be easily enabled and disabled to obtain all sorts of scenarios. It is a sensible choice to start from the most basic sensor and environment setup and the simplest possible task, to first establish a baseline. Establishing a simple first baseline is important to be able to measure the impact of changes to hyperparameters or other aspects of the training.

4.1 Reward Shaping Techniques

Reward shaping in the context of reinforcement learning (RL) refers to the process of modifying the reward signal to facilitate the learning process and improve the performance of RL agents.
In RL, agents learn to maximize cumulative rewards by interacting with an environment. However, the original reward signal provided by the environment may be sparse or provide insufficient guidance for effective learning. A classic problem that challenged all DRL algorithms before is part of the ALE environments is Montezuma Revenge.
The sparsity of the reward is at the core of the challenge posed by this environment several techniques have been employed to alleviate the problem, among which, are various forms of Reward Shaping [25]
Reward Shaping techniques aim to address this limitation by incorporating additional rewards or penalties based on specific criteria.
In the given scenario, the distance between the boat and the target is used as a negative reward signal. The intention is to discourage the boat from being far away from the target, the negative signal also motivates the agent to finish as fast as possible.
To make the reward signal richer and more dense information, it is divided

into concentric rings around the target. Each ring alters the basic reward signal by adding bonuses or penalties. The configuration of these bonuses and penalties, along with their multipliers or fixed values, varies in different experiments. Being in faraway areas from the target is penalized more heavily than being in just far areas. The general sense of this approach is to enrich the reward signal so the agent can sense the difference between "good" and "really good" states or between "bad" and "really bad" states. The objective is to provide a more nuanced reward signal that guides the agent towards optimal behaviour. Also, reaching the external borders of the simulators and the surrounding areas has been penalized, a similar approach can be used to encourage the agent to avoid collisions.
By shaping the reward signal in this way, the RL agent can learn to navigate efficiently towards the target while avoiding undesirable areas. This technique helps in achieving better performance and can be applied in various RL domains, including navigation tasks, as shown in the experiment. I experimented with 4 different types of reward shaping: PLAIN, SQUARED, ADDICTIVE bonus and MULTIPLICATIVE malus. Here are the details of each:

- **PLAIN**

  The reward is equal to $-(distance)$ providing a linear signal.

- **SQUARED**

  The reward is equal to $-(distance^2)$ providing a non-linear signal that increases very fast with higher distances.

- **ADDICTIVE**

  The reward is computed such as PLAIN plus an additive bonus. The region around the target is divided into 3 rings respectively at 25, 50 and 75 meters. Being within a circle would award a fixed bonus of respectively +0.003, +0.002 and +0.001.

- **MULTIPLICATIVE**

  The reward is computed such as PLAIN plus an additive bonus. The

region around the target is divided into 6 concentric rings starting at 20 covering up to 300 meters, each ring having equal width. Each circle has a different multiplier associated starting from the closest to the target: 1, 2, 4, 8, 16, 32. When multiplied by the reward (negative of the distance) acts as a malus, penalizing far away areas much more than closer ones.

4.2 Consideration of Weather Conditions

The addition of wave generators and the simulation of forces such as water currents can increase the level of difficulty of a task in reinforcement learning (RL) and improve the safety skills of the RL agent. By incorporating wave generators and simulating water currents, the RL agent is exposed to more complex and dynamic environments.

These forces create disturbances and perturbations in the environment, making it more challenging for the agent to maintain stability and achieve its objectives. The RL agent needs to learn how to navigate through the waves, adjust its movements to counteract the wave-induced forces and maintain control of the system.

Simulating water currents further adds to the complexity of the task. Water currents can affect the agent's motion, creating additional resistance or pushing the agent in certain directions. The RL agent must learn to adapt its actions based on the direction and strength of the currents, making navigation more challenging and requiring advanced control strategies.

The system can be utilized by incorporating random variations of episodes during more advanced stages of training. By introducing episodes with wave generators and water currents, the agent can gradually learn to handle these complex conditions and improve its safety manoeuvres and energy-efficient behaviour. This approach allows the agent to gain experience and develop strategies to navigate through waves and currents effectively while optimizing its performance.

4.3 Task Definition

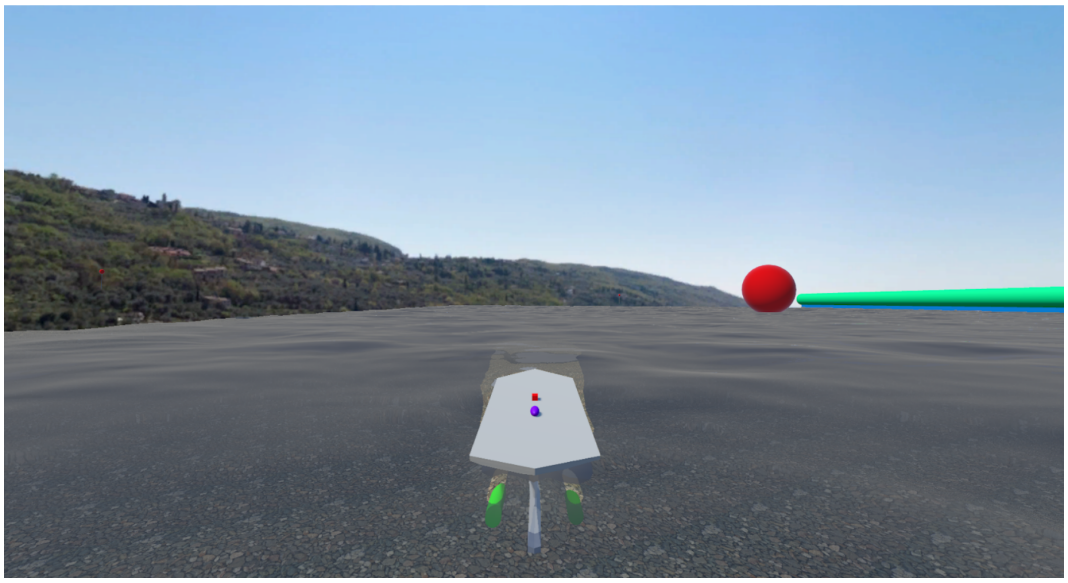

Figure 4.1: Waypoint navigation task setup

The goal of this work is to assess the replicability of the simulated scenarios while ensuring that DRL algorithms are capable of learning. I used a task of GPS waypoint navigation with fixed starting positions and rotations.
The agent is provided with a single GPS waypoint, which consists of latitude and longitude coordinates. The objective is to autonomously navigate from the agent's current location to each of the defined waypoints[51]. The agent needs to determine the optimal set of actions and adjust its movements to reach the waypoint as quickly as possible. The fixed positions and rotations in this task allow for controlled testing and evaluation of the agent's navigation capabilities.
This task served as a foundation for testing and refining various configurations of sensor data, reward-shaping techniques, algorithms, and hyperparameters.
The primary DRL algorithm I employed for this phase was REINFORCE. Given the limited computational power, I utilized short training sessions of 100 episodes of 500 steps each, to access the agent under various conditions. By experimenting with different combinations of sensor data, reward shaping techniques, algorithms, and hyperparameters, I aimed to identify the most effective configuration that maximized the agent's navigation capabilities in

the GPS waypoint navigation task. This approach enabled us to refine the system and optimize its performance before extending it to handle the more challenging task of GPS waypoint navigation with randomized locations.
All the experiments have been performed using a simple, single fixed random seed chosen randomly by hand. Optimizing and averaging over multiple random seeds, which is a common practice in DRL, aims to improve the performance of the Agent, which however is not the main focus of this work. In this experimental setup, the choice of use of a single random seed increases the replicability and comparability of the results.

4.4 Action-Space

In the context of Deep Reinforcement Learning (DRL), the choice between a discrete action space and a continuous action space has important implications. Discrete action spaces, which consist of a finite set of actions represented by discrete values or categories, are generally considered preferable to continuous action spaces. This is because discrete action spaces offer simplicity and sample efficiency.
Simplicity is a key advantage of discrete action spaces. They are easier to define and understand compared to continuous action spaces, making it simpler to design and implement the decision-making process of the agent. Discrete actions can be explicitly enumerated, allowing for clear and concise specification of the available actions.
Sample efficiency is another important factor. In DRL, agents learn through interactions with the environment, and discrete action spaces generally require fewer samples for effective learning. With a finite number of discrete actions, the agent can explore and learn the consequences of each action more efficiently. On the other hand, continuous action spaces with infinite possibilities often require a larger number of samples to discover optimal actions [53].
Inside the simulator both the engines provided work similarly: a force is applied along the longitudinal axis to the solid representing the engine, then,

via rigid body simulation, the force is transferred to the rest of the boat, and applied to the join.

- **Tilt Engine**

  In the case of the tilt engine, it offers 2 parameters: speed and torque. Speed would determine the amount of force applied to the engine, torque would refer to the angular velocity of the rudder. The engine, when tilting, applies the force to the boat at a different angle, making it turn.

- **Differential Engine**

  In the case of the differential engine, it offers 3 parameters: speed, torque and reverse ratio. While the speed remains the same as in the tilt engine, the torque, together with the reverse ratio, determines the power distribution between the 2 engines, when turning.

While it is true that the force applied is fixed and applied in pulses, the intrinsic drag of the boat with the water still yields a smooth result.
Being the engine's standalone components, they expose methods to perform basic actions and they are independent of the agent. The agent then can combine these basic actions and expose custom action space.
In this way, it is possible to offer a standardized high-level interface to control the boat, but the physical effects of that action will change based on the kind of engine used.
Initial experiments with a size 4 action space, which would provide FORWARD, BACKWARD, LEFT and RIGHT, resulted in the agent randomly alternating between FORWARD and BACKWARD, and the boat not really going anywhere, hindering effective learning and progress in the navigation task. To address this issue, the action space was reduced to 3 actions: FORWARD, FORWARD LEFT, and FORWARD RIGHT. By constraining the agent's actions to primarily move forward, the agent was compelled to make meaningful progress and navigate the environment effectively [26].
By reducing the action space, the agent's exploration became more focused and directed. This setup encouraged the agent to move forward while still

allowing slight deviations to the left or right. Consequently, the agent learned more efficiently and improved its navigation performance [26].
In summary, discrete action spaces are often preferred in DRL due to their simplicity and sample efficiency. They provide a clear and manageable set of actions for the agent to choose from, enabling more effective learning and decision-making. By shaping the action space appropriately, the agent's training can be improved, leading to better performance in the target task. While this setup works fine for the given tasks, it might not be appropriate for more complex tasks that may require precise maneuvering, to avoid collisions.

4.5 State-Space

For the current training, the number of inputs has been reduced to four key variables: DISTANCE, COMPASS, LATITUDE, and LONGITUDE. While not being the only sensible configuration, these variables are proven sufficient for the agent to navigate in the environment.
To ensure that the values provided to the agent are within a suitable range, some normalization and relative adjustments have been applied.

- **DISTANCE**

  The original distance value is divided by the maximum distance to obtain a relative distance value, denoted as DISTANCE_REL. Dividing by the maximum distance scales the values to a range between 0 and 1, which helps in training the agent more effectively.

- **COMPASS**

  Initially, the compass value was used as a single input. However, after several attempts, it was decomposed into two values: COMPASS_SIN and COMPASS_COS. The decomposition into sine and cosine components allows the agent to capture the direction in a more effective manner. By representing the compass as two values, the agent can better understand the angular relationship between its current heading and the

desired direction.

- **LATITUDE and LONGITUDE**

  These geographic coordinates are made relative to the boat's starting point. This relative adjustment is important because it allows the agent to focus on the relative position changes rather than the absolute latitude and longitude values. The adjusted values are denoted as LATITUDE_REL and LONGITUDE_REL.

Overall, the state space provided to the agent consists of five continuous values: DISTANCE_REL, COMPASS_SIN, COMPASS_COS, LATITUDE_REL, and LONGITUDE_REL. These normalized and relative values provide the agent with more easy-to-use information to make informed decisions and navigate the environment effectively. It's worth noting that the normalization and relative adjustments are crucial steps in preparing the input data for the agent. They ensure that the values are within appropriate ranges and facilitate effective learning and decision-making during the training process[43][27].

5 Empirical Evaluation

5.1 Empirical Methodology

This empirical evaluation has several goals, the first is to make sure that the simulator is suitable for training DRL agents. The second goal is to evaluate how reproducible those experiments are, given such a noisy environment. Given that some tests had to be run, I decided to make the best use of the training time. I took the opportunity offered by this testing, to explore some of the hyperparameters of the agent: ALGORITHMS, REWARD SHAPING, FREQUENCY UPDATE TARGET NETWORK.
While this remains an empirical test, I hope it will still provide a starting point for future work. Each of these settings has been modified independently to observe their impact on the overall performance of the agent.
The primary performance metric is the average reward of the last 100 episodes. Being a slow signal can compensate for noisy training and still offer a quick way to rate the current configuration. However, for each experiment, I report the rate of successes and failures, which gives a more complete view of the performance as well and it offers a backup metric when testing with different reward shaping techniques, as they artificially alter the main performance metric.

5.2 Empirical Test

- **Test 1: ALGORITHMS**

  This test aims to compare the capabilities of PPO and REINFORCE for the given task. The testing conditions are 1000 Episodes, 1000 Steps each. Both models are equipped with DNN with 2 hidden layers of 32 nodes each. The target network is updated every 10 episodes. Additionally, PPO will update the Critic model, every 40 Episodes.

- **Test 2: REWARD SHAPING**

  This test aims to have a first evaluation of the 4 Reward Shaping techniques mentioned before. I run some short sessions of 100 Episodes with 500 Steps each. Each experiment is repeated 3 times to assess reproducibility.

- **Test 3: FREQUENCY UPDATE TARGET NETWORK**

  This test aims to have a first evaluation of how changing the frequency of the update of the target network can impact performance. I chose 5, 10 and 15 epochs as samples. I run some short sessions of 100 Episodes with 500 Steps each. Each experiment is repeated 3 times to assess reproducibility.

5.3 Hardware Setup

As mentioned, the ZibraLiquid plugin posed the limitation of running on only Windows, which limited access to the Linux clusters for the training. Therefore all training has been done using my personal computer at home with the following hardware specifications.

CPU	AMD Ryzen 9 5900X, 3700 Mhz, 12 Core, 24 Threads
RAM	24Gb
Disk	1Tb NVMe
GPU	Nvidia 3090
OS	Windows 10 Pro

Table 5.1: Hardware Setup

5.4 Presentation and Analysis of the Results

Test 1: ALGORITHMS

This table summarizes the results of running PPO and REINFORCE over 1000 Episodes, with 1000 steps each.

	PPO	REINFORCE
Success	127	636
Failures	617	128
Average steps last 100	833	541
Average reward	-648	373
Average reward last 100	-149	803

Table 5.2: Summary of results for PPO and REINFORCE

The following charts provide a clearer overview of the training, which highlights rewards. Averaged reward and Last Step are averaged over the last 100 episodes.

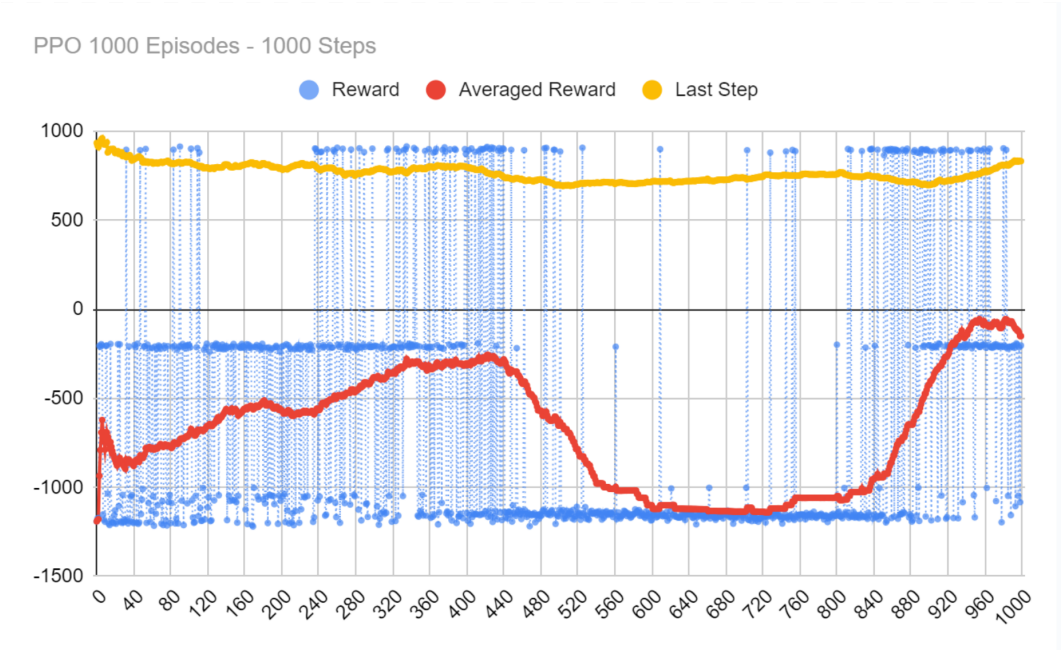

Figure 5.1: PPO detailed training plot

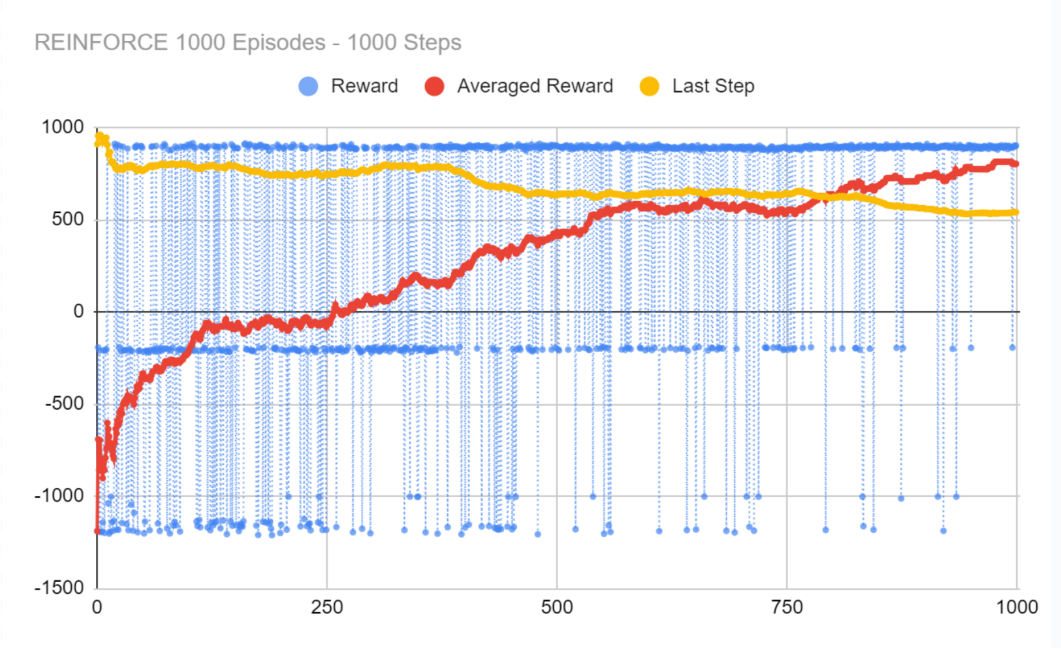

Figure 5.2: REINFORCE detailed training chart

Both algorithms are capable of learning from the simulated environment. REINFORCE shows steady improvement both in terms of Average reward and Number of steps. As is clearly represented in the chart by the red line, PPO appears to become unstable in the early phase of training and requires longer training to yield comparable results.

Test 2: REWARD SHAPING

This table summarizes the results of running REINFORCE over 100 Episodes, with 500 Steps each. Each table shows the overall result of each method.

PLAIN	Run 1	Run 2	Run 3
Success	11	15	16
Fail	7	3	2

SQUARE	Run 1	Run 2	Run 3
Success	28	22	27
Fail	3	3	4

MUL	Run 1	Run 2	Run 3
Success	25	29	22
Fail	3	2	4

ADD	Run 1	Run 2	Run 3
Success	29	26	24
Fail	7	1	2

Table 5.3: Summary of results for each method used

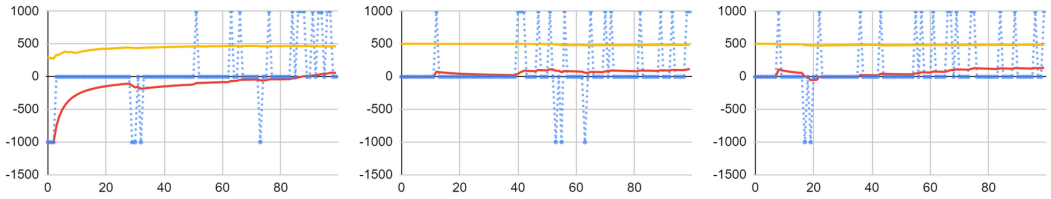

Figure 5.3: Training details for method PLAIN

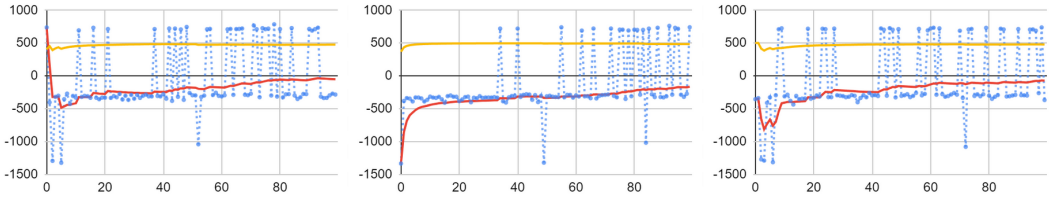

Figure 5.4: Training details for method SQUARE

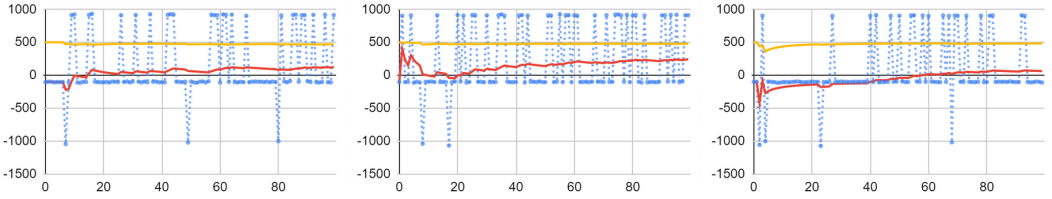

Figure 5.5: Training details for method MUL

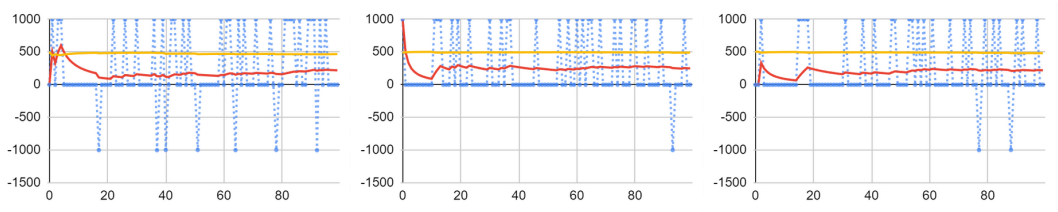

Figure 5.6: Training details for method ADD

The results show that the algorithm can produce overall comparable results, however as shown in the details for each method, the water simulations still appear to introduce some variance in the experiments, despite the water particles always being initialized from the same state.

This empirical test also shows that despite the agent being provided with a dense reward signal, it may benefit from some form of Reward Shaping.

Test 3: FREQUENCY UPDATE TARGET NETWORK

This table summarizes the results of running PPO and REINFORCE over 1000 Episodes, with 1000 steps each. Each table shows the overall result of each frequency of update.

UPDATE 5	Run 1	Run 2	Run 3
Success	10	8	9
Fail	15	16	14

UPDATE 10	Run 1	Run 2	Run 3
Success	11	5	8
Fail	26	26	29

UPDATE 15	Run 1	Run 2	Run 3
Success	2	2	5
Fail	32	33	34

Table 5.4: Summary of results for each UPDATE frequency

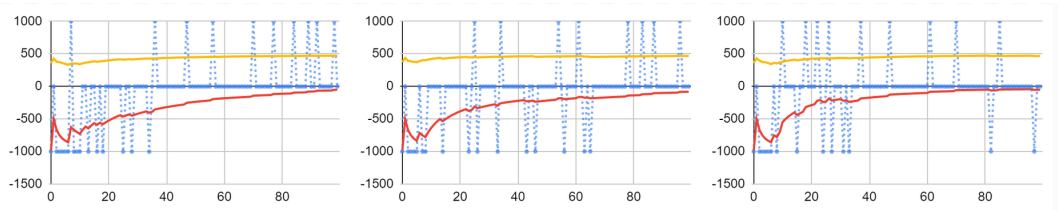

Figure 5.7: Training details for frequency UPDATE 5

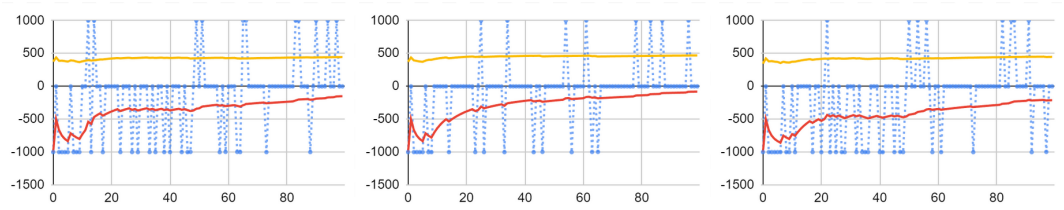

Figure 5.8: Training details for method UPDATE 10

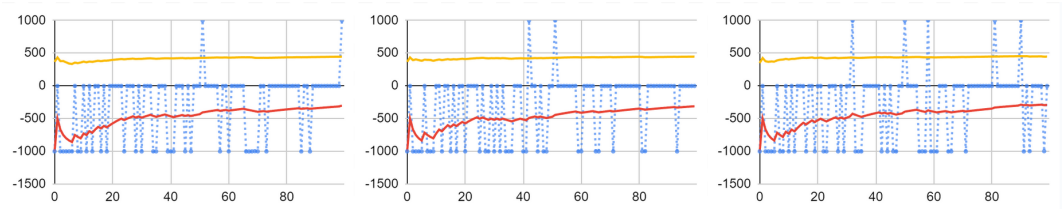

Figure 5.9: Training details for method UPDATE 15

The results show that the algorithm is capable of producing stable and comparable results, across multiple runs of the same experiment, showing very small variance.

This empirical test also shows some initial findings that suggest that REINFORCE, in these testing conditions, performs better with a higher update frequency of the target network.

6 Discussion and Conclusions

6.1 Summary of Findings

The main goal of the project is to build a boat simulator capable of providing realistic conditions for successfully training DRL agents, starting from the task of autonomous GPS waypoint navigation.
The second goal of the project is to evaluate the level of reproducibility of the results, given the noisy learning environment produced by the fluid simulation.
The third, side goal, of the project is to provide an initial assessment of suitable algorithms and other hyperparameters that can serve as a base for future developments.

Training

The overall results confirm that the simulator's suitable environment contributes to the expansion of Deep Reinforcement Learning (DRL) algorithms in the domain of autonomous boat navigation. By designing and implementing DRL agents to autonomously navigate and learn in the simulated environment, the project demonstrates both the capabilities of the simulator and the feasibility and partially assesses the effectiveness of using DRL techniques for autonomous waypoint navigation tasks.

Stability

The simulator demonstrates it is capable of a good level of reproducibility, despite the very noisy environment generated by the physics of the water. While my testing was of an empirical nature, and cannot really assess the level of reproducibility of its precision, the environment seems to yield similar overall results across multiple runs or experiments.
More in general, despite the complex and challenging nature of the environment, the simulator proves to be suitable for training DRL agents, enabling them to learn and adapt effectively. These positive experimental results highlight the reliability and effectiveness of the simulator,

emphasizing its potential for facilitating the development of robust and capable DRL agents.

Further consideration can be made by looking at Test 2 and Test 3 is possible to notice how in Test 3 there appears much less variance between episodes. During the development of the simulator, I noticed that the behaviour of the water dramatically changed by altering a seemingly unconnected parameter, the current Frame-Per-Second (FPS) of the scene, which was later confirmed by the ZibaAI team.

Despite setting it to a fixed value inside the simulator, I'm induced to believe that a temporary decrease in performance during the simulation may induce some variances in the dynamics of the liquid, which then introduces variance in the scene. The reason is that the results of Test 3 were entirely produced by night, where most of the experiments from Test 2 were conducted during the day, while I was using the computer.

While intuition has to be verified with the ZibraAI team, in the meantime it is recommended to keep the load stable on the computer running the simulator.

Talking about stability, it is worth mentioning that the simulator itself has never crashed or frozen throughout the experiments.

Sensors

The current setup shows that autonomous waypoint navigation can be achieved solely by the use of GPS and Compass sensors. The GPS sensor provides precise location information. The Compass sensor determines the orientation and heading of the vehicle, which otherwise should be extrapolated by the agent by using GPS coordinates between subsequent steps.

Algorithms

The data also confirms that my initial assumptions were accurate: PPO is more challenging to train than REINFORCE and exhibits lower performance at the beginning of the training. The results show that PPO requires more training iterations to achieve comparable performance, indicating a slower

convergence. Additionally, PPO demonstrates higher variance during the initial phase of the training, making the learning process less predictable and more unstable. The findings also highlight the sensitivity of PPO to hyperparameter tuning, further emphasizing its difficulty. In summary, the experimental results validate my suppositions, emphasizing the need for careful design and optimization when using PPO in reinforcement learning training.

6.2 Limitations and Future Research Directions

Tasks

The next step in the development of the simulator is to increase the number of available tasks and testing scenarios

- **GPS Waypoint navigation Randomized**
  By randomizing the starting position for the target and the position and orientation of the boat. The agent should be able to generalize the task.
- **GPS Waypoint navigation with fixed submerged obstacles**
  This task requires adding the Sonar to the Action-state. By using the sonar the boat can learn to avoid submerged or partially submerged obstacles.
- **GPS Waypoint navigation with the coastline as obstacle**
  This task requires adding the Sonar to the Action-state. By using the sonar the boat can learn to sense shores and docks. In combination with the reading coming from the visual bathymetry inside the simulator, the sonar will be also able to realize it is approaching the shore.
- **GPS Waypoint navigation with water-level obstacles**
  This task requires adding a lidar or a camera (or both) to the Action-State. An ideal task is to assess how noisy the lidar or the camera is,

on a constantly tilting object. The task can be randomized to increase generality.

- **GPS Waypoint navigation Randomized with multiple boats**
  This task requires adding a lidar or a camera (or both) to the Action-State. By spawning multiple boats, in the same wide training area, each having a randomized starting point and target, it would be possible for the agents to learn about collision avoidance with other floating, moving obstacles. This setup will also allow us to leverage parallelism.

- **Autonomous Search Tasks: Image**
  This task requires adding a camera (or both) to the Action-State. By extending the simulator it should be possible for an advanced agent to autonomously explore the environment searching for items, for example, a type of fish. The camera can be placed both under or above water.

- **Autonomous Search Tasks: Molecules**
  This task requires adding a molecular/particle sensor to the Action-State. By extending the simulator it should be possible for an advanced agent to autonomously explore the environment tracing for a substance, simulated using a sparse particle system. The sensor can be placed both under or above water.

Implementing these new tasks and training effectively on them should allow for the agent to be tested in real life. The use of Safe DRL is particularly interesting for the task of collision avoidance. Adverse weather conditions, like water currents and waves, can be added to all previous tasks.

Safe DRL

Another interesting evolution of this work is the use of Safe Deep Reinforcement Learning (Safe DRL), specifically Constrained Policy Optimization (CPO). CPO is a general-purpose policy search algorithm designed for constrained reinforcement learning tasks, where both a reward function and constraints are specified for the agent's behaviour [1].

CPO enables the training of neural network policies for high-dimensional control tasks while ensuring that the policy behaviour remains within the specified constraints throughout the training process. It introduces a novel theoretical result that establishes a bound between the expected returns of two policies and the average divergence between them. This bound forms the basis for the guarantees provided by CPO, ensuring that the policy remains close to the desired constraints. Thanks to this approach the reward shaping connected to critical actions, could be instead expressed with hard constraints.

Future Projects

The development of this multipurpose boat navigation simulator has the potential to significantly speed up the development of future projects based on autonomous boat navigation. Like the one recently terminated which would focus on water quality monitoring which was funded by the European Union to the University of Verona [23][13].
By creating a virtual environment for testing and experimentation, it becomes easy to rapidly iterate on algorithms and hyperparameter settings, without the need for physical boats and real-world resources. This allows for quicker development cycles, reduces costs, and accelerates the learning process. Additionally, the simulator's versatility makes it easy to adapt, or eventually extended for new applications.

Parallelism

Implementing parallelism by utilizing a multiagent environment seems to be the most promising next step. Parallelism enables multiple agents to interact and learn simultaneously, leading to faster and more efficient training. By running multiple agents in parallel, the training process can be accelerated, and the agents can learn from each other's experiences.
By combining the parallelism of the multiagent environment and utilizing PPO, the agent's training can be further optimized and serve as an ideal foundation for tackling more complex tasks[26].

The main limiting factor is the lack of support for Linux. While the team is confident to release it in the future, at the time of writing it is not yet available. This is generally not a favourable factor as traditionally all the software stack for RL runs on Linux. Practically limiting access to computational power.
In order to address the issue some preliminary tests have been done. I managed to use the Wine compatibility layer and actually run the simulator and it seems stable, however, the FPS are not nearly enough to be usable. From the initial finding, looking into DXVK support in the Wine environment is likely to yield good results. By using DXVK it would be possible to run it on Linux seamlessly, simplifying the whole process and allowing the use of dedicated servers. [12][22].
Another possible solution that remains interesting for future iterations of this project is to publish the game on Valves's Steam, an online gaming platform, that offers a Protons compatibility layer out of the box.

Reusability

By training agents to learn navigation strategies through interactions with the environment, the project demonstrates the ability of DRL algorithms to adapt and learn in complex field scenarios.
Moreover, several components have been developed and manually tested while developing the simulator and they have not been used or accessed during the training. Those sensors can enable new capabilities for the boat or they can be directly reused into the Unity3D project, contributing to the Unity ecosystem.

Environment Challenges

Several environmental elements have been disabled, like the collidable docks, and the Wave Generator. For example, in the case of the Wave Generator, the system has been tested and is ready to be used, however has proven to be too challenging for the agents trained during this work. The complexity introduced by wave generators and water currents often overwhelms the agent's learning capabilities and hinders its performance. During the

experiments, the wave generators, at times, ended up sinking the boat. While this correctly simulates real-life scenarios. From my observations it appears to be detrimental to the learning process of the agent, at this stage of advancement, therefore has not been included in this experimental setup.

Citations & References

References

[1] J. Achiam et al. "Constrained policy optimization". In: (2017). URL: https://dl.acm.org/doi/10.5555/3305381.3305384.

[2] *AI Habitat Website*. URL: https://aihabitat.org/.

[3] *Blender Documentation*. URL: https://docs.blender.org/manual/en/latest/getting_started/about/introduction.html.

[4] *Blender Website*. URL: https://www.blender.org/features/modeling/.

[5] *BlenderGIS Website*. URL: https://github.com/domlysz/BlenderGIS.

[6] M. Bojarski, D. Del Testa, and D. Dworakowski et al. "End to End Learning for Self-Driving Cars". In: (2016). URL: https://arxiv.org/abs/1604.07316.

[7] L. Buşoniu et al. "Reinforcement learning for control: Performance, stability, and deep approximators". In: *Annual Reviews in Control* 46 (2018). URL: https://doi.org/10.1016/j.arcontrol.2018.09.005.

[8] L. Christensen, J. de Gea Fernández, and M. et al. Hildebrandt. "Recent Advances in AI for Navigation and Control of Underwater Robots". In: *Curr Robot Rep* (2022). URL: https://doi.org/10.1007/s43154-022-00088-3.

[9] DeepMind. *AlphaStar: Mastering the real-time strategy game StarCraft II*. 2019. URL: https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii.

[10] *DeepMind Control Suite*. URL: https://www.deepmind.com/open-source/deepmind-control-suite.

[11] *DeepMind Lab*. URL: https://www.deepmind.com/open-source/deepmind-lab.

[12] *DXVK (DirectX-over-Vulkan) stability improvements & potential performance boost: installation guide & details*. URL: https://steamcommunity.com/sharedfiles/filedetails/?id=2461019058.

[13] *Europe - Development and application of Novel, Integrated Tools for monitoring and managing Catchments*. URL: https://cordis.europa.eu/project/id/689341/results.

[14] F. Ferreira, A. Quattrini Li, and Ø. J. Rødseth. "Editorial: Navigation and Perception for Autonomous Surface Vessels". In: *Frontiers in Robotics and AI* (2022). URL: https://doi.org/10.3389/frobt.2022.918464.

[15] M. Fortunato, M. G. Azar, and B. Piot et al. "Noisy Networks for Exploration". In: (2017). URL: https://arxiv.org/abs/1706.10295.

[16] *GazeboSim Website*. URL: https://gazebosim.org/home.

[17] A. Gosavi. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Springer, 2018. URL: https://link.springer.com/book/10.1007/978-1-4757-3766-0.

[18] Yewen Gu et al. "Autonomous Vessels: State of the Art and Potential Opportunities in Logistics". In: *NHH Dept. of Business and Management Science Discussion Paper* 2019/6 (2019). URL: http://dx.doi.org/10.2139/ssrn.3448420.

[19] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. URL: https://doi.org/10.48550/arXiv.1801.01290.

[20] D. Horgan, J. Quan, and D. Budden et al. "Distributed Prioritized Experience Replay". In: (2018). URL: https://arxiv.org/abs/1803.00933.

[21] C. D. Hubbs, H. D. Perez, and O. et al. Sarwar. "OR-Gym: A Reinforcement Learning Library for Operations Research Problems". In: (2020). URL: https://doi.org/10.48550/arXiv.2008.06319.

[22] *Improve Your Wine Gaming on Linux With DXVK*. URL: https://linuxconfig.org/improve-your-wine-gaming-on-linux-with-dxvk.

[23] *Intcatch Website*. URL: https://www.intcatch.eu/.

[24] P. Januszewski. "Best Benchmarks for Reinforcement Learning: The Ultimate List". In: *Neptune.ai* (2023). URL: https://neptune.ai/blog/best-benchmarks-for-reinforcement-learning.

[25] A. Juliani. "On "solving" Montezuma's Revenge Looking beyond the hype of recent Deep RL successes". In: (2018). URL: https://awjuliani.medium.com/on-solving-montezumas-revenge-2146d83f0bc3.

[26] A. Kanervisto, C. Scheller, and V. Hautamäki. "Action Space Shaping in Deep Reinforcement Learning". In: (2020). URL: https://doi.org/10.48550/arXiv.2004.00980.

[27] *Leveraging Geolocation Data for Machine Learning: Essential Techniques. A Gentle Guide to Feature Engineering and Visualization with Geospatial data*. URL: https://towardsdatascience.com/leveraging-geolocation-data-for-machine-learning-essential-techniques-192ce3a969bc.

[28] T. P. Lillicrap, J. J. Hunt, and A. et al. Pritzel. "Continuous control with deep reinforcement learning". In: (2015). URL: https://arxiv.org/abs/1509.02971.

[29] D. Mankowitz and T. Hester. "Challenges of Real-World Reinforcement Learning". In: (2019). URL: https://arxiv.org/abs/1904.12901.

[30] *MLAgents code repository*. URL: https://unity-technologies.github.io/ml-agents/.

[31] *MLAgents documentation*. URL: https://docs.unity3d.com/Packages/com.unity.ml-agents@2.0/manual/index.html.

[32] *MLAgents Website*. URL: https://github.com/Unity-Technologies/ml-agents.

[33] V. Mnih, A. P. Badia, and M. Mirza et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *ArXiv preprint* (2016). URL: https://doi.org/10.48550/arXiv.1602.01783.

[34] V. Mnih, K. Kavukcuoglu, and D. et al. Silver. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015). URL: https://doi.org/10.1038/nature14236.

[35] *MuJoCo Website*. URL: https://mujoco.org/.

[36] A. Nowé, P. Vrancx, and YM. De Hauwere. "Game Theory and Multi-agent Reinforcement Learning". In: *Reinforcement Learning. Adaptation, Learning, and Optimization*. Vol. 12. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-27645-3_14.

[37] *ROS Website*. URL: https://www.ros.org/.

[38] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 4th US ed. 2022. URL: https://aima.cs.berkeley.edu/.

[39] Daniel Russo and Benjamin Van Roy. "An information-theoretic analysis of Thompson sampling". In: *Journal of Machine Learning Research* 17 (2015). URL: https://theinformaticists.com/2021/03/19/an-application-of-information-theory-in-reinforcement-learning/.

[40] M. Savva, A. Kadian, and O. et al. Maksymets. "Habitat: A Platform for Embodied AI Research". In: *arXiv preprint* (2019). URL: https://doi.org/10.48550/arXiv.1904.01201.

[41] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. URL: https://doi.org/10.48550/arXiv.1707.06347.

[42] D. Silver, A. Huang, and C. et al. Maddison. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016). URL: https://doi.org/10.1038/nature16961.

[43] *Spatial Distance and Machine Learning. Distance Metrics and Feature Engineering Using Longitude and Latitude Data*. URL: https://towardsdatascience.com/spatial-distance-and-machine-learning-2cab72fc6284.

[44] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018.

[45] A. Szot, A. Clegg, and E. et al. Undersander. "Habitat 2.0: Training Home Assistants to Rearrange their Habitat". In: *arXiv preprint* (2021). URL: https://doi.org/10.48550/arXiv.2106.14405.

[46] *Unity Signs Multi-Million Dollar Contract To Help U.S. Army And Defense Agencies*. URL: https://kotaku.com/unity-new-contract-us-government-military-army-engine-1849403118.

[47] *Unity3D Website*. URL: https://www.unity.com/.

[48] C. Wang et al. "Collision avoidance for autonomous ships using deep reinforcement learning and prior-knowledge-based approximate representation". In: *Frontiers in Marine Science* 9 (2023), p. 1084763. URL: https://doi.org/10.3389/fmars.2022.1084763.

[49] *Wikipedia: Edward Thorndike*. URL: https://en.wikipedia.org/wiki/Edward_Thorndike.

[50] *Wikipedia: Unity game engine*. URL: https://en.wikipedia.org/wiki/Unity_(game_engine).

[51] *Wikipedia: Waypoint*. URL: https://en.wikipedia.org/wiki/Waypoint.

[52] K. Zhang, Z. Yang, and T. Başar. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms". In: *ArXiv preprint* (2019). URL: https://arxiv.org/abs/1911.10635.

[53] Jie Zhu, Fengge Wu, and Junsuo Zhao. "An Overview of the Action Space for Deep Reinforcement Learning". In: (2022). URL: https://doi.org/10.1145/3508546.3508598.