CLEI electronic journal, Volume 27, Number 2, Paper 5, July 2024

Simulation-based Reinforcement and Imitation Learning for Autonomous Sailboat Navigation in Variable Environmental Conditions

Agustín Rieppi
Universidad de la República, Facultad de Ingeniería,
Montevideo, Uruguay, 11300,
arieppi@fing.edu.uy

and

Florencia Rieppi
Universidad de la República, Facultad de Ingeniería,
Montevideo, Uruguay, 11300,
fireppi@fing.edu.uy

and

Mercedes Marzoa
Universidad de la República, Facultad de Ingeniería,
Montevideo, Uruguay, 11300,
mmarzoa@fing.edu.uy

and

Gonzalo Tejera
Universidad de la República, Facultad de Ingeniería,
Montevideo, Uruguay, 11300,
gtejera@fing.edu.uy

Abstract

In light of escalating concerns over climate change, harnessing oceanic data becomes increasingly urgent. Oceans serve as linchpins in understanding the intricate dynamics governing climate phenomena, exerting pivotal influence over global weather patterns and ecological systems. Despite scientific consensus on climate change impacts, including temperature shifts and acidification, a lack of data infrastructure hampers understanding marine ecosystems. This paper presents an adaptive control system for autonomous sailboats that aims to navigate efficiently in varied conditions by favoring oceanographic data acquisition. The controller emulates human decision-making by combining reinforcement and imitation learning, enabling robust navigation. While showcasing promising results, challenges persist in adverse, varied conditions. This challenge is exacerbated by the necessity of adaptive control mechanisms resilient to wear and tear. Simulators are vital for training due to the vast amounts of data required. Real-world data collection is costly and risky, while simulations accelerate learning. This study employs a simulator operating at ten times real-time speed, significantly simplifying scenario generation by adjusting factors such as sailboat position, orientation, target location, water current, and wind. Nevertheless, this novel approach signifies progress in addressing climate challenges and advancing oceanic research using advanced computational methods.

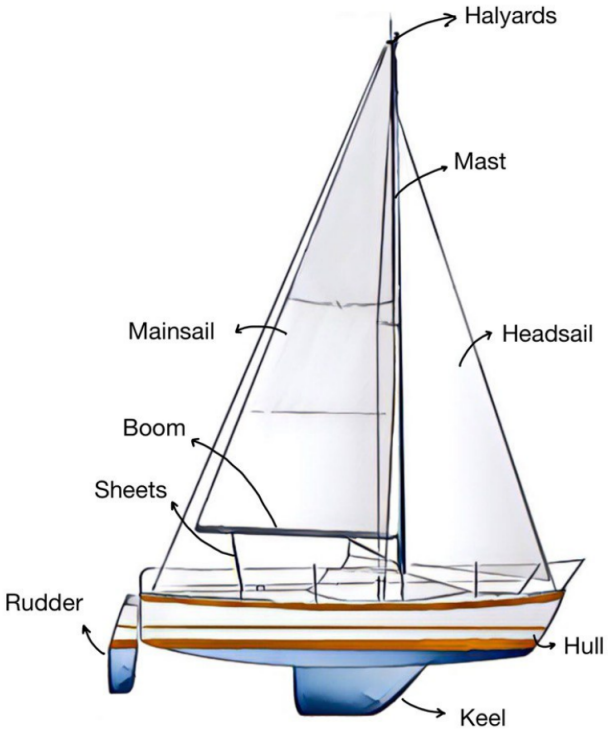

Figure 1: Sailboat parts

Keywords: Reinforcement learning, Imitation learning, ROS, Robotics, Sailboats, Simulators, Autonomous water vehicles, Machine learning.

1 Introduction

Presently, the interest in dealing with climate change has led to the need to obtain and analyze data from the ocean. Oceans play a crucial role in grasping the complex and intertwined processes that govern these phenomena. Oceans and rivers play a key role in regulating the planet's climate, weather, and ecology. A well-established scientific consensus exists regarding the repercussions of climate change on the global ocean. This encompasses a discernible temperature alteration, an escalation in acidification levels, the deoxygenation of water masses, and shifts in nutrient availability and biomass productivity. Collectively, these alterations are poised to profoundly influence virtually all life forms within the ocean, leading to far-reaching consequences for food security, ecosystem services, and coastal communities. Despite the myriad impacts identified, there remains a deficiency in scientific data and infrastructural support necessary for a comprehensive understanding and quantification of the ramifications of these perturbations on the marine ecosystem. Recent advances in computer sciences and applied mathematics, such as machine learning, artificial intelligence, and scientific computation, among others, have produced a revolution in our capacity for understanding the emergence of patterns and dynamics in complex systems. At the same time, the complexity of these problems poses significant challenges to computer science itself. The key factor in deciding the success or failure of applying these methods is having sufficient and adequate data. The systematic extraction of these samples presents significant challenges. Ocean sensing has been typically done with satellites, airplanes, buoys, research vessels, or ships of opportunity. However, satellites and airplanes are limited by cloud cover, temporal/geographical coverage, and spatial resolution. Moreover, oceanographic vessels have been used for this task. However, their high cost and limited availability make them inefficient. Using unmanned autonomous aquatic vehicles has emerged as an alternative for this task. Among these vehicles, sailboats stand out because they do not require energy sources other than wind for their propulsion. On the other hand, they present a disadvantage and a high complexity in their navigation. There is a wide variety of sailboats with different characteristics. However, some elements of the vessel are common and indispensable for all of them: the hull, the sails, the rudder, and the keel or centerboard. These elements are presented in Figure 1.

The position of the sail and rudder is a significant part of what determines the sailboat's course, so in order to move in a given direction, it is of great importance to have control of both. The different courses that

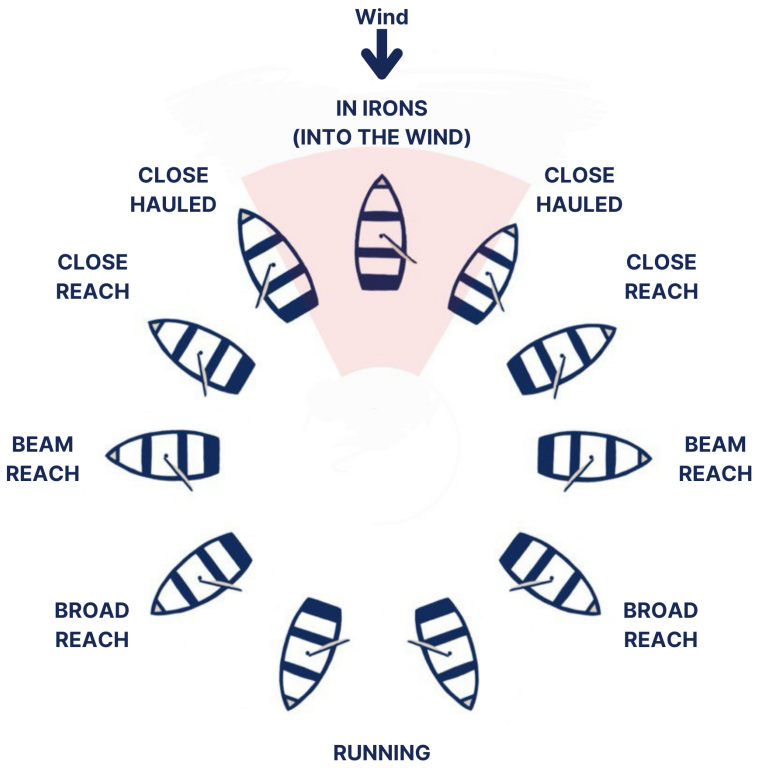

Figure 2: Sailing courses with respect to the wind.

can be sailed are shown in Figure 2.

One of the biggest challenges for autonomous sailboats is designing a robust controller capable of handling the multiple steering controls and adapting to the vessel's natural wear and tear. This is especially difficult if the sailboat is expected to sail on the open ocean and face adverse weather such as storms, large waves, high winds, and currents.

This paper presents an adaptive controller for a sailboat, which is responsible for driving the sailboat to a target point and staying there to obtain data, navigating from different initial locations and under different weather conditions. The proposed controller is based on a reinforcement learning model combined with imitation learning. Combining those methods could overcome the need to program a particular controller for every boat as it would be capable of replicating, to some degree, the learning process of human skippers and sailors. Imitation learning is intended to use expert knowledge to accelerate training. The goal of using reinforcement learning is to create a controller that can generalize to adapt to different environments and sailboat wear over time, as well as adapt to different types of sailboats. A simulator is fundamental to carry out this type of learning since a large volume of data is required to train the model. Obtaining this data from the real world entails a very high cost since it is necessary to have the agent, in addition to the possible breakage and damage of the agent due to errors made during training. In addition to the high cost, the learning time in a natural environment is longer than in simulation since the latter can accelerate the time and even parallelize learning.

2 Related work

Simulators are of great importance to this work since it is through them that training and experimentation are conducted, underscoring the crucial role of simulator fidelity to reality in the subsequent success. While there are several simulators for watercraft, the options become more limited when it comes to sailboats, especially when considering only open-source ones. It is also noteworthy to highlight existing controllers for watercraft, specifically focusing on controllers for sailboats and those utilizing reinforcement learning.

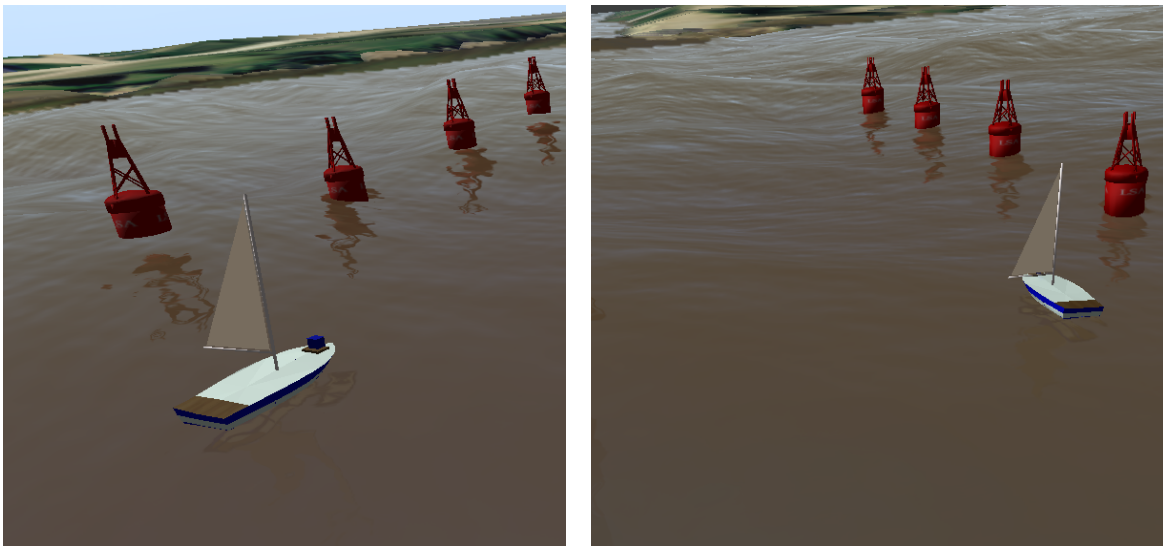

Figure 3: Screenshots from the simulator USVSim using the sailboat model.

2.1 Simulators

Among the open-source sailboat simulators, one of the most complete is USVSim [1] (Figure 3), a modular unmanned surface vehicle (USV) simulator to model different scenarios and boats, including sailboats composed of a sail, a keel, and a rudder. USVSim integrates the wind model, a hydrological model to simulate water currents that apply forces to the boat, and the buoyancy effects. Seeking to expand the open-source Gazebo simulator's capabilities, the Virtual RobotX simulator [2] emerged. It is a general-purpose gazebo-based simulator capable of simulating the behavior of USVs in oceanic environments. The application of this simulator was highlighted in the simulation-based robot competition designed to complement the physical Maritime RobotX Challenge. The simulator provides a catamaran model with differential actuators.

Other simulators based heavily on the Robot Operating System (ROS) framework and its tools, including Evain work [3], present Autonomous Sailboat, a work to simulate and control an autonomous sailboat. It mainly seeks to simulate control algorithms and implements a controller for triangular races and another to be able to remain at a given point. The sailing vessels project [4] is also based on ROS and its tools. Although the main objective is not the creation of a simulator, they create a simulator to test an autonomous sailboat virtually in different conditions. They simulate the speed of the sailboat based on the wind speed and some initial parameters, such as the minimum speed. They also simulate the position based on the heading and wind speed. The new course is computed based on the sailboat's speed and the rudder angle. It also provides the apparent wind speed and direction. There is also a GPS, which keeps a record of the time and position of the sailboat. Finally, is SailBoatRos [5], an open-source sailboat simulator. It has a GPS and sensors for inertial measurements of the sailboat. It also gives the speed and direction of the wind and provides the angle of both the sail and the rudder.

On the other hand, there are works that, although they do not simulate sailboats, are of interest to highlight, given their similarities in simulating aquatic vehicles. Among them are those that allow the simulation of underwater vehicles, such as UWSim [6], an open-source underwater simulation tool that is highly configurable by the user, allowing the configuration of essential parameters such as the color of the water, visibility or floating particles. UWSim enables the use of standard modeling software and allows the addition of underwater vehicles, robotic manipulators, surface vessels, and simulated sensors. The interfaces with external software are implemented through ROS interfaces. Another work with underwater vehicles is Free-Floating-Gazebo [7], which integrates Gazebo and UWSim to achieve a realistic multi-robot simulation rendering.

Table 1 summarizes the main characteristics of the simulators described.

2.2 Controllers

A controller is a system that manages the vehicle's navigation, operating at different levels. It ranges from the lowest level, where the sole objective is to maintain the boat on an arbitrary course, to the highest level, where a global route is planned and the boat is guided to follow it.

Simulator	Waves	Buoyancy	Currents	Wind	Fulid dynamics	Sailboats
UWSim [6]	✗	✓	✗	✗	✗	✗
FreeFloating-Gazebo [7]	✗	✓	✓	✗	✗	✗
Robotx [2]	✓	✓✓	✗	✓	✗	✗
USVSim [1]	✓	✓✓	✓✓	✓✓	✓	✓
Autonomous Sailboat [3]	✗	✓	✗	✓	✗	✓
Sailing Robot[4]	✗	✗	✗	✓	✗	✓
SailBoatROS [5]	✗	✗	✗	✓✓	✗	✓

Table 1: Simulator comparison. The following characteristics are taken into account: the simulation (✓) or not (✗) of waves, the simulation of winds and currents in a constant way (✓), variable (✓✓) or non-simulation (✗), the simulation (✓) or not (✗) of fluid dynamics, whether buoyancy is simulated in a plane (✗), in a plane and vertically (✓) or in both planes and includes rotation (✓✓), whether it has the ability to simulate sailboats (✓) or not (✗). An entry is highlighted in green when the corresponding simulator has the highest capacity in the category, in red the lowest and in yellow an average.

Controller	Sailboat	Rudder	Sail/Motor	Global Planning	Local Planning	ROS	Open-source
Ctrl. [13]	✓	✓	✓ (Linear function)	✗	✓ (HF, LF)	✓	✓ (Python)
Ctrl. [14]	✓	✓ (PID)	✓ (Linear function)	✗	✓ (LF)	✗	✗
Ctrl. [17]	✓	✗	✗	✗	✓ (LF)	✗	✗
Ctrl. [15]	✓	✓ (P, PD , CS)	✓ (Linear function)	✗	✓ (LF)	✗	✗
Ctrl. [16]	✓	✓ (PID)	✓ (Linear function)	✓✓ (QL)	✓	✓	✗
Ctrl. [11]	✓	✓ (PD)	✓✓ (DT)	✓ (Euclidean distance)	✓ (HF)	✗	✗ (C)
Ctrl. [12]	✓	✓✓ (QL)	✓✓ (QL)	✗	✓ (HF)	✗	✗
Ctrl. [8]	✗	✓✓ (QLDNN, GDDP)	✓✓ (QLDNN, GDDP)	✗	✓ (LF)	✗	✓ (Python)
Ctrl. [9]	✗	✓	✓	✗	✓✓ (PPCMSE)	✗	✗ (Matlab)
Ctrl. [10]	✗	✓	✓	✗	✓✓ (QLDNN)	✗	✗

Table 2: Controller comparison. The following characteristics were taken into account: whether it was a controller for a sailboat or other water vehicle, whether it controls the rudder and sail (or motor) statically (✓), adaptively (✓✓) or not at all, whether it performs static local and global planning (✓), adaptive (✓✓), or does not do so (✗), whether it is integrable (✓) or not (✗) with ROS and whether it is open source (✓) or not (✗). If applicable, the algorithm used is clarified: HF (Head Follower), LF (Line Follower), QLDNN (Q Learning with Deep Neural Networks), GDDP (Gradient Deep Deterministic Policy), DT (Decision Trees), PD (Proportional-Differential), PID (Proportional-Integral-Derivative), QL (Q Learning), CS (Controlling Sine), PPCMSE (Probabilistic Predictive Control Model with Sampling Efficiency).

It is useful to distinguish static vs. adaptive controllers. The distinction lies in the controller's ability to adapt to various environmental and sailboat state conditions. Adaptive controllers make modifications and adjustments during navigation, while static controllers define a set of rules that remain unchanged throughout the voyage.

There are several controllers; some of the existing works specifically focus on the control of motorboats [8, 9, 10], while others are designed for sailboat control. Among the latter, some incorporate adaptive control techniques for managing the sail [11], or both the rudder and sail [12], whereas others employ static control methods [13, 14, 15, 16]. Additionally, some controllers do not address the control of these elements [17]. The remaining evaluated characteristics can be observed in Table 2.

Among them, the controller presented in [12] stands out, where different reinforcement learning techniques are applied to control a sailboat, which aims to sail on a given course with a constant wind. Directional speed is used for the reward function. The actions are the position of the sail, defining its angle, and the rudder's position. The state is represented by the sailboat's speed, the relative wind, and the directional error. Different techniques are implemented to direct and control the sailboat, among which is continuous-state reinforcement learning, with which they obtain good results.

Although the latter is the most similar to our work, one of the main differences is that our controller seeks to navigate to a specific point with changing environmental conditions.

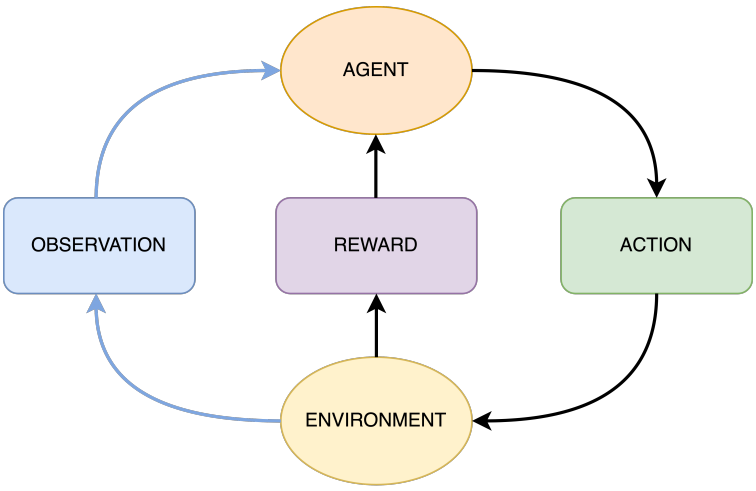

Figure 4: Interaction diagram between the agent and the environment.

3 Conceptual background

This section aims to explore and analyze the principles and essential aspects associated with reinforcement learning and imitation learning. First, the concept of reinforcement learning is introduced, outlining its foundations in state-of-the-art algorithms. Subsequently, imitation learning is presented, emphasizing its relationship with observational learning in the human context and its integration into intelligent systems for replicating expert behaviors. This introduction sets the context, objectives, and relevance of the topic to be discussed, as well as the general structure of the section.

3.1 Reinforcement Learning

Reinforcement learning (RL) is a field of machine learning, constituting one of the three fundamental paradigms alongside supervised and unsupervised learning. Broadly speaking, RL is the study of agents and how they learn through trial and error. The agent is either rewarded or penalized based on the impact of its actions on the environment, influencing the likelihood of repeating or avoiding that behavior in the future. To proceed with a more detailed description of an RL problem, some key terminologies that are part of reinforcement learning need to be understood, as described below.

State and Observation Space: The state is the complete description of the world where the agent exists without omitting any information. An observation is a partial state description, which may omit information. The most common way to represent states and observations is through vectors or matrices of real values.

Action Space: It is the set of valid actions that varies depending on the environment and the agent. Actions can be discrete or continuous. In discrete actions, the agent has a finite number of available moves, while in continuous actions, the possible actions are, a priori, infinite. This distinction has important consequences, as some algorithm families can only be applied in certain cases.

Policy: The policy is the agent's strategy to achieve the set goal. It is the rule by which the agent decides which actions to take in each state and can be deterministic or stochastic. Reinforcement learning involves parameterized policies, where the policy output is calculated through functions dependent on a set of parameters that can be adjusted using an optimization algorithm.

Reward: The definition of the reward function is crucial in reinforcement learning, as it seeks to define the agent's behavior. It depends on the current state, the action taken, and the next state. The reward is the feedback the agent receives from the environment, representing how well the last action aligns with the objective task.

Recapping, the general configuration of a reinforcement learning problem is defined as a discrete-time stochastic control process, where the agent interacts with its environment as follows: the agent starts in an arbitrary state of the environment, $s_0 \in S$, obtaining an initial observation $w_0 \in \Omega$. At each time step t, the agent takes an action $a_t \in A$, leading to three consequences: (i) the agent receives a reward $r_t \in R$, (ii) the transition from state s_t to $s_{t+1} \in S$ occurs, and (iii) the agent receives an observation $w_{t+1} \in \Omega$ from the new state s_{t+1}. This interaction is graphically reflected in Figure 4.

The four considered algorithms for this work are part of the state-of-the-art in reinforcement learning for continuous control. These are:

- Advantage Actor Critic (A2C) [18]
- Proximal Policy Optimization (PPO) [19]
- Twin Delayed Deep Deterministic Policy Gradients (TD3) [20]
- Soft Actor Critic (SAC) [21]

3.2 Imitation Learning

Imitation learning seeks to copy the behavior of an expert in a specific task. The agent is trained to perform a task, taking expert demonstrations as input. This technique is gaining popularity as it simplifies the teaching of complex tasks without the need for expert knowledge.

Traditional approaches, like Behavioral Cloning (BC) [22] and Inverse Reinforcement Learning (IRL) [23], have several issues, such as the requirement for large amounts of data and the assumption that the learned behavior is optimal, which is a strong assumption not always true.

In response to the flaws shown in the traditional techniques, new approaches emerge to address their limitations. Below, there are two novel imitation learning techniques which are going to be used in this work. Both employ a reinforcement learning algorithm, and what is particularly interesting is the potential to continue training the algorithm using traditional learning, providing an opportunity to combine both techniques for model training.

- Generative Adversarial Imitation Learning (GAIL) [24]
- Adversarial Inverse Reinforcement Learning (AIRL) [25]

3.3 Robot Operating System

Robot Operating System (ROS) [26] is an open-source meta-operating system that provides a set of tools for software development for robots. ROS offers most of the expected services of an operating system, including hardware abstraction and device control. It comprises a series of tools and packages to facilitate the development of complex and robust robot behaviors.

ROS is composed of nodes following a graph architecture, where each node represents executable code. Nodes can be located on the same computer or distributed among computers and robots. The advantage of the structure used is that each node can handle a unique aspect of the system. The primary communication mechanism between nodes is through the sending and receiving messages, which are transmitted over channels called topics. Typically, a node publishes messages on a topic, and other nodes subscribe to the same topic. Whenever a message is published on it, all subscribed nodes receive it. There is a second form of communication that, unlike messages, which are unidirectional interactions, is bidirectional. This communication is called services, and the interaction begins with a node requesting information from another and then receiving a response.

Gazebo is a simulation software that, while not part of ROS, can be used through ROS. This tool enables the creation of highly realistic simulations with a wide range of options for modification. For example, it provides the ability to add and configure sensors and allows the modification of the simulation environment.

4 Proposed Solution

The proposed solution involves defining the problem, specifying the task that the agent must perform to achieve the goals set by the controller, and subsequently designing the corresponding solution. The decisions made and the components used to develop the controllers are outlined below.

4.1 Task definition and problem context

The principal objective is to achieve autonomous sailing of a sailboat to an arbitrary point and take samples there. While not mandatory, it is desirable for the sailboat to stay as close as possible to the target point when taking samples. To accomplish this, the sailboat's task is defined as following the fastest route from an initial point to the target point while staying as close as possible to it.

Several environmental conditions that are changed during each execution are taken into account. These include the target point, wind speed and direction, water current speed and direction, and the initial orientation of the sailboat. In addition, minimum and maximum values for wind speed and water current are established to avoid unnavigable conditions. For the wind speed, the minimum is 2 m/s and, the maximum is 5.5 m/s, the maximum water current speed is 0.4 m/s; these limits are based on the Beaufort wind force scale[27] and simulation scenarios created by the USVSim simulator's authors[1].

4.2 Observation and Action Spaces

The definition of the observation space is crucial for successful learning. It should include all the necessary information to be able to take the best decision while also avoiding an excess of data, which would cause a slowdown in the learning process.

Observation Space:

- Distance to target: The distance between the sailboat and the target point.
- Heading error: The angle between the orientation of the sailboat and the direction it should be pointing to align with the target point.
- Sailboat velocity direction: The direction in which the sailboat is moving.
- Sailboat speed: The speed at which the sailboat is moving.
- Wind direction: The direction of the wind.
- Wind speed: The magnitude of the wind velocity vector.
- Current direction: The direction of the water current.
- Current speed: The magnitude of the water current velocity vector.

All angles and velocities are with respect to the orientation of the sailboat.

The definition of the action space must specify which actions the controller can take and the accepted values for these actions.

Action Space:

- Rudder angle: Takes values between $[-\frac{\pi}{2}, +\frac{\pi}{2}]$.
- Sail angle: Takes values between $[-\pi, +\pi]$.

4.3 Action execution and observation obtaining

The appropriate waiting time to obtain the observation after an action execution is crucial and not trivial. If the wait time is too short, the actuators may still be in motion, or the impact on the environment may not have occurred. If the wait is too long, relevant information for sailboat control is lost. This issue significantly affects learning, especially for sailboat control, where an action's consequences and timing are highly variable and dependent on environmental conditions.

For this problem, B. Chen [28] proposes an innovative solution; however, it is not employed in this work due to its additional complexity. Instead, the average time required by the sailboat to move the actuators to the desired position is used. This is defined through a test involving a thousand randomly executed actions in a scenario with low environmental disturbances, resulting in an average value close to 0.5s.

4.4 Reward function

The complexity of the reward function can vary significantly. In general, a simpler version is easier to implement but may result in slower learning, while the complex version can accelerate learning but requires a deeper understanding of the problem.

Significant expertise and research are needed to comprehend navigation dynamics and formulate a complex reward function that represents the desired objective. Due to this reason, the decision is to use relatively straightforward reinforcement functions. Considering the task, the function rewards the sailboat when it approaches the target point and penalizes it when it moves away.

The reward function is divided into three different scenarios, each with an associated reward:

- $Reward = 1$: When the distance to the target is less than the previous observation.
- $Reward = 0$: When the distance to the target is equal to the previous observation.
- $Reward = -1$: When the distance to the target is greater than the previous observation.

The sailboat is considered to have moved if the difference between the previous and current distance is greater than an arbitrary epsilon.

4.5 Episode termination

A fixed horizon is used, where each episode concludes after a fixed number of steps. It is necessary to determine a sufficient number of steps to reach the target points through reasonably accurate navigation. For this, the number of steps it takes for the static controller to sail close-hauled against the wind and stay at the point for a few seconds is calculated. The choice of this heading is because it is one of the slowest ones. Twice the number of steps taken to reach the point is considered to ensure it is adequate for staying at the point for some time.

4.6 Reinforcement Learning and Imitation Learning Library

The choice of the library has a significant impact on the rest of the solution, so it is relevant to understand the key points upon which this decision is made. The first point is whether the library provides state-of-the-art algorithms, specifically the four algorithms selected in this work: PPO, A2C, TD3, and SAC. Another point is the quality of the official documentation, as good documentation can accelerate the library's implementation process and facilitate the resolution of potential issues. An additional, though not mandatory, consideration is whether the library provides imitation learning algorithms, especially those selected for this work: AIRL and GAIL. If the library lacks these algorithms, it is necessary to evaluate the complexity of integrating an external library that does provide them. A final point for comparison is understanding how complex it is to interface the library with custom environments.

Considering these comparison points, a search for libraries that meet the most expected requirements is conducted, focusing on non-academic articles since there are no formal works on this comparison. Based on the articles [29, 30, 31], the choice is narrowed down to three possible libraries: Stable Baseline 3 [32], Tensorforce [33], and TF Agents[34]. Of these, Stable Baseline 3 is selected as it offers better documentation compared to the others. While it does not provide imitation algorithms, a separate imitation learning library called Imitation[35] implements them and is designed to be used in conjunction with Stable Baseline 3.

4.7 Architecture

The two main components are the Adaptive Controller and the USVSim Simulator [1]; there is also an auxiliary component called Metrics. The Adaptive Controller component includes the "Training Algorithm" module, where, among other configurations, the type of training is determined by whether imitation learning is used or not, the reinforcement learning algorithm used, and if manual control is desired. As shown in Figure 5, this module communicates with the "Gym" module through the functions of the Gym interface *step* and *reset*.

At the beginning of each episode, the "Gym" is responsible for configuring the variables that define the episode's environment. Then, at each step, it takes an action and returns the information required by the *step* function. This includes obtaining environmental information, processing it, and returning an observation, as well as the reward for the action taken and whether the episode has terminated. To accomplish this, it communicates with the "Simulator Interface", which provides all the necessary information. Communication with this interface is primarily through the functions 'set_up_env,' 'set_action,' and 'get_obs.' As the names suggest, these functions correspond to setting up the initial conditions of the environment before starting an episode, configuring the rudder and sail angle at each step, and obtaining information about the environmental conditions at each step.

To provide the required information, the "Simulator Interface" communicates with the USVSim Simulator. This communication is done using the topics and services provided by ROS. Specifically, the 'sailboat/state' topic is utilized to obtain information about the sailboat, such as its position and velocity. Then, the configuration of the currents in each episode and obtaining this information at each step is done through the 'gazebo/current' topic, and for the wind, the 'windCurrent' service is employed. The communication of the angle at which the actuators should be positioned at each step is accomplished through the 'sailboat/joint_setpoint' topic. Configuring the initial orientation and position of the sailboat in each episode is

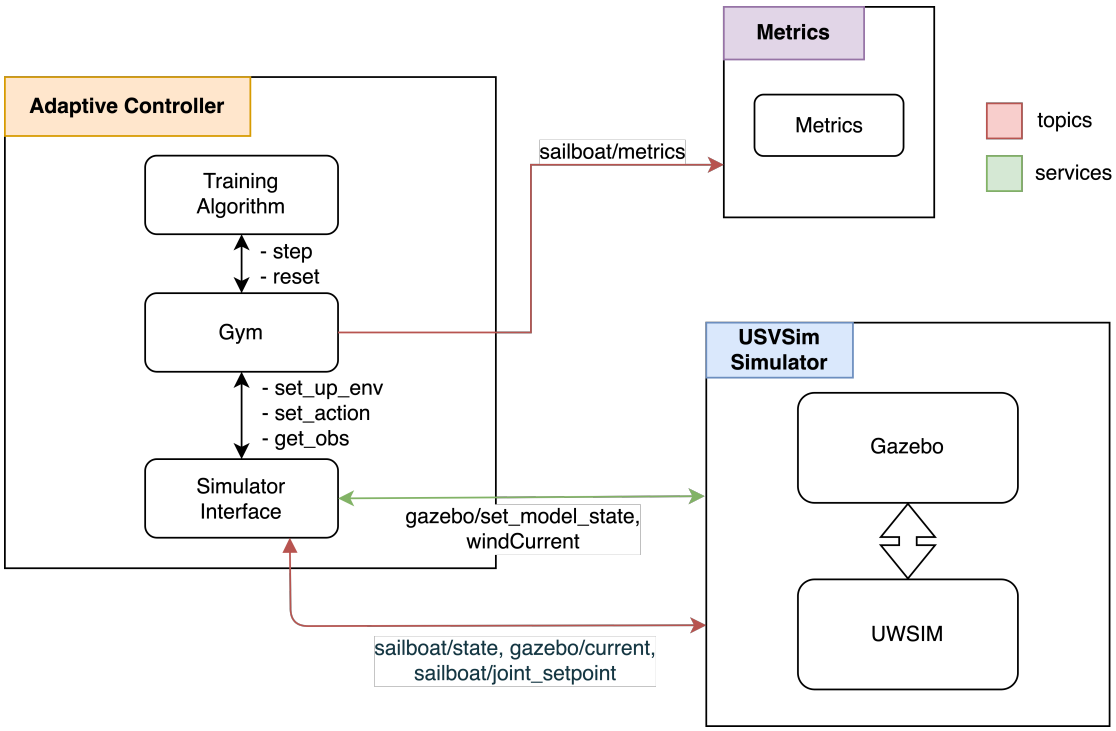

Figure 5: Architecture of the proposed solution

achieved through the 'gazebo/set_model_state' service. Detailed information about communication within the simulator can be found at USVSim Simulator Documentation [1].

Finally, we have the Metrics component, which fulfills the need to save training information. For this purpose, a topic named 'sailboat/metrics' is created, where information about observation, reward, actions, and the episode is published at each step.

5 Experimentation

Reinforcement learning techniques require interacting with the environment at each step to obtain the observation of the performed action, making the learning process time-consuming. In this study, training takes place in a simulator, which is adjusted to operate at approximately ten times the real-time speed.

However, despite this enhancement, the times required to achieve convergence in learning are very extensive. Due to this issue, testing all ideas and approaches in a final experiment becomes extremely expensive. Therefore, the choice is made to conduct pre-experimental tests, where simplified scenarios are generated to validate and discard ideas before proceeding to the final experimentation.

Below are detailed metrics, scenario generation, pre-experimental tests, and, finally, experimental tests.

The experimentation was conducted on an MSI PS63 Modern 8RC laptop with the following specifications:

- Operating System: Ubuntu 18.04
- ROS version: ROS Melodic Morenia
- RAM : 16GB DDR4-2666
- CPU: Intel Core i7-8565U
- GPU: Nvidia GeForce GTX 1050 Ti Max-Q

5.1 Metrics

In this section, three metrics are defined to compare the results of the different controllers. Depending on the type of test being conducted, a subset or the entirety of these metrics is chosen. This distinction is detailed in the description of each test.

Average Reward: This metric is calculated as the average of the reward obtained by the agent at each step. This metric is the most traditional in reinforcement learning and does not directly refer to the task to be performed but rather to the defined reward function. High values of this metric indicate how well the agent is able to behave with respect to the defined reward function. This distinction is important because the definition of the reward function may not accurately reflect the task to be performed.

Let n be the number of episodes, and let r_i be the cumulative reward of episode i. The metric used is as follows:

$$average\_reward = (\sum_{i=1}^{n} r_i)/n$$

Number of episodes: This metric is the number of episodes in which the sailboat, at some point during the entire episode, is at a distance less than a specified threshold from the target point. The chosen threshold is two and a half times the length of the sailboat. The goal here is to understand if the sailboat was able to reach the target at least once, regardless of whether it deviated significantly before or after reaching it.

Let L be the length of the sailboat, and the specified threshold $\epsilon = 2.5L$. Let d_t be the distance between the sailboat and the target point at time t during an episode. Let T be the total duration of the episode. Define $I(e)$ as an indicator function that returns 1 if the condition inside is true for episode e, and 0 otherwise. The metric used is as follows:

$$I(e) = \begin{cases} 1 & \text{if } \exists t \in [0, T] \text{ such that } d_t < \epsilon \\ 0 & \text{otherwise} \end{cases}$$

The metric $number\_of\_episodes$ can be defined as the sum of $I(e)$ over all episodes:

$$number\_of\_episodes = \sum_{i=1}^{n} I(i)$$

Average Cumulative Distance: For each episode, the Euclidean distance from the center of the sailboat to the target point is summed over each of its steps. The average across all episodes is then calculated. This metric is expressed in meters. This metric aims to understand how close the sailboat is to the target point throughout the entire episode.

Let d_{ij} be the distance between the sailboat and the target point at step i of episode j, n be the number of episodes evaluated, and m be the number of steps in each episode.

$$average\_cumulative\_distance = (\sum_{j=1}^{n} \sum_{i=1}^{m} d_{ij})/n$$

5.2 Scenarios

There are two methods of generation for the scenarios: manual and automatic. Each scenario begins with the sailboat in a fixed initial position and is defined by the tuple: (Target Point, Initial Orientation, Current, Wind).

Automatic generation: The scenarios are generated randomly, and the restrictions for generating the tuple are detailed below:

- Target Point: The target point is determined by the pair (x, y) representing coordinates in the global frame. The sailboat's starting point is taken as the reference center, from which a minimum and maximum radius are defined, forming a ring within which the x and y values must lie. The minimum and maximum radius are defined using as reference the distance of the points chosen in the trajectories of the manual controller, considering that the difference between the maximum and minimum can be covered in a few steps.
- Initial Orientation: This value is a number within the range [-π, π], which represents the orientation in radians of the sailboat at the beginning of the episode.
- Current: The water current is expressed as a vector (x, y). To generate it, a speed is randomly chosen between the maximum and minimum current speeds, along with an angle within the range [-π, π]. The speed is then decomposed into the x and y values.
- Wind: This value is analogous to the current, but using the maximum and minimum wind speed values.

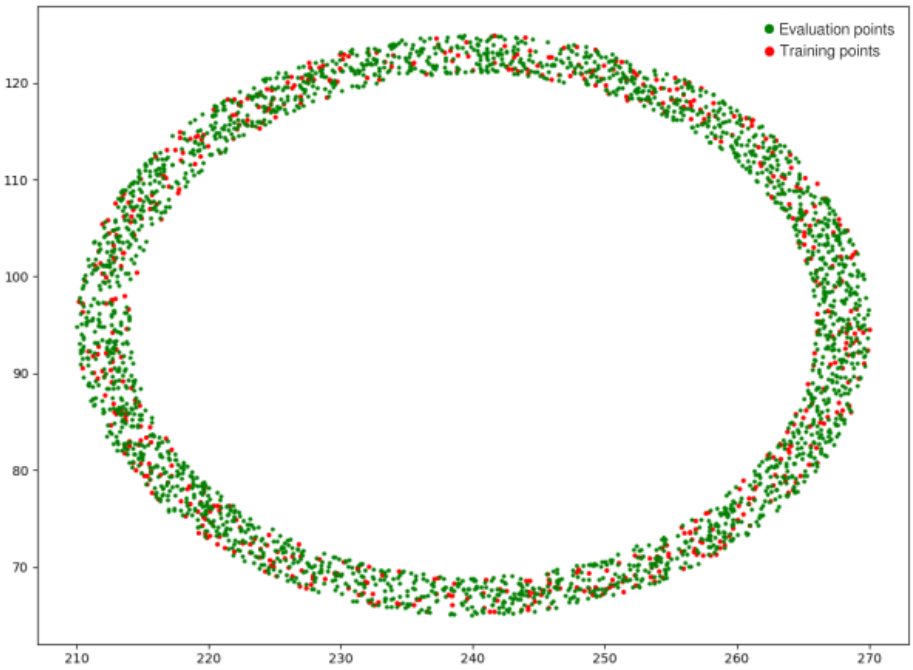

Figure 6: Example of 3000 randomly generated target points, with green representing those used in training and red representing evaluation points.

In Figure 6, an example of randomly generated target points is presented, where the sailboat starts from the center of the ring.

Manual generation: These scenarios are generated using the same tuple and constraints as in automatically generated scenarios, with the difference that values are manually chosen.

The different types of scenarios listed below are defined according to the test to be carried out:

- Scenarios for Manual Controller Trajectories:
  The objective is to generate trajectories for diverse scenarios where the manual controller performs well. To achieve this, eighteen scenarios are generated, attempting to cover all points of sail except dead zones and close-hauled sailing. Two moderate winds are used, with half of the scenarios featuring a wind speed of 2.0 m/s and the other half with a wind speed of 3.0 m/s. These values are chosen as the manual controller navigates effectively under these conditions. There is no water current because the manual controller does not take this information into account and fails to reach the target point if there is a significant current, as can be seen in the pseudocode of the controller used in the Manual controller 1.

- Scenarios for Pre-Experimental Imitation Tests:
  Due to the small number of training steps, scenario generation for these tests is done manually to simplify and expedite the learning process. In this scenario generation, reference points are taken from the chosen trajectories for the manual controller, with additional points added between them to introduce more variety. Fifteen target points are established. Three maritime current options are considered: no current, 0.1 m/s current along the x-axis, and 0.1 m/s current along the y-axis. The maritime current is included to introduce a variable that the manual controller does not consider without overly complicating the scenarios. To create a scenario, one of the fifteen points and one of the three established currents are selected, keeping wind speed and initial sailboat orientation constant across all scenarios.

  The proposed scenarios for evaluating these tests consist of four points, representing a beam reach and broad reach for each side. Additionally, two current possibilities are presented: no current and 0.05

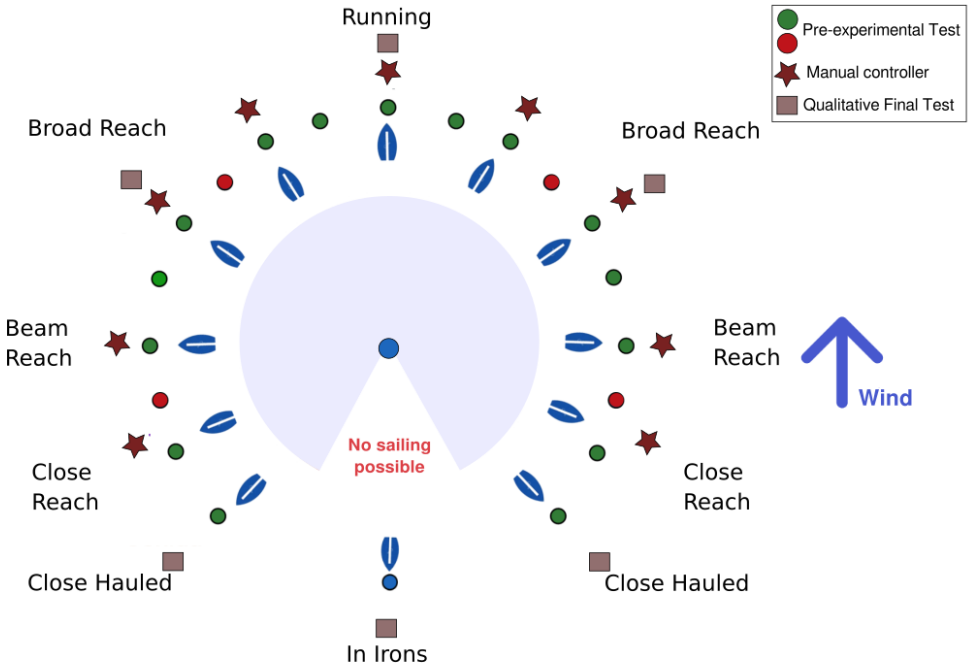

Figure 7: Manual generated scenarios: Represented with stars are the courses used for the manual controller trajectories. Represented with circles are the directions chosen for the pre-experimental imitation learning tests; the training objectives are in green, while the evaluation objectives are in red. Represented with squares are the selected courses for qualitative evaluation

m/s in both axes. This results in the generation of eight different scenarios.

- Scenarios for Qualitative Final Tests:
  These scenarios are generated to observe how the sailboat navigates in different situations. Six target points are proposed, three of which are expected to be less challenging: running and broad reach on both sides. The other three pose a greater challenge: into the wind and close-hauled on both sides.

  Two distinct weather situations are also considered—one representing calm sailing conditions and the other depicting turbulent conditions. For the calm scenario, a nearly no-current condition of 0.05 m/s and a moderate wind speed of 3.0 m/s are determined. The turbulent scenario features a significant current, with 0.2 m/s and a strong wind speed of 5.0 m/s.

Figure 7 illustrates the different types.

5.3 Manual controller trajectories

For imitation learning techniques like GAIL and AIRL, it is necessary to have trajectory data generated by the manual controller,in this work this controller follows the logic described in Algorithm 1. The data for each trajectory includes, for each step, the action taken, the observation, and the obtained reward. The chosen imitation library provides functionalities to collect this information, so it is only necessary to determine in which scenarios the trajectories will be obtained. When defining the scenarios, it is crucial that the trajectories reflect a sequence of good actions since the learning process is built upon them. Based on this, the scenarios for the manual controller trajectories described in 5.2 are established.

5.4 Pre-experimental tests

Pre-experimental tests are conducted to validate ideas with the aim of reducing the number of combinations for the final tests. The metrics used in these tests are average accumulated distance and the number of episodes. These tests have two main objectives. The first one is to define the reward function, and the second, is to try to understand if the use of imitation learning algorithms can be beneficial for learning the proposed task.

While both tests are pre-experimental, there are a few differences between them in terms of the amount of time dedicated to them and the scenarios used for their training and evaluation. Reward function tests are

Algorithm 1: Manual controller

Input: wind, target_pose, sailboat_pose
Output: sail_angle, rudder_angle

```
1  static previous_integral_term
   // proportional
2  Def P(error):
3      return K_P * error
   // integral
4  Def I(error):
5      previous_integral_term = previous_integral_term + K_I * error
6      return previus_integral_term
   // sail controller
7  wind_dir = atan2(wind.y, wind.x)
8  sail_angle = (wind_dir − sailboat_pose.yaw)/2
   // rudder controller
9  sp_angle = atan2(target_pose.y − sailboat_pose.y, target_pose.x − sailboat_pose.x)
10 err = sp_angle − sailboat_pose.yaw
11 rudder_angle = (P(err) + I(err))/2
```

conducted in a scenario closer to the final experimentation, as its definition has a big impact on the final experimentation, whereas imitation learning tests are more exploratory.

5.4.1 Imitation Learning tests

These experiments are conducted with the objective of understanding whether incorporating imitation learning into the final tests can be beneficial. For this, six types of training are defined, and each is tested with the four previously defined reinforcement learning algorithms. These are the details of the six training types:

- RL: The agent goes through the entire training using only reinforcement learning.
- GAIL: The agent goes through the entire training using only imitation learning with the GAIL technique.
- AIRL: The agent goes through the entire training using only imitation learning with the AIRL technique.
- GAIL + RL: This agent goes through the first 200,000 steps using imitation learning with the GAIL technique and then continues the remaining 200,000 steps using reinforcement learning.
- AIRL + RL: This agent goes through the first 200,000 steps using imitation learning with the AIRL technique and then continues the remaining 200,000 steps using reinforcement learning.

All the trying types, except for AIRL and AIRL + RL, are tested using the four algorithms described previously: SAC, PPO, A2C, TD3. AIRL and AIRL + RL are tested using only A2C and PPO since the AIRL algorithm is not compatible with SAC and TD3, all of these combinations result in 16 agents.

The experimentation involves training for 400,000 steps and evaluating the agents over 16 episodes. The amount of training step is determined to be a relatively low number but sufficient for most of the agents to reach certain points. Training and evaluation utilize scenarios from pre-experimental imitation tests outlined in Section 5.2.

In the results, there is no clear advantage of using imitation learning algorithms. The results strongly depend on which algorithm is analyzed. Particularly, with the PPO algorithm, better results are achieved when using the GAIL + RL agent and even more when using only the GAIL agent. In contrast, when analyzing TD3, there is a clear superiority in using RL compared to imitation techniques. Although RL has better results with SAC, these are similar to those obtained with agents using imitation. Lastly, A2C is not considered in the analysis due to extremely negative results across all agents.

This parity supports further investigation of the benefits of using imitation learning in the final experimentation. However, looking at the results obtained with the imitation learning algorithms AIRL and GAIL in their various combinations, it can be determined that GAIL is superior to AIRL. For this reason, considering

Agent		Metrics	
Algorithms	Training Type	Avg. Accumulated Distance [m]	Number of Episodes
A2C	RL	8504.75	0
	GAIL	8504.11	0
	AIRL	8653.42	0
	GAIL + RL	8503.48	0
	AIRL + RL	**8502.56**	0
PPO	RL	8515.20	0
	GAIL	**3673.72**	**13**
	AIRL	8595.47	0
	GAIL + RL	5113.34	7
	AIRL + RL	8104.93	0
SAC	RL	2231.63	**16**
	GAIL	**2114.26**	**16**
	GAIL + RL	3796.94	12
TD3	RL	**3952.24**	**12**
	GAIL	8504.59	0
	GAIL + RL	5706.33	4

Table 3: Results of the imitation learning tests.

the limitation of not being able to use the AIRL algorithm with two of the algorithms, the decision is to only utilize GAIL and GAIL + RL in the final experimentation.

5.4.2 Reward Function tests

These tests aim to define the reward function to be used, proposing three different functions. All functions share the following idea: when the sailboat moves away from the target, a reward of -1 is returned; when it approaches the target, a reward of 1 is returned, and if it neither, a reward of 0 is returned. The variation is the value returned once the sailboat reaches the goal. Reaching the goal is defined as being within a certain radius centered on the target point.

The three functions to be tested, as previously said, return different values when the sailboat is at the target point:

- Function 1: Returns 10 when it is at the target point.
- Function 2: Returns 1 when it is at the target point.
- Function 3: This function maintains the same calculation as when it is outside the target point.

For this test, 1000 different scenarios are generated, 800 for training and 200 for evaluation using the automated method detailed in Section 5.2. Each of the functions is tested with the four Reinforcement Learning algorithms already described: SAC, PPO, A2C, TD3.

When analyzing the results (Table 3 and Table 4) of the reward function tests, it can be seen that Function 3 achieves the best results or results not far from the best in all algorithms and in both metrics. Therefore, its performance is considered more consistent than the others, making it the preferred choice for conducting the final tests.

5.5 Experimental tests

Once the pre-experimental tests are completed, the final tests are carried out, where the controllers are trained in complex scenarios using a large number of steps, and their performance is evaluated using different metrics. The tests are divided into four parts, described below. Section 5.5.1 presents the training and quantitative evaluation of the controllers, comparing them with the manual controller. Section 5.5.2 presents the qualitative evaluation between the best controllers of the first part and the expert controller. Section 5.5.3 presents the quantitative evaluation of the best controllers but varying the dimensions of the sailboat, seeking to evaluate how the controllers adapt to these changes. Finally, Section 5.5.4 presents the tests and shows exploratory tests carried out to propose future work.

Agent		Metrics	
Algorithms	Function	Avg. Accumulated Distance [m]	Number of Episodes
A2C	Function 1	42968.94	**16**
	Function 2	43110.95	15
	Function 3	**42153.71**	15
PPO	Function 1	**17048.40**	**150**
	Function 2	47485.22	13
	Function 3	20612.99	138
SAC	Function 1	25416.65	109
	Function 2	16648.85	**135**
	Function 3	**15642.66**	128
TD3	Function 1	27085.93	71
	Function 2	20442.26	**122**
	Function 3	**16648.85**	120

Table 4: Results of the Reward Function tests.

Agent		Metrics		
Algorithm	Training	Average reward	Accumulated distance [m]	Number of episodes
TD3	RL	-586.56	90576.18	55
	GAIL	-485.77	**48400.30**	58
	GAIL + RL	**-480.16**	49985.42	**81**
SAC	RL	**249.05**	**11834.66**	**445**
	GAIL	-468.75	47607.80	58
	GAIL + RL	-31.54	26718.31	174
PPO	RL	**-53.03**	**19244.34**	**418**
	GAIL	-304.29	43372.99	156
	GAIL + RL	-230.85	28368.07	354
Manual	-	-275.85	24188.99	402

Table 5: Quantitative evaluation results in 600 episodes

5.5.1 Training and quantitative evaluation

Taking as input the results of the pre-experimental tests, three types of training applied to three algorithms (SAC, PPO, TD3) are defined, where it is important to highlight that the A2C algorithm is not taken into account for the final tests due to the poor results and for the computational cost of including it in them. Furthermore, the AIRL learning technique is not considered by what is argued in the conclusions of the imitation tests.

The three training types used are:

- RL: This agent performs the entire training only with reinforcement learning.
- GAIL: This agent performs the entire training only with imitation learning using the GAIL technique.
- GAIL + RL: This agent performs half of the steps with imitation learning using the GAIL technique and then continues the remaining steps using reinforcement learning.

For these tests, 3000 different scenarios are generated using the automatic method, detailed in Section 5.2. 2400 are used for training and 600 for evaluation. Having defined the nine agents, the scenarios and the metrics, the training is carried out, which is made up of two million steps, and then 600 episodes are carried out for evaluation.

Table 5 shows that the imitation learning technique does not obtain good results with any of the algorithms used. With two of the algorithms, PPO and SAC, the results get worse when using the GAIL + RL or the GAIL agent and are even worse when using the latter. That is not the case with the TD3 algorithm, which shows similar results with the three agents.

Although the results obtained with imitation learning were poor, it does not necessarily mean that these techniques are not good; instead, multiple factors can affect their performance. In this specific case, two

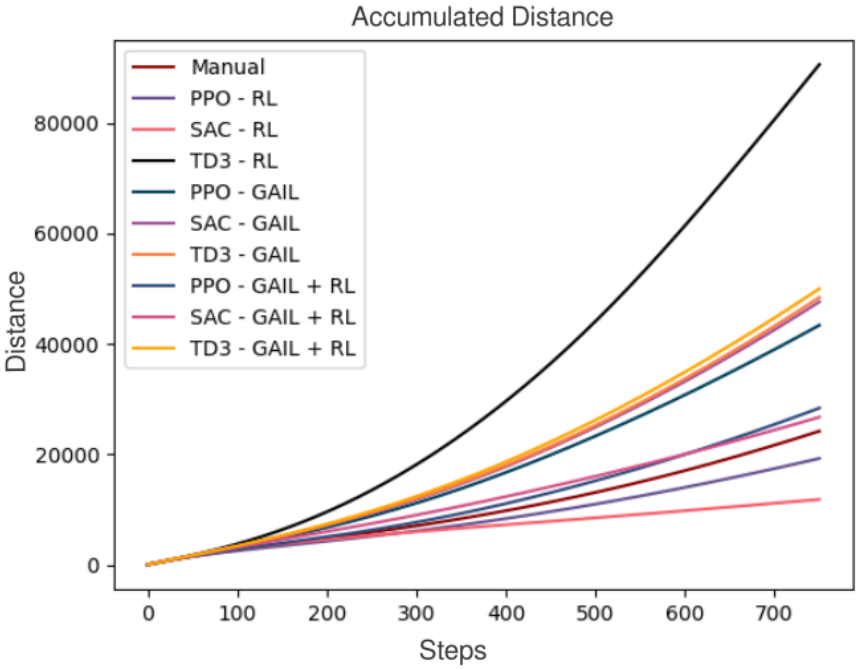

Figure 8: Results of the quantitative evaluation using the average cumulative distance metric which is expressed in meters

factors that may have affected are the manual controller, which is taken as the expert for the imitation, and the number and variety of trajectories used.

Focusing only on the training that uses the RL agent, it can be seen that TD3 does not achieve good results, while PPO and SAC do achieve good results, better than those obtained with the manual controller with the three metrics used. In comparison with the latter, it can be seen that SAC has a vastly superior performance in the average reward and average accumulated distance; however, in the number of times the PPO objective is reached, it is not far from SAC.

When observing Figure 8, which shows the average accumulated distance metric, it can be seen that SAC stands out above the rest by staying close to the goal after 400 steps. From this point on, the rest of the agents begin to increase the average accumulated distance considerably. This may indicate that SAC manages to learn to stay close to the objective throughout the episode and not only in a part of it.

5.5.2 Qualitative evaluation

The two controllers that present the best results in the quantitative test, SAC and PPO, together with the RL technique, are selected and qualitatively evaluated with the manual controller. From now on, it will be referenced as its algorithm's name. This qualitative testing aims to analyze the controllers' performance in arbitrary scenarios to try to understand what navigation capabilities they generated.

Six specific points are proposed, three of which should not be very difficult to navigate: downwind and on a broad reach on both sides. The other three pose a greater challenge: upwind and on a close-hauled course on both sides. The mentioned courses can be observed in Figure 7. Additionally, two different weather conditions are considered, one with low disturbances and the other with high disturbances. For each point, two scenarios are generated, one for each of the mentioned weather conditions. In total, twelve scenarios are determined.

Table 6 presents the results obtained in each of the scenarios evaluated with the different metrics used. In them, the objective points are identified according to the course and the environment, divided according to the two climatic conditions: high and low disturbances.

There are differences in results when looking at the cumulative reward or the remaining metrics. In the case

Target point		Final reward			Cumulative distance [m]			Step to reach the objective		
Environment	Course	SAC	PPO	Manual	SAC	PPO	Manual	SAC	PPO	Manual
High Disturbance	Upwind Left	**313**	-561	-751	**17208.94**	23886.35	61399.29	-	-	-
	Bow	**305**	-751	-736	**26338.63**	35120.63	47893.62	-	-	-
	Upwind Right	**-52**	-432	-302	33446.41	111902.65	**21370.36**	-	-	-
	Right Length	**158**	-476	-499	**12340.63**	42079.17	40042.23	120	**67**	70
	Stern	**298**	-577	-279	**10941.31**	13813.81	35792.36	239	**78**	-
	Left Length	**220**	-551	-265	19518.06	41155.45	**17271.61**	182	**83**	320
Low Disturbance	Upwind Left	93	-163	**126**	9290.22	6592.39	**6586.11**	312	221	**196**
	Bow	**271**	-11	-90	20099.34	**10022.44**	25597.88	-	**320**	-
	Upwind Right	**443**	-154	106	13626.66	8259.97	**6375.34**	-	273	**193**
	Right Length	**432**	-82	9	12404.20	7746.01	**5642.47**	527	288	**89**
	Stern	**741**	-495	-68	14963.87	8721.61	**4967.55**	712	101	**91**
	Left Length	**328**	-324	77	20046.97	7840.99	**7065.70**	712	145	**117**

Table 6: Qualitative test results. The target point is determined according to the heading and the environment, which is divided according to two weather conditions: high disturbance and low disturbance.

of reward, SAC is substantially superior to PPO and Manual, whereas the latter has similar performance. The results do not hold if the cumulative distance metric is observed, where SAC has the best results in high disturbance scenarios, but for low disturbance scenarios, it is surpassed by PPO and Manual. Finally, when analyzing the remaining metric, the results are reversed since SAC obtains the worst results in the number of points at which it reaches the objective and in the speed at which it reaches it. On the other hand, PPO, in this case, has better results both in the number of points you reach and in the speed you do so.

When analyzing the headings that the different controllers manage to reach, it can be seen that all of them reach the target point in an environment of low disturbances and downwind, that is, long and downwind headings. Something similar happens in a high-disturbance environment, where all three controllers manage to reach the target points on the long heading, and all but the manual one achieves the down heading. A totally different panorama is given when analyzing the upwind and bow courses, courses that are very difficult to navigate. None of the controllers manage to reach them when navigating in a high-disturbance environment, however, the PPO controller manages to reach all of them when faced with a low disturbance environment. Even achieving zigzag navigation is necessary to reach the point that is against the wind.

To conclude, SAC obtains a very good reward by approaching the objective slowly, arriving late, or not arriving at all. One of the causes of this behavior is that staying on the point is a challenging task and, in many cases, more complex than navigating to a point. So, in terms of the task that is sought to be achieved, the behavior of SAC is not what was intended. On the other hand, PPO does not obtain good results in the reward, but when analyzing its navigation, it performs better. Although SAC surpasses PPO in the number of points achieved in the quantitative tests, by observing how some points are achieved, it is estimated that PPO has more realistic navigation by observing how some points are achieved.

5.5.3 Evaluation changing the configuration of the sailboat

Using the same controllers as in the qualitative evaluation, the aim is to evaluate the adaptation of the controllers to the change of some of the components of the sailboat, modifying the sail, rudder, and keel.

Since the simulator multiplies them by the area of the component in question to calculate the forces applied to each component, it is decided to modify the area to simulate different configurations of the sailboat. The areas used in training are taken as a basis, and from these, the value is increased to analyze how this change affects the controller's performance.

Below are the configurations of the sailboat to be evaluated:

- Original: Sail = $1.44m^2$, Keel = $1.00m^2$, Rudder = $1.00m^2$
- Dimension 1: Sail = $1.80m^2$, Keel = $1.25m^2$ Rudder = $1.25m^2$
- Dimension 2: Sail = $2.16m^2$ Keel = $1.50m^2$, Rudder = $1.50m^2$

The scenarios for this test are the same 600 used for the quantitative evaluation carried out previously, where the same three metrics are also used.

When analyzing the results (Table 7) of the adaptive tests, it can be seen that both SAC and PPO worsen as the dimensions of the sailboat parts increase; however, this drop in performance is not seen in the Manual controller. Despite this decline in reinforcement learning controllers, it is important to mention that SAC maintains better results than the Manual controller in all cases, and PPO generally surpasses it.

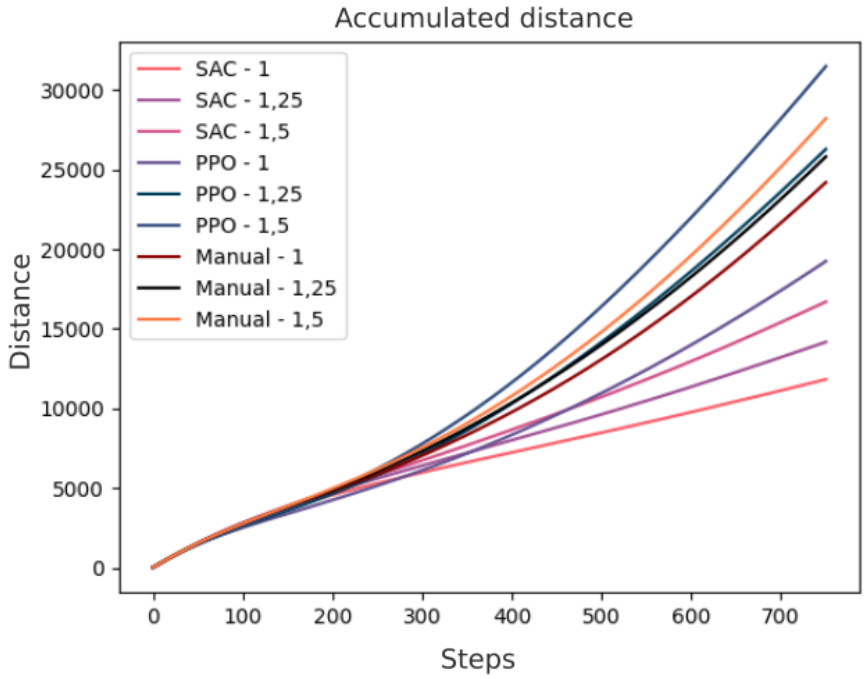

Figure 9: Results of the tests modifying the configuration of the sailboat using the average accumulated distance metric, which is expressed in meters.

Dimension	Controller	Average reward	Average distance [m]	Number of episodes
Original	SAC	**259.05**	**11834.66**	**445**
	PPO	-53.03	19244.34	418
	Manual	-275.85	24188.99	402
Dimension 1	SAC	**240.75**	**14175.04**	**432**
	PPO	-103.15	26276.04	414
	Manual	-260.53	25805.71	415
Dimension 2	SAC	**204.73**	**16696.89**	**408**
	PPO	-131.03	31471.47	399
	Manual	-279.86	28201.31	**408**

Table 7: Results of the tests modifying the sailboat configuration

5.5.4 Other Tests

In an effort to enhance the outcomes obtained, we conducted a series of tests to assess potential solutions for future endeavors. The objective is to develop a controller that exhibits heightened adaptability to various weather conditions and natural wear on different mechanical elements.

Variable Horizon Tests: It is decided to explore whether using a variable horizon makes learning to navigate to a point easier. The same training and evaluation as used in the final tests with the PPO and SAC algorithms are conducted. Since the horizon is variable, the relevant metric for comparison is the number of episodes. The reward is modified to reach the target point quickly. During the journey to the target, the reward remains the same as in the previous tests. Once the target point is reached, the subtraction between the limit of steps and the steps it took to reach the goal is performed, multiplied by two. The controller generated with the PPO algorithm reached 489 target points, 71 points more than the controller generated with this algorithm. On the other hand, considering the SAC algorithm, the result is 418 target points, 27 fewer points than the controller generated with this algorithm. While the controller does not improve in the number of target points in this case, observing its behavior suggests an improvement in navigation in easier-to-sail directions, advancing quickly toward the target point. The results of these minor tests are

inconclusive; however, this modification could be a promising path to achieve better results.

Training Extension Tests: It is desired to evaluate whether controllers achieve better performance with extended training, as the number of training steps along with hyperparameter configuration may be reasons for not achieving optimal learning performance. As mentioned earlier, obtaining the optimal hyperparameter configuration is time-consuming, so extending training is explored as a possible solution. For these tests, the selected PPO controller's training is extended by two million additional steps under the same conditions as its previous training. Subsequently, evaluation is performed using the 600 episodes used in the quantitative tests. The controller obtained after this extended training performs worse than the PPO controller before this training in all three metrics. The average accumulated distance increases from $19,244.34m$ to $23,839.81m$, the average reward obtained decreases from -53.03 to -72.28, and the number of episodes decreases from 418 to 390. This result suggests that increasing the number of steps is not a viable path to achieving better controllers.

6 Conclusions and future work

6.1 Conclusions

An adaptive controller for autonomous sailboats has been meticulously crafted, leveraging the principles of both reinforcement and imitation learning. This sophisticated design reflects a strategic integration of cutting-edge methodologies to ensure optimal performance in diverse scenarios.

The designed controller successfully navigates through a great variety of scenarios, surpassing manual control capabilities and even managing to sail against the wind. However, it does not achieve navigation in all presented weather conditions.

Regarding the use of imitation learning, it is concluded that it does not yield satisfactory results and, in some training sessions, even worsens performance. It would be beneficial to explore whether using another controller as an expert or adjusting the reward function improves overall performance.

Given the complexity of navigation as a task, results analysis suggests another possible approach: separating the behavior of sailing towards a target point from that of staying at a specific point. Experimentation with modifying the objective to sail towards a specific point shows promising results.

Another noteworthy conclusion relates to the validity of pre-experimental tests in such studies, as their results do not always align with those of experimental tests. One possible cause is the inherent nature of machine learning, where models need to converge towards an optimum for good results. Short tests may not allow sufficient time for convergence, leading to non-representative results compared to what could be achieved with continuous learning.

The "sim-to-real gap" in robotics refers to the differences between simulated environments and real-world conditions, which can hinder the direct application of simulated training to real robots. ROS helps minimize this gap by providing standard interfaces for both real and simulated robots, enabling seamless integration and testing of sensors, behavior, and actuators. These interfaces allow for more accurate calibration and validation, facilitating an iterative process that bridges the gap and enhances the reliability of robotic systems in practical applications.

6.2 Future work

The subsequent delineation elucidates a sequence of pertinent tasks that warrant thorough exploration for future endeavors in our research:

- Test the best controllers generated on real sailboats to evaluate their performance in the real world.
- The decision of the optimal time to wait to obtain the observation after performing an action has a significant impact on the reinforcement learning process. Therefore, it is interesting to incorporate the technique proposed in [28], which attempts to address this problem.
- Bearing in mind the costs associated with each training from scratch, it is interesting to incorporate techniques for reusing data and knowledge from previous training sessions. Two examples of these are *Experience Replay* [36] and *Reincarnating Reinforcement Learning* [37].
- While the majority of the related works we have detailed do not provide their code, it would be highly beneficial to expand the scope of comparison by incorporating additional techniques. Specifically,

include the static controllers developed in [13], and also explore the creation of more complex static controllers that incorporate a broader range of sensor data.

- Further refinement of the controller's adaptability is imperative, with an emphasis on addressing the specific weather conditions where navigation currently presents challenges. This entails a comprehensive analysis of the controller's responses to diverse weather scenarios and the implementation of targeted improvements.
- Separate the target tasks into two categories: one where the goal is to navigate towards a point and another where the goal is to remain at a specific point. Since both tasks individually pose high complexity, designing a reward function that considers both is not trivial.
- Focusing on the long-term durability of the controller is imperative. Evaluating its performance over extended periods and addressing any potential wear-and-tear issues will contribute to the sustained effectiveness of autonomous sailboat navigation.
- Explore alternative reward functions by incorporating the knowledge of navigation experts and attempting to align them with effective sailboat control better.
- Incorporate obstacle avoidance capabilities into the controller. That would involve several changes, especially adding the necessary information to the observation state and adjusting the reward function to penalize any type of collision.
- One notable aspect of sailboats as autonomous vehicles is their ability to be propelled by the wind without needing any external power source. However, an external energy source is necessary for the movement of actuators. Therefore, a future task involves penalizing the movement of actuators, as executing their motion requires energy that is desirable to minimize. Perhaps this could lead to additional benefits, as in large vessels where minimizing sail movement is also sought since each movement decreases the acquired speed.
- Study the use of transfer learning to improve the performance of controllers when transitioning between sailboats or modifying their dimensions.
- The IA techniques employed in this work can be straightforwardly extended to broaden the action space and incorporate additional controllable elements. By leveraging advanced control algorithms, autonomous sailboats can expand beyond the conventional reliance on rudder and boom angle control. Integration of more sophisticated control surfaces presents a promising avenue for enhancing sailing efficiency and adaptability.

  In prioritizing and advancing these tasks, our aim is to propel the autonomous sailboat controller towards a state of heightened adaptability, robust performance, and seamless navigation across an extensive range of challenging conditions.

References

[1] M. Paravisi, D. H Santos, V. Jorge, G. Heck, L. M. Gonçalves, and A. Amory, "Unmanned surface vehicle simulator with realistic environmental disturbances," *Sensors*, vol. 19, no. 5, p. 1068, 2019.

[2] B. Bingham, C. Agüero, M. McCarrin, J. Klamo, J. Malia, K. Allen, T. Lum, M. Rawson, and R. Waqar, "Toward maritime robotic simulation in Gazebo," in *OCEANS 2019 MTS/IEEE SEATTLE*. IEEE, 2019, pp. 1–10.

[3] Alexandre EVAIN, "Simulation of an autonomous sailboat with ROS," 2020. [Online]. Available: https://www.ensta-bretagne.fr/jaulin/rapport2020_evain.pdf

[4] S. Lemaire, Y. Cao, T. Kluyver, D. Hausner, C. Vasilovici, Z.-y. Lee, U. J. Varbaro, and S. M. Schillai, "Adaptive probabilistic tack manoeuvre decision for sailing vessels," *arXiv preprint arXiv:1903.06677*, 2019.

[5] Ulysse VAUTIER, Christophe VIEL. ROS nodes for plymouth sailboat project. Accessed: 5-04-2023. [Online]. Available: https://github.com/Plymouth-Sailboat/SailBoatROS

[6] M. Prats, J. Perez, J. J. Fernández, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 2577–2582.

[7] O. Kermorgant, "A dynamic simulator for underwater vehicle-manipulators," in *Simulation, Modeling, and Programming for Autonomous Robots: 4th International Conference, SIMPAR 2014, Bergamo, Italy, October 20-23, 2014. Proceedings 4*. Springer, 2014, pp. 25–36.

[8] J. M. P. Figueiredo and R. P. Abou Rejaili, "Deep reinforcement learning algorithms for ship navigation in restricted waters," *Mecatrone*, vol. 3, no. 1, 2018.

[9] Y. Cui, S. Osaki, and T. Matsubara, "Reinforcement learning boat autopilot: a sample-efficient and model predictive control based approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2868–2875.

[10] X. Zhang, C. Wang, Y. Liu, and X. Chen, "Decision-making for the autonomous navigation of maritime autonomous surface ships based on scene division and deep reinforcement learning," *Sensors*, vol. 19, no. 18, p. 4055, 2019.

[11] T. Ruzicka, "Model based design of a sailboat autopilot," 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:64633183

[12] P. J. Sterne and G. Hayes, "Reinforcement sailing," *Master of Science Artificial Intelligence School of Informatics University of Edinburgh*, 2004.

[13] C. Jegat, "Control of autonomous sailboats," 2019. [Online]. Available: https://www.ensta-bretagne.fr/jaulin/rapport2019_jeguat.pdf

[14] S. Yang, C. Liu, Y. Liu, J. An, and X. Xiang, "Generic and flexible unmanned sailboat for innovative education and world robotic sailing championship," *Frontiers in Robotics and AI*, vol. 8, p. 630081, 2021.

[15] J. Melin, "Modeling, control and state-estimation for an autonomous sailboat," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:109188728

[16] A. G. d. Silva Junior, D. H. d. Santos, A. P. F. d. Negreiros, J. M. V. B. d. S. Silva, and L. M. G. Gonçalves, "High-level path planning for an autonomous sailboat robot using q-learning," *Sensors*, vol. 20, no. 6, p. 1550, 2020.

[17] B. Clement, "Control algorithms for a sailboat robot with a sea experiment," *IFAC Proceedings Volumes*, vol. 46, no. 33, pp. 19–24, 2013.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[22] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[23] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, 05 2000.

[24] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[25] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," *arXiv preprint arXiv:1710.11248*, 2017.

[26] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[27] S. J. . R. Stul. Beaufort wind force scale. [Online]. Available: https://www.eoas.ubc.ca/courses/atsc113/sailing/met_concepts/08-met-waves/8e-beaufort-scale/index.html

[28] B. Chen, M. Xu, Z. Liu, L. Li, and D. Zhao, "Delay-aware multi-agent reinforcement learning for cooperative and competitive environments," *arXiv preprint arXiv:2005.05441*, 2020.

[29] M. F. Argerich. 5 frameworks for reinforcement learning on python. Accessed: 29-12-2023. [Online]. Available: https://towardsdatascience.com/5-frameworks-for-reinforcement-learning-on-python-1447fede2f18

[30] P. J. Vladimir Lyashenko. The best tools for reinforcement learning in python you actually want to try. Accessed: 29-12-2023. [Online]. Available: https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python

[31] T. Simonini. On choosing a deep reinforcement learning library. Accessed: 29-12-2023. [Online]. Available: https://blog.dataiku.com/on-choosing-a-deep-reinforcement-learning-library

[32] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021, http://jmlr.org/papers/v22/20-1364.html. Accessed: 2022-03-14.

[33] A. Kuhnle, M. Schaarschmidt, and K. Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Accessed: 29-12-2023. [Online]. Available: https://github.com/tensorforce/tensorforce

[34] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow," 2018, https://github.com/tensorflow/agents. Accessed: 2022-03-14.

[35] S. Wang, S. Toyer, A. Gleave, and S. Emmons. The `imitation` library for imitation learning and inverse reinforcement learning. Accessed: 04-05-2023. [Online]. Available: https://github.com/HumanCompatibleAI/imitation

[36] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," in *2018 56th annual allerton conference on communication, control, and computing (Allerton)*. IEEE, 2018, pp. 478–485.

[37] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Reincarnating reinforcement learning: Reusing prior computation to accelerate progress," *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 955–28 971, 2022.