



Yara: An Ocean Virtual Environment for Research and Development of Autonomous Sailing Robots and Other Unmanned Surface Vessels

Eduardo Charles Vasconcellos¹ · Álvaro Pinto Fernandes Negreiros² · André Paulo Dantas de Araújo^{1,4} · Raphael Guerra⁴ · Philippe Preux¹ · Davi Henrique dos Santos^{2,3} · Luiz Marcos Garcia Gonçalves² · Esteban Walter Gonzalez Clua⁴

Received: 4 June 2023 / Accepted: 11 December 2024
© The Author(s) 2025

Abstract

Overall, a big challenge in building a sailboat USV relies on the development of an autonomous system for guidance, navigation, and control (GNC) because both sail and rudder angle must be cooperatively adjusted to correct the navigation direction — traditional propelled boats can be more easily controlled with a straightforward control task to set the rudder angle. Moreover, sailing upwind requires special maneuvers to reach a given target in that unfeasible direction. Reinforcement learning emerges as a promising technique for building autonomous GNCs for sailing robots, but training the neural network with a real sailboat is impractical due to long periods of training and safety reasons. Even traditional control-based approaches are mainly tested in simulated environments due to the difficulties in building and operating a real sailboat. The issue that arises is the fidelity of these simulated environments. In this context, we propose Yara, an oceanic virtual environment with a reliable physics simulation for developing, training, and evaluating autonomous agents to operate digital twins of sailing robots in reinforcement learning and other paradigms. An autonomous sailing robot digital twin is available within the virtual environment, with the foil dynamics constructed based on a real sailing robot. We coupled these foil dynamics in Gazebo's physics engine to compute the lift and drag forces acting on the sail, rudder, and keel. The simulated world feeds sensors such as cameras, wind sensors, and GPS. The Robot Operating System communicates these sensors' data through topics, facilitating users' implementation and testing of new GNC solutions. Yara provides a reliable solution for foil dynamic simulated physics that achieves a simulation speedup of 300 times on an i7 laptop with 8 GB of RAM, powered by a Nvidia RTX 3060 and running Ubuntu 20.04. With this speedup, it is possible to complete a million time steps of deep reinforcement learning training in approximately eight hours. Evaluation scenarios were presented to highlight specific features of the simulator, like the maneuverability of the sailing robot digital twin and applications to train, evaluate, and compare reinforcement learning agents and other control solutions.

Keywords Digital twin · Unmanned surface vessel · Autonomous sailboat · Reinforcement learning

1 Introduction

The development of Unmanned Surface Vessels (USVs), also known as Autonomous Surface Vessels (ASVs), is currently an interesting and exciting research field, with a vast amount of applications such as surveillance, rescue missions, structural inspection, and environmental monitoring, among others. As a USV can be operated remotely or autonomously, it can perform dangerous and/or long-duration missions at a cost smaller than a manned vessel. In this work, we deal with a sailing robot, a particular type of USV where the main source

of propulsion is the wind. Developing and assembling such a project can be challenging from both perspectives, hardware and software [1]. This type of vessel has some advantages when dealing with specific applications, such as its low cost and self-sustainability. As its main propulsion force comes from the wind, a sailing robot performs long missions consuming less energy than other USVs [2, 3]. Works like the ones of Ang et al. [4] and Negreiros et al. [3] advocate sailing robots as a reliable instrument for different tasks in environmental monitoring.

Mendonça et al. [5], Plumet et al. [6], and Jing et al. [7] discuss important issues that arise when developing and testing autonomous systems. One of the main concerns is that any

Extended author information available on the last page of the article

unintended behavior can lead to severe damage to the vessel, the environment, and possibly cause injuries. Another significant challenge is the time-consuming nature of system development, particularly in the case of reinforcement learning (RL) applications where the agent interacts with the environment to create a policy to support its guidance, navigation, and control (GNC) decision, as stated by Lapan et al. [8].

To alleviate these issues, virtual environments and digital twins have been used as safe environments to develop and test autonomous vessels. Virtual environments not only offer a safe method for testing autonomous systems, but they also allow for the simultaneous evaluation of multiple solutions, quick and easy re-running of experiments, control over environmental conditions, and the simulation of rare real-world events [7]. However, creating a simulator for sailing robots always involves simulating certain aspects of physics, which are not typically included in the physics engines used by simulation tools. Generally, the physics engines used in robotic and vehicle simulations are built to calculate the dynamics of rigid bodies under the influence of gravitational force. However, in order to simulate a sailing boat accurately, it's necessary to account for hydrodynamic, hydrostatic, and aerodynamic forces.

To address the aforementioned issues, we introduce Yara, a new and effective ocean virtual environment (OVE) for the development, training (using RL), and assessment of autonomous GNC systems for sailing robots. Our OVE is open source and has been created within the Gazebo simulator [9], providing a high-quality 3D scene and accurate physics to simulate the maneuverability of sailing boats and their response to various wind conditions. In order to accurately simulate the responses of the sail, rudder, and keel to various environmental conditions, we have integrated the E-Boat, an actual sailing robot digital twin, into Yara. The dynamics of the foils have been added to Gazebo's physics engine to calculate the lift and drag forces on the sail, rudder, and keel. Yara simulates various sensors, including cameras, wind sensors, and GPS, as well as the realistic behavior of actuators that control the sail, rudder, and electric propeller. Sensor data and actuator commands are communicated through Robot Operating System (ROS) topics, which makes it simple for ROS developers to integrate and test new guidance, navigation, and control (GNC) solutions. We integrate our platform with Stable-baselines3 [10] using the Gymnasium [11] API to train reinforcement learning agents for GNC tasks.

Thus, the main contribution of our work is the USV simulator itself, which has in addition to traditional simulators the following novel and/or improved characteristics:

- wind variability extrapolated from meteorological data from a real large aquatic ecosystem in Brazil (the Guanabara Bay);
- wave pattern coupled to the wind direction and speed;
- a sailing robot digital twin modeled in a way its simulated dynamic behavior in response to different environmental conditions (such as wind, waves, and water currents) and actuator commands reflects the actual behavior of a real sailing robot.
- Python libraries that enable ready-to-go RL training using Gymnasium and Stable-baselines 3 APIs.
- a GitHub repository containing all Yara codes, physics models, scenarios, obstacles and the sail robot digital twin (https://github.com/medialab-fboat/Yara_OVE.git)

This document is organized as described in the following. Section 2 reviews state-of-the-art autonomous sailing systems and virtual environments for marine robotics. Section 3 presents the theory and functioning of some simulators used in autonomous vessel research. In Section 4, we propose our virtual sailing environment, introducing it with all its details. In Section 5, we introduce and discuss the responses of our virtual sailing robot in some maneuvers tests and depict the performance of three solutions to control the sailing robot through a seven-waypoints mission. Finally, in Section 6, we draw our conclusions and future research lines on this project.

2 Related Works

This section brings a careful review of the state-of-the-art, both on autonomous sailing systems (Section 2.1) and on virtual environments for marine robotics (Section 2.2).

2.1 Autonomous Sailing

It is not the intention of this section to do a deep review and discussion of autonomous GNC for sailing boats. However, this topic is relevant in the context of our proposed simulation tool.

Creating an autonomous system for guidance, navigation, and control (GNC) poses a significant challenge in building an unmanned sailboat, as discussed by Plumet et al. [12], Plumet et al. [6], Shen et al. [13], Silva Júnior et al. [14], and Shen et al. [15], among others. Steering and controlling a sailing robot involve much more intricate maneuvers than a motor boat. The aerodynamic forces acting on the sail are the main propeller of a sailing robot, and the poor management of these forces could reduce the robot's performance and pose

risks such as sinking or causing damage to other vessels or people.

In general, the primary approach to mobile robot navigation uses path planning algorithms, and for autonomous sailing robots, it is no different. In 2012, Pêtrés et al. [16] proposed a path-planning solution for unmanned sailing boats based on the artificial potential field algorithm. The authors split the solution into a global and a local potential field to consider the small-scale wind variation in a small region around the boat. Until today, the work of Pêtrés et al. is a reference for the field, being cited several times.

Silva-Junior et al. [14] propose a solution using Q-learning. Although the authors presented good results, their evaluation scenarios are overly simplistic and do not reflect realistic wind behavior or the maneuverability of a sailing boat. Still, the overall results point out that Q-Learning is a promising technique for sailboat path planning.

Lin Zhou et al. [17] present an advanced optimization method, the Beetle Swarm Optimization (IBSO), for autonomous sailboat trajectory planning. The proposed method uses mathematical simulation modeling to validate the IBSO method against Particle Swarm Optimization (PSO) and BSO algorithms regarding convergence speed and accuracy. The focus is on trajectory optimization to avoid obstacles with improved efficiency and precision. The optimized path is defined based on the wind, obstacle distribution, and the goal. The authors presented results using a four-degree-of-freedom (4-DOF) simulator and an actual robot sailing in a controlled environment.

Liu et al. [18] proposes a path-finding algorithm using Q-Learning based on a Gaussian process (GP-QL). The authors sought to improve the adaptability and reliability by modeling the Q-Learning value functions as a Gaussian process. The authors utilized a 3-DOF simulation to conduct their tests. The results show that the method produces a consistent path but exhibits a slightly irregular trajectory.

Shen et al. [15] propose a path-planning solution based on the fusion of Enhanced Adaptive Ant Colony Optimization (ACO) and the Rolling Window Method (RWM) for autonomous sailboats in complex maritime environments. The authors claimed that by combining the global search of ACO with the RWM, the resultant algorithm can provide a dynamic path computation in the presence of moving obstacles. They present results to validate their proposal generated by simulations. Although the results seem to support their claims, the simulation used is simplistic and may not represent a sailing boat's complex motion.

The primary approach to mobile robot navigation typically involves path-planning algorithms. While these solutions have seen significant development, there is still room for improvement. For instance, the QAPF algorithm, proposed by Orozco-Rosas et al. [19], aims to enhance Q-Learning by using artificial potential fields. This approach could

potentially be adapted to sailing boats by incorporating the concepts introduced by Pêtrés et al. [16]. Therefore, even though path planning is an important part of an autonomous GNC system, it can effectively respond to small-scale variations in the wave and wind patterns that affect a sailing boat. The complex and strategic operation of the rudder and the sail to enhance performance, avoid collisions, or prevent the boat from capsizing is managed by a control system. This control system can receive sensor data representing the environmental conditions at a given time. Tipsuwan et al. [20] conducts a comprehensive survey on autonomous sailboat control strategies, focusing on dynamic modeling, navigation systems, and control techniques for optimizing course and speed. The work leans more toward theoretical review and categorization of different control approaches used in autonomous sailboats without specifying or presenting a particular solution. Several authors address the sailboat control challenges with different strategies.

Méxas et al. [21] compare reinforcement learning and imitation learning algorithms in autonomous sailboats using the Unity platform and ML-Agents toolkit. The paper explores the performance of Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) under different wind conditions and obstacles, also examining the effect of imitation learning with Behavioral Cloning and Generative Adversarial Imitation Learning. Both algorithms demonstrate good capabilities in controlling the boat under steady wind conditions. The policy generated by PPO presented the most promising results.

Suda and Nikovski [22] describe the application of Deep RL algorithms to determine the optimal decision policy for upwind sailing; wind direction and speed vary according to an unknown stochastic process. The learned policy outperformed baseline control algorithms in simulations considering uncertain and dynamic environments. However, they do not consider the effects of waves, currents, and other environmental factors that can affect a sailboat's performance. They assume that the stochastic process that generates the wind conditions is unknown to the learning algorithm. Still, sailors may have some knowledge or intuition about the wind patterns in a particular area. Additionally, they do not compare the performance of the Deep RL algorithms against other machine learning or control methods used for sailing control.

Liu et al. [23] propose a collaborative control method for the sail and rudder of an unmanned sailboat based on model predictive control. Their model considers the yaw angle to the desired trajectory and the restriction of roll angle as control targets, establishing a four-degree-of-freedom kinematics and dynamics model of the unmanned sailboat. They verified via simulation that the collaborative control has better effects on the yaw angle control and roll angle limitation and can obtain a more accurate path-tracking effect than the traditional separation control method of sail and rudder under

the same wind field conditions. Their work does not consider ocean currents' effect on the sailboat's motion.

The works presented in this section propose various approaches to contribute to an autonomous GNC system for sailboats. However, comparing the authors' results is sometimes difficult, as each author used a different kind of simulator. On top of that, the simulators are usually too simplistic and cannot guarantee a reliable representation of a sailboat's motion and maneuverability.

2.2 Virtual Environments

In the literature, we could not find a reliable simulation environment for sailing robots that provided a good graphical rendering of the surrounding environment with a high degree of visual realism and all the necessary physics. Several previous works rely on simplified physics models to simulate an environment to support the development and testing of autonomous sailing systems [6, 13, 15, 23, 24]. We noticed that these works had proposed neither a realistic graphical rendering of the surrounding environment nor an accurate simulation of collisions with the inclusion of essential elements such as ocean waves and wind. Some works [5, 25] seek to build a full 3D simulation combining graphical representation and physics. The Kelpie environment proposed by Mendonça et al. [5] seems currently unavailable. The environment proposed by Paravisi et al. [25] combines many wanted elements. Still, its graphical representation is outdated, and the physical model for the boat dynamics and the foil dynamics is not well-suited for a sailing boat.

In the broader context of marine robotics, autonomous boats are rapidly gaining prominence, offering cutting-edge solutions for various applications, from environmental monitoring to maritime logistics. The sophisticated nature of these vehicles requires extensive testing and validation of their control systems, navigation algorithms, and ability to adapt to dynamic marine environments. In this context, virtual simulation environments become fundamental, providing a safe and cost-effective platform for modeling complex scenarios and testing autonomous systems under diverse conditions. Many simulation tools, such as UWSim, VRX, USVSim, and UUV Simulator, propose solutions for autonomous surface vessel design and simulation, playing a significant role in this domain. Each offers distinct features and specialized functionalities, catering specifically to the nuanced requirements of marine robotics and the intricate development process of autonomous boats. Our proposed virtual environment architecture and solution contribute to this evolving landscape, offering an enhanced and comprehensive platform for the development and testing of these sophisticated marine vehicles.

The Underwater Simulator (UWSim) [26] is a solution specifically designed for underwater robotics. In this work,

the authors propose a 3D simulator that provides a realistic environment for testing and developing algorithms for underwater vehicles. This includes simulating the dynamics and physics of underwater environments, sensor feedback like sonar and camera, and the interaction of robotic vehicles with water currents and obstacles. For autonomous boats, UWSim can be particularly useful for developing sub-surface components and algorithms, especially if the vessel has functionalities that involve underwater exploration or interaction.

The UUV Simulator [27] is a Gazebo-based simulator focused on Unmanned Underwater Vehicles. The proposal includes a package implementing ROS capabilities for UUV applications. While it's more focused on underwater vehicles, the physics and environmental models may still be relevant for autonomous boats, especially for understanding and simulating the underwater portion of the vehicle or capabilities that cross the boundary between surface and subsurface operations.

Bingham et al. [28] introduce the Virtual RobotX (VRX) simulation, a tool for developing and testing unmanned surface vessels in oceanic environments. The simulated environment focuses on elements used in the RobotX competition tasks and offers sufficient fidelity to enable developers to prototype new solutions. The main boat model available is a differential boat designed to be a digital twin of the real RobotX autonomous vessel. The simulator design adheres to the premise that autonomous behavior that does not work in the simulated world will not work in the physical world, and autonomy that does work in the physical world will work in the simulated world. However, while the simulations provide a high level of fidelity, they may not perfectly replicate real-world conditions, and developers should still test their solutions in physical environments before deployment.

USVSim [25] implements a river-simulated environment with similar characteristics to VRX's. The simulation includes models for boat dynamics, surface water conditions, and interactions between the boat and the water surface. The 6-DOF dynamic model was implemented through the FreeFloat plugin [29]. The authors modify the original FreeFloat to include wave motion from the UWSim, so the boat responds to changes in the water surface. They also modeled the boat's response to environmental disturbances like winds and water currents. The USVSim offers four boat types: rudder boat, airboat, differential boat, and sailboat. All the boats are built from the same hull, and the authors used the Gazebo lift/drag plugin to implement the physics for the rudder, the keel, and the sail. In this way, the sail was modeled as a fixed wing, which does not reflect most of the sailing robots in the literature [23, 30–32].

These four virtual environments implement the vessel dynamic simulations using a 6-DOF model. The UUV and the UWSim focus on underwater vessels, while USVSim and VRX are dedicated to surface vessels. They all enable simu-

lated sensors and interface with ROS. However, the only one that offers a sailing boat model and parameters is USVSim.

Surface vessel simulations are not new, but an open-source, reliable sailing vessel simulation is hard to find. Despite its many qualities, the sailing vessel simulation offered within USVSim has poorly modeled aerodynamic forces, boat dynamics, and sail behavior. The only contribution of the aerodynamic force seems to be the forward component, and the sail is modeled as an airplane wing. We do not observe the torque or lateral drift caused by the lateral component of the aerodynamic force. In an actual sailing boat, the effects caused by the lateral component are resisted by the keel, but the boat still suffers some of its influence. Building on that, our work introduces an innovative virtual environment architecture and solution designed as an open platform for reinforcement learning tasks for USVs, focusing on sailing robots. We build a sailing robot digital twin, implementing reliable physics to simulate the boat dynamics and response to aerodynamic, hydrodynamic, and hydrostatic forces. Our solution provides full access to most physics simulation variables, computer vision tasks, and a simple and direct way to use RL APIs like Gymnasium [11] and Satble-baelines3 [10].

3 Physics Simulation for Sailing Boats

Most simulation engines have solutions for fast simulation of rigid body physics and force field computation. To simulate a surface vessel, one must add hydrodynamic, hydrostatic, and aerodynamic forces to the rigid body physics model. This Section discusses the required physics to simulate a sailing robot in different wind and sea surface conditions. A reliable physics simulation is required to improve the chances a GNC solution developed in a virtual world could work in the real world [28].

A sailboat sails under the influence of various hydrodynamic and aerodynamic forces. Its main propulsion is produced by the force of the wind on the sail. This force is generated by the sail trim (sail camber), which shapes the sail like an airfoil. The curvature of the sail causes a change in the airflow direction, which causes a reaction force called lift. A similar force also appears on the rudder and the keel when the boat moves through the water. Thus, to implement the digital twin (in the simulator), it is necessary to compute different forces acting on different parts of the boat.

Theoretical aspects of the physics simulation include aerodynamic and hydrostatic forces, current, and wind, among other physics constraints that act on the whole system. These aspects are further described in the next subsections so the reader understands the key aspects of modeling the physics of a simulated sailing robot.

3.1 Boat Dynamic Model

The motion of a boat traveling through the water can be described with the help of a six-degrees-of-freedom (6-DOF) vectorial model for marine crafts [33]. The motion equation has the following form:

$$\underbrace{\mathbf{M}_A \dot{\mathbf{v}}_r + \mathbf{C}_A(\mathbf{v}_r) \mathbf{v}_r + \mathbf{D}(\mathbf{v}_r) \mathbf{v}_r}_{\text{hydrodynamic forces}} + \underbrace{\mathbf{g}(\boldsymbol{\eta})}_{\text{hydrostatic forces}} + \underbrace{\mathbf{M}_{RB} \dot{\mathbf{v}} + \mathbf{C}_{RB}(\mathbf{v}) \mathbf{v}}_{\text{rigid body forces}} = \tau_{propeller} + \tau_{wind} + \tau_{waves} \quad (1)$$

where

$$\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^T \quad (2)$$

$$\mathbf{v} = [u, v, w, p, q, r]^T \quad (3)$$

The position ($\boldsymbol{\eta}$) and velocity (\mathbf{v}) vectors include the USV 6 degrees-of-freedom: surge u , sway v , heave w , roll p , pitch q , and yaw r (Fig. 1). While the velocities \mathbf{v} are described in the robot's frame b , the robot position $\boldsymbol{\eta}$ is related to a reference frame that could be Earth-centred, Earth-fixed, fixed on the water surface, fixed on a base station, etc. Table 1 depicts the notation for position and velocities [34]

The hydrodynamic forces in Eq. 1 describe forces generated by the water flowing around the hull as the boat moves. These forces include an inertial and a drag term. When the boat moves, the contact between the hull and the water causes resistance due to radiation-induced potential damping, skin friction, lifting forces, vortex shedding, and wave drifting [33]. The effects of the resistance forces are compiled in the hydrodynamic damping matrix $\mathbf{D}(\mathbf{v}_r)$.

Friction between the hull and the water causes water shear, i.e., part of the water near the contact surface will gain velocity. In other words, as the boat moves, it “pulls” a certain mass of water. The influence of this water movement on the boat

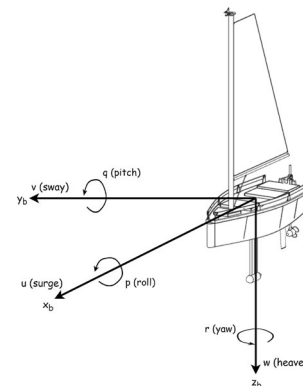


Fig. 1 The 6-DOF velocities u , v , w , p , q and r in the body-fixed reference frame $b = (x_b, y_b, z_b)$

Table 1 The SNAME's notation for marine vessels [34]

DOF		Linear and angular velocities	Positions and Euler angles
1	motion in the x direction (<i>surge</i>)	u	x
2	motion in the y direction (<i>sway</i>)	v	y
3	motion in the z direction (<i>heave</i>)	w	z
4	rotation about the x axis (<i>roll</i> , heel)	p	ϕ
5	rotation about the y axis (<i>pitch</i> , trim)	q	θ
6	rotation about the z axis (<i>yaw</i>)	r	ψ

Adapted from Fossen's book [33]

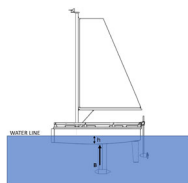
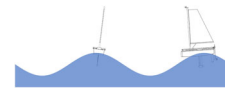
motion is represented as an additional mass. In Eq. 1, the additional mass and its inertial effect are described by the added mass matrix \mathbf{M}_A and the Coriolis-centripetal matrix $\mathbf{C}_A(\mathbf{v}_r)$ [28, 33].

The hydrostatic force acting on the hull can be simplified as the buoyancy force caused by the difference in pressure between the water line and the deepest part of the hull. Archimedes' principle says that the buoyancy force acting on a body equals the weight of water displaced by the body when totally or partially submerged (Fig. 2).

In a flat water surface, the water line says how much of our boat should be submerged so the buoyancy force matches the boat's weight making it float in equilibrium. However, in the presence of waves, the water surface is moving, and consequently, the water pressure distribution on the hull changes over time (Fig. 3).

3.2 Aerodynamic Forces

Under the presence of wind, the main sail of a sailing boat acts as an airfoil, changing the airflow direction and generating two aerodynamic forces: the lift (L) and the drag (D). These two forces depend on the respective lift and drag coefficients (C_L and C_D), the velocity of the air flowing by the sail (V_A), the sail area (A), the angle between the sail and the airflow (α - attack angle), and the air density (ρ_{air}). Equations 4 and 5

**Fig. 2** Buoyancy on a flat water surface**Fig. 3** Wave effect on the boat equilibrium

depict the modulus of the resultant lift and drag forces acting on the sail. The lift and drag coefficients are intrinsic to the sail shape and material and are functions of the attack angle (α). Figure 4 shows the aerodynamic forces as a function of the attack angle.

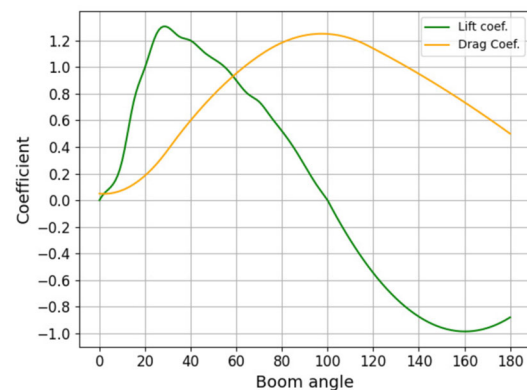
$$L = \frac{1}{2} \rho_{air} V_A^2 A C_L(\alpha) \quad (4)$$

$$D = \frac{1}{2} \rho_{air} V_A^2 A C_D(\alpha) \quad (5)$$

As a sailing boat moves, an airflow is generated through the sail that depends on the boat velocity (V_B) and the wind speed in the resting referential (V_T). The speed of the resultant airflow is called apparent wind speed (V_A). Figure 5 depicts a simple vector representation of the true and apparent wind speeds and the lift and drag forces. The resultant aerodynamic forces on the sail (R) can be decomposed into a forward (F_{fwr}) component that propels the boat forward (surge) and a lateral component (F_{LAT}) that pushes the boat sideways (sway).

3.3 Hydrodynamic Forces on Under Water Elements

To complete the whole dynamic model of a sailing boat, a key aspect is the hydrodynamic forces on the keel and the rudder. The keel's main objective is to balance the effects caused by the resultant lateral force acting on the main sail. The boat suffers from two main effects: a sideways movement and a rotation (Fig. 6). By design, the keel has a weight in its bottom part and a symmetric foil shape. While the extra

**Fig. 4** Lift and Drag forces as a function of the attack angle

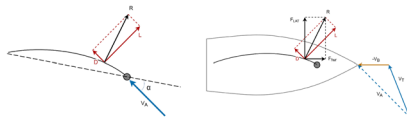


Fig. 5 Wind forces acting on the main sail (L and D) and being transmitted to the boat (F_{fwr} propelling the boat forward, and F_{LAT} pushing the boat sideways)

weight helps to lower the center of gravity of the sailing robot, it also opposes the rotational motion caused by the lateral component of the aerodynamic forces. On the other hand, its foil shape produces lift and drag forces as the boat moves through the water and resists the sideways motion caused by the aerodynamic forces on the sail. In Fig. 6, the water hydrodynamic resistance force R_{LAT} contains contributions from the resistance of the water to the lateral movement of the hull, the keel, and the rudder.

The rudder is designed as a symmetric foil to produce lift and drag forces as the boat moves. The resultant of these induced forces generates a torque in the hull that modifies its direction. As we turn the rudder, we modify the angle of attack, i.e., the angle between the fluid velocity and the rudder chord line. By changing the attack angle, we modify the resultant force, increasing or decreasing the torque transmitted to the boat hull. Figure 7 presents a simple illustration of the force decomposition in the rudder. The inflow or fluid (water) velocity in Fig. 7 is mainly caused by the boat's motion; in the boat's referential, the water moves with a velocity equal to the boat's velocity but in the opposite way. However, natural water velocity can occur when the boat travels under waves and/or currents. Therefore, as for the wind, the inflow represents an apparent water motion speed composed of the natural water movement and the movement due to the boat motion.

4 Proposed Sailing Virtual Environment

Our proposed simulated environment extends the functionalities of the Gazebo Sim [9] to support sailing robots. A key

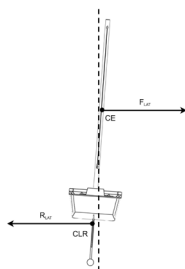


Fig. 6 Aerodynamic lateral force component F_{LAT} and the hydrodynamic resistance force R_{LAT}

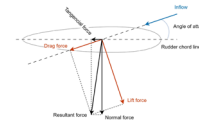


Fig. 7 Force decomposition in the rudder

to achieving this goal is representing the influence of both water and air on the robot's behavior. We implement new physics models to describe the forces acting on the sail, the rudder, and the keel. We adapt the wave and dynamic models implemented by the VRX project [28] to reflect the unique properties of a sailing robot. In this way, the movement of our sailing robot digital twin is subjected to the ocean waves and wind conditions, as well as to the aerodynamic forces acting on the sail. Each moving part in the digital twin is treated as a different component with its own center of gravity and mass distribution. In this way, the boat's balance is affected by the waves and the forces acting on its different



Item	Description
Length	2500 mm
Width	830 mm
Draught	1322 mm
Mast length	3130 mm
Hull weight	65 kg
Keel weight	37 Kg
Batteries weight	25.50 Kg + 25.50 Kg
Number compartments	4
Number waterproof wallets	7
Sail area	3188320.84 mm ²
Sail type	Sailcloth - Dacron
Stored power	105 A + 105 A (12V)
Solar panel power	100 Wp
Research group	UFF/RJ and UFRN/RN

Fig. 8 The F-Boat (top left), the E-Boat (top right), and specification table (bottom)

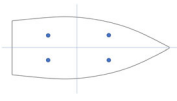


Fig. 9 Points where the buoyancy force is applied

components. Through this Section, we describe our proposed virtual environment and all its features.

4.1 Sailing Boat Model

Our simulated sailing robot model, named E-Boat, was built to be the digital twin of our real sailing robot named F-Boat [3]. As such, E-Boat shares the same physical characteristics, actuators, and F-Boat's sensors (see Fig. 8).

In the simulation, the E-Boat's main source of propulsion is the wind, but it also relies on an electric propeller when needed. To describe its motion through the water, we must consider the action of rigid body static and dynamic forces as much as contributions from hydrodynamic, hydrostatic, and aerodynamic forces (see Section 3). We approximate the simulation of this motion using an adaptation of the Fossen six-degrees-of-freedom (6-DOF) vectorial model for marine crafts (Eq. 1) implemented and maintained by the VRX project team [28].

For our application, the wave motion of the water surface influences the motion control and the perception of the sensors, like images captured by cameras. For this reason, we kept a simplified implementation where the boat's response to the wave motion is restricted to heave, roll, and pitch velocities. The algorithm provided by Bingham et al. [28] emulates the unbalance caused by a wave by splitting the hull in a grid with size $2 \times N$, where the user can select N . The buoyancy force is computed for each boat segment, enabling responses as illustrated in Fig. 3. For the E-Boat, we use $N = 2$ (Fig. 9).

Although the 6-DOF model from VRX is suitable for describing the hull behavior, it is missing four vital hydrodynamic forces for a sailing boat: the lift and the drag generated by the water moving around the keel and the rudder (see Section 3.3). Thus, we modify the original VRX model by adding these four extra forces.

As the main source of propulsion is the wind, we also added the aerodynamic force acting on the sail. In the original VRX model, the wind force (τ_{wind}) interacts only with

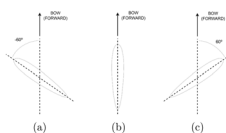


Fig. 10 Rudder maximum positions (a) and (c), and resting position (b)

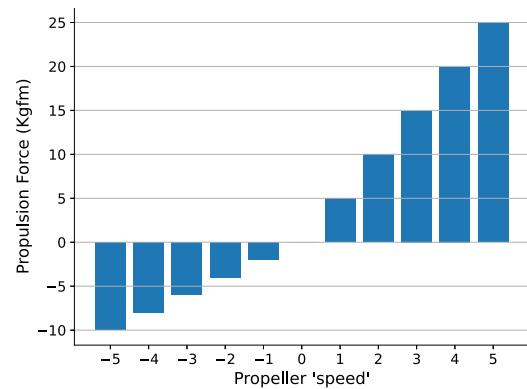


Fig. 11 Propulsion force intensity as a function of the propeller speed

the hull. In our model, the wind is commissioned as a main propulsion force, and its contribution is computed as described in Section 3.2.

4.1.1 Actuators and Control Interfaces

The E-Boat navigation is performed by controlling three actuators: the boom to which the sail is attached, the rudder, and the electric propeller. Within the Yara OVE the E-Boat's actuators were defined to emulate the behavior of those of the F-Boat.

The rudder can take any position from -60 to +60 degrees (Fig. 10), and a proportional controller is used to set the angle as required. The controller defines the angular velocity of the rudder, and the Gazebo physics engine applies the necessary force.

The propeller control interface implements a discrete function with five forward and five backward speeds. For each of these ten speeds, a force is applied to the center of the propeller joint. The intensity of the applied force is set according to the propeller manufacturer manual [35], and it is depicted in Fig. 11. Figure 12 illustrates the propulsion force generated by the propeller. As the force is applied on the joint connecting the propeller and the electric engine body, it is as if, when activated, the propeller is pushing the engine body.

While the rudder and the propeller are operated by engines acting on their respective joints, the boom operation differs. In the F-Boat (the real sailing robot), the boom is connected to a cable that can be released or pulled using an electric



Fig. 12 The red arrow indicates the force the propeller generates

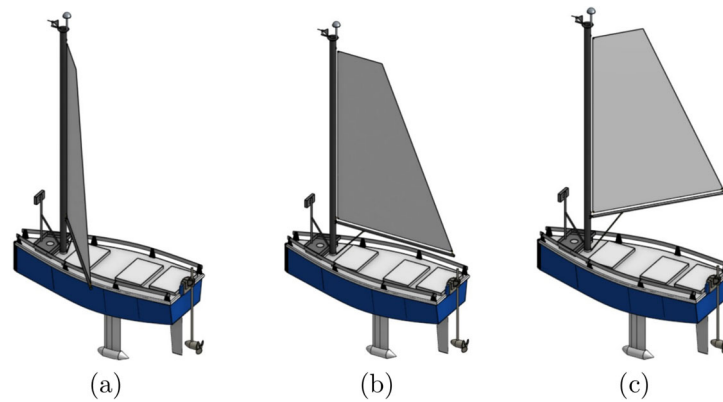


Fig. 13 Sail opening to left (a), right (c), and closed in its resting position (b)

motor. The motor does not act on the joint that connects the boom to the mast. Under the influence of the wind, the boom moves to the left or the right, and the length of the boom cable controls the amplitude of this movement. Figure 13 illustrates different boom positions. In our simulated environment, the emulation of the actual boom controller is implemented as described in Algorithm 1. At line 1, we set the boom angular velocity, defined by the cable releasing rate. As the algorithm is executed in every simulation loop (the loop in which the Gazebo physics engine computes the next state of the models, see Fig. 18), the value $sov = 0.29$ means that the boom cable releasing/pulling rate is twenty-nine degrees per second. Using the upper and lower limits of the boom joint (lines 6, 7, 9, and 10), we represent the restriction imposed by the cable to the boom motion inside Gazebo. The $required_boom_angle$ must be a positive value between 0 and 90 degrees. Each time a new $required_boom_angle$ is set, the algorithm adjusts the boom joint limits. In this way, the boom is free to move with the wind from right to left, only limited by the absolute value of the $required_boom_angle$ (Fig. 13). Algorithm 1 is implemented within the Boom Controller Plugin, and it is executed in every physics engine time step, as we will detail in Section 4.2.

4.1.2 Sensors

The virtual sensor array within the E-Boat measures apparent wind speed and its direction, the linear velocity in the bow

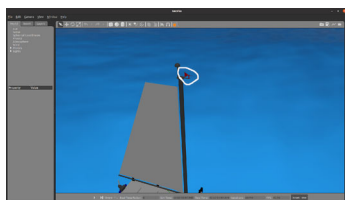


Fig. 14 Picture of the wind sensor on our Virtual World

Algorithm 1 Emulate boom operation. See text for explanations.

```

1:  $sov \leftarrow 0.29$ 
2: if  $required\_boom\_angle > 90.0$  then
3:    $required\_boom\_angle \leftarrow 90.0$ 
4: end if
5: if  $required\_boom\_angle > joint\_upper\_limit + sov$  then
6:    $joint\_upper\_limit \leftarrow joint\_upper\_limit + sov$ 
7:    $joint\_lower\_limit \leftarrow joint\_lower\_limit - sov$ 
8: else if  $required\_boom\_angle < joint\_upper\_limit - sov$  then
9:    $joint\_upper\_limit \leftarrow joint\_upper\_limit - sov$ 
10:   $joint\_lower\_limit \leftarrow joint\_lower\_limit + sov$ 
11: end if
12: if  $joint\_upper\_limit > 90.0$  then
13:   $joint\_upper\_limit \leftarrow 90.0$ 
14:   $joint\_lower\_limit \leftarrow -90.0$ 
15: end if

```

direction (surge, see Fig. 1), and roll angle. The wind sensors are positioned at the top of the mast (Fig. 14), while the other sensors are assembled below the deck. A stereo camera is assembled at the bow to complete our sensor array (Figs. 15 and 16).

4.2 System Architecture

The proposed virtual sailing environment is implemented as a ROS package. We use ROS launch files to launch the virtual world in Gazebo as ROS nodes and populate them with models. Currently, our virtual world can be populated

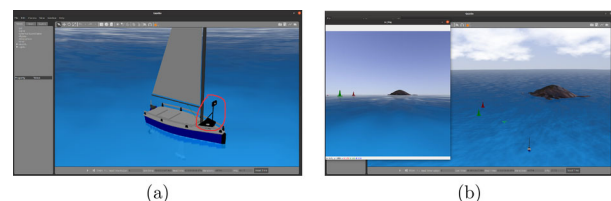
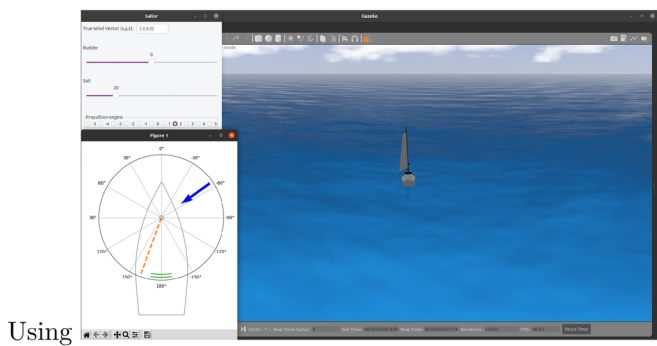


Fig. 15 Bow camera: (a) location of the camera. (b) Example of a scene provided by the camera

Fig. 16 The figure shows some human interfaces of Yara. The GUI (Graphical User Interface) to enable the E-Boat manual control is on the top right. Below the controller GUI, we find a HUD displaying wind sensor data (blue arrow) and boom and propeller status. The orange dashed line represents the boom position, and the green solid lines represent the propeller speed



by four model types: the E-Boat, signal buoys, the ocean, and an island. The E-Boat and signal buoys are 3D models created with the help of Onshape (<https://www.onshape.com/en/>). The E-Boat characteristics, actuators, sensors, and physical properties are set using URDF and SDF description files. The ocean model implements the sea surface simulation using OpenGL shaders and an implementation of the Gerstner wave model [28, 36–39].

Figure 17 depicts the overall structure of our virtual sailing environment. We wrote or adapted plugins to implement some physics, boat controls, sensor arrays, and environmental control. These plugins are executed within the simulation loop in which Gazebo physics engine computes the new state of the models populating the virtual world. Figure 18 illustrates the simulation looping in a simplified way. We detail these plugins in the following sections.

4.2.1 Physics Plugins

Some of the physics used in the simulation are not defined within the Gazebo's physics engine. This additional physics is implemented as plugins written in C++ and compiled within the ROS package. They are: (i) the hydrodynamic plugin, (ii) the wave plugin which implements the ocean wave model responsible for the sea surface movement [28, 37–

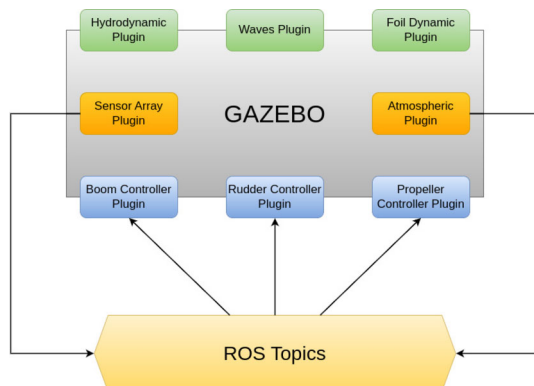


Fig. 17 Overall system architecture. See Sec. 4.2 introduction for a description of the components

39], (iii) and the foil dynamic which implements the lift and drag forces acting on the sail, the rudder, and the keel. In Yara OVE, we combine the hydrodynamic and wave plugins with our original foil dynamic plugin to extend the Gazebo physics engine to create suitable physics to simulate sailing vessels.

There are two physics plugins to add hydrodynamic, hydrostatic, and aerodynamic forces to the Gazebo simulation and one more for ocean waves. The hydrodynamic plugin was adapted from the Virtual RobotX Challenge (<https://github.com/osrf/vrx>), adding the 6-DOF maneuvering model to the E-Boat simulation. The hydrodynamic force is computed based on the physical characteristics of the E-Boat, like added mass and drag coefficients. On the other hand, the hydrostatic force computation considers the sea surface state as defined by the wave plugin.

The foil dynamic plugin computes the lift and drag forces acting on the sail, rudder, and keel. These forces are computed at each physics engine time step following Eqs. 4 and 5. For the sail, the fluid velocity, represented by V_A is the addition of the negative E-Boat velocity and the environment wind velocity (see Fig. 5). For the keel and the rudder,

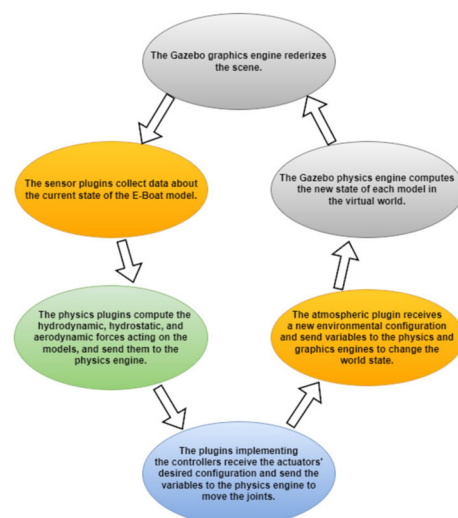


Fig. 18 Simplistic visualization of simulation looping

the fluid velocity is the addition of the negative velocity of the E-Boat and the natural fluid velocity caused by natural water currents. Figure 19 shows the vector operation used to compute the apparent fluid velocity (V_A). In the figure, V_A is the fluid apparent velocity (called Inflow in Fig. 7), V_T is the natural fluid velocity (wind true speed/water current or wave motion), u is the surge velocity, v is the sway velocity, and V_B is the boat velocity. Therefore, the vector operation can be written in the following way: $\vec{V}_A = \vec{V}_T - \vec{V}_B$.

The wave plugin was also adapted from the Virtual RobotX Challenge. This plugin generates a wave field using a summation of Gerstner waves from a horizontal reference location $P_0 = (x_0, y_0)$ with no vertical displacement $z = 0$. The wave field is described by a horizontal (P) and vertical (z) displacement related to the reference location P_0 as:

$$P = P_0 - \sum_{i=1}^N q_i A_i \sin(k_i \cdot P_0 - \omega_i t + \phi_i) \quad (6)$$

$$z = \sum_{i=1}^N A_i \cos(k_i \cdot P_0 - \omega_i t + \phi_i) \quad (7)$$

where q_i is the steepness, A_i is the amplitude, k_i is the wave vector, ω_i is the angular frequency, and ϕ_i is the wave phase.

4.2.2 Control and Sensors Plugins

The control and sensor plugins are responsible for replicating the system that manages the E-Boat actuators and sensors. The control plugins are subscribers of three ROS topics, one for each actuator: boom, rudder, and propeller. At each time step of the physics engine, these plugins check for new commands and apply the required changes to the position of the actuators, as discussed in Section 4.1.1.

The Sensor Array plugin collects and transmits data from different sources within the E-Boat. It publishes all data in a ROS topic every four seconds, but the user can change the publishing frequency. The data vector is composed of distance to the current waypoint, trajectory angle, linear velocity, apparent wind speed, apparent wind angle, current

boom angle, current rudder angle, current propeller speed, and roll angle.

4.2.3 Atmospheric Control Plugin

Usually, environmental conditions such as wind speed and direction, wave height and frequency, and fog are set when the simulation is launched. The atmospheric plugin was designed to allow users to change some environmental parameters during the execution. At the moment, we can only manipulate the wind speed and direction. However, we plan for the users to be able to include fog and manipulate the light conditions at any time. The user will also be able to control the ocean wave height and frequency, for example, and whether the wind speed and direction will automatically define the wave parameters.

4.3 Reinforcement Learning APIs

In terms of implementation, the standard approach for training and evaluating RL agents is by using an environment known as Gym [40]. A Gym environment is essentially an API that lets the learning agent interact with its environment. The environment may be a piece of software or some physical device. In our case, the environment is a software simulation of the sea and wind conditions encountered by the E-Boat. This API defines a few primitive calls to initialize the environment, get the current state/observation from the environment, specify the action to perform, and obtain three pieces of information. This information concerns the new state/observation, the return, and a flag indicating whether the task has been completed. The Gym is migrating to a new API named Gymnasium [11]. Essentially, Gymnasium is an upgraded version of the Gym to cope with some weaknesses. As Gymnasium is the current supported API to support Gym environments, we chose it so the environment of the E-Boat could provide resources for RL training and evaluation.

Making a Gymnasium environment with Gazebo and ROS is not a direct approach. We must consider the ROS messages, topics, communication delays, environment reset, actuators reset, launch files, etc. Gym-Gazebo [41] is based on OpenAI Gym and provides several tools for conducting the experiments. As we chose to work with Gymnasium, using Gym-Gazebo is not viable. Therefore, we developed our own Gymnasium-Gazebo environment class.

Once we have our Gymnasium environment, we must connect an RL agent to be trained and evaluated. There are several libraries for training and evaluating RL agents like RL-Berry [42], RL_Coach [43], MushroomRL [44], CleanRL [45], Tensorforce [46], and KerasRL [47]. In the first phase of this work, we have chosen to use the Stable-Baselines3 library [10], which is largely used in the reinforcement learning community.

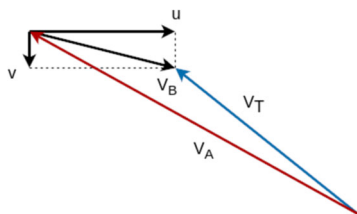


Fig. 19 Diagram depicting the vector operation to compute the apparent fluid velocity V_A

Table 2 Yara's Gymnasium environments

environment	electric propeller	object detection
Eboat62-v0	NO	NO
Eboat63-v0	YES	NO
Eboat93-v0	YES	YES

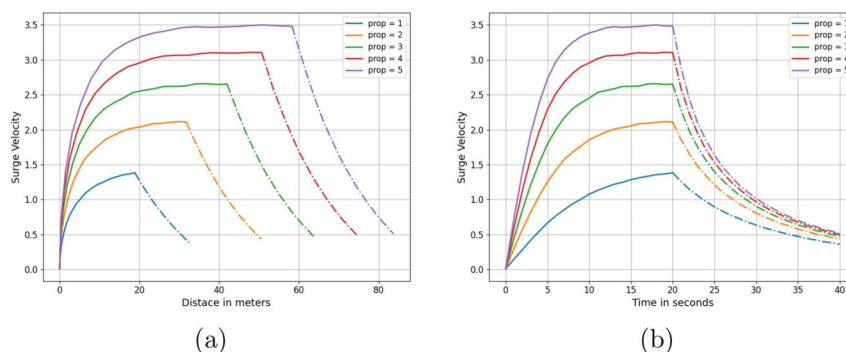
We provide, within Yara, three Gymnasium environments to train agents using deep RL algorithms. The solution is in Python, and the base class can be easily used to develop new training environments. Table 2 depicts the main characteristics of each one of Yara's Gymnasium environments. Eboat62-v0 enables training in a pure sailing boat, in which the electric propeller is not available. In Eboat63-v0, the propeller is available, but the designed return function penalizes its use. Eboat93-v0 implements object detection in a 120-degree arc ahead of the boat. These environments provide an easy way to train new decision policies by changing the return function, the neural network architecture, the wind direction/speed sampling, etc.

5 Experimental Results

The main objective of our environment is to provide a set of tools to enable the development and testing of RL agents in a simulated environment with reliable physics. In this section, we present some experiments that describe the EBoat's maneuverability (Section 5.1) and some autonomous navigation experiments using RL agents and a PID-based controller (Section 5.2).

5.1 E-Boat response in maneuvers

To validate the E-Boat maneuverability, we performed a qualitative evaluation of different behaviors as shown in Figs. 20, 21, 22, 23, and 24.

**Fig. 20** E-Boat acceleration for different propeller speeds

We can observe a clear trend by examining Fig. 20a, which illustrates the relationship between velocity and distance. As the intensity of propulsion increases from 1 to 5, the E-Boat experiences a gradual acceleration, covering larger distances within the same time intervals.

Figure 20b depicts the relationship between velocity and time, where we can discern a similar pattern. As the E-Boat's propulsion intensity increases, its velocity rises steadily. The velocity stabilization happens after 10 seconds at the highest propeller speed while taking 20 seconds for a slow-pace propeller. This observation further reinforces the understanding of the effects of varying propulsive forces on the boat speed.

From a qualitative perspective, the graphs confirm the intuitive understanding that breaking the initial inertia requires an initial burst of energy. Subsequently, maintaining momentum becomes a delicate balance between the energy consumed by hydrodynamic and aerodynamic forces, including those from the atmosphere (assuming wind speed is zero). This interplay between propulsive and resistive forces is a familiar concept to experienced sailors, who rely on such knowledge to optimize their performance on the water.

Figure 21 depicts the coordinates of a stationary boat that starts to propel itself using its propeller at different intensities. In Fig. (21a), the boat makes a curve to starboard with a rudder angle of 60° , while in Fig. (21b), it makes a curve to port with a rudder angle of 60° . It is possible to observe that with the highest propeller intensity (violet plot), the boat develops a complete circle with a radius close to its size, which is an expected behavior.

When analyzing the results, it becomes evident that the boat's turning radius decreases as propulsion velocity increases. This behavior aligns with the hydrodynamics of the rudder, where lower water velocities result in reduced maneuvering capabilities. These observations highlight the relationship between propulsion velocity and a sailboat's turning abilities, enhancing our understanding of boat maneuvering dynamics.

Figure 22 shows the coordinates of a stationary boat propelled solely by the electric propeller. The boat successfully

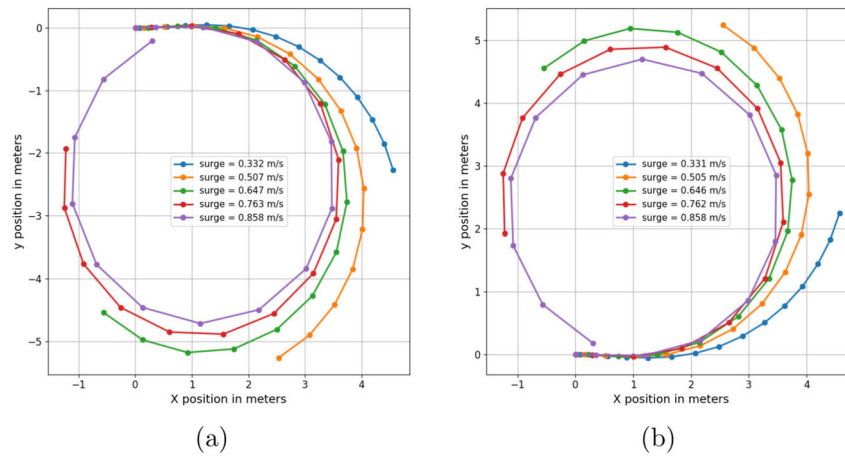


Fig. 21 E-Boat turning under different velocities. The rudder angle is fixed in $60^\circ/-60^\circ$, and we tried five runs with different propeller speeds. Each run lasts fifteen seconds. The position and surge velocity data are collected every second

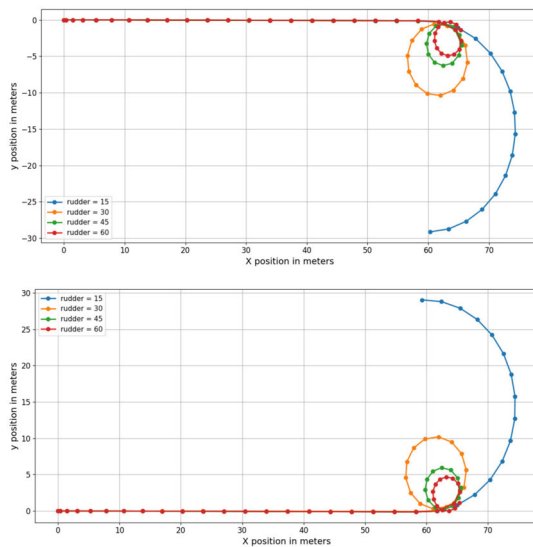


Fig. 22 E-Boat turning maneuver executed after the virtual boat reaches maximum velocity. The propeller is set to its maximum power, and the boat's velocity right before the turning is around 3.5 m/s

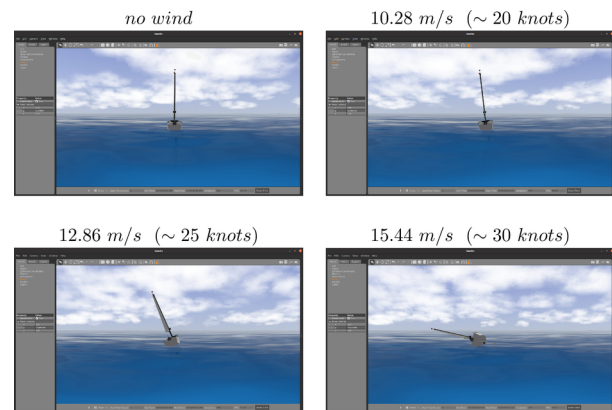
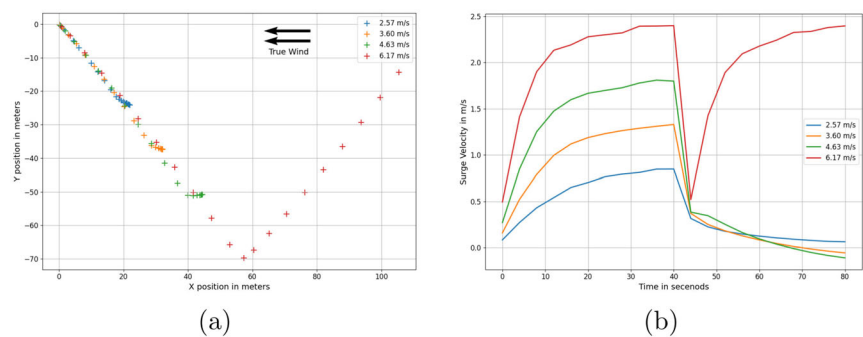


Fig. 24 The listening of the E-Boat under the influence of different wind speeds. The wind blows perpendicular to the sail plan. We can observe the Y component of the wind velocity vector in each picture in m/s

Fig. 23 Tack maneuver under different wind speeds. The legend depicts the true wind speed. Figure (a) shows the boat's position, while Figure (b) shows the boat's surge velocity. The position and velocity values were collected every four seconds



performs curves to both starboard (top) and port (bottom), demonstrating satisfactory maneuverability under the given conditions. These results provide valuable insights into the boat's maneuverability and the hydrodynamics of its rudder. They demonstrate the effectiveness of the boat's steering mechanism in executing curves.

Figure 23a depicts the coordinates of a stationary boat propelled solely by the wind with various tested velocities. As the simulation approaches approximately 40 seconds, the boat attempts to perform a maneuver known as a tack. Figure (23b) displays the boat velocity over time. Although the first graph implies it, the second graph shows that the boat can only execute the tack maneuver at 2.3 m/s or higher speeds. This may depend on different boat characteristics, but it is well known that it cannot perform any maneuver if the boat has a small velocity course. However, these results do not align well with the qualitative expectations. It was anticipated that, under real conditions, the boat would still be able to perform the tack maneuver at speeds as low as 1.5 m/s. Nevertheless, the obtained results are still closer to the expectations.

Figure 24 showcases the effects of wind intensity on the sail and keel of the sailboat. Four wind intensities are iterated: No wind, 20, 25, and 30 knots. As a result, we can observe how the wind impacts the sail and the sailboat's keel. These observations align with the expected behavior, considering that real sailboats often have a low keel and a significant ballast relative to the total weight of the vessel. This configuration results in a low center of gravity, minimizing the inclination caused by the demonstrated winds. In sailing, 30 knots is considered dangerous and may cause a situation similar to the one presented in Fig. 24, depending on the weight and keel.

5.2 Autonomous Navigation

Control tasks for a sailing robot present unique challenges not encountered with traditional motor-powered USVs, such as performing a tacking maneuver while sailing upwind or adjusting the jib when sailing downwind. Additionally, ensuring the boat's safety by adjusting the course and sail position to avoid any potentially dangerous situations is crucial (see Fig. 24).

In this section, we describe some experiments performed with three different control agents: a PID-based controller, a policy-based controller trained with Proximal Policy Optimization (PPO) [48], and a policy-based controller trained with Soft Actor-Critic (SAC) [49]. Our aim is not to present a performance comparison but rather to showcase the implementation of RL training and evaluation using Yara and to

demonstrate how ambient factors such as waves and wind can impact mission execution.

5.2.1 PID Controller

A PID control can be used for a simple control of the boat's movement. To reach a desired waypoint in the platform workspace, a simple strategy is to have two decoupled PID controllers, one for the heading and the other for the boat's forward speed (surge). The PID consists of calculating the error of the control variable, heading or speed in this case, that is, the error between a reference input x_d and the current value x of the variable, as shown in Eq. 8. A higher-level navigation system sets the reference input.

$$e_x = x_d - x \quad (8)$$

The next step is to apply the proportional, integral, and derivative gains to the error, as shown in Eq. 9, to find the appropriate commands for the actuators, in this case, the rudder angle and motor speed.

$$\delta_r = K_p e_x(t) + K_i \int e_x(t) dt + K_d \frac{de_x}{dt} \quad (9)$$

For sailing robots, the heading control is less robust to lateral drifts caused by both the water current and the wind, and in some situations, a course control, where the velocity vector is the control variable.

5.2.2 RL Agents

We trained two RL agents that are available within Yara. To begin with, we took a straightforward approach for the initial environment test. As a navigation path can be divided into multiple waypoints, we trained the agent to reach a single waypoint located 100 meters away. Once the agent masters the skill of reaching a waypoint, it can progress to the next location in the navigation path simply by changing the waypoint each time the sailing robot reaches one.

We chose PPO and SAC, which are well-known to fit such settings as the observation space and the continuous action space.

For this experiment, the observational space is composed of five parameters: the current distance from the waypoint; the angle between the E-Boat forward direction and the position of the waypoint; surge velocity (see Fig. 1); apparent wind speed; the apparent wind angle. The action space is composed by all three actuators within EBoat: rudder, boom, and propeller.

As the waypoint position and wind conditions could change from mission to mission, the agent should learn how to behave in different initial conditions. Using a uniform distribution, we randomly sample the wind speed, wind direction, and boat orientation for every environment reset. The wind speed is sampled for the interval $[2, 6]$, the wind direction from $(-180, 180]$, and the boat orientation from $(-180, 180]$.

The return function is described in Algorithm 2. All actions that decrease the distance from the goal are associated with a positive return, while those increasing the distance are associated with a negative return, hence penalized. It is a desired behavior that the agent uses the electric propeller only in situations where the wind force is not enough to push the boat forward or when a maneuver against the wind is needed. Therefore, returns earned by the agent are decreased by a value proportional to the current propeller speed (PS_t).

Algorithm 2 The function that computes the immediate return r at time t , r_t .

```

1:  $\Delta D \leftarrow D_{t-1} - D_t$   $\triangleright D_t$  is the distance from the goal in time  $t$ .

2:  $\triangleright D_{t-1}$  is the distance from the goal in time  $t - 1$ .

3:  $D_{max} = 1.25 * D_0$   $\triangleright D_0$  is the distance from the goal in the initial state,

4:  $t = 0$ .

5: if  $D_t < 5$  then

6:    $r_t \leftarrow 1$ 

7: else if  $D_t > D_{max}$  then

8:    $r_t \leftarrow -1$ 

9: else if  $\Delta D > 0$  then

10:   $r_t \leftarrow \frac{\Delta D}{D_{max}} * \left(1.0 - 0.9 * \frac{|PS_t|}{5.0}\right)$   $\triangleright PS_t$  is the propeller speed at time  $t$ 

11: else

12:   $r_t \leftarrow \frac{\Delta D}{D_{max}} - 0.01 * |PS_t|$ 

13: end if

```

We trained a functional agent using two neural networks, each made of two hidden layers with thirty-two ReLU units each. All other PPO parameters were set to the standard values used in Stable-baselines3, except that `ent_coef` parameter was set to 0.001. The training length was 5.0×10^5 steps. We performed all our experiments training the agent on a laptop running Ubuntu 20.04 and powered by an 11th Gen Intel Core i7-11800H (2.30GHz), 16GB DDR5 RAM, and an Nvidia RTX 3060 GPU. The whole training process took three hours and forty minutes.

In this preliminary experiment, we aim for a policy to enable the agent to reach the goal using the wind as the preferred propulsion, i.e., with little use of the propeller. Figure 25 shows the average surge velocity of the boat for different wind speeds. In the worst-case scenario, the E-Boat should travel with a surge velocity of about one meter per second. The expected return should be at least 1.6, and an episode should not exceed fifty steps. Figure 26a displays the average return per episode during training, while Fig. 26b displays the ratio between the episode length and the expected worst episode length (50 steps). The experimental data shows that, on average, the agent exceeded our expectations.

5.2.3 Seven Waypoints Mission

In this experiment, we evaluate the execution of a seven-waypoint mission. Throughout the mission, the EBoat must navigate under different wind conditions. For our evaluation, we built a scenario with a wind speed of approximately 6 m/s and relatively calm ocean waters.

The trajectory of the E-Boat during mission execution is shown in Fig. 27. The PPO policy and PID-based controller follow a very similar course. However, differences become apparent when we analyze the mission execution data presented in Fig. 28. The SAC policy does not perform as well as the other two, and to avoid obscuring other data, we have plotted the SAC data as a dotted line. Between 0 and 154 seconds, the performance achieved by PID and PPO is very similar when the E-Boat is sailing in running, broad reach, or beam reach (see Fig. 29). During close reach sailing (periods 154 to 193 and 306 to 368), PPO achieves higher surge velocities than PID. However, between 193 and 306 seconds, PID reaches higher speeds when sailing against the wind.

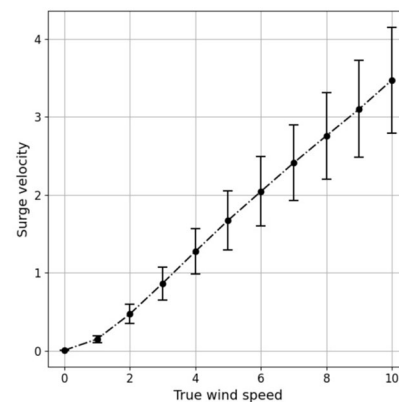


Fig. 25 Average surge velocity as a function of the apparent wind speed. The values represent the velocity in a linear course averaged over different wind directions

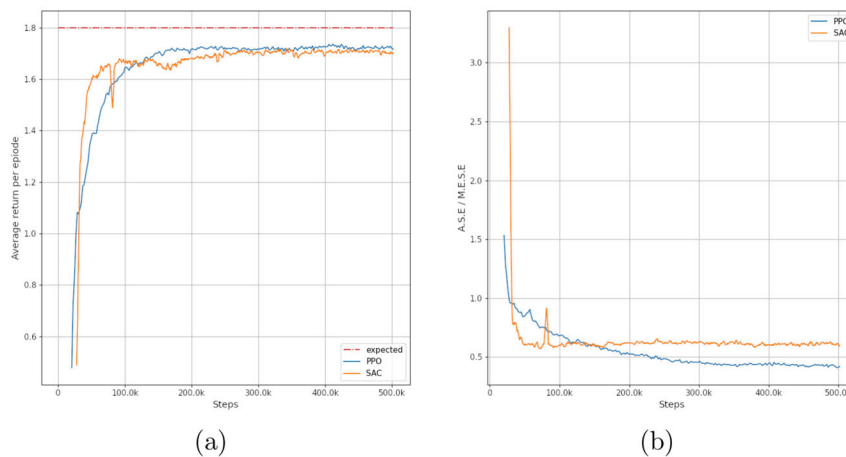


Fig. 26 Training data. Figure (a) shows the agent's average return value per episode. Figure (b) shows the ratio between the average steps per episode (A.S.E.) and the maximum expected steps per episode (M.E.S.E.), which is equal to fifty

Figure 28d shows us that PID applies more power to the propeller than PPO, indicating that PPO achieves similar results using less energy. These results demonstrate how our environment can aid in the development, evaluation, and comparison of various approaches to implementing autonomous control for sailing robots.

6 Conclusion

There are not many open-source OVEs available, and only USVSim seems to offer a sailboat simulation. Different authors proposing new GNC solutions usually implement simplistic simulations or use simulations that do not represent the complex motion of a sailboat. Yara comes as tool to address this issue, offering different resources to make easy developing and testing GNC solutions with a reliable

physics simulation. The digital twin E-Boat was modeled with a 6-DOF model parametrized to emulate a actual sailing robot. In this way, path-planning algorithms and other control solutions, such as Deep Reinforcement Learning (DRL), can be evaluated in different environmental conditions, with dynamic, static or no obstacles. As the use of DRL methods are growing in the literature, Yara facilitates training and testing of Reinforcement learning applications are enabled through Python classes that implement Gymnasium environments and integration with Stable-baselines3. We could not find other open-source OVE with a read-to-go integration with RL APIs.

Within Yara, we build the E-Boat, digital twin of real sailing robots built in Brazil for environmental monitoring. The physics models implemented within the E-Boat enable a reliable simulation of its motion under different ambient conditions, such as waves, water currents, and wind. The sail is modeled as a standard sail, not a rigid sail.

Our experiments demonstrate that E-Boat's simulated maneuverability and response of the sailing boats under different wind conditions closely resembled real-world behavior, showcasing the accuracy of the physics modeling implemented in Yara. Additionally, the integration with Gymnasium and Stable-baselines3 allowed us to train RL agents for autonomous GNC decision-making successfully. The experiments depicted in Section 5.2.3 demonstrate the practical applicability of Yara as a reliable tool for implementing and testing different GNC strategies and comparing them.

In future work, we will improve the wave model by creating an interface between it and the wind so the wind conditions modify the water surface conditions. We will also

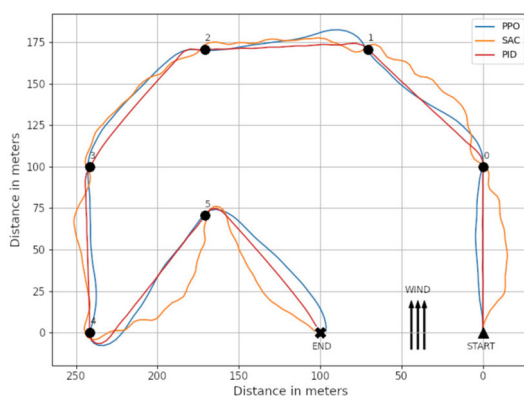


Fig. 27 E-Boat's trajectory

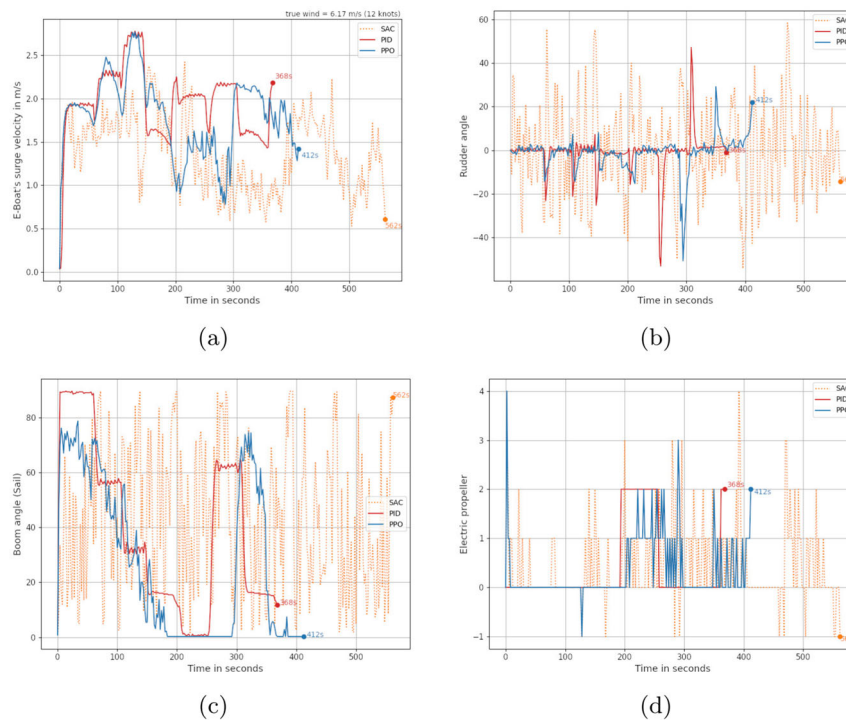


Fig. 28 Mission data

include fog and rain conditions. We will also include fog and rain conditions. Other parameters of the PPO will also be worked on in the future implementations.

Concerning GNC's decision policies, the initial results were very promising. Suda & Nokovski [22], Mexas et al. [21] and Rieppi et al. [31] presented some promising results using RL agents to control sailing boats. We will investigate new return functions that can improve the results we already have. Two challenging scenarios happen when the waypoint is in the “no go zone” (see Fig. 29) and in strong wind conditions. Suda & Nokovski [22] tested PPO and SAC to learn optimized tack maneuver, but they used a 3-DOF simulation, and their approach must be checked in a more complex environment as Yara. We believe the answer is a hierarchical controller combining RL-trained policies with

other classic types of controllers. We will test that hypothesis in future work.

Finally, the results obtained with the Yara OVE support its effectiveness and value as a tool for developing, training, and evaluating autonomous GNC systems for sailing robots on an RL-based approach. With further refinements and future research, Yara has the potential to contribute significantly to the advancement of autonomous sailing robot technology, ultimately leading to improved performance and capabilities in real-world applications.

Supplementary information Our Yara OVE is available at https://github.com/medialab-fboat/Yara_OVE.git

Acknowledgements We acknowledge and be thankful to CAPES AMSUD (proj. 88881.522966/2020-01 NVIDIA, CNPq-FNDCT-MCTI (405535/2022-8) and the City of Niterói (Brazil) for supporting and funding this research.

Author Contributions All authors contributed to the conception, design, and evaluation of the Yara ocean virtual environment and the validation experiments. Yara was implemented by Eduardo Charles Vasconcellos with the contributions of Álvaro Pinto Fernandes de Negreiros, André Paulo Dantas de Araújo, and Raphael Guerra. The E-Boat maneuverability was evaluated by Álvaro Pinto Fernandes de Negreiros, Esteban Walter Gonzales Clua, and Luiz Marcos Garcia Gonçalves. Eduardo Charles Vasconcellos and Philippe Preux were responsible for the RL agent design and evaluation and for integrating the RL APIs into the environment. Eduardo Charles Vasconcellos, André Paulo Dantas de Araújo, and Philippe Preux wrote the first draft of the manuscript. All authors commented on previous versions of the manuscript. They read and approved the final manuscript.

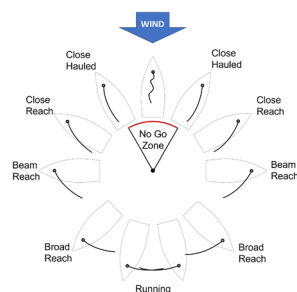


Fig. 29 Sailing points

Funding This work was supported by CAPES AMSUD (proj. 88881.522966/2020-01 NVIDIA, CNPq-FNDCT-MCTI (405535/2022-8) and the City of Niterói (Brazil).

Data Availability The trained agent and the training data are available within Yara's code at https://github.com/medialab-fboat/Yara_OVE.git

Declarations

Conflicts of Interest The authors have no relevant financial or non-financial interests to disclose.

Ethical Approval not applicable

Consent to Participate not applicable

Consent for Publication not applicable

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

References

1. Theja, V.B., Yachameni, P., Shakeera, S., Venkataraman, H.: Integration of gazebo and ros for underwater vehicle environment. In: Proc. of OCEANS 2022 - Chennai, vol. 1, pp. 1–6 (2022)
2. Sun, Q., Qi, W., Liu, H., Ji, X., Qian, H.: Toward long-term sailing robots: State of the art from energy perspectives. *Front. Robot. AI* **8**, 416 (2022)
3. Negreiros, A.P.F., Correa, W.S., Araujo, A.P.D., Santos, D.H., Vilas Boas, J.M., Dias, D.H.N., Clua, E.W.G., Gonçalves, L.M.G.: Sustainable solutions for sea monitoring with robotic sailboats: N-boat and f-boat twins. *Frontiers in Robotics and AI*, 93 (2022)
4. Ang, Y.-T., Ng, W.-K., Chong, Y.-W., Wan, J., Chee, S.-Y., Firth, L.B.: An autonomous sailboat for environment monitoring. In: 2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 242–246 (2022). IEEE
5. Mendonça, R., Santana, P., Marques, F., Lourenço, A., Silva, J.a., Barata, J.: Kelpie: A ros-based multi-robot simulator for water surface and aerial vehicles. In: 2013 IEEE International Conference on Systems, Man, and Cybernetics, pp. 3645–3650 (2013)
6. Plumet, F., Pêtres, C., Romero-Ramirez, M.-A., Gas, B., Ieng, S.-H.: Toward an autonomous sailing boat. *IEEE J. Ocean. Eng.* **40**(2), 397–407 (2015)
7. Jing, P., Hu, H., Zhan, F., Chen, Y., Shi, Y.: Agent-based simulation of autonomous vehicles: A systematic literature review. *IEEE Access* **8**, 79089–79103 (2020)
8. Lapan, M.: Deep Reinforcement Learning Hands-On: Apply Modern RL Methods to Practical Problems of Chatbots, Robotics, Discrete Optimization, Web Automation, and More. Packt Publishing Ltd, Birmingham, UK (2020)
9. Gazebo: Gazebo Web Page: Understand how to simulate underwater vehicles. (2024). https://gazebo.org/api/gazebo/5.0/underwater_vehicles.html
10. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.* **22**(1), 12348–12355 (2021)
11. Farama Foundation: Gymnasium. (2024). <https://github.com/Farama-Foundation/Gymnasium>
12. Plumet, F., Saoud, H., Hua, M.-D.: Line following for an autonomous sailboat using potential fields method. In: Proc. of 2013 MTS/IEEE OCEANS - Bergen, pp. 1–6 (2013)
13. Shen, Z., Wang, S., Yu, H., Guo, C.: Online speed optimization with feedforward of unmanned sailboat via extremum seeking without steady-state oscillation. *Ocean Eng.* **189**, 106393 (2019)
14. da Silva-Junior, A.G., Dos Santos, D.H., de Negreiros, A.P.F., Silva, J.M.V.B.S., Gonçalves, L.M.G.: High-level path planning for an autonomous sailboat robot using q-learning. *Sensors (Switzerland)* **20**(6) (2020)
15. Shen, Z., Ding, W., Liu, Y., Yu, H.: Path planning optimization for unmanned sailboat in complex marine environment. *Ocean Eng.* **269**, 113475 (2023)
16. Pêtres, C., Romero-Ramirez, M.-A., Plumet, F.: Reactive path planning for autonomous sailboat. In: 2011 15th International Conference on Advanced Robotics (ICAR), pp. 112–117 (2011). IEEE
17. Zhou, L., Chen, K., Dong, H., Chi, S., Chen, Z.: An improved beetle swarm optimization algorithm for the intelligent navigation control of autonomous sailing robots. *IEEE Access* **9**, 5296–5311 (2021)
18. Liu, L., Wang, C., Gao, H., Shen, D., Liao, Y.: High-level path planning of unmanned sailboat for sailing championship and innovative education. In: 2022 IEEE International Conference on Unmanned Systems (ICUS), pp. 1557–1562 (2022)
19. Orozco-Rosas, U., Picos, K., Pantrigo, J.J., Montemayor, A.S., Cuesta-Infante, A.: Mobile robot path planning using a qapf learning algorithm for known and unknown environments. *IEEE Access* **10**, 84648–84663 (2022)
20. Tipsuwan, Y., Sanposh, P., Techajaronjit, N.: Overview and control strategies of autonomous sailboats—a survey. *Ocean Eng.* **281**, 114879 (2023)
21. Méxas, R.P., Leta, F.R., Clua, E.W.G.: Comparison of reinforcement and imitation learning algorithms in autonomous sailboat digital twins. *IEEE Lat. Am. Trans.* **20**(9), 2153–2161 (2022)
22. Suda, T., Nikovski, D.: Deep reinforcement learning for optimal sailing upwind. In: 2022 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2022)
23. Liu, S., Yu, Z., Wang, T., Chen, Y., Zhang, Y., Cai, Y.: Mpc-based collaborative control of sail and rudder for unmanned sailboat. *J. Mar. Sci. Eng.* **11**, 460 (2023)
24. Saoud, H., Hua, M.D., Plumet, F., Amar, F.B.: Routing and course control of an autonomous sailboat. In: 2015 European Conference on Mobile Robots (ECMR), pp. 1–6 (2015)
25. Paravisi, M., Santos, D.H., Jorge, V., Heck, G., Gonçalves, L.M.G., Amory, A.: Unmanned surface vehicle simulator with realistic environmental disturbances. *Sensors* **19**(5) (2019)
26. Prats, M., Perez, J., Fernández, J.J., Sanz, P.J.: An open source tool for simulation and supervision of underwater intervention missions. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2577–2582 (2012). IEEE
27. Manhães, M.M.M., Scherer, S.A., Voss, M., Douat, L.R., Rauschenbach, T.: Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In: OCEANS 2016 MTS/IEEE Monterey, pp. 1–8 (2016). IEEE
28. Bingham, B., Aguero, C., McCarrin, M., Klamo, J., Malia, J., Allen, K., Lum, T., Rawson, M., Waqar, R.: Toward maritime robotic

- simulation in gazebo. In: Proceedings of MTS/IEEE OCEANS Conference, Seattle, WA (2019)
29. Kermorgant, O.: A dynamic simulator for underwater vehicle-manipulators. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots, pp. 25–36 (2014). Springer
 30. Chacon Mosquera, E.F.: Autonomous sailboat prototype sensors and electronics implementation with machine learning for navigation. PhD thesis, UPC, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial (2021). <http://hdl.handle.net/2117/344154>
 31. Rieppi, A., Rieppi, F., Marzola, M., Tejera, G.: Autonomous sailboat control based on reinforcement learning for navigation in variable conditions. In: 2023 XLIX Latin American Computer Conference (CLEI), pp. 1–9 (2023)
 32. Zhou, C., Wang, Y., Wang, L., He, H.: Obstacle avoidance strategy for an autonomous surface vessel based on modified deep deterministic policy gradient. *Ocean Eng.* **243**, 110166 (2022)
 33. Fossen, T.I.: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, New Jersey, USA (2011)
 34. SNAME, T.: Nomenclature for treating the motion of a submerged body through a fluid. The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin (1950), 1–5 (1950)
 35. Marine Sports: Reference Data for Motor Phantom SW with Display for Salt Water (2023). <https://www.marinefishing.com.br/produto/phantom-sw-com-display-para-agua-salgada-1714.html>
 36. Thon, S., Dischler, J.-M., Ghazanfarpour, D.: Ocean waves synthesis using a spectrum-based turbulence function. In: Proc. of Computer Graphics International, 2000, pp. 65–72 (2000)
 37. Tessendorf, J., et al.: Simulating ocean water. Simulating nature: realistic and interactive techniques. *SIGGRAPH* **1**(2), 5 (2001)
 38. Fréchet, J.: Realistic simulation of ocean surface using wave spectra. In: Proceedings of the First International Conference on Computer Graphics Theory and Applications (GRAPP 2006), pp. 76–83 (2006)
 39. GPU-GEMs: GPU GEMs: Chapter 1. Effective Water Simulation from Physical Models. Available at <https://developer.nvidia.com/gpugems/gpugems/> (2023)
 40. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint [arXiv:1606.01540](https://arxiv.org/abs/1606.01540) (2016)
 41. Zamora, I., Lopez, N.G., Vilches, V.M., Cordero, A.H.: Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. arXiv preprint [arXiv:1608.05742](https://arxiv.org/abs/1608.05742) (2016)
 42. Domingues, O.D., Flet-Berliac, Y., Leurent, E., Ménard, P., Shang, X., Valko, M.: rlberry - A Reinforcement Learning Library for Research and Education (2021). <https://github.com/rlberry-py/rlberry>
 43. Caspi, I., Leibovich, G., Novik, G., Endrawis, S.: Reinforcement Learning Coach (2017). <https://doi.org/10.5281/zenodo.1134899>
 44. D'Eramo, C., Tateo, D., Bonarini, A., Restelli, M., Peters, J.: Mushroomrl: Simplifying reinforcement learning research. *J. Mach. Learn. Res.* **22**(131), 1–5 (2021)
 45. Huang, S., Dossa, R.F.J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., Araújo, J.G.M.: Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *J. Mach. Learn. Res.* **23**(274), 1–18 (2022)
 46. Kuhnle, A., Schaarschmidt, M., Fricke, K.: Tensorforce: a TensorFlow library for applied reinforcement learning (2023). <https://github.com/tensorforce/tensorforce>
 47. Plappert, M.: keras-rl. GitHub (2023). <https://github.com/keras-rl/keras-rl>
 48. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint [arXiv:1707.06347](https://arxiv.org/abs/1707.06347) (2017)
 49. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870 (2018). PMLR

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Eduardo Charles Vasconcellos has a master's degree in Applied Computing from the National Institute for Space Research (INPE, Brazil) and a PhD. in Computer Science from Fluminense Federal University (UFF, Brazil) with an internship at Georgia Institute of Technology (GATECH, USA). He worked as a researcher fellow at INPE from 2011 to 2013 and at UFF from 2017 to 2022. He developed a post-doctoral research position at INRIA from 2022 to 2023, and is currently a project manager at UFF.

Álvaro Pinto Fernandes Negreiros holds a degree in Science and Technology and Computer Engineering, and has a PhD. in Electrical Engineering from Federal University of Rio Grande do Norte (UFRN, Brazil). His research was related to designing and constructing low-cost hardware for educational robotics, unmanned autonomous vehicles, and machine learning.

André Paulo Dantas de Araújo is a Software Engineer. He has a master's degree in computer science from the Federal University of Rio Grande do Norte (UFRN, Brazil) and he is a PhD candidate in computer science at the Fluminense Federal University. He worked for the Brazilian Navy for 8 years as a software engineer.

Raphael Guerra is a professor at Universidade Federal Fluminense (UFF, Brazil). He has a master's degree in computer science from UFF and a Doctoral degree (with Hons.) in electrical and computer engineering from Technische Universität Kaiserslautern (Germany). His primary research interest is embedded system applications in IoT.


Philippe Preux is a Computer Science professor at the Université de Lille (France). He belongs to the Centre de Recherche en Informatique, Signal et Automatique de Lille (CRISTAL) (UMR CNRS 9189), and the Inria-Lille research center. He has PhD in Computer Science from Université de Lille. His research concerns machine learning, in particular, sequential decision-making under uncertainty (reinforcement learning, bandits). More generally, his interests are data science, algorithmics, and data structures in settings where things evolve over time.

Davi Henrique dos Santos is a professor at the Computer Systems Department of the Computer Science Center of the Federal University of Paraíba (UFPB, Brazil). He has a degree in Computer Engineering from the Federal University of Rio Grande do Norte (UFRN, Brazil), a master's degree from the UFRN, and a PhD from the UFRN. He has experience in Robotics, Mechatronics, and Automation, with an emphasis on Robotics Mechatronics and Automation, working mainly on the following topics: USV, particulate matter, sailboat, model deployment, and web interface.

Luiz Marcos Garcia Gonçalves is a Full Professor in the Department of Computer Engineering and Automation at the Federal University of Rio Grande do Norte (UFRN, Brazil). He has a PhD in Systems and Computer Engineering from the Computer Graphics Laboratory (LCG) of COPPE-UFRJ (Brazil), including a two-year internship at the Perceptual Robotics Laboratory of UMASS (USA). His research interests include several aspects of Computing Methodologies and Techniques, working mainly with Perceptual Information Systems for Robotics.

Esteban Walter Gonzales Clua is a full professor at Universidade Federal Fluminense and a CNPq top-level researcher. He has a master's and doctoral degrees from PUC-Rio. His main research areas are Digital Games, Virtual Reality, GPUs, and Visualization. In 2015, he was named NVIDIA CUDA Fellow, and in 2020, NVIDIA Ambassador. In 2024, he received the Friends of the Navy medal from the Brazilian Navy for the importance of his research for the institution. Today, he is the general president of IFIP (International Federation of Information Processing) Technical Committee 14 (TC14 Digital Entertainment), in addition to being the representative for Brazil, and he is editor-in-chief of the CLEI Journal and a member of several editorial committees.

Authors and Affiliations

Eduardo Charles Vasconcellos¹  · **Álvaro Pinto Fernandes Negreiros²** · **André Paulo Dantas de Araújo^{1,4}** · **Raphael Guerra⁴** · **Philippe Preux¹** · **Davi Henrique dos Santos^{2,3}** · **Luiz Marcos Garcia Gonçalves²** · **Esteban Walter Gonzalez Clua⁴**

✉ Eduardo Charles Vasconcellos
charles.edu@gmail.com

Álvaro Pinto Fernandes Negreiros
alvaro.negreiros.107@ufrn.edu.br

André Paulo Dantas de Araújo
andrelda@id.uff.br

Raphael Guerra
rguerra@ic.uff.br

Philippe Preux
philippe.preux@univ-lille.fr

Davi Henrique dos Santos
davihenriqueds@gmail.com

Luiz Marcos Garcia Gonçalves
lmarcos@dca.ufrn.br

Esteban Walter Gonzalez Clua
esteban@ic.uff.br

¹ Inria, Université de Lille, Villeneuve d'Ascq, Lille 59650, France

² Elec. and Comp. Eng. Graduate Program, Univ. Fed. do Rio Grande do Norte, Natal 59078-970, RN, Brazil

³ Systems Engineering and Robotics Laboratory, Universidade Federal da Paraíba, João Pessoa 58051-900, PB, Brazil

⁴ Institute of Computing, Fluminense Federal University, Niterói 24210-346, RJ, Brazil