Exploring UUV Development with NauSim: An Open-Source Simulation Platform

1st César Antonio Ortiz Toro
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
ca.ortiz@upm.es

2nd Cristina Cerrada-Collado,
Departamento de Ciencias de la Computación y Control Automático
UNED
Madrid, Spain
criscerrada@dia.uned.es

3rd David Moreno-Salinas
Departamento de Ciencias de la Computación y Control Automático
UNED
Madrid, Spain
dmoreno@dia.uned.es

4th Dictino Chaos-García
Departamento de Ciencias de la Computación y Control Automático
UNED
Madrid, Spain
dchaos@dia.uned.es

5th Karen Lyn García-Suárez
I. para el Desarrollo Tecnológico y la Innovación en Comunicaciones
ULPGC
Las Palmas, Spain
karen.garcia101@alu.ulpgc.es

6th Pablo Otero-Roth
Instituto de Ingeniería Oceánica, E.T.S. de Ingeniería de Telecomunicación
UMA
Málaga, Spain
pablo.otero@uma.es

7th Juan Manuel Vidal-Pérez
Escuela de Ingenierías Marinas, Náutica y Radioelectrónica
UCA
Cádiz, Spain
juan.vidal@uca.es

8th Miguel Ángel Luque-Nieto
Instituto de Ingeniería Oceánica, E.T.S. de Ingeniería de Telecomunicación
UMA
Málaga, Spain
luquen@uma.es

9th Ana Isabel Vázquez
Escuela de Ingenierías Marinas, Náutica y Radioelectrónica
UCA
Cádiz, Spain
anaisabel.vazquez@uca.es

10th José Jesús Fraile-Ardanuy
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
jesus.fraile.ardanuy@upm.es

11th Vicente Negro-Valdecantos
Escuela Técnica Superior de ing. de caminos canales y puertos
UPM
Madrid, Spain
vicente.negro@upm.es

12th Eugenio Jiménez-Yguacel
I. para el Desarrollo Tecnológico y la Innovación en Comunicaciones
ULPGC
Las Palmas, Spain
eugenio.jimenez@ulpgc.es

13th Joaquín Aranda-Almansa
Departamento de Ciencias de la Computación y Control Automático
UNED
Madrid, Spain
jaranda@dia.uned.es

14th Santiago Zazo-Bello
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
santiago.zazo@upm.es

15th Pedro José Zufiria-Zatarain
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
pedro.zufiria@upm.es

16th Luis Magdalena-Layos
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
luis.magdalena@upm.es

17th Juan Parras-Moral
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
j.parras@upm.es

18th Alvaro Gutiérrez-Martín
Escuela Técnica Superior de Ingenieros de Telecomunicación
UPM
Madrid, Spain
a.gutierrez@upm.es

***Abstract*—**

This article introduces NauSim, an open-source simulation tool designed for developing control algorithms Unmanned Underwater Vehicle (UUVs). NauSim is targeted at researchers and developers in underwater robotics, with a focus on Machine Learning (ML) based applications. Key design principles include a clean, flexible, and modular architecture that can easily integrate with various existing control paradigms. The simulation tool is made in Python, acknowledging its prominence in ML research. NauSim emphasizes simplicity in deploying control algorithms from the simulator to target hardware. To ensure applicability in real-world scenarios, NauSim strives to provide an experience that closely mimics real-life conditions, encompassing both sensory and physical interaction models.

Keywords*— *Simulation, UUVs, Multi-vehicle systems, Sensors and actuators, Robot Navigation, Programming and Vision

I. INTRODUCTION

The development of marine robotics is emerging as a interdisciplinary field that integrates various disciplines, including engineering, computer science, and marine sciences. This field focuses on the advancement and deployment of both autonomous and remotely operated underwater drones, which have become essential tools in a variety of marine applications. These technological innovations have been pivotal in supporting numerous projects, including maintenance missions as seen in [11] and [9], Additionally, reef monitoring, as highlighted in [19], utilizes underwater drones to collect data and monitor the health of coral reefs, contributing significantly to conservation efforts. In the realm of aquaculture, fish farm monitoring, discussed in [4] and [3], benefits from these drones by providing continuous surveillance and management of fish stocks, thereby improving yield and sustainability. Moreover, underwater drones are employed in water quality control, as described in [13], to monitor and ensure the health of aquatic

This work has been supported by Grant PID2020-112502RB-C41, PID2020-112502RB-C42, PID2020-112502RB-C43 and PID2020-112502RB-C44 funded by MCIN/AEI/10.13039/501100011033.

environments, detecting pollutants and other harmful substances.

Despite the growing importance of underwater robotics, the research and development of these technologies are fraught with challenges, mainly due to their dependence on aquatic environments. Any project in this area requires access to suitable bodies of water, such as rivers, seas, or lakes, which come with associated logistical problems. In addition, the variable and often unpredictable nature of these environments adds a layer of uncertainty to the test conditions, making it difficult to repeat tests or ensure the safety of the equipment. The alternative, a large enough water tank, is a costly option with limited availability. Additionally, the confined space of a tank, while providing a controlled environment, may not fully replicate the complexities of natural aquatic conditions, which can limit the effectiveness of testing.

Furthermore, the characteristics of an underwater environment make test monitoring particularly challenging. Visibility may be limited, and the environment restricts the use of traditional monitoring equipment. There is also a risk of losing or damaging vehicles during testing due to unforeseen problems, such as technical failures or environmental hazards. This risk requires a cautious and often time-consuming approach to experimentation, where each step must be carefully planned and executed to mitigate potential losses.

These challenges demand the use of simulators for the design and preliminary testing of vehicle behavior. Simulators mitigate the dependence on access to a water environment prior to the deployment phase and allow observation of underwater tasks when direct observation of the system is not feasible. This has led to numerous developments. Among the most recognized are the UWSim software, presented in [16], and the Gazebo UUV Simulator extension developed in [14], although these have not been updated for some time. With the aim of utilizing well-known platforms, there are packages that integrate simulation functionalities within MATLAB™ and Simulink™, such as the work presented in [22] and Simu2VITA in [6]. In recent years, the trend has been towards providing visual fidelity by leveraging the capabilities of commercial 3D engines, as evidenced in [15] (HoloOcean) and [1] (UNav-Sim), albeit at the cost of making these simulators somewhat dependent on the structure and limitations of the engines. Ultimately, these developments are frequently associated with specific projects, and their features do not always align with the requirements of other endeavors.

In this article we present NauSim, an open source simulation tool for underwater unmanned underwater vehicle (UUVs), designed according to the following guidelines:

- It is a software designed with the objective of developing control algorithms for autonomous underwater vehicles, either individually or as group behavior, aimed at researchers and developers in underwater robotics, with emphasis, but not limited to Machine Learning (ML) based developments. The architecture has to be clean, flexible and modular.
- It must be easily integrated with different existing control paradigms. The control algorithms are external to the simulator. Taking into account the importance that Python has acquired as a reference language (see [22] or [17]) in ML, the simulator must be made or be compatible with this language.
- The deployment on the target hardware of the control algorithms developed in the simulator has to be as simple as possible.
- Since the results are oriented to use in real environments, it should provide an experience as close as possible to reality to the drone/drones, either in the sensing model, including the visual domain, or as a physical model of interaction.

II. ARCHITECTURE

The core of the NauSim architecture is based on the sensor-controller-actuator model, a fundamental framework in drone design and operation that ensures efficient and accurate execution of tasks. This model divides the functionality of the drone into three interconnected layers: sensors, controllers, and actuators. The sensors are responsible for collecting data from the environment, such as temperature, pressure, and visual information. The controllers process this data, making decisions based on pre-programmed algorithms or artificial intelligence to determine the appropriate actions. Finally, the actuators execute these actions, performing tasks such as maneuvering the drone, adjusting its position, or manipulating objects.

This approach allows for continuous feedback and real-time adjustments, configuring the drone as a versatile and adaptable tool. Sensors provide ongoing data to the controllers, which continuously evaluate this information and send commands to the actuators, ensuring the drone can respond dynamically to changes in its environment. This architecture not only enhances the drone's operational efficiency but also increases its reliability in performing complex tasks. Additionally, the sensor-controller-actuator architecture isolates the different components of the drone, promoting the reusability of already implemented parts. This modularity makes it easier to upgrade or replace individual components without redesigning the entire system. It also facilitates a seamless transition between the simulated and real robot.

The interaction of this architecture with the simulated world is managed through a physical model, which translates the actions of the actuators into changes in the robot's state within the simulated space. A diagram illustrating the general organization of this development is shown in Fig. 1.

A. Simulated enviroment

A robotics simulator, regardless of its specific functionalities, is typically organized around a virtual space that represents the real world. This virtual space is where the simulation process takes place, allowing developers to test and refine their robotic systems in a controlled and repeatable environment. The visual representation of this virtual environment in the NauSim simulator is handled by the Panda3D engine.

Developed by Disney Interactive in 2002 for its theme park virtual reality division, Panda3D [8] (originally ‘Platform Agnostic Networked Display Architecture’, although its use as an acronym has been lost over time) was released under BSD license in 2008, and has since been maintained and

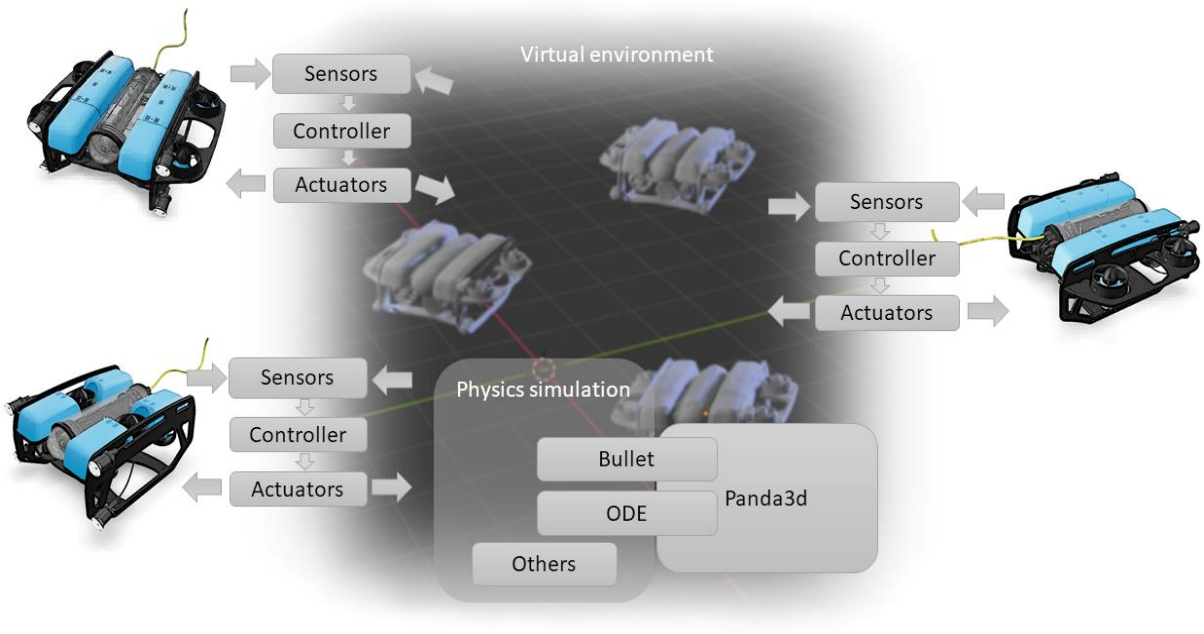

Fig.1. Schematic representation of the NauSim simulator architecture

extended by an active community of users. Panda3D offers a complete set of functionalities, is fully cross-platform, and features an interface fully developed in Python.

Panda3D is a scene graph-based engine. A scene graph is a general data structure commonly used in vector graphics editing applications and 3D engines. It imposes a hierarchy on the logical and often spatial representation of a graphical scene, organizing it as a collection of nodes forming a tree structure. A node may have multiple children but only one parent. Any effect applied to a parent node automatically propagates to all its child nodes. In a 3D engine, this typically involves associating a geometric transformation matrix to each and to determine the simulated space position, either relative to the global axes or to its parent node, concatenate those matrices.

This hierarchical structure is well-suited to the abstract definition of space and the entities forming the simulated environment. For instance, a drone within the simulator can be 'loaded' with multiple sensors that can move or pivot relative to the parent object (the drone itself) while maintaining actualizing synchronization with the parent object.

The configuration of a test scenario is defined using a .json, where multiple participating robots can be defined, each with different sensors, controllers and associated physical models.

The simulator is not limited to any specific scenario editor, providing flexibility in creating and defining virtual environments. Virtual scenarios are defined using glTF (GL Transmission Format) files, which are commonly supported by most 3D modeling software. These glTF format generally consist on a text file (.gltf) using a json structure, that describes the scene, along with separate files containing the geometry and texture data of the objects. This format is extensible through tags, allowing developers to define specific functions for the simulator. For example, they can include simplified geometry for collision detection, add invisible 'walls' to limit the simulation area, or import geometry based on height maps directly into the 3D engine. Some examples of scenarios created for NauSim can be seen in Fig. 2.

B. Sensors

Sensors are the first layer, responsible for collecting data from the drone's environment. The real-time information provided by the sensors, together with possible communications with other drones are the only data input the drone has, form the basis for the decision-making processes within the drone control system and are therefore critical for the behavior of the drone either in an autonomous mission or during a training process.

The sensors encompass both simulations of real-world sensors and virtual sensors. Real sensors might include accelerometers, sonars, GPS units, cameras, and lidar systems, which provide essential data about the drone's position, orientation, speed, and surrounding environment. Virtual sensors, sensors that have no real-world equivalents but designed to assist in the development and testing of for the development of control models, especially those related with machine learning. For example, a collision sensor can be simulated to detect potential impacts with obstacles, enabling the development of collision avoidance algorithms.

To accurately simulate real conditions on a drone, sensors operate in a separate thread where the update rate of each sensor is defined independently. This approach reflects the variability in how different sensors collect data. For example, a GPS unit might update its position data once per second, while an accelerometer might provide data hundreds of times per second. This allows for more accurate modeling of real-world scenarios. Different sensors have different levels of precision and latency, and these characteristics can be replicated in the simulation environment ensuring that the simulated conditions closely match those the drone will encounter in the real world. Also, running the sensors in a separate thread, the simulator can minimize the impact of

computationally expensive sensor simulations on overall performance.

Sensors are defined as independent components and can be reused in new simulated robot models. There is no distinction between ‘simulated’ and ‘real’ sensors (defined as an ‘interface’ for accessing data from the corresponding hardware’), which makes it possible, once an ML model has been trained, to switch transparently between the sensors used in the simulator for training and their real counterparts.

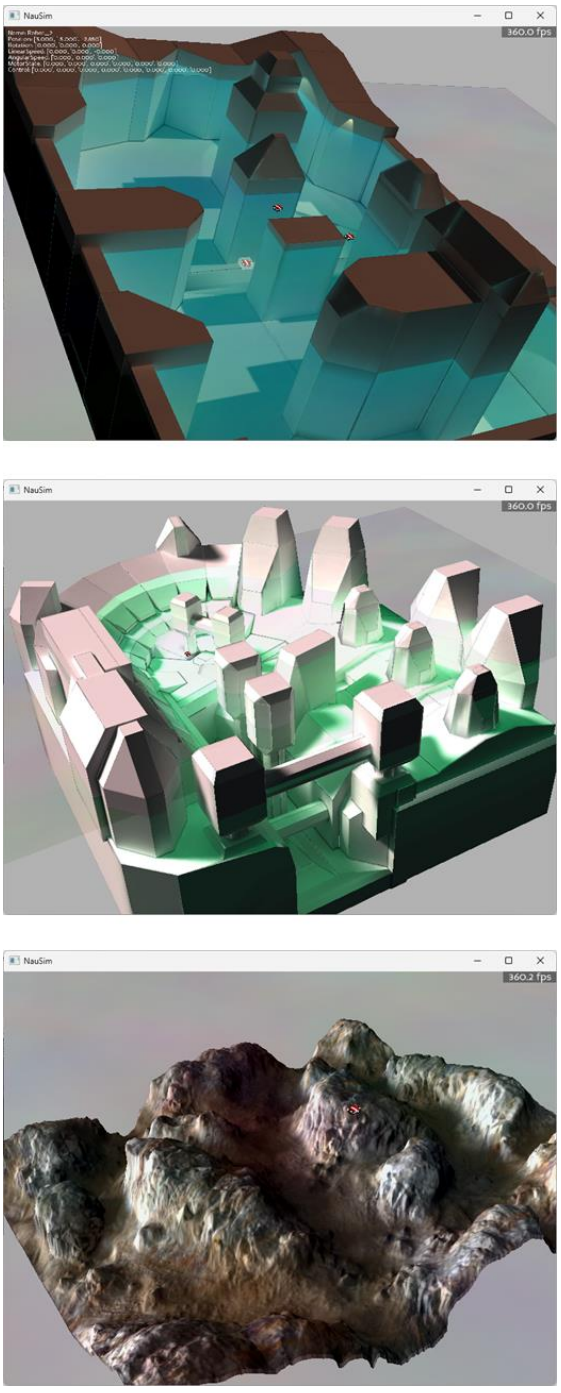

Fig.2. Some examples of different scenes developed for NauSim, as they appear in the simulator.

C. Controllers

The controller serves as the second layer, functioning as the brain of the drone. It interprets input data to determine the current state of the drone and plans the necessary actions to achieve its objectives. This can involve vehicle stabilization, navigation, obstacle avoidance, and specific mission tasks. In the sensor-controller-actuator model, the controller does not interact directly with the environment; instead, it receives information via sensors and, in particular cases such as drone swarm simulations, through inter-robot communication. The interaction with the environment is mediated through actuators, enabling the use of the same controller for both a simulated drone and its real-world counterpart.

It processes raw data from various sensors, such as GPS for location tracking, gyroscopes for orientation, accelerometers for movement detection, cameras for visual input, and other specialized sensors depending on the drone's application. This data is analyzed to assess the drone's current state, including its position, velocity, orientation, and environmental conditions. Actuators, such as motors, servos, and other mechanical devices, execute the commands issued by the controller. This layer of abstraction allows the same controller software to be used in both simulated environments and real-world applications. Once validated, these controllers can be easily transferred to the objective hardware.

The controller concept does not imply any specific technique; it can range from a minimal program designed "ad-hoc" for a specific situation to complex learning algorithms, such as genetic algorithms. The simulator's design allows for extending a drone's control functionalities using a hierarchy of controllers, where a parent controller's role is to select which child controller to deploy at any given moment.

D. Actuators

The final layer of the sensor-controller-actuator architecture is comprised of actuators, which execute the commands generated by the controller. Actuators include motors, servos, and other mechanical components that physically adjust the drone's position and orientation. Actuators include motors, servos, and other mechanical components that physically adjust the drone's position and orientation. The specific type of actuators used depends on the specific drone

A simulated actuator functions within a virtual environment to replicate the behavior of real actuators. It translates the controller's commands into values that the simulation's physical engine can process. For example, in a simulation, the actuator might adjust the power levels of virtual motors to change the drone's position and orientation in the simulated space. This allows developers to test and refine control algorithms in a safe and controlled environment, ensuring that they function correctly before deploying them on a real drone.

In a real-world setting, actuators are usually defined as an interface drone's hardware components. These interfaces can be managed through direct connections to the drone's control systems or through external control libraries that act as middleware such as MAVLink [10] and ROS2 [12] .

Fig. 3. BlueROV2 in its heavy configuration.

E. Physics engine

Interaction with the virtual world is achieved through the simulation of a physical model associated with each robot. A physical model replicates the behaviors, properties, and dynamics of real-world objects within a virtual environment.

The primary function of the physical model is to translate the commands generated by virtual actuators into changes in the robot's position, orientation, and other physical attributes within the simulated space. I. e. when a virtual actuator adjusts the power levels of motors, the physical model calculates the resulting movement of the robot based on principles such as Newtonian mechanics, inertia, and buoyancy.

NauSim does not associate the concept of a physical model with any specific algorithm or technique; it is possible to develop a model for each type of robot with the desired level of complexity.

As an alternative, Panda3D integrates an interface with two independent external physical models:

- Open Dynamics Engine (ODE): Developed by Russell L. Smith and presented in [20], Open Dynamics Engine (ODE) is an open-source physics engine, robust and versatile, with a long history of use in various fields such as games, robotics, virtual reality, and engineering-oriented simulations. ODE is optimized for CPU performance, especially through multithreading support.
- Bullet: Developed in its current form by [5], Bullet is a benchmark in the field of real-time physical simulation, known for its precision, adaptability, and computational efficiency. While this engine supports multithreaded execution, its overall design has been made with a view to using SIMD optimizations to leverage GPU capabilities.

III. USE CASES

The development of this simulator is framed within the NAUTILUS project (Swarms of uNderwAter aUTonomous vehIcLes gUided by artificial intelligence: ItS time has come). This project aims to develop swarms of small, low-cost autonomous vehicles responsible for managing coordinated activities, supporting research in the area, providing services such as positioning, data collection, and battery recharging for fixed or mobile nodes deployed without direct human intervention. The swarm will act as a single, decentralized system where collective information will be disseminated among the individuals. The swarm will autonomously decide where to deploy each individual and adapt its spatial coverage based on the environmental state and its needs.

As a vehicle, NAUTILUS uses the popular BlueROV2 [18] in its heavy configuration (Fig. 3). This version of the ROV features four thrusters for horizontal locomotion and four thrusters for vertical locomotion, allowing six degrees of freedom in maneuvers. The vehicle is controlled by a Raspberry Pi and integrates an inertial measurement unit (IMU), a magnetometer, and a pressure sensor on an external expansion board (the "Navigator"). The vehicle's software is distributed as open source, allowing it to work with a wide variety of hardware, such as sonar sensors, cameras, and an inertial navigation system. Moreover, although the BlueROV2 is a tethered underwater vehicle, the use of open-source code opens the possibility of extending the vehicle's control software to convert it into a UUV.

As an extension to the basic sensors configuration, each vehicle has been equipped with an echo-sounding device and a mechanical scanning sonar for navigation and image acquisition. Developing realistic models of these sensors is part of the simulator's development.

For the physical model simulation of this vehicle, two versions have been adapted and implemented, based on the mathematical simulation models of BlueROV2 dynamics developed in [23], derived from the work presented in [7].: a simplified one for the BlueROV2 basic configuration and a complete one, that also includes the blueROV2 heavy configuration.

Some of the scenarios and developments associated with this project are shown below.

A. Flocking

One of the first controllers implemented in NauSim was a PID controller based on virtual GPS positioning. A Proportional-Integral-Derivative (PID) controller is a control mechanism for dynamic feedback systems used in industrial and engineering applications. It is designed to minimize the error between the desired and actual state by adjusting the control inputs through three types of actions: the proportional component, which adjusts the output proportionally to the current error; the integral component, which takes into account the accumulation of past errors to eliminate steady-state deviations; and the derivative component, which predicts future errors based on the rate of change.

Using this PID controller, a scenario is proposed where one of the drones acts as a leader and six others as followers. The leader drone is configured to move, by means of a PID controller, along a predetermined route, defined by a series of points. In the rest of the drones, over the PID controller a modification to the classical flocking rules is implemented, in order to maintain a formation. These rules include maintaining a safe distance to avoid collisions, aligning their direction and speed with the leader and nearby drones, and staying close to

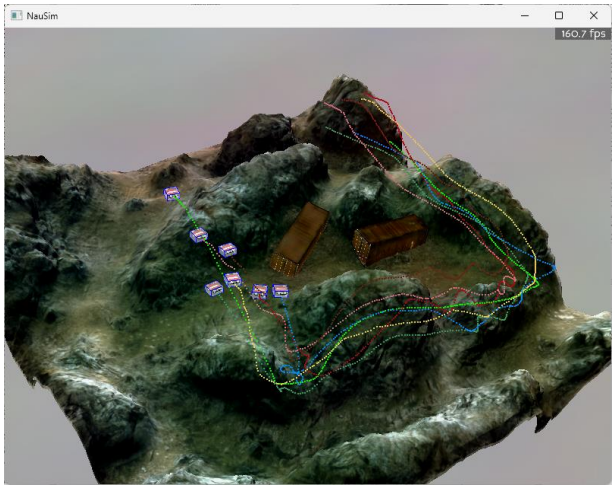

Fig.4. Flocking simulation using NauSim. The trajectories followed by each drone are shown as dotted lines. The leader's trajectory is shown in light green.

the center of the group. As the leader drone navigates the terrain, the followers dynamically adjust their positions to create a cohesive and synchronized flight pattern. Results of running this scenario can be seen in Fig. 4.

Of course, this scenario is not intended to be realistic, since its execution depends on two of the major problems faced by AUVs, communication and positioning. However, by modifying the transfer rate, noise and accuracy of the communication module and the virtual positioning sensors, it is possible to have a baseline to assess the performance needs, robustness and reliability of the system under various conditions.

B. Sonar simulation

While a basic approach is relatively simple to implement, the complexities associated with sub-acoustic acoustic phenomena such as reflections, propagation characteristics and scattering make a realistic implementation a very complex task, especially in the area of high-frequency sonar with which AUVs are often equipped. However, in an autonomous drone the sonar is one of its main windows to the world, so a realistic simulation of this contributes directly to the development of these vehicles and associated marine technologies, providing a tool with which to virtually test these sonar systems in various navigation and data acquisition scenarios.

Thus, based on the method presented in [6], a sonar model based on Screen Space Reflections (SSR) has been developed. SSR is a method commonly used in the generation of real-time 3D graphics to compute realistic reflections in the environment. SSR approximates reflections by tracing rays in screen space, rather than in the entire 3D scene, which significantly reduces the computational burden. By capturing the interactions of sound waves with objects and surfaces in the screen space, the use of SSR can effectively represent realistic reflections and refractions of sonar signals, as well as alternative paths and secondary reflections, all in real time. A comparison of the results with real sampling can be seen in Fig. 5.

C. Wall-tracking

We propose the design of a wall-tracking controller, where the output will be deployed on the target vehicle. The test scenario is a swimming pool, so we can assume flat walls and right angles. The wall tracker acts as a state machine where the vehicle first orients itself towards the first structure it encounters, approaches a predetermined distance from it and starts moving parallel to the structure, maintaining the distance. If it detects a blockage in its path, it rotates until the obstacle disappears and starts again in the initial state. If it

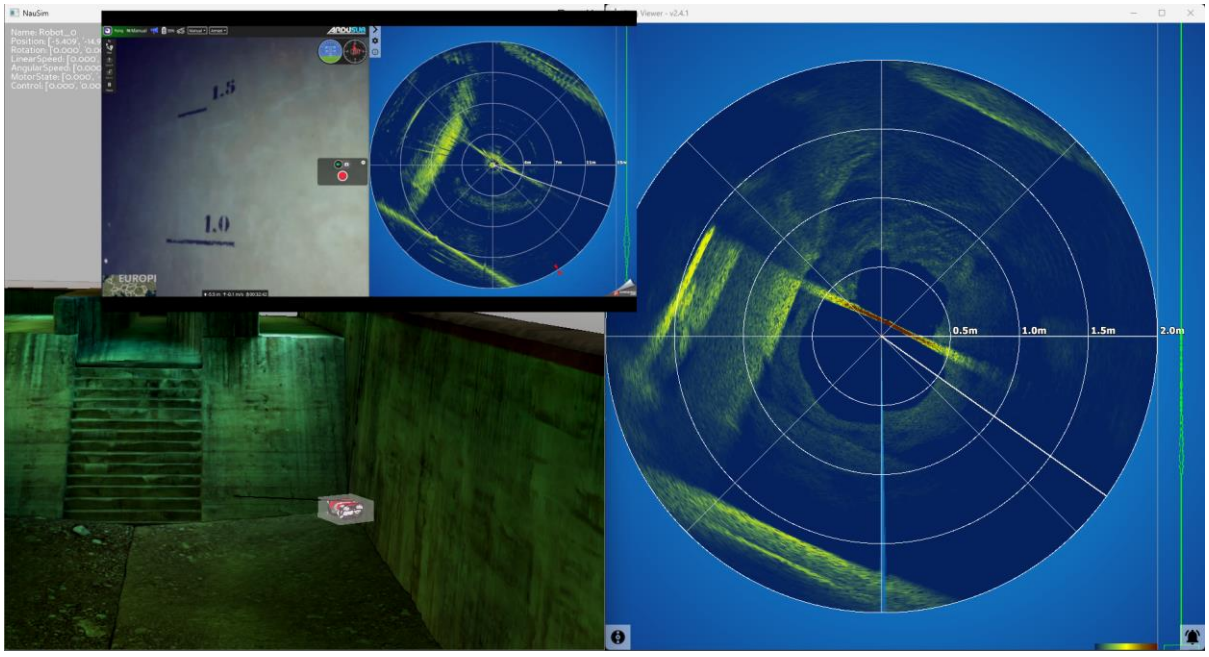

Fig. 5. Sonar simulation compared to real data. The left side shows the virtual scenario together with the real data (video and sonar overlay). On the right side the simulation results are shown. Real and simulated results are visualized using PingViewer™.

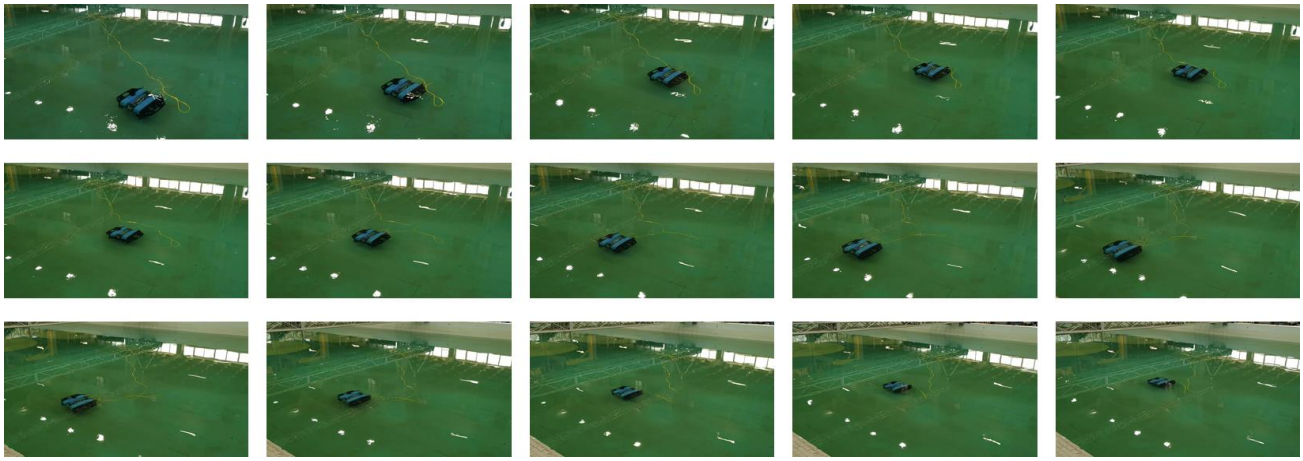

Fig. 6. Sequence of images showing the results of the wall-tracking controller in a real scenario. It can be seen how the vehicle moves parallel to the left wall of the pool, while moving back and forth to adjust its distance.

misses the wall or is unable to maintain its orientation, the vehicle starts again in the initial state.

The main constraints in this scenario are the result of the limitations of mechanical sonar, which is used to detect walls. A mechanical sonar needs to physically move the head to sample at a given angle; to cover the necessary arc of view for this controller is required so that there will only be one complete update of the world around the vehicle every 2 seconds. Taking into account the maximum speed of the drone this means more than 2 meters of distance travelled.

Moreover, the nature of sonar makes it difficult to differentiate between giving a distance to an obstacle if other sources of reflections are present, such as the ground or the air-water boundary. Having straight walls makes the task easier, but the detecting algorithm (based on finding the maximum sum subarray in the sonar signal, defined the subs arrays by a fixed window, once the noise has been cleaned.) has to be properly validated. Therefore, the sonar simulation presented in the previous section has been used in the validation of the controller, modified to take into account the delay imposed by the hardware.

In a non-simulated environment, the controller behaves (except for the differences in starting conditions) similarly to the simulated version, resulting in comparable trajectories and the same obstacle behavior. Fig. 6 shows a snapshot of a test using the objective hardware in a real environment .

IV. CONCLUSIONS

In this work, we present NauSim, a simulation environment developed with the purpose of providing researchers and students with a platform for testing, developing and verifying sensor configurations and control algorithms in underwater vehicles, with special emphasis on the use of ML-based controllers. The simulation environment provides a fast, simple and, most importantly, cost-effective alternative to pool and sea testing, facilitating faster and less costly development of underwater robotic solutions. NAUTILUS covers a wide range of possible usage scenarios, including rapid virtual prototyping, design testing, or control algorithm development, both for individual robots and large heterogeneous swarms. It should be noted that this simulation environment is still under development (currently in revision 59), with plans to extend the functionalities with new sensors, controllers and other robot models.

REFERENCES

[1] Amer, A., Álvarez-Tuñon, O., Ugurlu, H. I ̇., Sejersen, J. L. F., Brodskiy, Y.,Kayacan, E., 2023. Unav-sim: A visually realistic underwater robotics simulator and synthetic data-generation framework. In: 2023 21st International Conference on Advanced Robotics (ICAR). IEEE, pp. 570-576.

[2] Betancourt, J., Coral, W., Colorado, J., 2020. An integrated rov solution for underwater net-cage inspection in fish farms using computer vision. SN Applied Sciences 2 (12), 1946.

[3] Cerqueira, R., Trocoli, T., Neves, G., Joyeux, S., Albiez, J., Oliveira, L., 2017. A novel gpu-based sonar simulator for real-time applications. Computers & Graphics 68, 66-76.

[4] Cheng, L., Tan, X., Yao, D., Xu, W., Wu, H., Chen, Y., 2021. A fishery water quality monitoring and prediction evaluation system for floating uav based on time series. Sensors 21 (13), 4451.

[5] Coumans, E., 2015. Bullet physics simulation. In: ACM SIGGRAPH 2015 Courses. p. 1.

[6] de Cerqueira Gava, P. D., Nascimento J ́ unior, C. L., Belchior de Franc, a Silva,J. R., Adabo, G. J., 2022. Simu2vita: A general purpose underwater vehicle simulator. Sensors 22 (9), 3255.

[7] Fossen, T. I., 2011. Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons.

[8] Goslin, M., Mine, M. R., 2004. The panda3d graphics engine. Computer 37 (10), 112-114.

[9] Hu, S., Feng, A., Shi, J., Li, J., Khan, F., Zhu, H., Chen, J., Chen, G., 2022. Underwater gas leak detection using an autonomous underwater vehicle (robotic fish). Process Safety and Environmental Protection 167, 89-96.

[10] Koub ̂ aa, A., Allouch, A., Alajlan, M., Javed, Y., Belghith, A., Khalgui, M.,2019. Micro air vehicle link (mavlink) in a nutshell: A survey. IEEE Access 7, 87658-87680.

[11] Liniger, J., Jensen, A. L., Pedersen, S., Sorensen, H., Mai, C., 2022. On the autonomous inspection and classification of marine growth on subsea structures. In: OCEANS 2022-Chennai. IEEE, pp. 1-7.

[12] Macenski, S., Foote, T., Gerkey, B., Lalancette, C.,Woodall,W., 2022. Robot operating system 2: Design, architecture, and uses in the wild. Science robotics 7 (66), eabm6074

[13] Madeo, D., Pozzebon, A., Mocenni, C., Bertoni, D., 2020. A low-cost unmanned surface vehicle for pervasive water quality monitoring. IEEE Transactions on Instrumentation and Measurement 69 (4), 1433-1444.

[14] Manhaes, M. M. M., Scherer, S. A., Voss, M., Douat, L. R., Rauschenbach, T., 2016. Uuv simulator: A gazebo-based package for

underwater intervention and multi-robot simulation. In: OCEANS 2016 MTS/IEEE Monterey. Ieee, pp. 1–8.

[15] Potokar, E., Ashford, S., Kaess, M., Mangelson, J. G., 2022. Holoocean: An underwater robotics simulator. In: 2022 International Conference on Robotics and Automation (ICRA). IEEE, pp. 3040–3046.

[16] Prats, M., Perez, J., Fern´andez, J. J., Sanz, P. J., 2012. An open source tool for simulation and supervision of underwater intervention missions. In: 2012 IEEE/RSJ international conference on Intelligent Robots and Systems. IEEE, pp. 2577–2582.

[17] Raschka, S., Patterson, J., Nolet, C., 2020. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information 11 (4), 193.

[18] Robotics, B., 2016. Bluerov2: The world's most affordable high-performance rov. BlueROV2 Datasheet; Blue Robotics: Torrance, CA, USA.

[19] Rofallski, R., Tholen, C., Helmholz, P., Parnum, I., Luhmann, T., 2020. Measuring artificial reefs using a multi-camera system for unmanned underwater vehicles. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives 43 (B2), 999–1008.

[20] Smith, R., et al., 2005. Open dynamics engine.

[21] Sultonov, S., 2023. Importance of python programming language in machine learning. International Bulletin of Engineering and Technology 3 (9), 28–30.

[22] von Benzon, M., Sorensen, F. F., Uth, E., Jouffroy, J., Liniger, J., Pedersen, S., 2022. An open-source benchmark simulator: Control of a bluerov2 underwater robot. Journal of Marine Science and Engineering 10 (12), 1898.

[23] Wu, C.-J., 2018. 6-dof modelling and control of a remotely operated vehicle. Ph.D. thesis.