Corporate sensitivity
PUBLIC

Development of Adaptive Human-Machine Interaction and Multi-User Capabilities in LOTUSim for Maritime Human-Drone Teaming

Author: DUBROMEL Marie - FISE 2025 - Robotics
Naval Group Supervisor: Pr. BUCHE Cédric
Master Jury President: Pr. HARDOUIN Laurent
ENSTA Supervisor: Pr JAULIN Luc

Report content checked by: BUCHE Cédric	**Check Date**	**Signature**
	21/07/25	CB

(École Nationale Supérieure des Techniques Avancées (ENSTA))
(31/03/2025–05/09/2025)

© Naval Group Pacific. All rights reserved 2025. Both the content and the form of this document and its appendices are the property of Naval Group Pacific and/or of third parties. It is therefore prohibited to copy, duplicate, extract or transmit all or part of the document or its appendices, by any means and in any form, without the prior written consent of Naval Group Pacific. The information contained in this document and its appendices is provided for information purposes only. Naval Group Pacific shall not be held liable for any damage resulting from the use of this document or its appendices. Naval Group Pacific – SAS with capital of 562 000 000 euros – RCS Paris 418 163 895 – SIRET 418 163 895 00135 – APE 3011 Z

1 Acknowledgements

I would like to express my deepest gratitude to **Professor Cedric BUCHE** for his exceptional guidance, advice, and support throughout this internship. His mentorship has been instrumental not only in supporting my technical and professional development but also in helping me adapt and thrive in life outside of work during my time in Australia. I am truly grateful to have been part of a project I deeply believe in. Thank you for entrusting me with meaningful tasks and for always encouraging curiosity and initiative.

I am also sincerely thankful to **S.B.** for warmly welcoming me into the company, making me feel safe and confident, and for introducing me to the tools and professional practices essential to evolving in a large-scale industrial environment.

A heartfelt thanks to **J.G.** for her continuous support during the entire internship. Her technical guidance, availability, and willingness to share her experience made a significant impact on my progress and confidence.

I would also like to thank **B.N.M.** for taking the time to explain the strategic and organisational challenges of the defense sector. Her perspective helped me step back from the technical details and better appreciate the broader picture and societal implications of our work.

Many thanks to **B.D.** as well, with whom it was a pleasure to collaborate and combine efforts on our shared objectives.

Finally, I extend my deep appreciation to **Professor Luc JAULIN** for introducing me to the world of robotics, for his inspiring teaching, and for encouraging me to pursue research beyond borders. His passion for robotics continues to motivate me to delve further into this field.

©Naval Group SA property, 2025, all rights reserved.

Abstract

This report presents the development and evaluation of LOTUSim, a modular and real-time simulation platform designed to support research in multi-agent autonomous robotics and human-machine interaction. The system enables collaborative operations between multiple users and robotic agents, such as Unmanned Underwater Vehicles (UUVs), surface drones, and aerial systems, within a shared, high-fidelity 3D environment. Key contributions include the integration of physiological and cognitive user tracking tools (such as eye trackers, leap motion), enabling adaptive interfaces that respond to the operator's state in real time. Additionally, the implementation of a multi-user architecture, to facilitate distributed simulation and shared situational awareness among operators.

Comprehensive benchmarks were conducted to validate the simulator's performance across various domains, including underwater coordination, real-time AI training with accelerated time, and large-scale agent deployment. The results demonstrate LOTUSim's capacity to support both system-level evaluations and human-centered experimentation. The platform lays the groundwork for future extensions, and iterative improvements, such as energy modeling, camera integration, and broader applications in defense, environmental monitoring, and robotics research. LOTUSim represents a valuable and extensible tool for advancing the state of the art in autonomous systems and human-machine collaboration.

Keywords : Marine Robotics, Simulation, Multi-Agent System, Autonomous Agents, User Tracking Sensors, Benchmarking

Corporate sensitivity
PUBLIC

Contents

Corporate sensitivity
PUBLIC

©Naval Group SA property, 2025, all rights reserved.

Corporate sensitivity
PUBLIC

List of Figures

2 Introduction

2.1 Contextualising the Emergence of Autonomous Drones

Autonomous drones have become a cornerstone of modern technological advancement, revolutionising sectors ranging from environmental monitoring to defense, logistics, and disaster response. In marine contexts especially, autonomous systems, including surface and underwater drones, offer the potential to conduct long-duration missions in harsh environments where human access is limited or risky. As the complexity of these systems increases, so too does the need for robust, scalable, and realistic simulation tools that enable safe development and testing. The risk lies in the technological feasibility of deploying such systems, reliability, efficiency, and safety implications of autonomous operations in real-world maritime environments.

2.2 Presentation of Naval Group Pacific, and partnership the Crossing Laboratory and the LOTUSim platform

In this context, the company Naval Group Pacific is deeply committed to addressing these challenges through innovation-driven collaboration. As a subsidiary of Naval Group, Naval Group Pacific aims to become a strategic industrial partner in Australia for research and innovation. By accelerating R&D efforts, Naval Group Pacific plays a vital role in an open innovation ecosystem that promotes academic-industry synergies. A significant driver of this initiative is the CROSSING International Research Laboratory (IRL), established in February 2021. Created in partnership with the French National Centre for Scientific Research (CNRS) and leading academic institutions, CROSSING serves as a hub for upstream research in human-autonomous systems teaming and is a big partner for Naval Group Pacific. This lab not only coordinates research and technology strategies but also facilitates funding acquisition and long-term collaborative partnerships between scientific and industrial actors. During this internship, they have made available all the sensors treated in section 4 of this study.

Moreover, simulation plays a central role in marine robotics operations, offering a cost-effective and repeatable framework for developing and evaluating robotic systems operating across surface, underwater, and aerial domains. It has been a big topic and tool in studies for both Naval Group and CROSSING. Effective simulators must support human-in-the-loop interaction, empowering users to actively engage in the simulation process, a capability that proves particularly valuable for operator training and teleoperation interface development. Moreover, realistic simulation environments are essential for training artificial intelligence (AI) models, as they enable the generation of large-scale, diverse, and controllable datasets that are often difficult or impossible to gather through physical experimentation. To fulfill these goals, simulation systems must accurately model physical phenomena such as hydrodynamics, buoyancy, drag, thrust, and other domain-specific behaviors.

In this context, Naval Group Pacific has started to develop LOTUSim, Learning Operational Teaming with Unmanned Systems simulation, a real-time, cross-domain maritime simulation platform designed for human-vehicle interaction and multi-agent experimentation [11]. Developed in Unity and Gazebo, LOTUSim combines immersive user experience with realistic surface, underwater, and aerial physics. The platform supports various agent types, including unmanned underwater vehicles (UUVs), surface vessels,

Corporate sensitivity
PUBLIC

and aerial drones, enabling the design and testing of complex scenarios involving co-operative or autonomous behaviors. Thanks to its modular and scalable architecture, LOTUSim accommodates both small-scale and large-scale experiments involving up to hundreds of agents. Furthermore, its human-centered design allows operators to pilot vehicles in real-time while simultaneously capturing rich data streams for AI training, algorithm validation, or scenario testing. As such, LOTUSim serves as both an experimental playground for human-autonomous teaming and a powerful tool for advancing autonomy in marine systems.

2.3 Objectives of this study

This internship takes place within the context of developing advanced features for LOTUSim, with the aim of enhancing its multi-user and human-machine interaction capabilities (section 3). A key objective is to enable distributed simulation experiences through real-time communication systems and operator-specific perspectives. This approach supports collaborative mission scenarios where multiple users share situational awareness and coordinate decisions. Additionally, the project investigates a variety of physiological and cognitive tracking technologies, including heart rate monitors (Empatica E4), VR headsets (Meta Quest 2), hand tracking (Leap Motion), and eye-tracking systems (Tobii Eye Tracker 5 and Glasses Pro 3) to create adaptive interfaces (section 4). These sensors enable the system to adjust dynamically based on the user's state, promoting more intuitive control and enhanced situational engagement.

The internship also involves benchmarking other simulation platform (section 5) using agents such as BlueROVs and LRAUVs. This includes rigorous testing of the performance and time-accelerated AI training simulations. Furthermore, the internship contributes to the long-term development of LOTUSim by establishing proof-of-concept features, supporting the integration of PhD research contributions, and preparing tools for energy-aware simulations (section 8). Through these efforts, the work aims to advance LOTUSim toward becoming a cutting-edge platform for autonomous system development and immersive human-robot teaming experimentation, in order to be a strong tool for researchers.

This report had to be written a month and a half before the official end of the internship, as Naval Group Pacific requires a security check to be completed one month prior to the report submission date. As a result, additional work has since been carried out, including further benchmark results as well as the developments described in sections 6 and 8. These contributions will not appear in this written document but will instead be presented during the oral presentation in September.

Corporate sensitivity
PUBLIC

3 Multi User Support

Before the beginning of this work, the LOTUSim platform was offering one simulation, where several agents could be spawned and move respecting their physics, and the physic of the environment. But the user can only monitor the simulation with one computer, and cannot interact with other user in the simulation. Therefore, the first part of this internship was to develop a multi-user support, with distributed operator views and real-time communication systems for multi-user environments in order to have collaborative scenarios. These systems should then enhance coordination and decision-making among operators by providing shared situational awareness and communication capabilities.

3.1 State of the Art

To develop a real-time collaborative support in a simulation platforms such as LOTUSim, the selection of a robust and scalable multiplayer framework is critical to ensure low-latency communication, flexible user synchronisation, and seamless integration within Unity. Photon Unity Networking 2 (PUN2) [13] is a widely adopted solution that provides an optimised client-server architecture for Unity-based applications. Compared to other networking frameworks such as Unity's deprecated UNet, Mirror, or MLAPI (now Unity Netcode for GameObjects), PUN2 offers several advantages, particularly in terms of integration simplicity, cloud-hosted services, and built-in support for room-based matchmaking. Following studies found in the literature, the comparaison of performance has mostly be done between PUN2 and Mirror, as they are the two most used software. Overall, these two networking softwares are very effective and depend a lot of the needs of the developer. For the LOTUSim, as this support will be at first a proof of concept, the chosen software needs to be integrated quickly with a light structure. So even thought in a controlled evaluation conducted at FIT Turku Centre [1], where Mirror and PUN2 were directly compared under simulated VR workloads, Mirror demonstrated lower synchronisation delay and more stable frame rates in local-area network setups, particularly for small groups, PUN2 exhibited higher latency in movement updates: 80–120 ms (cloud optimised) for PUN and 40–80 ms (Local Area Network), 150 ms (Wide Area Network), this remained within acceptable bounds for many applications including LOTUSim, and it is for a VR context which won't always be the case fot LOTUSim. Complementing these results, a Toxigon 2025 industry benchmark [24] highlighted the strengths of PUN2 in deployment speed and global scalability. The cloud-hosted architecture of PUN2 ensures consistent round-trip latencies under 100 ms between regions, thanks to automatic selection of optimal server nodes when Mirror can be a bit more complex to setup.
Considering the goals of this internship to create a first version of multi-user support in LOTUSim, the choice of Photon Unity Networking 2 (PUN2) was based on practical needs. Although Mirror shows slightly better performance in local networks, PUN2 offers much faster and easier integration into Unity, along with built-in matchmaking and cloud-based hosting. These features made it easier to quickly build a working prototype. Its support for multiple platforms, user-friendly tools, and reliable global servers provide a solid base for enabling collaborative use of the LOTUSim platform, where real-time communication and shared awareness between users are key.

Corporate sensitivity
PUBLIC

3.2 Overall Functioning

The integration of a multi-user support has been done in different steps, and the first one consisted of creating a multi-user game/environment from a basic Unity scene, with few elements, to develop the correct code, and then integrate it into a more complex scene and scenario of LOTUSim. At the end of this phase, and after reading the documentation of the PUN2 plugin [7], the code developed was integrated into a larger scene and the Unity project part of the LOTUSim project called `SilentStorm` (Fig 1). This scene contains various elements and agents such as :

- An Island
- PHA (porte-hélicoptères amphibies)
- FREMM (frégates multimissions)
- LRAUV
- BlueROV
- Mines
- Drones X500
- WAMV (Wave Adaptive Modular Vehicle)

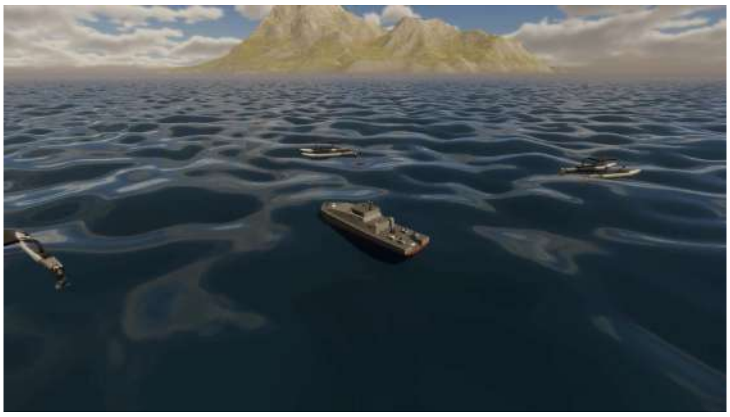


Figure 1: Illustration of the Silent Storm scene

All these entities are scattered into three different domains: underwater, on the surface of the sea, and in the air.

At this point, the project supports for both **players** (controlling robots) (paragraph3.3) and **spectators** (observing freely) (paragraph 3.40. Each script plays a key role in establishing connectivity, spawning users, and enabling network-synchronised interaction.

Corporate sensitivity
PUBLIC

3.2.1 Launcher and Connection Setup

The simulation begins in the **Launcher scene**, where users:

- Enter their name (`PlayerNameInputField.cs`)
- Choose a role: player or spectator (through a Toggle object in Unity to tick ir not)
- Specify ROS IP/port (stored via `PlayerPrefs`) : By default, the `ROS_IP = 127.0.0.1` (local ip adress) and the `ROS_PORT = 10000`
- Launch the simulation with feedback from the loading animation (`LoaderAnime.cs`)

The `Launcher.cs` script then connects to Photon and loads the main scene (`SilentStorm`) using `PhotonNetwork.LoadLevel`.

Key Parameters:

- `LoaderAnime.cs`
  - `speed` (float) — Angular speed in degrees per second.
  - `radius` (float) — Radius of the rotating particle effect.
  - `particles` (GameObject) — Reference to particle system.
- `PlayerNameInputField.cs`
  - `playerNamePrefKey` (string) — Key used for saving the player name in `PlayerPrefs` (Fig 2).

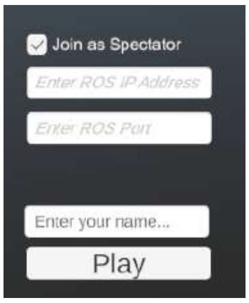


Figure 2: Illustration of the new lobby created to start a simulation

---

3.2.2 ROS Integration

The `ROSConnectionConfigurator.cs` reads the stored ROS parameters and dynamically sets up a `ROSConnection` component to communicate with the ROS2 backend. This enables publishing/subscribing to topics like `/it/position` during runtime.

Key Parameters:

- (Typically parameters like ROS IP address, port, and topic names are configurable here.)

This part have been implemented in order to merge the multi-user support with the rest of the LOTUSim.

Corporate sensitivity
PUBLIC

3.2.3 Game Management and Player Instantiation

Once inside the room:

- `GameManager.cs` checks whether the user is a player or spectator.
- Players are spawned using `PhotonNetwork.Instantiate` with their robot avatar.
- Spectators activate the `SpectatorCamera` only.

The `SetupCamera()` method ensures only the correct camera (robot-attached or free-flying) is active and tagged as `"MainCamera"`.

Key Parameters:

- `GameManager.cs`
  - `playerPrefab` (GameObject) — Prefab for the player robot (Fig 3).
  - `spectatorCameraRobotPrefab` (GameObject) — Prefab for the spectator camera.

Figure 3: Kyle Robot Fbx Asset

3.2.4 Player Control and Networking

Each player:

- Is instantiated with `PlayerManager.cs`, which manages input, health, beam interactions, and scene transitions.
- Uses `PlayerAnimatorManager.cs` to trigger animations like walking and jumping in sync with movement inputs.
- Controls a beam weapon (Fig 4) via input (`Fire1`) and gets damaged when intersecting with other players' beams.

Networking state (like `IsFiring` and `Health`) is synchronized via the `IPunObservable` interface.

Key Parameters:

- `PlayerManager.cs`
  - `Health` (float) — Player health.

Corporate sensitivity
PUBLIC

   - `playerUiPrefab` (GameObject) — Player UI prefab for health and name display (Fig 5).
   - `beams` (GameObject) — Beam weapon GameObject.

- `PlayerAnimatorManager.cs`
   - `directionDampTime` (float) — Damp time for directional input smoothing.

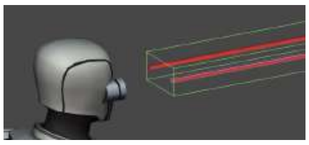


Figure 4: Illustration of the Robot Kyle Firing

3.2.5 User Interface and Visual Feedback

Each player is followed by a UI canvas:

- `PlayerUI.cs` instantiates a nameplate and health bar.
- The UI follows the robot's position and becomes transparent if the player isn't visible.
- Player names come from the Photon nickname set in the launcher.

Key Parameters:

- `PlayerUI.cs`
   - `screenOffset` (Vector3) — Pixel offset of UI above player.
   - `playerNameText` (TMP_Text) — Text component for player name.
   - `playerHealthSlider` (Slider) — Slider component for health.

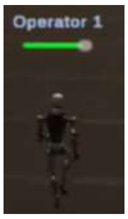


Figure 5: Illustration of the Robot with name and health bar

Corporate sensitivity
PUBLIC

3.2.6 Spectator Camera

Spectators are not associated with any robot:

- Instead, `SpectatorCamera.cs` gives them full 6DOF navigation using keyboard (WASD + QE) and mouse look.
- The camera has acceleration with `Shift`, customizable speed, and clamped pitch for stability.
- This role is useful for supervisors or passive observers.

Key Parameters:

- `SpectatorCamera.cs`
  - `baseMoveSpeed` (float) — Base movement speed.
  - `lookSpeed` (float) — Mouse look sensitivity.
  - `accelerationRate` (float) — Shift key acceleration rate.
  - `maxShiftMultiplier` (float) — Maximum speed multiplier when accelerating.

The system is designed for **modularity**, **role separation**, and **network synchronisation**. Indeed, for the LOTUSim Simulator, this support is a feature which can be enabled for different purpose, and more sub features could be added. Here some example of small features have been added, such as the health bar, the beams, to show that it is possible. As this simulator should be a tool for researchers, they will be able to keep developing certain part of the simulation to serve their work. But whether controlling a robot or spectating, each user connects to a unified simulation with ROS-ROS2 integration and consistent visual/audio state. In the end, the architecture is scalable and can be extended for additional robot types, control interfaces, or collaborative tasks.

3.3 Player Mode

Now, as described in the previous section, two different modes are available in this multi-user support. The first one, the player mode, allows a user to take direct control over a simulated robot, enabling hands-on training in mission scenarios that reflect real-life challenges. This mode is particularly valuable in high-risk domains such as defense operations or environmental interventions, for instance during the maintenance of offshore wind turbines, where quick decision-making and operational accuracy can be critical. The player is spawned using a prefab (`playerPrefab`) and is controlled via inputs such as movement and action triggers. Camera control is handled by the `CameraWork.cs` script, which follows the player during the simulation. With this mode, several players can interact in and with their environment, but also with other players (Fig 6), connected to the same simulation.

Corporate sensitivity
PUBLIC

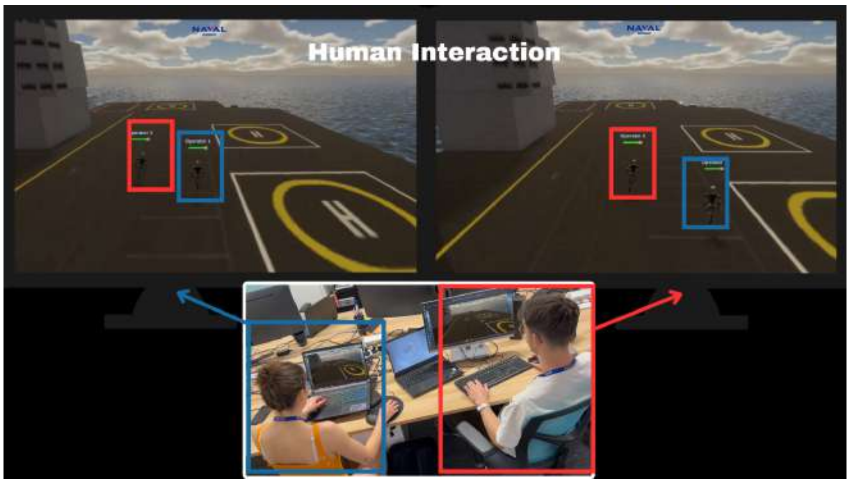


Figure 6: Illustration of the multi-user support with two players

3.4 Spectator Mode

The second mode, called the spectator mode can be used with or without operators connected to the simulation through the player mode (Fig 7). It offers a flexible and non-intrusive way to observe missions from various perspectives, providing trainers, supervisors, or engineers with situational awareness without directly participating in the simulation. This is essential for mission debriefing, team coordination, and scenario validation, especially in collaborative or educational settings. Therefore, the spectator camera (`SpectatorCamera.cs`) can be freely moved and rotated using configurable parameters such as `baseMoveSpeed`, `lookSpeed`, and acceleration settings (`accelerationRate`, `maxShiftMultiplier`). When enabled, this mode disables player instantiation and instead activates an orbiting or free-flying camera that provides a holistic view of the simulated environment.

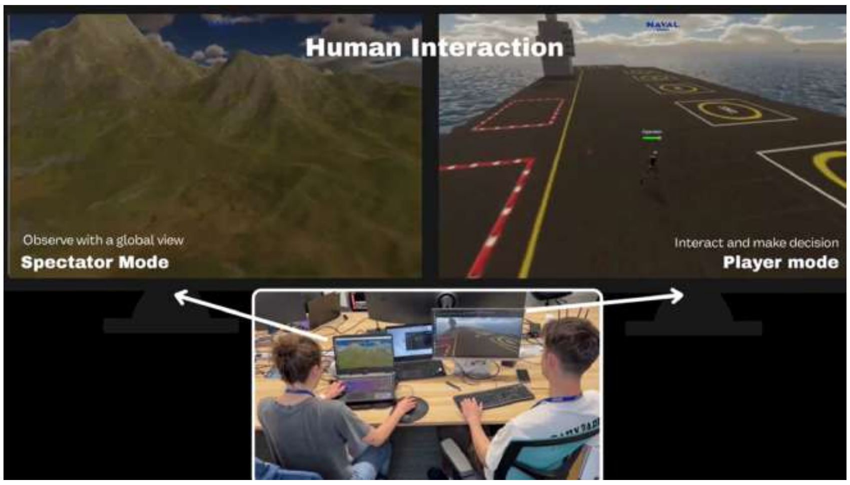


Figure 7: Illustration of the multi-user support with a spectator and a player

4 Human-Machine Tracking and interaction

As the LOTUSim is a real-time maritime simulation platform for human-drone teaming, which would be a base for very diverse research works, one of the goal is to make sure the operators benefits from an immersive interface to experiment the simulation platform and to train themselves with new kinds of human-autonomous agents collaborative scenarios. In this section, multi-sensor systems (such as eye trackers, heart rate monitors, cameras) to capture real-time data for analysing human factors have been integrated. This data will serve as the foundation for developing adaptive interfaces that dynamically respond to the operator's physiological and cognitive state. Additionally, offer a range of interaction methods (like eye control, gesture recognition...) to enhance situational awareness and facilitate more intuitive navigation of the tactical situation.

An important thing to note is that the sensors which could be possibly integrated in the LOTUSim are only the ones available wether through the CROSSING Lab, or directly via Naval Group Pacific. This explains why some state of art have a restrictive list of sensors. As all these implementations are proof of concepts for research purposes, it is not necessarily in the company's best interest to spend a lot of money for the best and latest sensors.

4.1 Empatica Watch

As the CROSSING Lab is one of Naval Group Pacific's partners, the Empatica E4 watch (Fig 8) and its sensors were made available for potential integration into the simulator. Integrating the Empatica E4 watch into the LOTUSim platform could have brought valuable insights into human physiological and cognitive states during simulation. The E4 is a medical grade wearable device equipped with multiple sensors, including electrodermal activity (EDA), heart rate, skin temperature, and accelerometer data, all of which can be exported in CSV format for further analysis. These real-time physiological signals can help detect stress, cognitive workload, or fatigue levels, providing a deeper understanding of operator states during human-drone collaboration scenarios or for research work about human fatigue in simulations. Such data would be essential for building adaptive interfaces that respond to the current condition of the user, enhancing immersion, decision making, and overall performance.

Figure 8: Picture of the Empatica E4 watch

However, as stated in the official notice from Empatica [10], the E4 software suite, including E4 Connect, E4 Manager, E4 Realtime App, E4 Streaming Server, and E4 SDK, has been officially discontinued and can no longer collect or store data, rendering the provided services inoperative. And unfortunately, without the aforementioned software, the data cannot be extracted from the watch, and therefore not analysed. The only

Corporate sensitivity
PUBLIC

solution would be to upgrade to their latest model, which was not the priority of the company at this stage of development of the platform.

4.2 Immersive environment : VR headset

For the implementation of this sensor, the design considerations were driven by the necessity to incorporate both a VR headset and a Leap Motion controller (paragraph 4.3.2) to enable hand-tracking capabilities within the immersive environment

4.2.1 State of the Art: Comparison of Head-Mounted Displays for VR-Enhanced Simulation

In the context of immersive simulation environments such as LOTUSim, the integration of Virtual Reality (VR) headsets plays a crucial role in enhancing user presence, natural interaction, and spatial awareness. Among the most relevant Head-Mounted Displays for this purpose and available for Naval Group Pacific, are the **Oculus Rift S**, **Meta Quest 2**, and **Microsoft HoloLens 2**. Each device presents a different trade-off between immersion, tracking capabilities, computational independence, and compatibility with hand-tracking peripherals such as the Leap Motion Controller.

The **Oculus Rift S** (Fig 9), launched by Oculus in 2019, offers high-quality inside-out tracking with five onboard cameras and a refresh rate of 80 Hz. It requires a tethered connection to a PC, allowing access to high-performance rendering and simulation engines. While the Rift S has been discontinued in favor of newer devices, its support for Unity, low latency, and compatibility with the Leap Motion (via USB integration) make it a technically robust but legacy choice for desktop-based simulation platforms. However, the lack of native support and manufacturer discontinuation raises concerns regarding long-term viability in research and development.

In contrast, the **Meta Quest 2** (Fig 10) represents a more recent VR solution, supporting both untethered operation and Oculus Link for PC-based simulation. With a resolution of 1832$\times$1920 per eye and a 90-120 Hz refresh rate, it delivers higher visual fidelity compared to the Rift S. Its inside-out tracking system and growing developer ecosystem make it suitable for both local and remote training applications. Importantly, its support for Unity XR plugins and OpenXR ensures straightforward integration with simulation software like LOTUSim. Although Leap Motion support on the Quest 2 requires custom integration (since it lacks USB-A ports), recent research demonstrates successful attachment via external mounts and USB-C adapters, enabling hand-tracking in VR environments. Or another possible implementation would be to directly use the hand controllers of the head set.

The **Microsoft HoloLens 2** (Fig 11), meanwhile, differs fundamentally as it implements a mixed reality (MR) paradigm rather than full VR. It uses spatial mapping and see-through holographic displays, allowing users to interact with virtual overlays while maintaining perception of the physical environment. Although it features robust hand tracking and eye tracking without external peripherals, its limited field of view (approximately 52° diagonal) and relatively low holographic resolution make it less suited for fully immersive training simulations. Moreover, HoloLens 2 is optimised for industry-specific AR workflows rather than fully immersive VR scenes, which are central to LOTUSim's objective of simulating high-stakes scenarios like offshore wind turbine maintenance or defense training.

Corporate sensitivity
PUBLIC

Figure 9: Illustration of the Occulus Rift S headset

Figure 10: Illustration the Meta Quest 2 headset

Figure 11: Illustration of the Microsoft HoloLens 2

Therefore, for implementing VR atop the LOTUSim simulator while maintaining compatibility with the Leap Motion Controller and achieving a high level of immersion, the **Meta Quest 2** offers the most balanced solution. Its performance, developer support, and hybrid tethering make it suitable for both research and deployment. The Rift S, while still viable, is becoming obsolete, and the HoloLens 2, although technologically impressive, aligns more closely with augmented rather than immersive virtual training environments.

4.2.2 Integration : Meta Quest 2

A Unity simulation was developed with a basic environment in which the Meta Quest 2 headset was successfully integrated. However, during the attempt to integrate the headset into the existing LOTUSim simulator, the process failed due to compatibility issues with the High Definition Render Pipeline (HDRP) package. Indeed, LOTUSim relies on a highly realistic visual environment, including detailed waves, sky, and clouds, which results in graphics quality that exceeds what the Meta Quest 2 can render effectively. Displaying such content in the headset would require a significant reconfiguration of the HDRP rendering pipeline, which was outside the scope of this task. Therefore, it was decided to postpone this integration effort and proceed with the next planned development phase.

4.3 Leap Motion : Natural Interaction

To enhance the natural interaction within the simulator's spectator mode, a Leap Motion hand tracking (Fig 12) was integrated to enable operators to navigate without relying on traditional keyboard inputs. This approach allows operators to move freely by simply gesturing with their hands in physical space, which is particularly useful for training scenarios in confined or constrained environments such as command boats or maintenance rooms. This integration is a proof of concept, and the device was lent again by the French Australian CROSSING Lab.

4.3.1 Leap Motion Working Principle and Justification

The Leap Motion Controller (LMC) operates using **stereo infrared cameras** in combination with **projected infrared (IR) light** to create a high-resolution, low-latency 3D interaction space extending several tens of centimeters above the device [4]. Specifically,

Corporate sensitivity
PUBLIC

Figure 12: Picture of the Leap Motion

the system captures images at high frame rates (up to 120 Hz) using two 640×240 IR cameras spaced approximately 40 mm apart. The depth estimation is achieved through stereo-vision algorithms supported by LED illumination in the near-infrared spectrum (around 850 nm) according to Daniel Bachmann, Frank Weichert, and Gerhard Rinkenauer's paper [4]. This hardware configuration enables the detection and tracking of up to 27 hand elements, including finger joints, palm position, orientation, and velocity (Fig 13), which is more than enough for the LOTUSim and will guarantee precise gesture recognition and responsive control.

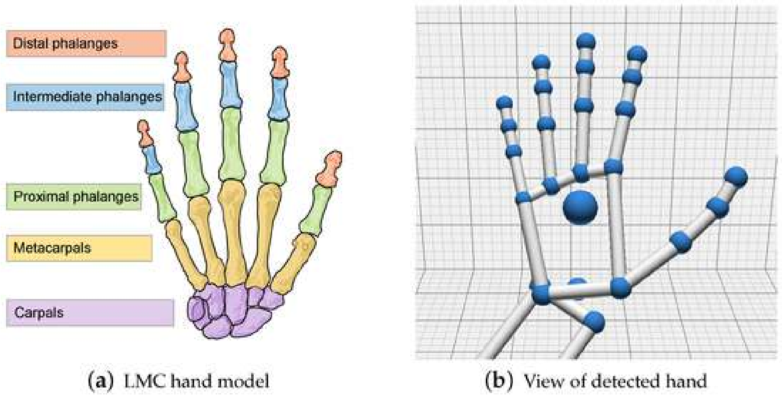


Figure 13: The Leap Motion Controller (LMC) hand model provides access to the positions of individual bones in the tracked hand. The system tracks the metacarpal, proximal phalanx, intermediate phalanx, and distal phalanx for each finger (the thumb is modeled with zero-length metacarpal). (a) Hand model based on the original diagram by Marianna Villareal[1], used by the LMC SDK; (b) view of a detected hand.

According to Weichert [4], the Leap Motion Controller offers **sub-millimeter accuracy** in static conditions and approximately **1–1.2 mm error** in dynamic scenarios. While the interaction space is limited—commonly ranging from 80 mm to 300 mm above the sensor within a $150° \times 120°$ field of view—this is sufficient for natural, contactless hand-gesture control.

Integrating this technology into the simulator enhances the **spectator mode navigation** by providing a **hands-free and intuitive control interface**. This allows operators to navigate complex environments, such as ship bridges or wind turbine nacelles, simply through hand gestures, without needing physical input devices.

As a result, this interaction method is highly appropriate for LOTUSim, espacially thanks to its precision and low latency, perfect to control the free-fly camera in real time.

[1]https://commons.wikimedia.org/wiki/File:Scheme_human_hand_bones-en.svg

Corporate sensitivity
PUBLIC

4.3.2 Leap Motion Integration for Spectator Mode Navigation

The implementation follows the Ultraleap Unity integration guidelines [25], including adding Unity's XR Origin to the scene for proper camera positioning relative to the XR device (in case the LOTUSim would one day have a VR interface), and configuring the project's scripting backend and architecture to ensure compatibility with Leap Motion hardware. Using the Leap Motion SDK, several `PoseDetector` game objects were created to recognise specific hand poses corresponding to movement directions: forward, back, left, right, up, and down.

What can also be noted is that the Leap Motion can be head mounted (usefull for VR) and desk mounted. For this project, the Leap is set on the desk, the cable pointing to the left so that the user is completely free and doesn't have to wear any device.

The core of the movement logic is implemented in a `LeapMotionMovement` script, which reads the detected poses each frame and translates them into directional movement vectors. Specifically, for each detected pose p_i, the corresponding movement vector $\mathbf{v}_i$ is added to a cumulative movement vector $\mathbf{v}$:

$$\mathbf{v} = \sum_i \mathbf{v}_i \cdot 1_{p_i} \tag{1}$$

where 1_{p_i} is an indicator function that equals 1 if the pose p_i is currently detected, and 0 otherwise.

For example:

- The "move forward" pose adds the camera's forward vector $\mathbf{f}$.
- The "move left" pose subtracts the camera's right vector $\mathbf{r}$.
- Vertical movement uses the world up and down vectors $\mathbf{u}$ and $-\mathbf{u}$.

The resulting vector $\mathbf{v}$ is then scaled by a speed factor s and the frame time Δt, producing the displacement $\Delta\mathbf{x}$:

$$\Delta\mathbf{x} = s \times \mathbf{v} \times \Delta t \tag{2}$$

This displacement is applied via Unity's `CharacterController.Move()` method, allowing smooth and physics-aware movement of the spectator camera.

To ensure reliable pose detection and control responsiveness, hand positions were calibrated using the Leap Motion Pose Recorder, storing each pose (Fig 14) in dedicated `HandPoses` assets. This enables accurate detection of the operator's intended movements while minimising unintended gestures triggering the camera. Additionally, the movement speed is adjustable in the `LeapMotionMovementController` object within the scene to accommodate operator preferences and training requirements.

Overall, this implementation provides a natural and intuitive interface for simulator navigation (Fig 15), which can significantly improve operator immersion and training efficiency in real-life critical mission scenarios, and it works for for Windows and Linux.

4.4 Eye Tracking

Another way to enhance the simulator and could help measuring the quality of the experience of the user in the simulator would be to be able to track the gaze of the user.

Corporate sensitivity
PUBLIC

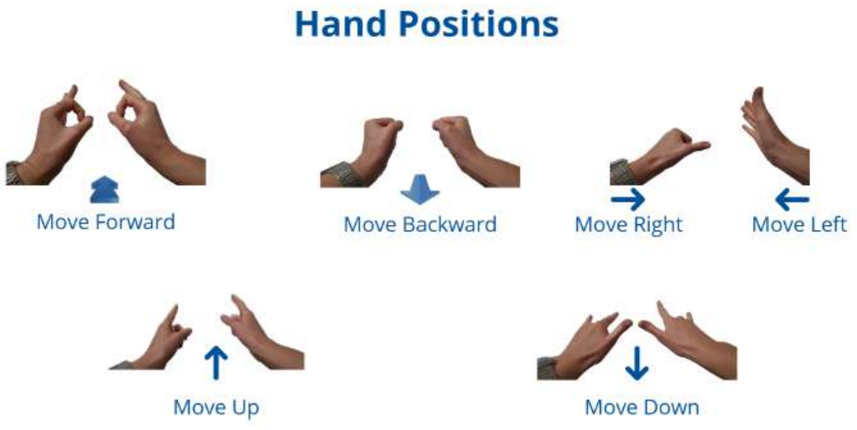


Figure 14: Different hand poses implemented for the LOTUSim

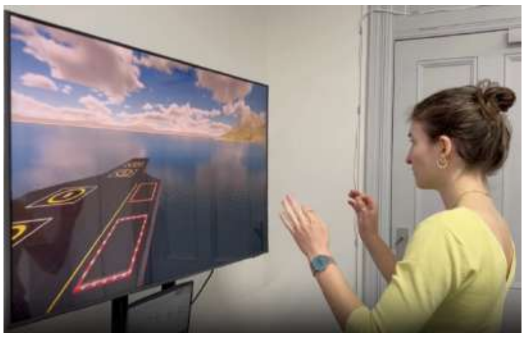


Figure 15: Picture of a user using the Leap Motion integrated in LOTUSim

In fact, as a research tool, the LOTUSim could be used in the medical field, where some researchers could be interested in knowing the stress, fatigue or interest of the user while using the simulator. Furthermore, such a feature could enable safety protocols where the simulator detects when the user is too stressed or fatigued to continue making critical decisions. In such cases, the system could trigger a replacement process, an important safeguard that could potentially save lives during high-risk missions, such as those simulated in rescue operations.

The following paragraph describes the implementation of two different devices. The first one is the Tobii Glasses, owned by the Crossing Lab, which were the sensors initially requested for integration into the simulator as part of this project.

4.4.1 Tobii Glasses Pro 3

The Tobii Pro Glasses 3 (Fig 16) are wearable eye trackers designed to capture gaze data in dynamic, real-world scenarios [20]. In this research, they are employed to determine and visualise where a user is looking within a Unity-based simulation environment. The glasses support a sampling frequency of up to 100 Hz and offer wide-angle scene coverage while maintaining comfort and natural behavior like Thibeault explain in his paper [19]. For data collection and analysis, the optional Tobii Pro Lab software is available. It provides a comprehensive research platform for experiment design, gaze data recording, synchronisation with biometric sensors, and precise timing accuracy. Alternatively, third-party software developed with the Tobii Pro SDK [22] researchers could be used, allowing

Corporate sensitivity
PUBLIC

full access to the raw gaze data stream and integration with custom Unity applications. But in this project, the goal is to extract gaze coordinates in real time and overlay them within the Unity-based LOTUSim simulation. To achieve this, the Tobii Pro Glasses 3 Web API Interface was deemed more suitable, as it provides access to the live scene view, real-time eye images, and built-in tools to test API functionality. Additionally, the Web API facilitates data transmission over the network, for an easier integration of gaze data into the Unity environment.

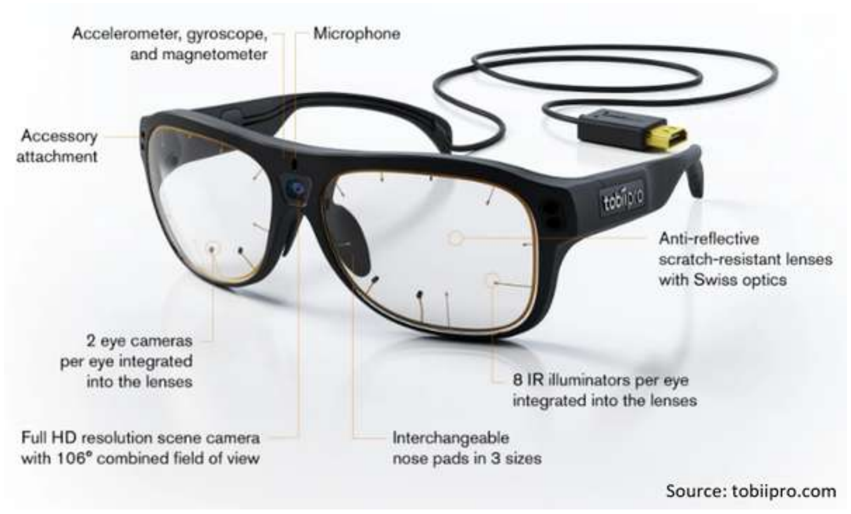

Accelerometer, gyroscope, and magnetometer
Microphone
Accessory attachment
Anti-reflective scratch-resistant lenses with Swiss optics
2 eye cameras per eye integrated into the lenses
8 IR illuminators per eye integrated into the lenses
Full HD resolution scene camera with 106° combined field of view
Interchangeable nose pads in 3 sizes
Source: tobiipro.com

Figure 16: Picture of the Tobbi Pro Glasses 3

Step 1: Real-Time Gaze Data Acquisition Using the Tobii Pro Glasses 3 Web API

In order to visualise the user's point of gaze in real time within the Unity, which is used to render the LOTUSim simulator, the Tobii Pro Glasses 3 were interfaced through their dedicated Web API. The device was connected to the computer via an Ethernet cable, enabling stable and low-latency communication. The API Web Interface was accessed through a local network address (`http://tg03b-080201135331.local`), which allowed full access to live eye images, system diagnostics, and streaming endpoints.
Among the various available options for communication, the WebSocket protocol was selected over HTTP due to its capability for persistent, bidirectional data transmission, an essential feature for streaming eye-tracking data at a frame rate of approximately 30 Hz.

On the Unity side, the integration was implemented using the `NativeWebSocket` library[22]. A custom C# script, `TobiiG3NativeWebSocketClient`, was developed to handle connection, authentication, and continuous polling of gaze samples via WebSocket requests. Each gaze sample message was structured as a JSON object containing `gaze2d` (normalised coordinates on the viewport) and `gaze3d` (a 3D vector indicating gaze direction), along with detailed information for each eye such as gaze origin, direction vectors, and pupil diameter.

Two types of messages can be used: one for the **Inertial Measurement Unit (IMU)** and another for **gaze tracking data** which will be the one needed for the eye tracking.

Corporate sensitivity
PUBLIC

But before staring the simulation, a calibration needs to be performed. To do so, the Web API can be used, with a special **calibration card** (Fig 17). Then the glasses can be launched with a play button in the Web API, and after that, they are ready to go.

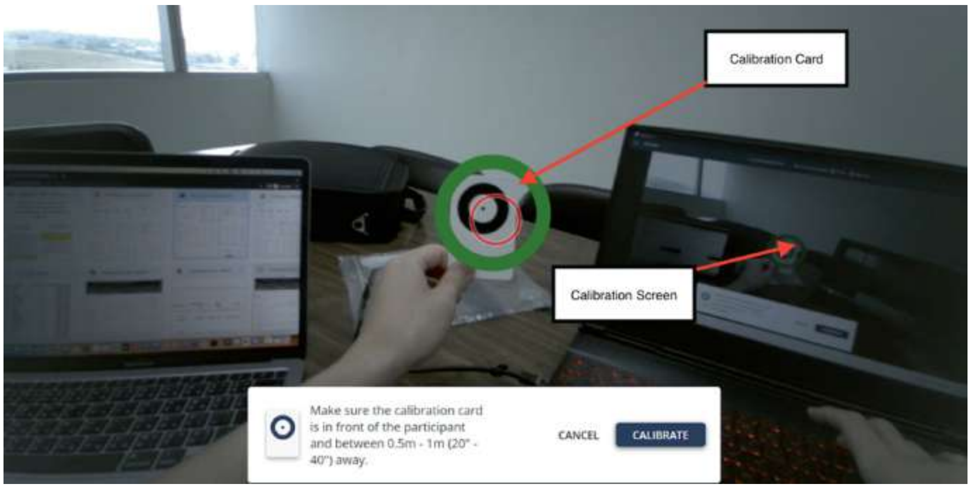

Calibration Card
Calibration Screen
Make sure the calibration card is in front of the participant and between 0.5m - 1m (20" - 40") away.
CANCEL
CALIBRATE

Figure 17: Illustration of the calibration process for the Tobii Pro Glasses 3

- IMU Sample Request (just for further research)
To access the latest IMU sample, the following message is sent:

```
string msg = "{ \"path\":\"rudimentary.imu-sample\", \"id\":2, \"method\":\
   "GET\" }";
```

Listing 1: IMU Sample Request

The corresponding response includes a timestamp and two 3D vectors:

- `accelerometer`: linear acceleration in m/s^2.
- `gyroscope`: angular velocity in °/s.

```
{
  "id": 2,
  "body": {
    "timestamp": 37.580546,
    "data": {
      "accelerometer": [1.24, 9.70, -0.52],
      "gyroscope": [2.01, -1.53, 1.59]
    }
  }
}
```

Listing 2: IMU Response Sample

- Gaze Data Sample Request
To obtain the latest gaze sample, the following command is used:

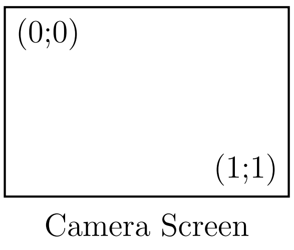
```
string msg = "{ \"path\":\"rudimentary.gaze-sample\", \"id\":2, \"method\":
   \"GET\" }";
```

Listing 3: Gaze Sample Request

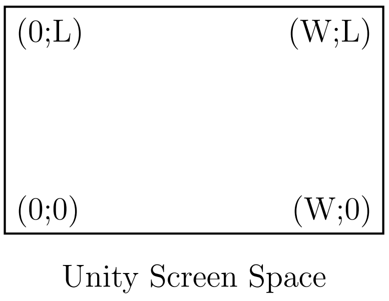
Corporate sensitivity
PUBLIC

The response includes normalised 2D and 3D gaze data (Fig 18), as well as eye-specific origin, direction, and pupil diameter:

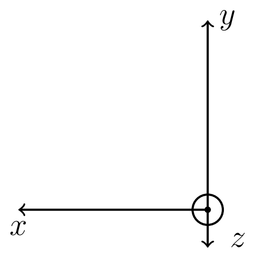
- `gaze2d`: normalised 2D coordinates on the scene camera (range: 0 to 1).
- `gaze3d`: 3D gaze vector in millimeters.



(a) Normalised screen coordinates (gaze2d)



(b) Unity screen coordinates (pixels)



(c) Tobii 3D coordinates (gaze3d)

Figure 18: Comparison between Normalised (gaze2d), Unity (pixel), and Tobii 3D (gaze3d) coordinate systems

- `eyeleft` and `eyeright`: each includes:
  - `gazeorigin` (mm)
  - `gazedirection` (unit vector)
  - `pupildiameter` (mm)

```
{
  "id": 2,
  "body": {
    "timestamp": 5.652236,
    "data": {
      "gaze2d": [0.4398, 0.6069],
      "gaze3d": [60.46, -53.80, 510.17],
      "eyeleft": {
        "gazeorigin": [33.05, -9.23, -27.56],
        "gazedirection": [0.05, -0.10, 0.99],
        "pupildiameter": 3.15
      },
      "eyeright": {
        "gazeorigin": [-30.37, -8.07, -26.80],
        "gazedirection": [0.16, -0.05, 0.98],
        "pupildiameter": 3.16
      }
    }
  }
}
```

Listing 4: Gaze Response Sample

These messages and formats were identified using the Tobii Web API interface. To render the gaze location in Unity, a `Canvas` was configured in `Screen Space - Overlay`

Corporate sensitivity
PUBLIC

mode with a dedicated `RectTransform` for the gaze pointer. The normalised gaze coordinates received from the WebSocket stream were converted to screen space (like in Fig 18) to be finally displayed in a Unity scene as a red ring, representing the user's gaze.

The conversion from normalised gaze coordinates to Unity screen coordinates is done by flipping the y-axis and scaling by the screen resolution:

$$\begin{aligned} x_{\text{unity}} &= x_{\text{norm}} \cdot \text{Screen\_width} \\ y_{\text{unity}} &= (1 - y_{\text{norm}}) \cdot \text{Screen\_height} \end{aligned} \tag{3}$$

The overall setup, when combined with calibration and synchronisation procedures detailed in the developer documentation[21], ensured a responsive and intuitive user feedback loop during immersive simulation tasks.were then integrated into the Unity-based **LOTUSim** simulation to visually overlay gaze data in real time.

Step 2: Least Square Resolution

Now at this point of the work, in a setup with a screen resolution of 1920×899, the top-right corner should ideally correspond to $(1920, 899)$ in screen space, but the Tobii Glasses reported $(1313.86, 774.67)$, and therefore the red rings obtained when the four are corners of the Unity screen/computer screen are being pointed by the user, are these Fig19:

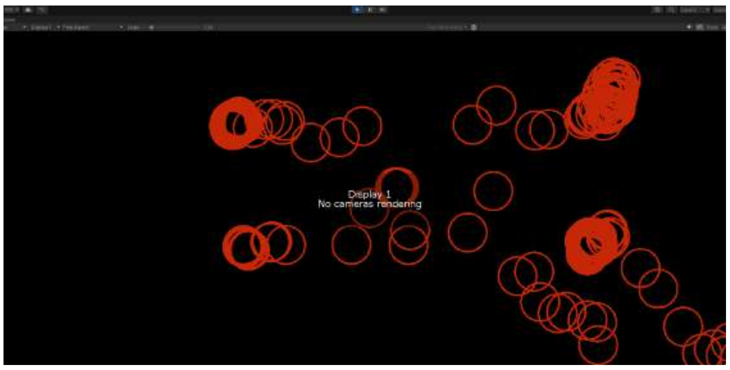


Figure 19: Illustration the red rings representing the user's gaze in Unity

The observed issue is that gaze2d are the coordinates of the user's gaze's position through the camera of the glasses, therefore, despite the conversion from normalised gaze coordinates to Unity screen coordinates with Equation (3), the gaze points did not align with the corners of the Unity window as expected. This discrepancy was consistent across all corners and varied between simulation runs. This is because when the user look at the screen of its computer, the screen doesn't take its entire field of view like the final rendering would in Unity. Therefore, even after being converted to the screen resolution, the gaze2d coordinates still won't match the corner of the computer's screen, but the real positions of each corner in the glasses' camera view. This idea is illustrated in Fig 20.

To address this, a linear least-squares regression was applied to estimate the affine transformation between the gaze view coordinates of the screen's corners and the real screen coordinates.

We chose a data-driven approach, by collecting seven sets of empirical gaze data where the user was asked to fix precisely each corner of the Unity window: the top-left, top-right, bottom-left, and bottom-right corners of the Unity display. For each of these, the

Corporate sensitivity
PUBLIC

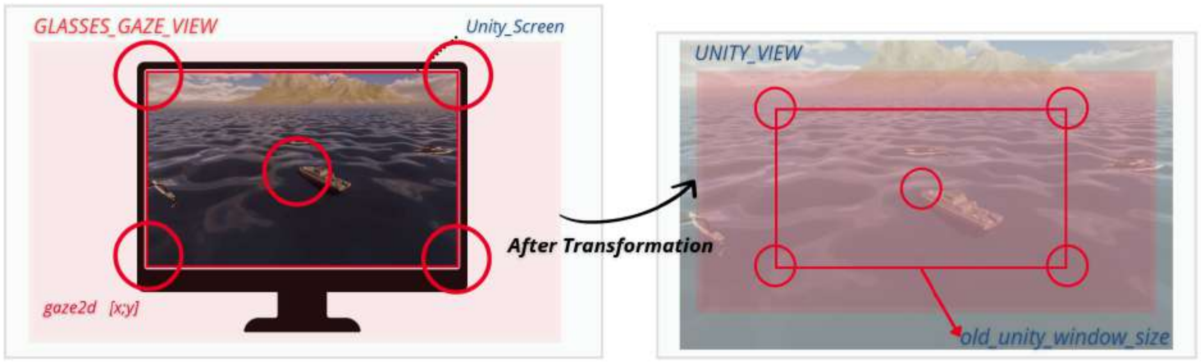


Figure 20: Illustration of the conversion issue between gaze2d and the rendering of the red ring (user's gaze trace in Unity in LOTUSim)

corresponding experimental screen coordinates (in pixels) were recorded from the gaze projection, denoted as:

$$\begin{aligned} r_{\text{top\_left}} &= (x_{\text{tl}}, y_{\text{tl}}), \\ r_{\text{top\_right}} &= (x_{\text{tr}}, y_{\text{tr}}), \\ r_{\text{bottom\_left}} &= (x_{\text{bl}}, y_{\text{bl}}), \\ r_{\text{bottom\_right}} &= (x_{\text{br}}, y_{\text{br}}), \end{aligned}$$

Having these data, the transformation model to solve has the two following system **System 1 (for horizontal mapping — screen X)**

$$\begin{cases} a \cdot x_{\text{tr}} + b &= 1920 \\ a \cdot x_{\text{tl}} + b &= 0 \\ a \cdot x_{\text{br}} + b &= 1920 \\ a \cdot x_{\text{bl}} + b &= 0 \end{cases} \tag{4}$$

Let's rewrite this as a matrix system:

$$\begin{bmatrix} x_{\text{tr}} & 1 \\ x_{\text{tl}} & 1 \\ x_{\text{br}} & 1 \\ x_{\text{bl}} & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1920 \\ 0 \\ 1920 \\ 0 \end{bmatrix}$$

Then compute the least squares solution using the normal equation:

$$\boxed{x = (A^T A)^{-1} A^T y} \tag{5}$$

With:

$$x = \begin{bmatrix} a \\ b \end{bmatrix}, \quad A = \begin{bmatrix} x_{\text{tr}} & 1 \\ x_{\text{tl}} & 1 \\ x_{\text{br}} & 1 \\ x_{\text{bl}} & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1920 \\ 0 \\ 1920 \\ 0 \end{bmatrix}$$

Corporate sensitivity
PUBLIC

We can calculate the inverse and multiply numerically, but to shortcut the algebra, the final values which have been averaged from the seven experiments for more precision are:

$$a \approx 2.6502, \quad b \approx -1591.08$$

System 2 (for vertical mapping — screen Y) Same logic for the y:

$$\begin{cases} c \cdot y_{\text{tr}} + d &= 899 \\ c \cdot y_{\text{tl}} + d &= 899 \\ c \cdot y_{\text{br}} + d &= 0 \\ c \cdot y_{\text{bl}} + d &= 0 \end{cases} \tag{6}$$

Matrix form:

$$\begin{bmatrix} y_{\text{tr}} & 1 \\ y_{\text{tl}} & 1 \\ y_{\text{br}} & 1 \\ y_{\text{bl}} & 1 \end{bmatrix} \cdot \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} 899 \\ 899 \\ 0 \\ 0 \end{bmatrix}$$

Final result (after same process):

$$c \approx 3.6121, \quad d \approx -2331.79$$

Final Equations

$$\boxed{\begin{cases} \text{ScreenX} &= 2.6502 \cdot x - 1591.08 \\ \text{ScreenY} &= 3.6121 \cdot y - 2331.79 \end{cases}} \tag{7}$$

To conclude, even if this approach significantly improved the alignment between the projected gaze and the Unity display, the red rings are still not always perfectly fitting the screen of the simulation. After some analysis it can be concluded that this misalignment arises because the gaze data is reported in a coordinate system that is head-relative and not dynamically corrected for head position and viewing angle. Consequently, the rectangle formed by the four gaze-reported corners was neither accurately scaled nor positioned.

Because this issue is intrinsic to the eye-tracking glasses, and in order to still move forward with the project and implement an eye-tracking system nonetheless, the following paragraph presents an alternative method that was developed during this project.

4.4.2 Tobii Eye Tracker 5

In order to avoid this head-related coordinate system, the Tobii Eye Tracker 5 (ET5 Fig 21) was selected for integration into the simulator primarily due to the fact that the device is mounted directly on the screen of the computer, its accessibility, ease of use, and compatibility with real-time visualisation tools such as *Tobii Ghost*. While it is a consumer-grade device, it offers sufficient tracking accuracy and robustness for many interactive simulation and user interface applications, especially where qualitative or semi-quantitative insights are sufficient. Its unobtrusive form factor and USB interface make it ideal for quick deployment in experimental setups without requiring extensive

Corporate sensitivity
PUBLIC

calibration or specialised hardware like the Tobii Glasses. Initially, a quote was requested from Tobii for the higher-end *Tobii Pro Spark* [23] as it is not available to the public, which offers access to raw gaze data and more advanced analytics; however, the cost was too high for the project scope. As this phase was intended to serve as a proof of concept for the simulator, the organisation accepted after this analysis to purchase the Eye Tracker 5, which provided an adequate balance between functionality and budget. Despite restrictions on accessing raw gaze data via the Pro SDK, the ability to visualize gaze behavior in real time within Unity using *Tobii Ghost* made the ET5 a practical and cost-effective choice for validating early-stage user attention modeling in the simulation.

Figure 21: Illustration of the Tobii Eye Tracker 5

It operates based on the principle of *pupil center corneal reflection* (PCCR), using near-infrared (NIR) light to create reflections on the cornea (glints), which are detected by its built-in infrared cameras. By identifying the relative position of the pupil and the corneal reflection, the tracker computes the gaze vector and estimates the user's point of regard on the screen with a sampling rate of up to 90 Hz[2].

For its accuracy, several analysis can be found in the literature, including one from the Human-Computer Interaction book. In their independent evaluation study, the Tracker's average gaze error is approximately 35 pixels, which corresponds to about 0.74° of visual angle, with a standard deviation around 18 pixels (0.39°) (for a test involving three participants across 21 fixed points and 252 measurements). In this paper, the researchers also present a possibility to improve these results with a Multi-Layer Perceptron Regressor (MLP Regressor). In the context of eye tracking, this neural network is trained to correct inaccuracies in the gaze data collected from eye-tracking devices. So when the raw gaze data does not perfectly align with where the user was actually looking (due to device limitations, calibration errors, etc.), the MLP Regressor learns a mapping from the measured gaze positions to the true gaze targets (usually provided during a calibration procedure). After training, it can then be used to reassign or adjust the predicted gaze points, making them better match the expected or intended gaze location on the screen. This could potentially be added to the LOTUSim in the future if experiments requiring more accuracy was needed.

But for this project, as it is a proof of concept, only the gaze trace was added to the simulator, using the Tobii Experience Driver v1.133 [3] to do the calibration (with or without glasses, and depending the size of the screen), and the Tobii Ghost v1.14.1 application [3] to set the type, size, color of the gaze trace (Fig 22 and Fig 23).

Here are some illustrations of its integration in LOTUSim :

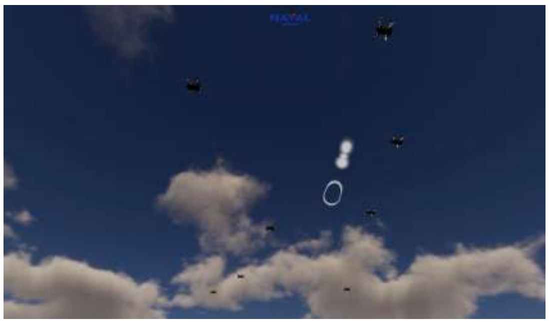


Figure 22: Illustration of the Eye Tracker 5 in LOTUSim in bubble mode

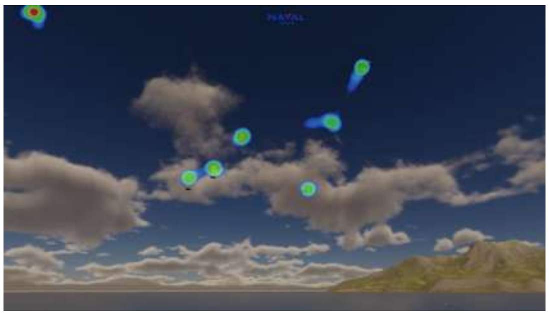


Figure 23: Illustration of the Eye Tracker 5 in LOTUSim in heatmap mode

5 LOTUSim Benchmark

In the previous sections, the work presented primarily focused on the implementation of features and proof-of-concept developments for LOTUSim. This section now provides a more detailed overview of how LOTUSim operates by highlighting the results of performance benchmarks. These benchmarks were carried out to compare LOTUSim's performance with other existing simulators, in the context of preparing a scientific paper submission to the IEEE International Conference on Robotics and Automation (ICRA). The benchmarking of LOTUSim was conducted by J.G., a post-doctoral researcher. This section begins with an explanation of LOTUSim's architecture and design, followed by a presentation of two benchmarking campaigns performed during this intership on the UUV Simulator and LRAUV Simulator, in order to compare their results with J.G.'s evaluation of LOTUSim.

5.1 Overview of LOTUSim

5.1.1 LOTUSim Architecture and Integration

LOTUSim is a distributed multi-agent simulation framework that integrates ROS 2, Gazebo, and Unity to offer a modular and flexible simulation environment. ROS 2 is used to manage inter-agent communication and facilitate the connection with real robotic systems. Gazebo provides realistic physics-based simulation capabilities, while Unity offers high-fidelity rendering for visualisation.

The primary goal of LOTUSim is to support scalable and distributed simulations across diverse domains, with a particular focus on maritime applications. The architecture is organised around a client-server model, where Gazebo acts as the central orchestrator for managing all simulated assets. Various interface modules are connected to Gazebo via custom plugins that delegate specific tasks such as physics computation, rendering, and agent interaction.

The core control unit, referred to as the multi-agent simulation controller, is interfaced with three major subsystems represented in Fig 24:

- **Physics Computation:** During each simulation update cycle, Gazebo queries the physics client to compute dynamics based on the asset's state and the elapsed timestep. A standardised physics interface abstracts the underlying engine require-

Corporate sensitivity
PUBLIC

ments, enabling interoperability with external engines like LOTUSim-Xdyn. Communication is typically conducted over gRPC or WebSocket.

- **Rendering:** Visualisation is handled through Unity by default. The rendering client receives positional and event-based data (creation or destruction of assets) at each simulation step. Users may also integrate custom rendering features, such as visual effects, or even disable visualization when not required (during AI training for instance).
- **Agent Interaction:** This is managed via ROS 2, allowing agents to operate independently and interact through DDS-based messaging. Users can interface with the system using various programming languages by publishing to topics or using action servers.

This modular and distributed setup also enhances scalability. Computational loads can be offloaded to external machines, and agents can be dynamically spawned or removed during runtime. The physics plugin randomizes asset update order to reduce scheduling bias, and communication latency naturally introduces non-determinism into the simulation.

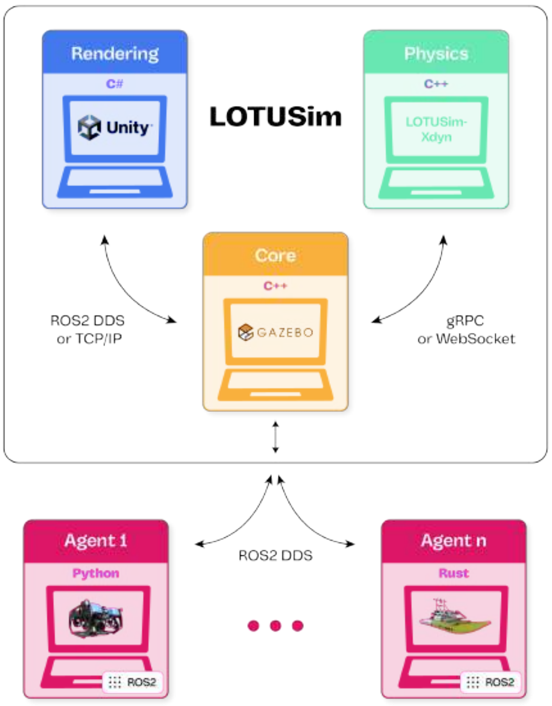


Figure 24: LOTUSim Architecture Overview

5.1.2 Multi-Agent Control System

The LOTUSim architecture facilitates robust and flexible multi-agent simulations by allowing each entity to operate semi-independently. The MAS (Multi-Agent System) plugin in Gazebo serves as a central controller and exposes a ROS2 action server capable of managing asset spawning and behavior updates.

Corporate sensitivity
PUBLIC

Agents can be deployed across different machines in a network and interact asynchronously. Commands for spawning, moving, or removing agents are processed at each update step, with execution order being inherently randomised due to network delays.

5.1.3 Environment Modeling in LOTUSim

- **Surface** : To simulate surface physics, LOTUSim integrates *Xdyn8*, an open-source, ship simulator that models real-time vessel dynamics at sea, created by **SIREHNA** [17]. Xdyn8 computes vessel motion based on *Fossen's equations of motion* [8], incorporating detailed hydrodynamic effects such as Froude–Krylov and diffraction forces as described in their documentation [18]. LOTUSim allows the export of environment data from Xdyn as 2D or 3D grids for offline analysis. In the current official release of Xdyn, surface elevation data is already available [18].

- **Underwater** : For subsurface ocean currents, an Ekman layer model was implemented by a former intern, divided in the followng three layers according to his report:
  - The **surface layer**, where wind stress and the Coriolis effect produce the characteristic Ekman spiral.
  - The **bottom layer**, where seabed friction and bathymetry generate a bottom spiral and vertical flows.
  - The **geostrophic interior**, where flow balances pressure gradients and Coriolis forces, largely free from friction.

- **Aerial**: The winds dynamics have been implemented also by an intern, through a Gazebo wind plugin

5.2 First tests on LOTUSim

In parallel with this report, two scientific papers are currently being prepared. The first, which will be submitted to the ICRA conference in mid-September, focuses on the multi-domain simulation framework for marine robotics. It presents the simulator's architecture, the implementation of the environment, the modeling choices, and preliminary benchmark results. The second paper will provide a more in-depth analysis, dedicated entirely to the detailed results of the three benchmark studies conducted during the internship.

Before starting to implement benchmarks on other simulators, an important step was to try to perform few tests with J.G's benchmark on the LOTUSim, in order to get a bit more familiar with the architecture, and the core of the simulator, to then be able to create the two other benchmarks for this study.

5.2.1 Benchmark Tests and Settings

Settings During these tests, a few features and settings have been noted to keep in mind for the future benchmarks to perform.

The important elements which are gonna have to be replicated in the other benchmarks, are the *.sdf* files, describing the different agents of the simulators, but also the *.world* files, where the Gazebo worlds are described, as all the compared simulators work

Corporate sensitivity
PUBLIC

with Gazebo. Indeed, the goal here is to compare the LRAUV and BlueROV simulated in LOTUSim, with the one simulated in BlueROVSim, and LRAUVSim, therefore, In the end here are the shared parameters we kept: For the LRAUV and BlueROV:

- **Masses**: LRAUV = 120kg, and BlueROV = 10kg
- **Sensors**: For the commun comparaison, only the IMU was kept
- **Rigid body inertia matrix I**: for a rectangular cuboid in the body frame is:

$$\mathbf{I} = \begin{bmatrix} 0.2 \cdot m \cdot y_{\text{size}}^2 + 0.2 \cdot m \cdot z_{\text{size}}^2 & 0 & 0 \\ 0 & 0.2 \cdot m \cdot x_{\text{size}}^2 + 0.2 \cdot m \cdot z_{\text{size}}^2 & 0 \\ 0 & 0 & 0.2 \cdot m \cdot x_{\text{size}}^2 + 0.2 \cdot m \cdot y_{\text{size}}^2 \end{bmatrix}$$

  where m is the mass, and $x_{\text{size}}, y_{\text{size}}, z_{\text{size}}$ are the dimensions along each principal axis.

- **Linear damping matrix**: defines resistance forces that are proportional to the vessel's velocity, modeling viscous drag effects at low speeds. It has been calculated during the work done in paragraph 5.3, and according to the coefficient found in the Fossen book [8]. Here is the from of the matrix:

$$\mathbf{D}_{\text{lin}} = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_w & 0 & 0 & 0 \\ 0 & 0 & 0 & K_p & 0 & 0 \\ 0 & 0 & 0 & 0 & M_q & 0 \\ 0 & 0 & 0 & 0 & 0 & N_r \end{bmatrix}$$

  where X_u, Y_v, Z_w are the linear damping coefficients for translational motion, and K_p, M_q, N_r for rotational motion. The coefficient recommended by Fossen were : 11.7391, 20, 31.8678, 25, 44.9085 and 5.

- **Quadratic damping matrix**: accounts for resistance forces that increase with the square of velocity, capturing turbulent flow effects at higher speeds. It has also been calculated in the paragraph 5.3, and here is its from:

$$\mathbf{D}_{\text{quad}} = \begin{bmatrix} X_{|u|u} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{|v|v} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{|w|w} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{|p|p} & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{|q|q} & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{|r|r} \end{bmatrix}$$

  where the coefficients $(X_{|u|u})$ represent damping proportional to the square of the velocity components. For the benchmarks, all coefficient were set to 0.

- **Added Mass:** When an underwater vehicle accelerates, it must also accelerate some surrounding fluid. This effect is captured by the *added mass*, which represents the inertia of the fluid that the vehicle must move along with itself. But we set this to zero, because if this was working in UUVSim, is was causing some issues in LOTUSim, as the calculs of the physics by Xdyn were diverging.

Corporate sensitivity
PUBLIC

- **`max_step_size`** = 0.2: defines the maximum simulation time step (in seconds), controlling the physics update granularity and stability. In this work, the real-time update threshold for the simulation was set to 200 ms to ensure responsiveness suitable for soft real-time operation. Prior research indicates that update delays longer than 200 ms may impair system responsiveness and user interaction [12]. For scenarios involving direct control of physical robots, the stricter threshold of 30 ms is commonly cited to maintain stable hard real-time performance, for instance, closed-loop control in ROS-based systems typically runs at 30–100 Hz (10–33 ms cycle times) [16]. Therefore, setting the 200 ms maximum for simulation updates aligns with best practices for simulation-based training, while keeping in mind the tighter 30 ms constraint required for real-world robot control.

- **`real_time_factor`** = 1: sets the ratio between simulation time and real-world time (1.0 means real-time).

- **`real_time_update_rate`** = 5: specifies the desired number of physics updates per second in real time.

Evaluation

To assess LOTUSim's, UUVSim's and LRAUVSim's performance, we monitor the following key metrics:

The RTF is a metric used to evaluate the performance of a robotics simulator such as Gazebo. It measures how fast the simulation is running compared to real time. The It is defined as:

$$\text{RTF} = \frac{\text{Simulated Time}}{\text{Real Time}} \tag{8}$$

More precisely, for each simulation update step:

$$\text{RTF} = \frac{\Delta t_{\text{sim}}}{\Delta t_{\text{real}}} \tag{9}$$

where:

- Δt_{sim} is the amount of simulated time advanced in one update step (0.001 s for a 1 ms timestep),
- Δt_{real} is the amount of wall-clock (real) time required to compute that update.

For example, if the simulator advances the simulation time by 1 ms while requiring 2 ms of real time to compute it, then the RTF would be:

$$\text{RTF} = \frac{0.001}{0.002} = 0.5 \tag{10}$$

This indicates that the simulation is running at half the speed of real time.

The **`realUpdateDuration` or `update_rate`** in Gazebo measures the wall-clock time consumed during a single simulation update iteration on the server side. This duration includes the computational cost of the following elements:

Corporate sensitivity
PUBLIC

- **Physics updates:** Execution of the physics engine (ODE, Bullet, DART).
- **Sensor updates:** Generation of synthetic sensor data (sonar, IMU, and camera data), excluding rendering.
- **World plugins:** Execution of world-level behaviors such as environmental dynamics or global controllers.
- **Model plugins:** Execution of model-level behaviors such as autopilots, thruster dynamics, or sensor controllers.
- **Communication overhead:** Inter-process or intra-process message passing inside the Gazebo server (gzserver).

However, it is important to note that `realUpdateDuration` **does not include** these, but done by the measruement of **the FPS**:

- **Rendering time:** All operations related to visual rendering, which are handled by the client (gzclient).
- **GUI plugin time:** Execution of graphical user interface plugins.
- **File logging time:** Unless file Input/Output operations (I/O) is explicitly executed inside plugins, logging time is excluded from this duration.

To assess these aspects in LOTUSim, UUVSim, and LRAUVSim, two dedicated evaluation procedures were introduced:

- Algorithm 1 is used to evaluate real-time performance in the context of human-in-the-loop interaction.
- Algorithm 2 is designed to measure simulation acceleration, which is critical for efficient training of AI models.

5.3 Benchmark on UUV Simulator

The UUV Simulator (UUV Sim) is an open-source underwater robotics simulation framework built on top of ROS and Gazebo (Fig 26), designed to test and evaluate underwater vehicle behaviors (like the BlueROV in Fig 25) and control strategies in realistic marine environments [26]. In this work, UUVSim was used on Ubuntu 18.04, which relies on ROS Melodic and Gazebo 9 (detailed in Table 1) (can be cloned from their GitHub [6]). The benchmarking experiments were conducted in the `empty_underwater` world, which provides a minimal environment for evaluating performance without additional rendering or physics complexity. The benchmark setup involves three separate terminals: the first one is used to launch the ROS core; the second runs a custom ROS node responsible for initialising and spawning a predefined number of agents in the underwater world for a given simulation duration; and the third terminal runs a script to apply external current forces in the simulation, mimicking realistic marine currents such as those observed in the LOTUSim environment. Importantly, all agents are initially spawned underwater and float naturally to the surface due to their buoyancy, replicating real-world underwater behavior.

Corporate sensitivity
PUBLIC

Algorithm 1 Real-Time Performance Benchmarking

```
Require: Set of agent population $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$
Configure simulator:
    • max_step_size ← 200 ms
    • RTF ← 1
    • T ← 5 min (Simulation Duration)
Initialize simulation with $n = 1$ agent
$\overline{\texttt{RTF}}_n \leftarrow \overline{\texttt{RTF}}_1$ measure
$\overline{\texttt{update\_rate}}_n \leftarrow \overline{\texttt{update\_rate}}_1$ measure
if $\overline{\texttt{RTF}}_n \approx 1$ and $\overline{\texttt{update\_rate}}_n < \texttt{max\_step\_size}$ then
    for each $n \in \mathcal{A}$ do
        Initialise simulation with $n$ agents
        Measure over duration T (after agent spawned):
            • $\texttt{update\_rate}_n$
            • $\texttt{FPS}_n$
            • $\texttt{RTF}_n$
        Store $(n, \overline{\texttt{update\_rate}}_n, \overline{\texttt{FPS}}_n, \overline{\texttt{RTF}}_n)$
    end for
end if
```

Algorithm 2 Accelerated-Time Performance Benchmarking

```
Require: Set of agent population $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$
Configure simulator:
    • max_step_size ← 30 ms
    • RTF ← $MaxValue$   (as-fast-as-possible)
    • T ← 5 min (Simulation Duration)
Initialise simulation with $n = 1$ agent
$\overline{\texttt{RTF}}_n \leftarrow \overline{\texttt{RTF}}_1$ measure
if $\overline{\texttt{RTF}}_n > 1$ then
    for each $n \in \mathcal{A}$ do
        Initialise simulation with $n$ agents
        Measure $\texttt{RTF}_n$ over duration T (after agent spawned)
        Store $(n, \overline{\texttt{RTF}}_n)$
    end for
end if
```

Corporate sensitivity
PUBLIC

Figure 25: Illustration of the BlueROV

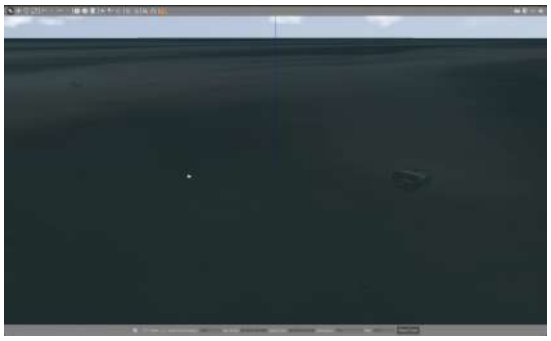


Figure 26: Illustration of the UUV Simulator

Specification	UUVSim	LOTUSim
OS	Ubuntu 18.04.6 LTS	Ubuntu 22.04.5 LTS
Processor	Intel Core i7-11800H (11th Gen)	Intel Core i9-13980HX (13th Gen)
CPU Clock Speed	Up to 4.6 GHz	Up to 5.6 GHz
GPU	NVIDIA GeForce GPU	NVIDIA GeForce RTX 4090 Laptop GPU
GPU VRAM	8 GB	16 GB
NVIDIA Driver	470.223.02	570.133.07
CUDA Version	11.4	12.8
Python Version	Python 2.7.17	Python 3.10.12
Gazebo Version	Gazebo 9	Gazebo Harmonic
ROS Version	ROS1 Melodic	ROS2 Humble

Table 1: Benchmark system specifications comparison: UUVSim vs LOTUSim

5.3.1 Real Time Simulations on BlueROVs

The benchmark procedure for UUVSim, is to incrementally add more agents into the same simulation environment to assess system scalability. The simulation is considered to have failed when either the Real-Time Factor (RTF) drops below 1, indicating that the simulation is no longer running in real time, or when the real-time update duration exceeds a critical threshold of 200 ms.

So to be able to measure all these values, modifications were introduced directly into the simulator's source code to extract and continuously log three essential performance metrics: the **Real-Time Factor (RTF)**, the **Real Update Time**, and the **Frames Per Second (FPS)**. Each of these metrics was recorded into a separate `.csv` file to facilitate post-processing and visualisation.

Modification of Gazebo's Source Code

The extraction of RTF and Real Update Time was implemented at the simulation core level, specifically within the `World.cc` file. The `World::RunLoop()` function, which governs the main simulation loop, was modified to compute the *real update time* as the wall-clock duration required to simulate a single time step. This was accomplished by capturing timestamps before and after each call to `Update()`, using high-resolution time functions from the Gazebo time library. The real-time factor was then computed as the ratio between the simulation time increment (typically fixed at 1 ms or another configured timestep) and the measured real update time. These values were written to two separate files, `real_update_time.csv` and `rtf.csv`, through standard C++ file streams. Logging was configured to occur periodically (every second of simulation time) to avoid excessive I/O overhead.

Corporate sensitivity
PUBLIC

To capture the rendering performance on the client side, modifications were applied in the graphical user interface components, particularly in the `MainWindow.hh` and `MainWindow.cc` files. A periodic callback was introduced to access the frame rate data using the method `UserCamera::AvgFPS()` . This value, representing the average number of rendered frames per second, was appended to `fps.csv` at fixed intervals using a QTimer. Integration with the Qt event loop ensured that the logging process was non-blocking and did not interfere with GUI responsiveness.
Once all these modifications in the source code had been done, Gazebo 9 was entirely **recompiled** using `cmake` and `make`.
In total, these modifications enabled precise tracking of simulation performance under increasing load, particularly during large-scale experiments involving the spawning of hundreds of autonomous underwater vehicles.

5.3.2 Simulations for AI training : Accelerated Time Benchmarking

The next phase of the benchmarking process aimed to evaluate the simulation's capacity for **accelerated time**, which is essential for enabling efficient **training of AI** agents. This was particularly intended to demonstrate that UUVsim, and more importantly LOTUSim, can support such accelerated training workflows. To perform this test in UUVsim, the simulation's **`max_step_size`** parameter was set to **0.03**, same as the benchmark on LOTUSim's side, and the real-time factor (RTF) target was significantly increased, aiming for values up to 200%. Then, the number of agents (BlueROV vehicles) was incrementally increased in the simulation until the RTF dropped below 1, indicating a loss of real-time performance, and the limit of doing accelerated time.

Due to UUVsim's reliance on older frameworks, namely ROS 1 and Gazebo 9 on Ubuntu 18.04, agent spawning is notably slow, taking approximately 10 minutes to initialise 200 agents. Consequently, to ensure consistency in the benchmark results, only the RTF values measured after all agents were fully spawned were considered for analysis. This approach allows for a clear comparison of performance between UUVsim and LOTUSim under accelerated time conditions.

5.3.3 UUVSim Benchmark Results

Real Time Performance

We observed that the simulation reaches a critical point at approximately 333 agents. Although the terminal output indicated that all requested agents (340 or 400) were successfully spawned, only **333 models were visible in the Gazebo model list**. At 330 agents, the real-time update time was still below the critical 200 ms threshold (specifically 93.88 ms), suggesting that another bottleneck is responsible for the simulation limit.

This limitation appears to be primarily due to GUI and rendering overhead. Each new agent added to the simulation requires Gazebo's client interface (`gzclient`) to render new entities, manage associated resources (meshes, textures, sensors), and update the model list panel. This panel is implemented via Qt's `QTreeView`, which becomes increasingly inefficient when handling several hundred models. Around 300-400 models, the Qt tree view begins to lag, and in some cases, fails to display additional agents even if they exist in the simulation backend (`gzserver`). This GUI limitation can be mitigated by running the simulation in headless mode (without `gzclient`), in which case more agents can be simulated successfully.

Corporate sensitivity
PUBLIC

Further testing of system resources showed that at 333 BlueROVs, the system **memory usage** reached 92.7% (Fig 27), with some CPU cores at 100% utilisation. However, GPU memory usage remained low at only 4 MiB, suggesting that the bottleneck is not GPU-bound but primarily due to GUI rendering and CPU limitations.

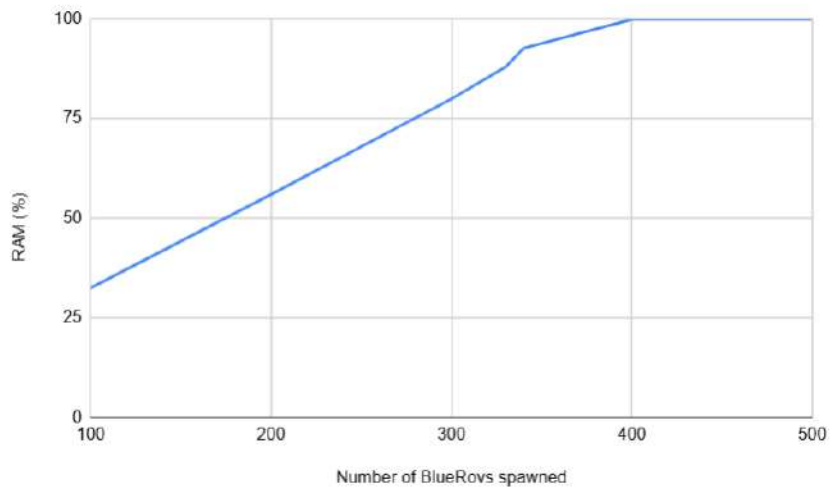


Figure 27: Graphic of the evolution of the RAM (%) depending on the number of BlueROVs spawned (visually, only 333 spawned)

For comparative benchmarking, the same experiment was conducted in the LOTUSim environment by J.G. Results indicate that LOTUSim can simulate up to 450 BlueROVs before reaching a similar perception limit (real-time update time exceeding 200 ms). Regarding physics time-step constraints, LOTUSim begins to exceed the 30 ms threshold with more than 30 agents, whereas UUV Simulator can sustain up to 130 agents before hitting this limit. These results (illustrated in Fig 28) demonstrate a significant advantage of LOTUSim in terms of scalability and performance, highlighting its suitability for large-scale multi-agent simulations and AI training tasks.

The performance of a simulator in terms of **rendering speed** is critical, particularly for immersive or interactive applications involving human operators. A widely accepted standard is that maintaining a framerate above 60 FPS ensures an acceptable user experience, while values near or above 120 FPS provide a seamless, "super-smooth" visual output [5]. In our evaluation of the UUV Simulator, we observed a significant degradation of framerate as the number of agents increased. With 100 agents, the mean FPS dropped to 11.89, and further declined to around 6–7 FPS for scenarios with 300 to 500 agents, which are not acceptable for a confortable user experience (Fig 29).

In contrast, LOTUSim, built on the Unity engine, demonstrates consistently superior rendering performance. Across identical scenarios ranging from 0 to 700 spawned autonomous agents, the FPS remained above 140. This performance not only guarantees smooth user interaction but also allows for high-fidelity visualisation necessary for real-time supervision, AI training with photorealistic feedback, and scenario replay. The architectural differences between the two platforms, Unity's GPU-accelerated rendering pipeline versus Gazebo's CPU-bound engine, largely explain this disparity, underscoring the importance of graphics backend design for scalable and responsive maritime simula-

Corporate sensitivity
PUBLIC

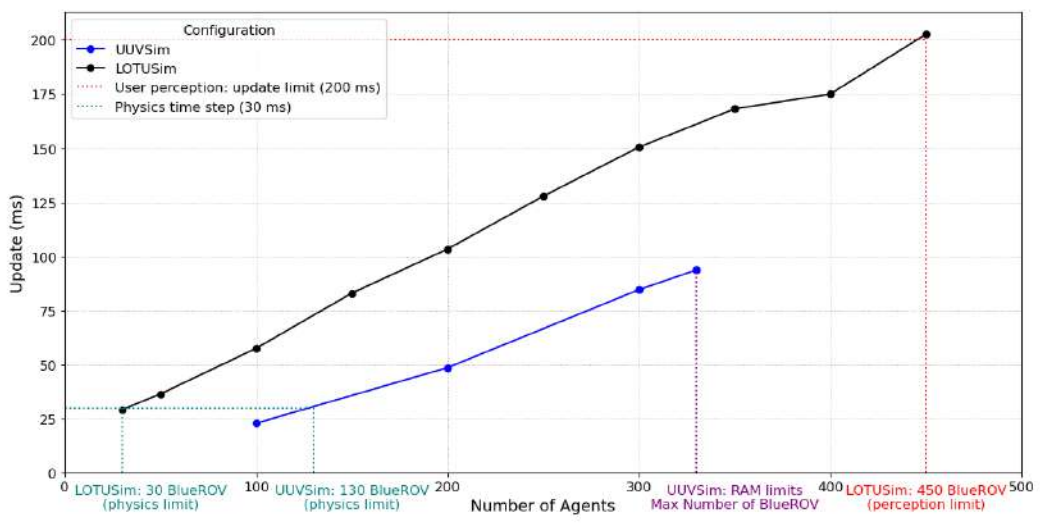


Figure 28: Simulations of BlueROV: UUVSim vs LOTUSim's Results

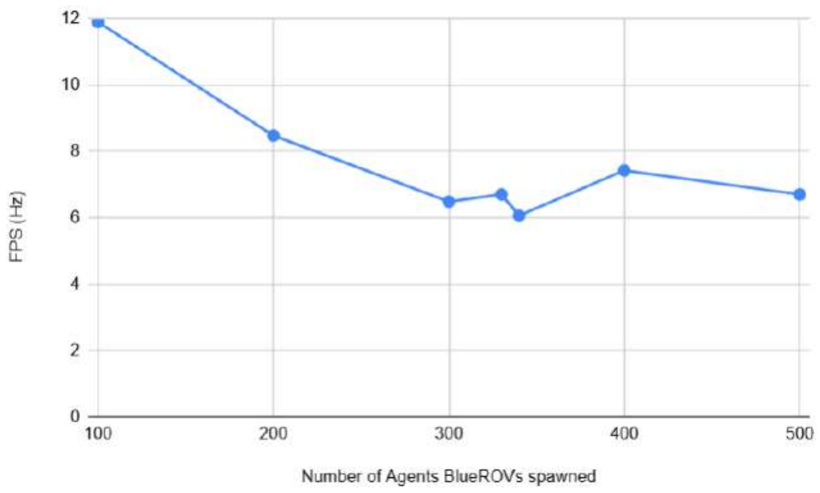


Figure 29: Graphic of the evolution of the FPS depending on the number of BlueROVs spawned (visually, only 333 spawned)

tion.
End in the end, the collected data revealed that simulation slowdowns were not strictly linked to real-time factor thresholds, but rather emerged due to accumulated CPU usage and graphical rendering limitations on the client side, and that the evaluation of the FPS is also a priority when operators are involved in a simulation.

Accelerated Time Performance

In order to assess the capability of both simulators for accelerated-time training of AI agents, a series of experiments was conducted where the `max_step_size` was fixed to 0.03 s and the simulation was run with a target Real-Time Factor (RTF) of up to 200%. Then, the number of BlueROV agents was gradually increased until the RTF dropped below 1.0, which is the minimal threshold for performing accelerated training. In these conditions, **UUVSim** was capable of handling up to **180 BlueROVs** before its RTF fell below

Corporate sensitivity
PUBLIC

1.0. In contrast, **LOTUSim** maintained an RTF greater than 1.0 up to **25 BlueROVs** (illustrated in Fig 30). While this number is lower compared to UUVSim, it is still highly effective for AI training purposes, as it enables simulations to run approximately 25 times faster than real-time. This demonstrates that both simulators are suitable for accelerated-time AI training, with UUVSim offering higher scalability, and LOTUSim remaining a viable and efficient alternative for smaller-scale, high-speed training tasks.

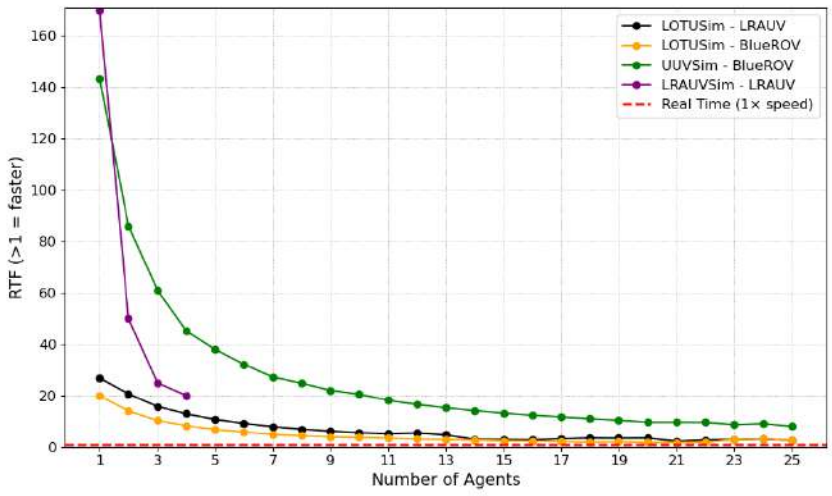


Figure 30: Simulations of BlueROV: UUVSim vs LOTUSim Accelarated Time's Results (RTF)

5.4 Benchmark on LRAUV Simulator

To extend the benchmark analysis to other state-of-the-art underwater simulators, the LRAUVSim framework was evaluated. This simulator is maintained by Open Robotics (their work on the simulator have been pubished in this paper [14]) and targets the simulation of Long-Range Autonomous Underwater Vehicles (LRAUVs). The installation followed the official instructions and was performed on an Ubuntu 20.04 system, using the `gazebo garden` simulation environment [9]. The benchmarking required installing the `gz-sim` tool (version 7 or higher) from the Garden suite.

LRAUVSim (Fig 31) itself was installed via its official repository [15], which provides a detailed walkthrough for compilation and dependencies. Once installed, the simulator was configured for performance testing under various mission scenarios to compare its runtime behavior with other platforms like UUVSim and LOTUSim.

5.4.1 Real Time Simulations on LRAUVs

For the LRAUVSim benchmark, the standard `.sdf` model of the LRAUV provided in the official simulator repository [15] was used, which is based on the design of the *Tethys* AUV. To simplify the setup and ensure fair comparisons with the other simulators (like in paragraph 5.2.1), only the IMU sensor in the model was retained, disabling other modules. In order to simulate realistic hydrodynamic conditions, a constant water current was manually introduced by modifying the `HydrodynamicsPlugin.cc` source file located in the LRAUV simulator folder. Specifically, we set the `waterCurrent` vector to a non-zero value:

Corporate sensitivity
PUBLIC

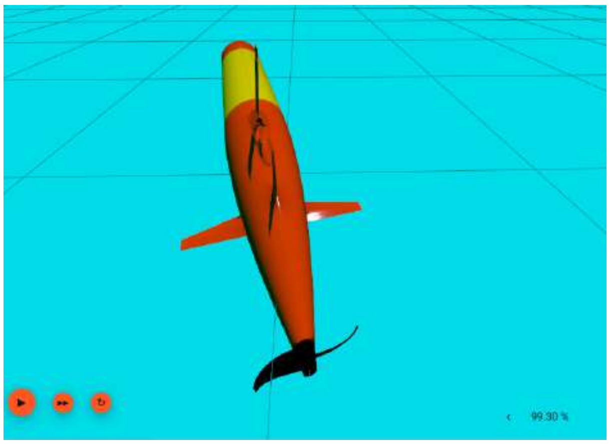


Figure 31: Illustration of an LRAUV in LRAUVSim

```
/// \brief Water current [m/s].
public: gz::math::Vector3d waterCurrent {0.5, 0.0, 0.0};
```

Although this hardcoded approach bypasses the more modular `/ocean_current` topic mechanism, it was selected due to project time constraints, as the benchmarks were performed in the context of an upcoming publication. A cleaner and more flexible implementation will be considered in future iterations.

Performance Metrics Collection in LRAUVSim To perform a rigorous benchmark of the LRAUV simulator, it was necessary to collect key performance metrics including Real-Time Factor (RTF), real-time update durations, and rendering frame rate (FPS) jus like UUVSim, and therefore modify Gazebo's source code, before recompiling it. Given the architecture of Gazebo Garden (used in LRAUVSim), the required values were not readily exposed, necessitating targeted source code modifications. To capture FPS values, the `MainWindow.cc` file was extended by connecting a lambda function to the `frameSwapped` signal emitted by the `QQuickWindow` instance. Each time a frame was rendered, the callback computed the elapsed time and wrote the resulting FPS and corresponding wall-clock time to a CSV log file (`render_fps.csv`). For the RTF and real-time update monitoring, modifications were made in the `WorldStats.cc` file. Simulation and real-time values were extracted from incoming messages, filtered using exponential smoothing, and used to compute the RTF as the ratio of simulated time over wall-clock time. Additionally, simulation timestamps, real-time values, and the computed RTF were exported to separate CSV files (`real_time.csv` and `rtf.csv`). These modifications allowed consistent logging and enabled accurate quantitative comparisons between simulators under test conditions. Unlike Gazebo 9, Garden's QML-based interface and plugin architecture required integration with Qt's signal system and the use of Gazebo's internal time representation utilities.

As Naval Group Pacific is a subsidiary of Naval Group working for defense, this report has to go through a security check a month before being submitted to ENSTA and the jury. Therefore, at the point of the intership (mid July), the results of this benchmark and work on LRAUVSim weren't finish. But they will be presented during the oral presentation in September.

Corporate sensitivity
PUBLIC

5.4.2 Simulations for AI training : accelerated time

The results for the accelarated time will as well be presented during the oral presentation as they also haven't been done yet at this point.

5.4.3 LRAUV Sim Benchmark Results

Although the benchmark results for LRAUVSim are not yet available, valuable insights into the simulator's performance can already be found in the literature and have been already added in the graph in Fig 30. In particular, Player et al. [14] present detailed performance evaluations of LRAUVSim in the context of accelerated development of multi-AUV missions. Their study demonstrates the simulator's ability to operate faster than real time while maintaining high-fidelity physical and environmental modeling. These findings support the relevance of LRAUVSim as a viable tool for AI training and mission planning, and the future work done on this benchmark before the end of this internship will aim to complement this existing data with additional metrics and comparisons under our specific test configurations.

©Naval Group SA property, 2025, all rights reserved.

6 Setting up the three domains integration

Now in the context of the development of the LOTUSim, Naval Group Pacific (based in Adelaide), works with another subsidiary called Naval Group Far Est (NGFE), based in Singapore. One of the developers in NGFE has to code the three domains integration, meaning guaranteeing that the physics, communications, and interactions with agents from the three different environments (air, surface, underwater) can all work together with the newest Multi-Agent-System (MAS).
This paragraph present the work that has been done to help with the intergration, of the three domains.

So to support the integration of the three domains a prototype of a dual-world simulation setup was created.
First, an **aerial Gazebo world** (Fig 32) was created, featuring X500 drones configured to interact with environmental wind forces using a dedicated **wind plugin**. In parallel, a **general simulation world** (Fig 33) was designed, containing all agents from the `Silent Storm` scenario (excluding the X500s), which relies on **Xdyn** to compute the physics for surface and underwater vehicles. This setup has now been sent to NGFE, and a major challenge in this new implementation lies in merging these two environments coherently, ensuring that the wind plugin and xdyn do not conflict in terms of external forces such as gravity or environmental interactions. Additionally, the network-level integration posed further complexity, as the current agent and entity managers in the simulator were not originally built to handle a multi-domain simulation architecture. This work lays the foundation for enabling seamless multi-environment coordination under the MAS framework.

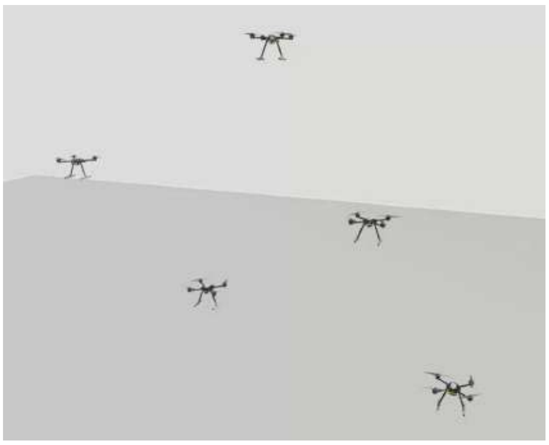


Figure 32: Illustration of the Aerial Gazebo World

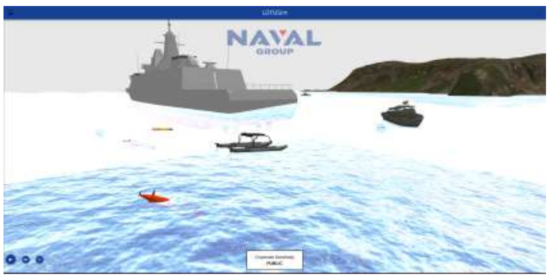


Figure 33: Illustration of the General Gazebo World from the Silent Storm scenario

Later on, in August, a big task done during this internship will be to implement the wind plugin in the Unity side, and also do the connection with the core of the simulator, through ROS2 nodes and bridges. This part will be detailed and explained during the oral presentation in September, again because of the early security check required by Naval Group.

7 Merging proof of concepts with a scenario and a ROS interface

All developments related to multi-user support and user tracking through sensor inputs have been successfully integrated into the work led by another intern, B.D., who focused on building a specialised scenario called `Silent Storm` (Fig 34). This scenario includes several types of agents and is notably used to demonstrate the capabilities of the simulator in more complex mission settings. To support this integration, the simulation relies on ROS 2 Humble for managing communication and physics computation. In the final setup, one Linux machine runs the core LOTUSim simulation, handling all the physics calculations and real-time dynamics. Meanwhile, multiple users, whether on Linux, Windows, or other platforms, can connect to the simulation through a Unity-built executable. By entering the appropriate ROS IP address and port number (described in paragraph 3.2.2), users are able to access the shared simulation scene, interact with it, and observe agent behaviors in real time. This architecture enables efficient distributed simulation while maintaining the high-fidelity physics model offered by LOTUSim.

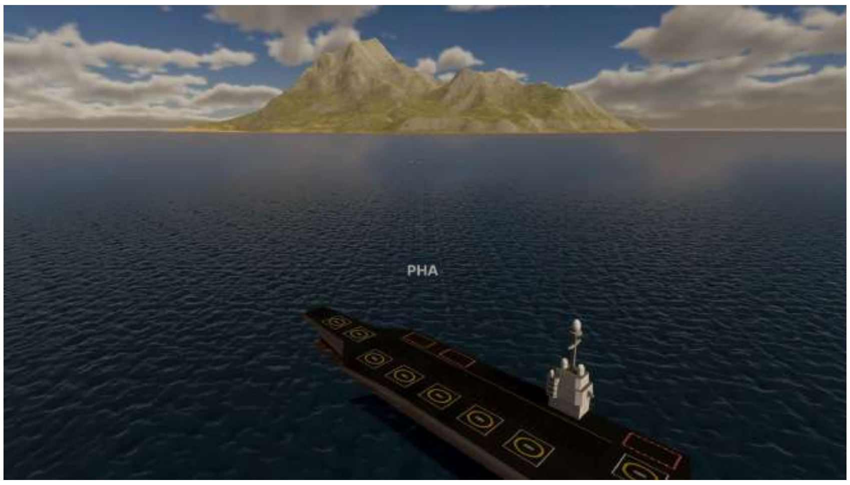


Figure 34: Illustration of the scenario

8 Future Development

Now for the month and a half remaining of internship, several other tasks will be done, and will also be continued during the end of the year.

8.1 Integration of PhD Students'work

One of the key objectives of LOTUSim's development is to serve as a modular and reusable tool for research applications. In the coming months, a set of dedicated ROS2 nodes will be developed to facilitate integration of custom algorithms. This will enable PhD students to directly insert and execute their own research code, such as AI training routines or SLAM algorithms within the LOTUSim environment. These developments aim to make simulation-based experimentation more accessible and flexible. Regular collaboration between the PhD students and the Naval Group Pacific team ensures that the architecture of LOTUSim remains well-suited to accommodate a wide range of research needs.

8.2 Camera Integration and Algorithm Testing

Another future task will focus on the development of a new ROS2 package aimed at integrating the work of a PhD student whose research involves testing a perception algorithm on an CDA (Fig 35) equipped with a camera. The objective is to enable real-time visualisation of the drone's camera feed directly within the Unity interface, providing an immersive and interactive way to observe the algorithm's behavior during simulation. This development will support the validation and debugging of vision-based algorithms in a realistic underwater environment, further enhancing the capabilities of LOTUSim as a research-oriented platform.

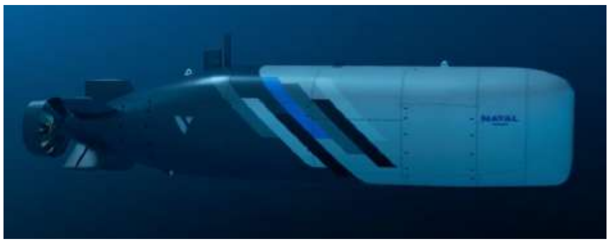


Figure 35: Picture of the CDA from Naval Group

8.3 LOTUSim-Energy: a Tool for Renewable Energy Infrastructure

In addition to underwater applications, the LOTUSim simulator is also being leveraged for broader **environmental and energy-related scenarios**. One such promising application is the simulation and planning of **offshore wind farms in Australia**, in synergy with the already widespread solar energy infrastructure. Given Australia's unique geography and high solar irradiance, solar power has become a dominant source of renewable energy. However, solar energy production is inherently intermittent and limited to daylight hours. Wind energy, particularly from offshore sources, offers a complementary solution by providing energy during periods of low solar availability, including at night

Corporate sensitivity
PUBLIC

and during cloudy conditions. Therefore, during this internship, some work around this topic has also been done, by preparing a workshop for companies working in the energy industry, and where demos and presentations of the LOTUSim were performed, to potentially work alongside on future projects. Indeed, the LOTUSim platform could be used to simulate complex offshore wind farm environments (like illustrated in Fig 36), enabling research and development in **autonomous maintenance** using **aerial and underwater drones**. These drones can be tasked with inspection (Fig 37), fault detection, and real-time maintenance, reducing operational costs and increasing safety. The simulator supports advanced functionalities such as AI training for autonomous navigation, the deployment and evaluation of fault detection algorithms, digital twin synchronisation, and predictive energy production monitoring under varying weather conditions. Such a comprehensive simulation capability is crucial for designing robust, efficient, and scalable offshore energy infrastructure. Ultimately, this work contributes to the vision of a resilient, fully renewable energy grid in Australia, where solar and wind energy sources operate in tandem to ensure continuous, sustainable power generation.

A third scientific paper presenting the application of the LOTUSim in an environment context is currently being written.

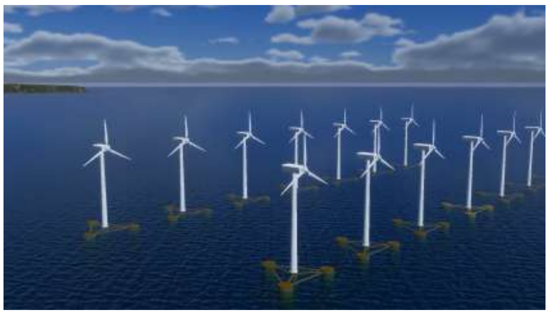


Figure 36: Illustration of an off-shore wind turbine farm created in LOTUSim

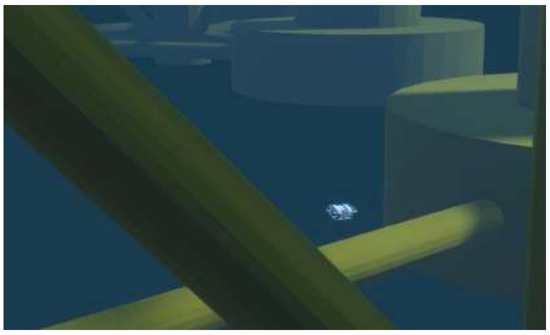


Figure 37: Illustration of bluerov patroling for a maintenace mission in awind turbine farm in LOTUSim

©Naval Group SA property, 2025, all rights reserved.

9 Conclusion

To conclude, this work presents a comprehensive exploration into the development and validation of LOTUSim, a real-time maritime simulation platform tailored for advanced robotics research and human-machine teaming. The platform integrates cross-domain robotic agents, underwater, surface, and aerial, and offers a unique framework for realistic interaction between autonomous systems and human operators. During this study a **multi-user support** was integrated, allowing collaborative scenarios to unfold in real time with synchronised views. This contributes to the study of coordinated decision-making and operator workload distribution in complex, multi-agent missions.

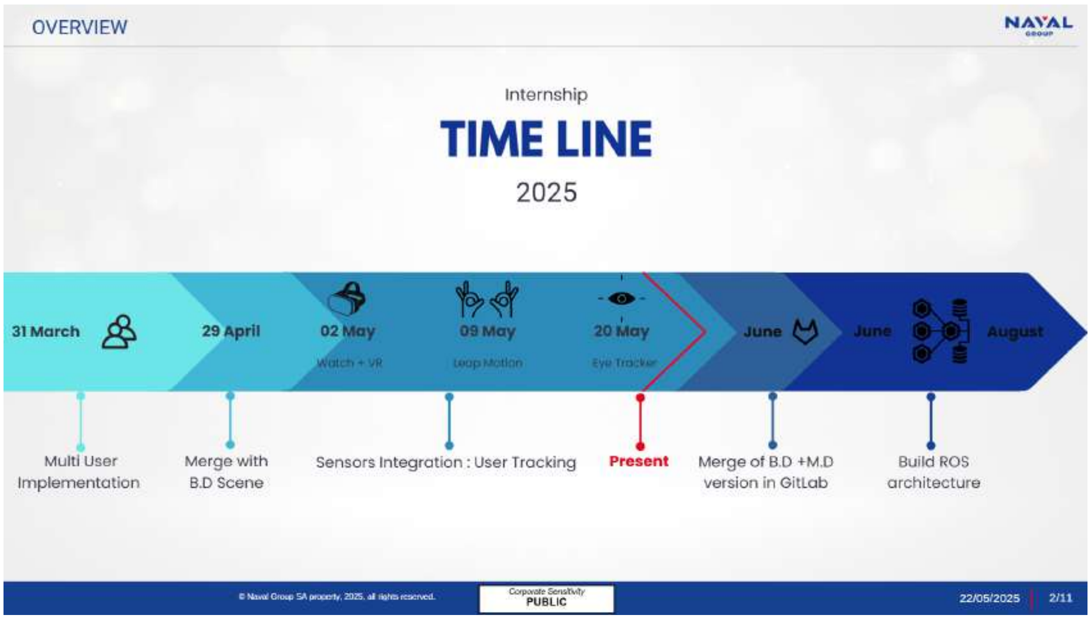
Another major contribution of this study lies in the **integration of human-state monitoring systems**. Devices such as the eye trackers and hand-tracking interfaces were incorporated into the simulation pipeline to enable adaptive interfaces. These systems allow real-time analysis of physiological and cognitive states, opening the path for dynamic human-machine interaction models that improve situational awareness and responsiveness.

Benchmarking efforts across different simulators demonstrated the LOTUSim's robustness and scalability in comparison to the other simulators. It successfully managed large-scale deployments, such as the spawning of hundreds of BlueROV agents, while maintaining real-time performance. Dedicated experiments on underwater and surface vehicles compared the physical accuracy of the simulators, reinforcing their relevance for both **AI training**, enabled by accelerated time, and human-centered evaluation.

Looking forward, the modular nature of LOTUSim opens vast avenues for future development. Integration with ongoing PhD research, such as camera-based perception modules and energy consumption modeling for sustainable maritime infrastructure, will further enrich the platform's capabilities. Beyond academic research, LOTUSim presents high potential for application in strategic defense simulations, operator training, environmental monitoring, and AI testing in safety-critical systems.

In conclusion, LOTUSim establishes itself as a versatile, research-grade simulation ecosystem, bridging the gap between realistic robotics experimentation, collaborative human interaction, and scalable AI development. As simulation continues to play a central role in robotics, platforms like LOTUSim will be instrumental in accelerating innovation, improving safety, and expanding the reach of autonomous systems across domains.

A Appendix



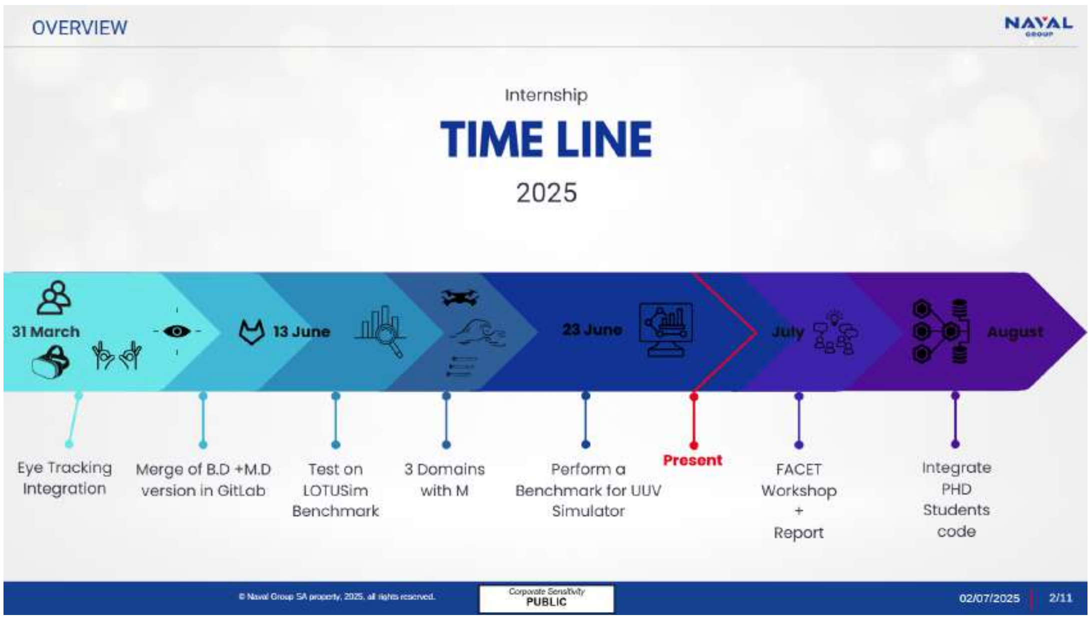
OVERVIEW

Internship
TIME LINE
2025

31 March — Multi User Implementation
29 April — Merge with B.D Scene
02 May Watch + VR / 09 May Leap Motion / 20 May Eye Tracker — Sensors Integration : User Tracking
Present
June — Merge of B.D +M.D version in GitLab
June — August — Build ROS architecture

© Naval Group SA property, 2025, all rights reserved. Corporate Sensitivity PUBLIC 22/05/2025 2/11

Figure 38: Personal Planning for Tasks First Period



OVERVIEW

Internship
TIME LINE
2025

31 March — Eye Tracking Integration
Merge of B.D +M.D version in GitLab
13 June — Test on LOTUSim Benchmark
3 Domains with M
23 June — Perform a Benchmark for UUV Simulator
Present
July — FACET Workshop + Report
August — Integrate PHD Students code

© Naval Group SA property, 2025, all rights reserved. Corporate Sensitivity PUBLIC 02/07/2025 2/11

Figure 39: Personal Planning for Tasks Second Period

B Scientific Paper Publications

Three scientific papers related to this work are currently in preparation. The first focuses on publishing the complete set of benchmarks and performance evaluations of the simulator.

The second paper will provide a more in-depth analysis, dedicated entirely to the detailed results of the three benchmark studies conducted during the internship.

And the third paper presents a practical application involving a wind turbine farm scenario. For confidentiality and plagiarism prevention reasons, the full content of these articles cannot be included in the appendix at this stage. They will be made publicly available once officially published.

Corporate sensitivity
PUBLIC

References

[1] Pasi Aaltonen. Networking tools performance evaluation in a vr application: Mirror vs. photon pun2. https://www.theseus.fi/handle/10024/755310, 2022.

[2] Tobii AB. Tobii eye tracker 5. https://gaming.tobii.com/product/eye-tracker-5/, 2024.

[3] Tobii AB. Tobii eye tracker 5 – get started - software and drivers. https://gaming.tobii.com/getstarted/, 2024.

[4] Daniel Bachmann, Frank Weichert, and Gerhard Rinkenauer. Review of three-dimensional human-computer interaction with focus on the leap motion controller. *Sensors*, 18(7):2194, 2018.

[5] K. Debattista, K. Bugeja, S. Spina, T. Bashford-Rogers, and V. Hulusic. Frame rate vs resolution: A subjective evaluation of spatiotemporal perceived quality under varying computational budgets. *Computer Graphics Forum*, 37(1):363–374, 2018.

[6] The UUV Simulator Developers. uuv_simulator: Underwater simulation with ros and gazebo. https://github.com/uuvsimulator/uuv_simulator, 2023.

[7] Photon Engine. Pun 2 introduction. https://doc.photonengine.com/pun/current/getting-started/pun-intro, 2025.

[8] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Chichester, UK, 2011.

[9] Gazebo Project. Gazebo garden documentation - installation on ubuntu. https://gazebosim.org/docs/garden/install_ubuntu_src/, 2025.

[10] Empatica Inc. E4 sunset: End of service announcement. https://www.empatica.com/research/e4-sunset/, 2024.

[11] Helene Lechene, Benoit Clement, Karl Sammut, Paulo Santos, Andrew Cunningham, and Cedric Buche. LOTUS: Learning from Operational Teaming with Unmanned Systems. In *2024 IEEE Oceans Conference*, pages 1–5, Singapore, April 2024.

[12] XiaoRui Liu, Juan Ospina, Ioannis Zografopoulos, Alonzo Russell, and Charalambos Konstantinou. Faster than real-time simulation: Methods, tools, and applications. *ArXiv*, 2104.04149, 2021. Faster-than-real-time simulation review.

[13] Photon Engine. Photon unity networking (pun2). https://www.photonengine.com/PUN, n.d.

[14] Timothy R. Player, Arjo Chakravarty, Mabel M. Zhang, Ben Yair Raanan, Brian Kieft, Yanwu Zhang, and Brett Hobson. From concept to field tests: Accelerated development of multi-auv missions using a high-fidelity faster-than-real-time simulator. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2023.

[15] Open Robotics. Lrauv simulator - installation guide. https://github.com/osrf/lrauv/wiki/Installation, 2025.

©Naval Group SA property, 2025, all rights reserved.

Corporate sensitivity
PUBLIC

[16] PickNik Robotics. Advances in ros 2 for real-time control. `https://picknik.ai/moveit/ros/2020/02/06/real-time-control.html`, 2020.

[17] SIREHNA. Xdyn8 - Lightweight and Modular Ship Simulator. `https://github.com/sirehna/xdyn`, 2024. Accessed: 2025-07-19.

[18] SIREHNA. Xdyn8 Documentation, 2024.

[19] M. Thibeault, M. Jesteen, and A. Beitman. Improved accuracy test method for mobile eye tracking in usability scenarios. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 63, pages 2226–2230, 2019. Original work published 2019.

[20] Tobii. Tobii pro glasses 3, 2024.

[21] Tobii. *Tobii Pro Glasses 3 Developer Guide*. Tobii AB, 2024.

[22] Tobii. Tobii pro glasses 3 sdk, 2024.

[23] Tobii AB. Tobii pro spark, 2024.

[24] Toxigon. Photon pun vs mirror: Which is right for your unity game? `https://toxigon.com/photon-pun-vs-mirror`, apr 2025.

[25] Ultraleap. Getting started with unity, 2024.

[26] Zekai Zhang, Jingzehua Xu, Jun Du, Weishi Mi, Ziyuan Wang, Zonglin Li, and Yong Ren. Uuvsim: Intelligent modular simulation platform for unmanned underwater vehicle learning. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2024.

©Naval Group SA property, 2025, all rights reserved.