

UNav-Sim: A Visually Realistic Underwater Robotics Simulator and Synthetic Data-generation Framework

Abdelhakim Amer, Olaya Álvarez-Tuñón, Halil İbrahim Uğurlu, Jonas le Fevre Sejersen, Yury Brodskiy and Erdal Kayacan

Abstract—Underwater robotic surveys can be costly due to the complex working environment and the need for various sensor modalities. While underwater simulators are essential, many existing simulators lack sufficient rendering quality, restricting their ability to transfer algorithms from simulation to real-world applications. To address this limitation, we introduce UNav-Sim, which, to the best of our knowledge, is the first simulator to incorporate the efficient, high-detail rendering of Unreal Engine 5 (UE5). UNav-Sim is open-source¹ and includes an autonomous vision-based navigation stack. By supporting standard robotics tools like ROS, UNav-Sim enables researchers to develop and test algorithms for underwater environments efficiently.

I. INTRODUCTION

Marine robotics is an expanding field with numerous applications, including exploring underwater ecosystems and inspecting underwater infrastructure [1]. Recent developments in robotics and autonomy have demonstrated the superior capabilities of artificial intelligence (AI) and vision-based algorithms in solving complex tasks, such as drone racing [2], [3] and inspection [4]. These achievements have shown promise in developing AI and autonomy for marine applications as well [5]. To mitigate the high costs involved in developing and testing such algorithms, photorealistic simulation environments are needed that can accurately model the complexity of underwater scenarios [6].

In this context, this paper presents UNav-Sim, the first open-source underwater simulator based on unreal engine 5 (UE5) to create photorealistic environments (see Fig. 1). Compared to existing underwater robotics simulators, [7]–[10], UNav-Sim provides superior rendering quality, essential for the development of AI and vision-based navigation algorithms for underwater vehicles. It supports robotics tools such as ROS 2 and autopilot firmwares making it suitable for robotics research and development. The simulator uses the following open-source AirSim [11] extensions: [12] to add custom vehicle models to AirSim, and [13] for integration of AirSim to UE5. UNav-Sim can be used to simulate a wide range of underwater scenarios and models. The paper also

A. Amer, O. Tunon, H. Uğurlu, J. Sejersen are with the Department of Electrical Engineering and Computer Engineering, Aarhus University, 8200 Aarhus, Denmark {abdelhakim, olaya, halil, jonas} at ece.au.dk. Y. Brodskiy is with EIVA a/s, 8660 Skanderborg, Denmark. {ybr} at eiva.com. E. Kayacan is with the Automatic Control Group, Department of Electrical Engineering and Information Technology, Paderborn University, Paderborn, Germany. {erdal.kayacan} at uni-paderborn.de

¹<https://github.com/open-airlab/UNav-Sim>

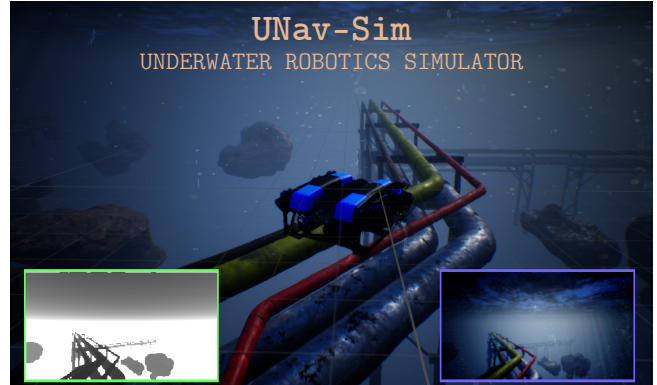


Fig. 1. UNav-Sim is an underwater robotics simulator utilizing Unreal Engine 5 (UE5) highly realistic environments. The simulator includes many features useful for roboticists, such as ROS 2, and a wide range of sensors and cameras. The bottom right displays the feed from a front-facing RGB camera, while the bottom left shows a corresponding depth image.

demonstrates its effectiveness for the development of vision-based localization and navigation methods for underwater robots.

The rest of the paper is structured as follows: Section II presents an overview of the state-of-the-art simulators and their respective capabilities. Section III describes the software architecture, physics, and models that comprise UNav-Sim. In Section IV, we describe a vision-based underwater navigation stack that was developed as a component of UNav-Sim. Then, we present a test case in Section V, where we showcase the abilities and features of our simulator in a vision-based pipe inspection scenario. Lastly, conclusions are drawn from this work in Section VI.

II. STATE-OF-THE-ART

Robotics simulation tools have significantly advanced in recent years, with a focus on providing high-fidelity and photorealistic visual rendering. IsaacSim [14], developed by Nvidia, is a recent example that includes both high-fidelity contact simulation and high-quality image rendering provided by Omniverse, making it suitable for simulating robotic grippers and walking robots. Another example is Microsoft’s AirSim [11], yet another popular robotics simulator, specifically designed for aerial vehicles. AirSim utilizes its Fastphysics engine for physics simulation and unreal engine 4 (UE4) for visualization.

While progress in robotics simulation tools has been rapid, underwater robotics simulation tools have lagged behind.

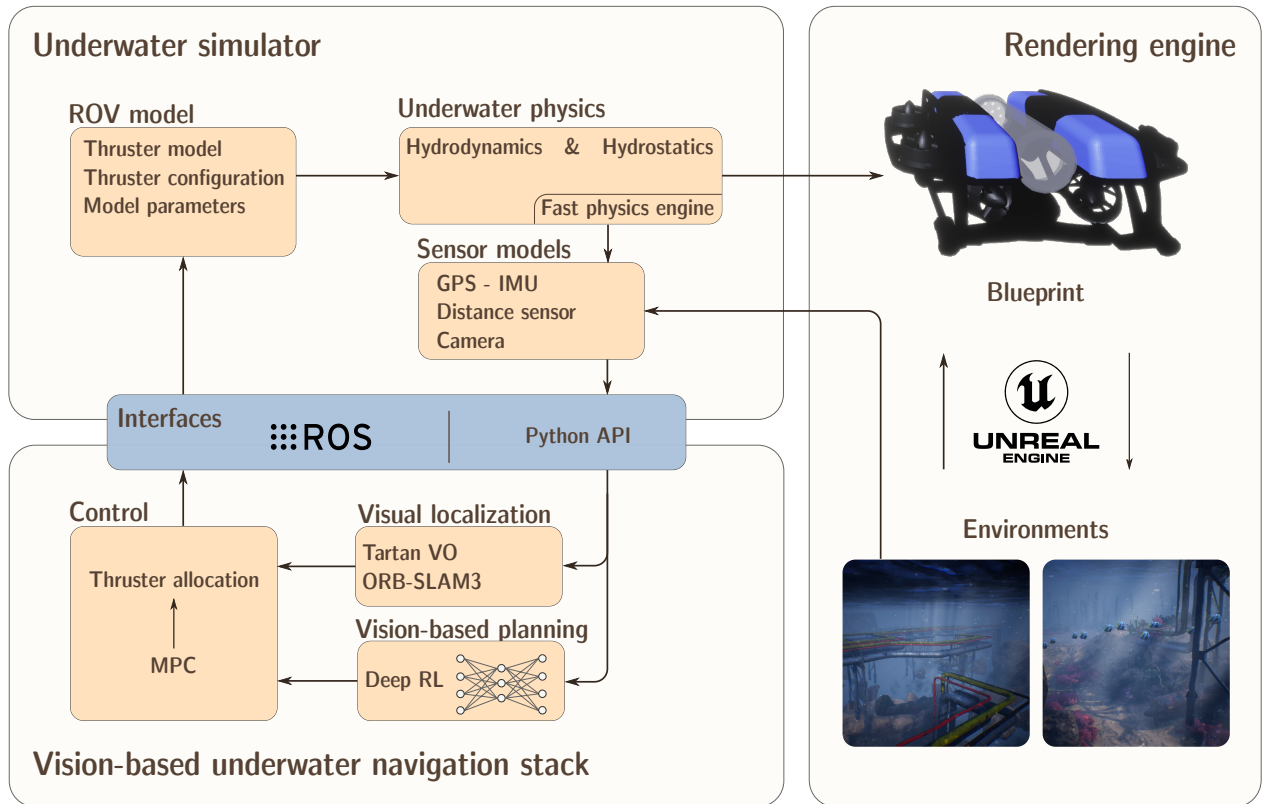


Fig. 2. UNav-Sim system architecture is designed to be modular, allowing flexibility in adapting the simulator to various underwater autonomy tasks. The system utilizes Unreal Engine 5 (UE5) to provide a high-fidelity rendering environment for increased photo-realism. A model predictive controller (MPC), combined with a deep reinforcement learning (DRL) planner and Visual SLAM are utilized for vision-based underwater navigation.

UWSim [15] and UUV Simulator [8] are the two most commonly used underwater simulators [16]; however, they are now discontinued. A more recent simulator, DAVE [9], was developed as a more modern version of the UUV simulator that supports more remotely operated vehicle (ROV) models and underwater sensors. However, the aforementioned simulators are based on Gazebo, which has the disadvantage of unrealistic rendering. This limits their usefulness for training and testing AI algorithms that often rely on image inputs. To address this issue, HoloOcean [7] was developed using UE4 for rendering and written in Python, but it lacks support for robot operating system (ROS) [17]. Another example is MARUS [10], which has not yet released its open-source implementation. The simulator uses Unity3D for visualization and integrates with ROS. However, it lacks support for essential robotics and AI tools, such as commercial autopilots [18], or OpenAI’s Gym environments [19], which are important tools for developing AI and control algorithms for autonomous vehicles.

A comparison of the capabilities of various open-source underwater robotics simulators, including the proposed simulator, is presented in Table I. Amongst all simulators evaluated, the present work, UNav-Sim, stands out for its superior rendering quality, achieved through the utilization of the unreal engine (UE)5 graphics engine. Additionally, UNav-

Sim supports a range of tools commonly used in developing robotics solutions, such as ROS, gym environments, and autopilot systems. Furthermore, UNav-Sim is compatible with both Windows and Linux operating systems.

III. UNAV-SIM SOFTWARE ARCHITECTURE

UNav-Sim is composed of three main components, as illustrated in Fig. 2: an underwater physics simulator, a state-of-the-art rendering engine, i.e. UE5, and an autonomy stack. The underwater physics simulator, which contains the lumped parameter ROV model and underwater dynamics equations, is modular and allows underwater vehicle motion simulation. It leverages the capabilities of AirSim, including the Fastphysics solver and a range of sensor models, such as GPS, IMU, cameras, and distance sensor. An API allows communication between the navigation stack and the physics simulator, with the former receiving essential sensor data and sending control commands. A ROS wrapper is also available, which enables ROS-based development and communication between different modules.

A. Underwater environment rendering

Underwater image formation can be modelled as a superposition of absorption, forward scattering, and backscattering effects at each pixel $\mathbf{x} = (u, v)$. The image intensity $I_c(x)$ in each color channel c can be expressed as [21]:

TABLE I
MARINE ROBOTICS SIMULATORS COMPARISON SHOWING UNAV-SIM'S SUPERIOR RENDERING QUALITY AND VERSATILITY.

Simulator	Year	Rendering quality	ROS Support	Autopilot	OS
UWSim [15]	2012	Low	ROS 1	None	Linux
UUV [8]	2016	Low	ROS 1	Ardupilot	Linux
URSim [20]	2019	Moderate	ROS 1	N/A	Linux
HoloOcean [7]	2022	High	N/A	N/A	Linux/Windows
DAVE [9]	2022	Low	ROS 1	PX4/Ardupilot	Linux
MARUS [10]	2022	Moderate	ROS 1,2	N/A	Linux/Windows
UNav-Sim (Ours)	2023	Highest	ROS 1,2	PX4/Ardupilot	Linux/Windows

$$I_c(x) = D_c(x) + F_c(x) + B_c(x) \quad (1)$$

In this equation, D_c represents the attenuated signal from the object due to absorption. The forward scattering component F_c captures the light from the object that reaches the camera with small-angle scattering. Lastly, the backscattering component B_c accounts for the degradation in color and contrast caused by the water scattering effect, where the light does not originate directly from the object. These effects can be modelled using different techniques and can vary based on the implementation within the rendering engine.

UNav-Sim utilizes UE5 as the rendering engine, which offers significant improvements over its predecessor, UE4. UE5 significantly boosts polygon handling to 10 billion, introduces real-time ray-based lighting with Lumen, and incorporates Temporal Super Resolution for high-quality textures with minimal performance impact, enhancing visual fidelity and efficiency.

UE5 underwater rendering module models scattering effects in underwater images (1), with Schlick Phase Functions [22], taking into account the Opaque or Masked water surface. The transparency of the water is implicitly handled within the volume shading model, and refraction is managed by reading the depth and color beneath the water surface to distort the samples. One of the main challenges in generating underwater renderings is the variety of imaging conditions that drastically change the environment's appearance [23]. UE5 allows users to define the scattering coefficients, absorption coefficients, phase function, and color scale behind the water, providing control over the water's appearance and thus allowing users to simulate their preferred environment's conditions.

Consequently, using UE5 within UNav-Sim, underwater environments that appear realistic can be created, where UE allows designers to place and manipulate assets in a 3D space. These assets can include terrain, static meshes, and lighting. They can be customized to create underwater virtual worlds, as shown in Fig. 2.

UE uses blueprints to define the physical representation and behaviour of an ROV. In UNav-Sim, the blueprint is linked to an external underwater physics engine to obtain kinematic information. The blueprint also defines cameras that gather visual information from the underwater environment, such as RGB and depth images.

B. Underwater physics

The core of the physics underlying underwater vehicles consists of the equations of motion that describe the different forces and moments acting on the vehicle's body. These forces and moments can be classified into three categories: hydrostatics, hydrodynamics, and externally applied forces.

The equation of motion in the body-fixed frame, originally presented in Fossen [24], can be expressed in SNAME notation [25] as,

$$\mathbf{M}_{RB}\dot{\nu} = \tau - \underbrace{\mathbf{C}_{RB}(\nu)\nu}_{\text{Coriolis term}} - \underbrace{\mathbf{M}_A\dot{\nu} - \mathbf{C}_A(\nu)\nu}_{\text{Added mass}} - \underbrace{\mathbf{D}(\nu)\nu}_{\text{Drag}} - \mathbf{g}(\eta). \quad (2)$$

The vehicle's pose, $\eta = [x, y, z, \phi, \theta, \psi]^T$, is described by a six-dimensional column vector where x , y , and z denote the vehicle's position in the North-East-Down (NED) frame, while ϕ , θ , and ψ represent its roll, pitch, and yaw angles, respectively (see Fig. 3). The linear and angular velocity vector in the body-fixed frame is denoted as $\nu = [u, v, w, p, q, r]^T$. The inertia matrix of the vehicle's body is represented by \mathbf{M}_{RB} . Hydrostatic forces, $\mathbf{g}(\eta)$, arise from gravity and buoyancy, along with associated moments and torques. Hydrodynamic forces arise from the interaction between the vehicle and the surrounding water and can significantly impact the vehicle's behavior. These forces include the Coriolis and centripetal forces caused by the rigid body's mass, $\mathbf{C}_{RB}(\nu)\nu$, the Coriolis forces, $\mathbf{C}_A(\nu)$, and moment of inertia, \mathbf{M}_A , arising from the added mass, and linear and quadratic damping effects, $\mathbf{D}(\nu)\nu$. External forces, τ , include the forces exerted on the ROV by its thrusters, as well as the disturbances, caused by the surrounding water flow. Multiple disturbance models, such as constant value, sinusoidal, and a combination of sine waves are implemented.

To efficiently solve the equations of motion presented above, AirSim's high-frequency physics engine is utilized with a computational frequency of 1000Hz. The engine uses the velocity verlet algorithm for numerical integration due to its computational benefits.

C. ROV model

The ROV is represented as a rigid body that is manipulated by an arbitrary number of actuators, N . These actuators are located at user-defined vertices of the vehicle, with

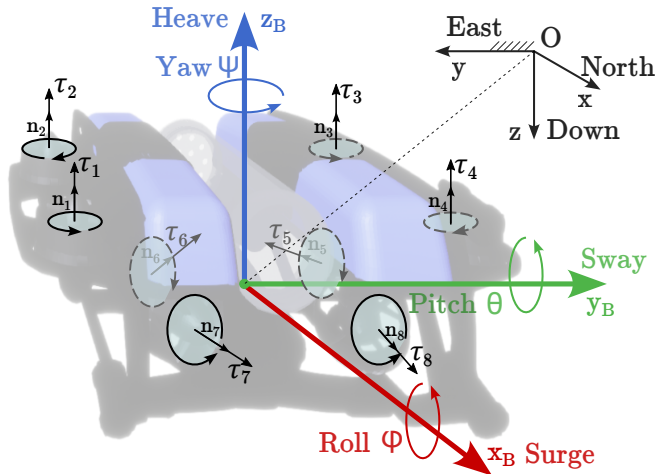


Fig. 3. ROV model defined in UNav-Sim: A force τ_i is applied at each thruster location i , where the thruster orientation is defined by a vector n_i .

corresponding normals and positions denoted by n_i and r_i , respectively, where $i \in \{1, \dots, N\}$ is the actuator number. At each vertex, the vehicle receives a unitless control input, $u_i \in (-1, 1)$. To account for actuator dynamics, a discrete low-pass filter with a time constant of t_c is applied to the control input. The filtered input, denoted as u_{fi} , is then used to calculate the thrust force using the relationship given as [11],

$$\tau_i = C_T \rho \omega_{max}^2 D^4 u_{fi}, \quad (3)$$

where τ_i is the thrust force on the i -th thruster, C_T is the thrust coefficient, ρ is the density of water, ω_{max} is the maximum thruster rotation speed, and D is the propeller diameter. In order to accurately compute the specific rigid motion of the ROV, as described by equations (2) and (3), several vehicle-specific parameters such as its inertia, hydrodynamic coefficients, and the maximum thruster rotation speed, must be configured by the user.

The model chosen to be implemented, the Blue Robotics BlueROV2 Heavy, is an over-actuated ROV with four vertical thrusters and four horizontal thrusters as shown in Fig. 3. The horizontal thrusters are oriented at 45 degrees and are responsible for the control of three degrees of freedom, namely, surge, sway, and yaw, while the vertical thrusters control heave, pitch, and roll. The model parameters are obtained from [26].

IV. VISION-BASED UNDERWATER NAVIGATION STACK

The vision-based underwater navigation stack of UNav-Sim includes planning, control, and SLAM. Many state-of-the-art AI algorithms in robotics, for example, as implemented in OpenDR toolkit [27] or PyPose library [28], primarily rely on vision for localization, planning, and control [29]. Therefore, in this study, we have chosen to utilize visual SLAM (VSLAM) and end-to-end deep reinforcement learning (DRL) algorithms to demonstrate the capabilities of the proposed simulator. To facilitate ease of integration with various autonomy algorithms and deployment on actual

hardware, all algorithms have been developed using ROS framework.

A. Vision-based planning

The recent developments in machine learning methods enable intelligent agents to learn navigation tasks end-to-end. Generally, a deep neural network policy takes sensory input, such as high-dimensional visual data, and generates feasible actions without explicitly mapping the environment. These learning-based methods require large amounts of data for training, which is impractical for real-world robotics systems. Hence, simulation environments are very substantial for enriching the required data in many cases [30]. Furthermore, DRL methods require demonstrations for exploration of the environment to learn a policy, which increases safety concerns for real-world learning [31]. Consequently, simulation environments with high-fidelity visual sensors and accurate physical dynamics are crucial for DRL research. OpenAI's gym [19] is a general standard for experimenting with learning-based sequential decision-making tasks. Therefore, we have provided a gym environment along with our simulator to augment its capabilities for benchmarking learning-based navigation algorithms.

B. Control

A model predictive control (MPC) strategy was utilized to control the ROV model, which consists of a two-step process involving an MPC followed by a control allocation algorithm. The motivation for the use of MPC is based on its ability to handle both input and state constraints explicitly, and intuitive tuning parameters [32]. The MPC is designed to solve an online optimization problem, aiming to determine the optimal body wrench forces and moments, given a particular robot pose and a desired reference pose.

The control allocation algorithm is then employed to obtain the individual control signal for each thruster by using a pseudo-inverse of an allocation matrix, which is vehicle-specific and depends on the thruster configuration of the ROV. ACADO toolkit [33] is utilized to implement the MPC as a ROS package, which is integrated into the ROS navigation stack.

C. Visual localization

Visual localization algorithms rely on cameras to retrieve the robot's state, which are widely used in the underwater robotics community [34]. Long-standing ROS packages for SLAM include `robot_localization`, which presents a classical filtering approach for sensor fusion [35], `hector_slam`, which implements occupancy grid maps for laser and IMU data [36], and `gmapping`, which leverages a Rao-Blackwellized particle filter for laser-based SLAM [37]. The availability of these state-of-the-art and ready-to-use algorithms has allowed outstanding progress in the robotics community [38]–[42], as they ease the implementation of future advances for SLAM and the benchmarking of their performance. However, the availability of off-the-shelf packages for visual SLAM remains an open problem.

Therefore, we propose `robot_visual_localization`, a ROS metapackage for deploying and benchmarking visual localization algorithms. We chose to implement the state-of-art methods ORB-SLAM3 [43] and TartanVO [44]. Each algorithm is implemented as a standalone ROS package within the `robot_visual_localization` metapackage, which takes as input the image stream, and outputs the camera trajectory and the map points for ORB-SLAM3, and the camera trajectory for TartanVO.

ORB-SLAM3 encompasses a geometry-based approach for SLAM. The ORB-SLAM3’s front end tracks ORB features across consecutive frames. The features are selected to be uniformly distributed across the image, and the matches search is performed according to a constant velocity model. The ORB-SLAM3’s back end builds a map with the sparse points tracked from the front end. Under tracking loss, the map is stored in memory as inactive, creating a new active map. The loop closure thread finds revisited areas under the active and inactive maps, merging them and propagating the accumulated drift.

Geometry-based algorithms such as ORB-SLAM3 still present the de-facto state-of-art for SLAM, due to their high precision and efficiency. However, they are highly dependent on feature detection and matching, and therefore sensitive to visual degradations such as repetitive patterns, textureless environments, and non-Lambertian surfaces. On the other hand, learning-based algorithms can be more robust against those challenging imaging conditions, but are highly dependent on the training data, and usually lack generalization ability.

The proposed ROS metapackage serves then as an accessible tool for the easy comparison of two of the main taxonomies in the SLAM’s state-of-art, which in the present work serves as a comparison of their performance under challenging underwater imaging conditions.

V. EXAMPLE USE-CASES

As a case study for UNav-Sim, we present an autonomous pipe inspection scenario. Pipe inspection, being the most common use case for ROVs, presents a relevant and challenging use case, where vision-based navigation is essential to achieve the required task. Furthermore, we assess the efficacy of our underwater autonomy stack and report its performance in executing the designated autonomous pipe inspection task. A video showing the pipe inspection demonstration can be found here².

A. Vision-based pipe following with DRL

In this experiment, the performance of UNav-Sim is evaluated in a pipe-following task. An agent utilizing DRL is trained with the gym interface provided by the simulator to generate position commands based on RGB image observations. An MPC controller subsequently executes the position commands. The convolutional neural network policy inputs 180×320 pixel RGB image observation, o_t , from a downward-looking camera on ROV and outputs an action, a_t ,

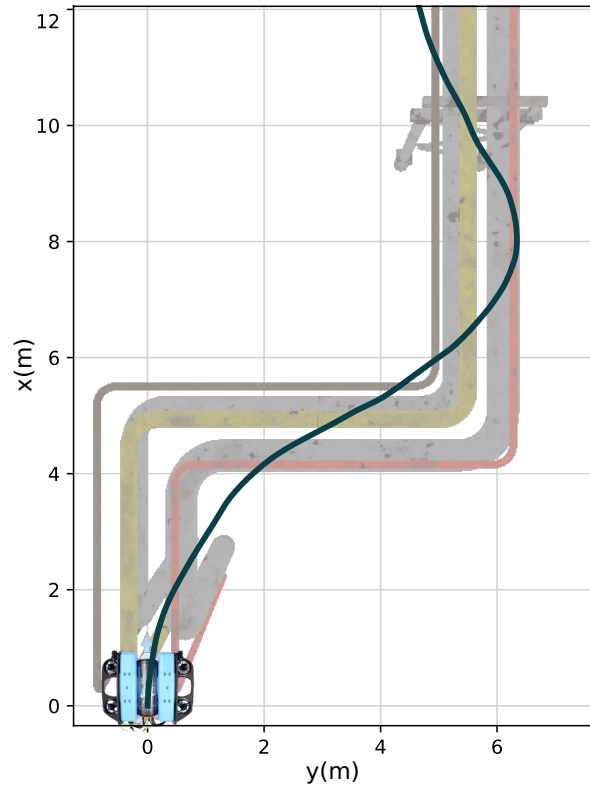


Fig. 4. Trajectory followed by the DRL agent (dark blue) along with the pipes from the top view. Starting point is the origin of the coordinate frame and is indicated with the ROV.

describing one meter away waypoint consisting of two values, $a_1, a_2 \in [-1, 1]$. The actions represent the direction of the position step and turn in the heading angle, respectively, similar to our previous work [31]. The reward is defined with respect to vertical divergence from the pipe unless the termination of episodes; $r_t = 10 - 2e_p^2 - 2e_\psi$ where e_p is the closest distance to the pipe in the horizontal plane and e_ψ is the error in heading with respect to pipe direction. An episode is terminated where the pipes are not in the camera’s field of view, which is $e_p > 2.5m$. The DRL agent is trained with proximal policy optimization [45] algorithm using `stable-baselines3` [46] package.

The trained policy is deployed on a pipe ~ 20 meters long with right and left turns. The trajectory of the agent along with the pipes is visualized in Fig. 4. While the agent is not accurately tracking the pipes due to the exploratory behavior of the DRL policy, it learns to make reasoning from image observations and successfully follows the pipe. This experiment demonstrates the utilization of high-fidelity image observations and accurate dynamics provided by UNav-Sim in a particular application.

B. Visual localization benchmarking

The vision-based trajectory generated in Section V-A, being carried out by the robot with the controller showcased in Section IV-B, has served as a test setup for the visual localization experiment. Two trajectories are carried out: a linear

²<https://youtu.be/unZS331CqpU>

trajectory where no area in the map is revisited (see Fig. 4), and a trajectory with a loop, where the robot navigates back to the starting point. The benchmarking is automatically performed by the `robot_visual_localization` metapackage using [47].

During runtime, the estimated trajectory P_i and the ground truth trajectory Q_i are recorded into separate files composing a sequence of time-synchronized spatial poses. The pose format is the one proposed in [48], composed of the three spatial coordinates with the orientation in quaternions. The metrics implemented are the absolute positioning error (APE) and the relative positioning error (RPE), which are automatically deployed over the recorded trajectories before shutdown. TartanVO presents a monocular visual odometry algorithm. Therefore, for a fair comparison, the ORB-SLAM3 experiment is executed under a monocular setup. The deployment of monocular algorithms implies that the orientation of the algorithm’s world frame and the trajectory’s scale is arbitrary. Therefore, the estimated trajectories are aligned with the ground truth by obtaining the transform $S \in Sim_3$ that best aligns P_i with Q_i .

With the deployment of the automatic stack for visual inspection proposed by UNav-Sim, benchmarking of visual localization algorithms becomes a straightforward task: the robot follows the pipeline autonomously under the planned trajectory, with the visual localization algorithms being automatically executed by the `robot_visual_localization` package, which on shutdown generates the results as depicted in Table II. It can be seen from the generated results that the two proposed algorithms depict a similar performance in the linear trajectory, but ORB-SLAM outperforms TartanVO under the presence of a closed loop. Despite ORB-SLAM’s high efficiency in state-of-the-art datasets, the realistic underwater conditions confront one of the main challenges for feature-based approaches: the lack of texture. Moreover, the pipes are the main source of features, which avoids their uniform distribution across the image. Without enough evenly-distributed features, the ORB-SLAM’s front end drifts. Nevertheless, the closed trajectory shows the convenience of the back-end’s loop closure algorithm: the absolute errors are significantly reduced for translation and rotation. On the other hand, TartanVO presents a drift similar to ORB-SLAM’s in the translations, but slightly higher for rotations. These results show the great potential of learning-based algorithms under imaging conditions that challenge geometry-based methods. Although the lack of generalization ability is the main source of drift in this case, TartanVO has been trained with high amounts of diversified data that explain its good performance in the proposed setup.

In conclusion, the framework proposed in UNav-Sim has enabled the automatic benchmarking of state-of-the-art visual localisation algorithms in a realistic underwater scenario. This has allowed the challenges and opportunities of these algorithms to be easily demonstrated in a geometry-based and learning-based manner.

TABLE II

VISUAL LOCALIZATION RESULTS IN THE PIPELINE TRACKING SCENARIO.

Trajectory	Algorithm	APE[m]	RPE[m]	APE[rad]	RPE[rad]
Linear	ORB-SLAM3	1.75	0.412	1.53	0.036
	TartanVO	1.67	0.489	1.95	0.108
With loop	ORB-SLAM3	0.078	0.372	1.62	0.006
	TartanVO	0.961	0.322	2.10	0.068

VI. CONCLUSION & FUTURE WORK

We have developed UNav-Sim, a novel open-source underwater simulator, which builds upon AirSim and UE5 and incorporates state-of-the-art robotics algorithms. UNav-Sim also supports ROS and multiple operating systems, facilitating a streamlined and efficient development process for robotics applications. Future work will include the incorporation of additional underwater sensors, vehicle models, and more custom environments.

ACKNOWLEDGEMENT

This work is supported by EIVA a/s and Innovation Fund Denmark under grants 2040-00032B and 1044-00007B, the European Union’s Horizon 2020 Research and Innovation Program (OpenDR) under Grant 871449 and the Marie Skłodowska-Curie (REMARO) under Grant 956200. This publication reflects the authors’ views only. The European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

- [1] Robert Bogue. Underwater robots: a review of technologies and applications. *Industrial Robot: An International Journal*, 2015.
- [2] Efe Camci, Domenico Campolo, and Erdal Kayacan. Deep reinforcement learning for motion planning of quadrotors using raw depth images. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.
- [3] Huy Xuan Pham, Ilker Bozcan, Andriy Sarabakha, Sami Haddadin, and Erdal Kayacan. Gatenet: An efficient deep neural network architecture for gate perception using fish-eye camera in autonomous drone racing. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4176–4183. IEEE, 2021.
- [4] Mohit Mehndiratta and Erdal Kayacan. Gaussian process-based learning control of aerial robots for precise visualization of geological outcrops. In *2020 European Control Conference (ECC)*, pages 10–16. IEEE, 2020.
- [5] Leif Christensen, José de Gea Fernández, Marc Hildebrandt, Christian Ernst Siegfried Koch, and Bilal Wehbe. Recent advances in ai for navigation and control of underwater robots. *Current Robotics Reports*, pages 1–11, 2022.
- [6] Olaya Álvarez Tuñón, Hemanth Kanner, Luiza Ribeiro Marnet, Huy Xuy Pham, Jonas le Fevre Sejersen, Yury Brodskiy, and Erdal Kayacan. Mimir-uw: A multipurpose synthetic dataset for underwater navigation and inspection. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.
- [7] Easton Potokar, Spencer Ashford, Michael Kaess, and Joshua G Mangelson. Holocean: An underwater robotics simulator. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3040–3046. IEEE, 2022.
- [8] Musa Morena Marcusso Manhães, Sebastian A Scherer, Martin Voss, Luiz Ricardo Douat, and Thomas Rauschenbach. Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–8. IEEE, 2016.

- [9] Mabel M Zhang, Woen-Sug Choi, Jessica Herman, Duane Davis, Carson Vogt, Michael McCarrin, Yadunund Vijay, Dharini Dutia, William Lew, Steven Peters, et al. Dave aquatic virtual environment: Toward a general underwater robotics simulator. In *2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*, pages 1–8. IEEE, 2022.
- [10] Ivan Lončar, Juraj Obradović, Natko Kraševac, Luka Mandić, Igor Kvasić, Fausto Ferreira, Vladimir Slošić, Đula Nađ, and Nikola Mišković. Marus-a marine robotics simulator. In *OCEANS 2022, Hampton Roads*, pages 1–7. IEEE, 2022.
- [11] Shital Shah, Debadepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.
- [12] BYU. byu vtol. <https://github.com/byu-magicc/vtol-AirSim>, 2013.
- [13] CodexLabsLLC. Colosseum. <https://github.com/CodexLabsLLC/Colosseum>, 2013.
- [14] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapon Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.
- [15] Mario Prats, Javier Perez, J Javier Fernandez, and Pedro J Sanz. An open source tool for simulation and supervision of underwater intervention missions. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 2577–2582. IEEE, 2012.
- [16] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.
- [17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.
- [18] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015.
- [19] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [20] Pushkal Katara, Mukul Khanna, Harshit Nagar, and Annapurani Panaiyappan. Open source simulator for unmanned underwater vehicles using ros and unity3d. In *2019 IEEE Underwater Technology (UT)*, pages 1–7. IEEE, 2019.
- [21] Olaya Álvarez-Tuñón, Alberto Jardón, and Carlos Balaguer. Generation and processing of simulated underwater images for infrastructure visual inspection with uavs. *Sensors*, 19(24):5497, 2019.
- [22] Philippe Blasi, Bertrand Le Saec, and Christophe Schlick. A rendering algorithm for discrete volume density objects. In *Computer Graphics Forum*, volume 12, pages 201–210. Wiley Online Library, 1993.
- [23] Derya Akkaynak, Tali Treibitz, Tom Shlesinger, Yossi Loya, Raz Tamir, and David Iluz. What is the space of attenuation coefficients in underwater computer vision? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4931–4940, 2017.
- [24] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.
- [25] TheSocietyofNavalArchitectureandMarineEngineers SNAME. Nomenclature for treating the motion of a submerged body through a fluid. *The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin*, pages 1–5, 1950.
- [26] Chu-Jou Wu. *6-dof modelling and control of a remotely operated vehicle*. PhD thesis, Flinders University, College of Science and Engineering., 2018.
- [27] S Pedrazzi, D Dias, F Ferro, O Green, E Kayacan, et al. Opendr: An open toolkit for enabling high performance, low footprint deep learning for robotics. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12479–12484. IEEE, 2022.
- [28] Chen Wang, Dasong Gao, Kuan Xu, Junyi Geng, Yaoyu Hu, Yuheng Qiu, Bowen Li, Fan Yang, Brady Moon, Abhinav Pandey, et al. Pypose: A library for robot learning with physics-based optimization. *arXiv preprint arXiv:2209.15428*, 2022.
- [29] Huy Xuan Pham, Halil Ibrahim Ugurlu, Jonas Le Fevre, Deniz Bardakci, and Erdal Kayacan. Deep learning for vision-based navigation in autonomous drone racing. In *Deep learning for robot perception and cognition*, pages 371–406. Elsevier, 2022.
- [30] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.
- [31] Halil Ibrahim Ugurlu, Xuan Huy Pham, and Erdal Kayacan. Sim-to-real deep reinforcement learning for safe end-to-end planning of aerial robots. *Robotics*, 11(5):109, 2022.
- [32] Utku Eren, Anna Prach, Başaran Bahadır Koçer, Saša V Raković, Erdal Kayacan, and Behçet Açıkmeşe. Model predictive control in aerospace systems: Current state and opportunities. *Journal of Guidance, Control, and Dynamics*, 40(7):1541–1566, 2017.
- [33] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.
- [34] Olaya Álvarez-Tuñón, Yury Brodskiy, and Erdal Kayacan. Monocular visual simultaneous localization and mapping:(r) evolution from geometry to deep learning-based pipelines. *IEEE Transactions on Artificial Intelligence*, 2023.
- [35] Thomas Moore and Daniel Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*, pages 335–348. Springer, 2016.
- [36] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE international symposium on safety, security, and rescue robotics*, pages 155–160. IEEE, 2011.
- [37] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2432–2437. IEEE, 2005.
- [38] Leyao Huang. Review on lidar-based slam techniques. In *2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*, pages 163–168. IEEE, 2021.
- [39] Naigong Yu and Bo Zhang. An improved hector slam algorithm based on information fusion for mobile robot. In *2018 5th IEEE International conference on cloud computing and intelligence systems (CCIS)*, pages 279–284. IEEE, 2018.
- [40] Weichen Wei, Bijan Shirinzadeh, Shunmugasundar Esakkiappan, Mohammadali Ghafarian, and Ammar Al-Jodah. Orientation correction for hector slam at starting stage. In *2019 7th International Conference on Robot Intelligence Technology and Applications (RiTA)*, pages 125–129. IEEE, 2019.
- [41] Yassin Abdelrasoul, Abu Bakar Sayuti HM Saman, and Patrick Sebastian. A quantitative study of tuning ros gmapping parameters and their effect on performing indoor 2d slam. In *2016 2nd IEEE international symposium on robotics and manufacturing automation (ROMA)*, pages 1–6. IEEE, 2016.
- [42] BLEA Balasuriya, BAH Chathuranga, BHMD Jayasundara, NRAC Napagoda, SP Kumarawadu, DP Chandima, and AGBP Jayasekara. Outdoor robot navigation using gmapping based slam algorithm. In *2016 moratuwa engineering research conference (mercon)*, pages 403–408. IEEE, 2016.
- [43] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.
- [44] Wenshan Wang, Yaoyu Hu, and Sebastian Scherer. Tartanvo: A generalizable learning-based vo. In *Conference on Robot Learning*, pages 1761–1772. PMLR, 2021.
- [45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [46] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [47] Michael Grupp. evo: Python package for the evaluation of odometry and slam. <https://github.com/MichaelGrupp/evo>, 2017.
- [48] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580. IEEE, 2012.