IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 April 2026, accepted 18 April 2026, date of publication 23 April 2026, date of current version 29 April 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3687084

RESEARCH ARTICLE

ASVSim (AirSim for Surface Vehicles): A High-Fidelity Simulation Framework for Autonomous Surface Vehicle Research

BAVO LESY[1], SIEMEN HERREMANS[1], ROBIN KERSTENS[2,3], JAN STECKEL[2,3], (Member, IEEE), WALTER DAEMS[2,3], (Senior Member, IEEE), SIEGFRIED MERCELIS[1], AND ALI ANWAR[1], (Member, IEEE)

[1]IDLab, Faculty of Applied Engineering, IMEC, University of Antwerp, 2000 Antwerp, Belgium
[2]Cosys-Laboratory, Faculty of Applied Engineering, University of Antwerp, 2020 Antwerp, Belgium
[3]Flanders Make Strategic Research Centre, 3920 Lommel, Belgium

Corresponding author: Bavo Lesy (bavo.lesy@uantwerpen.be)

This work was supported in part by the Horizon 2020 PIONEERS Project, funded by the European Commission under Agreement 101037564; in part by the HORIZON INNO2MARE project, funded by the European Commission under Agreement 101087348; and in part by the Research Foundation Flanders (FWO) under Grant 1SHAI24N.

ABSTRACT The transport industry has recently shown significant interest in unmanned surface vehicles (USVs), specifically for port and inland waterway transport. These systems can improve operational efficiency and safety, which is especially relevant in the European Union, where initiatives such as the Green Deal are driving a shift towards increased use of inland waterways. At the same time, a shortage of qualified personnel is accelerating the adoption of autonomous solutions. However, there is a notable lack of open-source, high-fidelity simulation frameworks and datasets for developing and evaluating such solutions. To address these challenges, we introduce AirSim for Surface Vehicles (ASVSim), an open-source simulation framework specifically designed for autonomous shipping research in inland and port environments. The framework combines simulated vessel dynamics with marine sensor simulation capabilities, including radar and camera systems and supports the generation of synthetic datasets for training computer vision models and reinforcement learning (RL) agents. Built upon Cosys-AirSim, ASVSim provides a comprehensive platform for developing autonomous navigation algorithms and generating synthetic datasets. The simulator supports research of both traditional control methods and deep learning-based approaches. Through experiments in waterway segmentation and autonomous navigation, we demonstrate the capabilities of the simulator in these research areas. ASVSim is provided as an open-source project under the MIT license, making autonomous navigation research accessible to a larger part of the ocean engineering community. See https://github.com/BavoLesy/ASVSim

INDEX TERMS Autonomous surface vehicles, computer vision, deep learning, open-source, path planning, reinforcement learning, vessel simulation.

I. INTRODUCTION

There is a significant research effort concerning autonomous navigation systems for USVs [1], [2]. However, developing and testing autonomous navigation systems in real-world environments is challenging due to high costs, safety risks, and the complex nature of maritime operations. For inland waterways, the complexity is further increased by narrow passages, locks, and dense traffic patterns and a frequent need to observe and avoid obstacles through the use of sensors. Infrastructure such as bridges, ports, and shore-based facilities may lead to sensor occlusion, creating blind spots that autonomous systems must account for. Additionally, varying wind and current conditions can significantly impact

The associate editor coordinating the review of this manuscript and approving it for publication was Wenbing Zhao.

a vessel, requiring robust control strategies to maintain safe navigation. Because of these challenges, simulation has emerged as a crucial tool for developing and validating navigation algorithms, in a safe and cost-effective manner [3], [4], [5]. For these purposes, it is important that the simulation closely represents the real world to reduce the effort needed to deploy simulated solutions on real vessels. However, a current gap exists in the availability of shipping simulators for autonomy research, with most existing simulators either being designed for human operator training, being proprietary, or lacking the visual realism (or fidelity) needed for accurate camera and sensor simulation. More recently, there is an increasing interest in the potential of deep learning methods in a shipping context, such as obstacle detection [6] or Reinforcement Learning (RL) for navigation [7]. RL has emerged as a promising approach for USVs, partly because of its ability to generalize across different vessel types and operating conditions [8]. However, the USV domain currently lacks accessible, high-fidelity simulation software that can accurately model vessel dynamics and generate realistic sensor data for development of autonomous systems that perceive their environment and navigate within it.

The main contribution of this paper is ASVSim,[1] an open-source, high-fidelity simulator specifically designed for autonomous shipping research. In the context of simulation, we use the term high-fidelity to refer to three complementary dimensions. First, photorealistic visual rendering enables the generation of realistic synthetic data for perception research (Sections V-A and VI-A). Second, physics-based vessel dynamics use the Fossen maneuvering model to accurately reproduce vessel motion (Section III). Third, sensor simulation provides camera rendering and a radar emulation that approximates marine radar imagery from LiDAR point clouds using a point spread function (PSF) (Section IV). The most notable modifications and extensions to the underlying Cosys-AirSim [9] framework are summarized in Fig. 1. ASVSim is tailored for Inland Waterway Transport (IWT), where the focus is more on the navigation of vessels through narrow passages and locks, and less on the dynamics of the vessel itself. The simulator currently provides 3-degree-of-freedom (3DoF) maneuvering dynamics, sensor simulations, and environmental conditions such as wind and currents. This enables researchers to develop, train, and test autonomous navigation algorithms in a configurable setting. As will be demonstrated throughout this work, ASVSim allows for the creation of high-fidelity environments, essential for image-based sensor research. Because of this visual fidelity, ASVSim can be used to generate realistic synthetic datasets to train computer vision models. We demonstrate how the simulator can be used for training and validation of autonomy solutions, using examples such as dataset collection and waterway segmentation. Autonomous navigation tasks are demonstrated as use-cases of the simulator, including

[1] https://github.com/BavoLesy/ASVSim

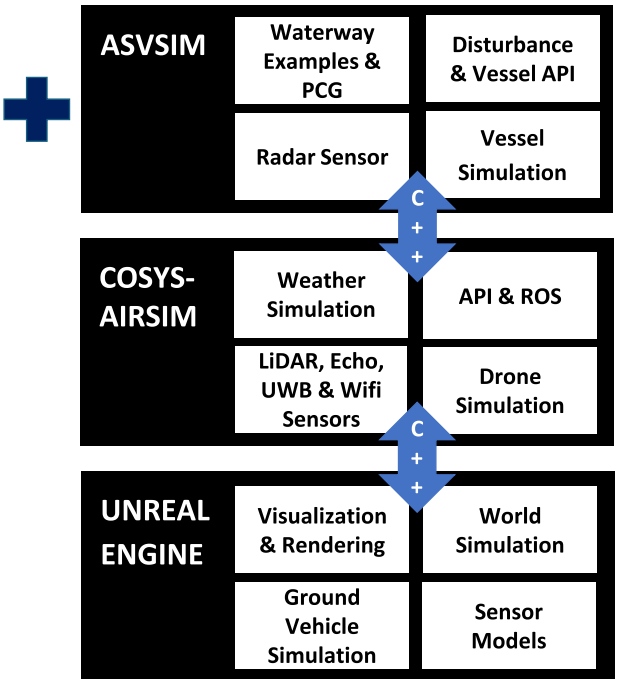


FIGURE 1. Overview of the ASVSim framework, showing the extensions and modifications to Cosys-AirSim. The simulator integrates specialized maritime dynamics models, radar simulation, and other features for autonomous vessel research.

autonomous channel navigation and the training of an RL agent to navigate a vessel to a goal whilst avoiding obstacles.

The paper is structured as follows: in Section II, we review related works in the field of shipping simulators. In Section III and IV, we describe the mathematical models underlying ASVSim, covering vessel dynamics and radar simulation. In Section V, we present the ASVSim framework in detail, including its dataset generation capabilities, procedural environment generation, and the Python and Matlab API for vessel control. In Section VI, we present the results of our autonomy experiments, covering waterway segmentation and path planning. In Section VII, we discuss the results and outline future research directions. Finally, in Section VIII, we provide a conclusion.

II. RELATED WORKS

Mission simulation has long been an essential tool in the shipping industry, primarily for training and certification of maritime personnel. The evolution of these simulators has been driven by the need for increasingly realistic training environments that can prepare personnel for operating modern vessels. Industry-standard simulators such as Kongsberg K-Sim [10] and Nautis [11] have become more and more present in maritime education, providing highly realistic environments with accurate vessel dynamics, environmental conditions, and operational scenarios. These commercial systems excel in their primary use case: training maritime professionals in vessel handling, navigation, and emergency

TABLE 1. Comparison of surface vessel simulators. Visual: camera-based perception sensors. Ranging: LiDAR and/or radar sensors.

Simulator	Engine	Visual Fidelity	Dynamics	Open-Source	Visual	Ranging	Primary Usage
Kongsberg K-Sim [10]	Proprietary	3D	Proprietary	×	×	×	Maritime training
Nautis [11]	Proprietary	3D (Photorealistic)	Proprietary	×	×	×	Maritime training
Wärtsilä R&D Sim. [12]	Proprietary	3D	Proprietary	×	×	×	Autonomy research
SIMTECH ASHSS [13]	Proprietary	3D (Photorealistic)	Proprietary	×	×	×	Autonomy research
AILiveSim [14]	UE	Photorealistic	Proprietary	×	✓	✓	Autonomy research
MSS [15]	Matlab	None	Fossen	✓	×	×	Control systems
uSimMarine [16]	C++	2D	Simple	✓	×	×	Autonomy research
VRX [17]	Gazebo	3D	Simple	✓	✓	✓	Autonomy research
ASVSim (ours)	UE5	3D (Photorealistic)	Fossen	✓	✓	✓	Autonomy research

procedures across various vessel types and maritime conditions. These simulators offer realistic physics models and visualization of maritime scenarios with detailed rendering of weather conditions and accurate representations of navigational sensors and port infrastructure. Many include replicas of actual bridge equipment and instrumentation, closely mirroring real-world vessel operations. However, as these systems were designed primarily for human operators, they are not directly suitable for autonomy research. Additionally, these simulators are mostly closed-source and licenses can be expensive.

The limitations of these traditional training simulators have become apparent as the maritime industry increasingly embraces automation. While platforms such as SIMTECH [13] and the Wärtsilä R&D Simulator [12] have begun incorporating features for autonomous vessel development, their proprietary nature and commercial focus create substantial barriers for academic research and open innovation. Entertainment-oriented simulators such as European Ship Simulator [18], though more accessible, lack the physical accuracy and sensor simulation capabilities essential for meaningful autonomy research. AILiveSim [14], built on Unreal Engine, provides camera, LiDAR, and radar sensor simulation for autonomous vessel development, but is not open-source. Several open-source maritime simulators have emerged to address the research gap in autonomous navigation, but each has significant limitations. MOOS-IVP (uSimMarine) [16], a widely used open-source platform, provides a lightweight framework for autonomous vessel control with a publish-subscribe communication architecture that makes it straightforward to integrate new autonomy modules. However, it relies on simplified vessel dynamics, 2D top-down visualization (via pMarineViewer), and lacks native ranging sensor simulation (e.g., radar or LiDAR). While data can in principle be logged through the MOOS publish-subscribe system, the 2D visualization and absence of 3D sensor modalities make it unsuitable for generating realistic synthetic perception datasets. These properties make uSimMarine well suited for rapid prototyping and lightweight control experiments where visual realism is not required, but limit its applicability to perception-driven and learning-based autonomy research. Similarly, the Marine Systems Simulator (MSS) [15], a Matlab/Simulink-based library for marine control systems development, offers elaborate physics models for various vessel types (ships, underwater vehicles, and unmanned underwater vehicles (UUVs)), but lacks integrated sensor simulation and comprehensive visualization capabilities. These limitations significantly constrain the effectiveness of both simulators for developing and training autonomous maritime navigation systems. VRX [17], the Virtual RobotX simulator built on Gazebo, provides an open-source platform for USV autonomy research with ROS integration, 3DoF dynamics with wave and wind plugins, and camera and LiDAR sensors, but lacks radar simulation and photorealistic rendering. Simultaneously, the field of autonomous driving has benefited greatly from several open-source simulators with high visual fidelity such as CARLA [19] and Cosys-AirSim [9]. These simulators excel at replicating real-world conditions through detailed graphics, precise physics models, and advanced sensor simulation. Such high-fidelity environments allow algorithms that were developed in simulation to transfer more effectively to real-world applications [20]. Another benefit of this fidelity is the ability and flexibility to effectively recreate real-world scenarios. Simulators such as Stonefish [21], OceanSim [22] and NauSim [23], offer realistic underwater environments and diverse sensor capabilities for UUV development and testing. Additionally, UNav-Sim [24] and HoloOcean [25] leverage the advanced capabilities of Unreal Engine [26] to further enhance simulation realism. However, these underwater-focused simulators are not suitable for inland shipping applications, as they primarily model underwater physics and UUV models and dynamics, in addition to the absence of dynamic obstacles and inland waterway environments.

Therefore, as summarized in Table 1, a critical gap exists in the inland shipping domain particularly for USVs: the lack of open-source, high-fidelity simulators that have the physical realism and the functionality required for inland autonomy research. Notably, this shortage is also present in high-quality datasets, with maritime data typically remaining limited, fragmented, or proprietary. For an overview of marine datasets, the reader is referred to [27] and [28]. ASVSim addresses these challenges by providing an open-source

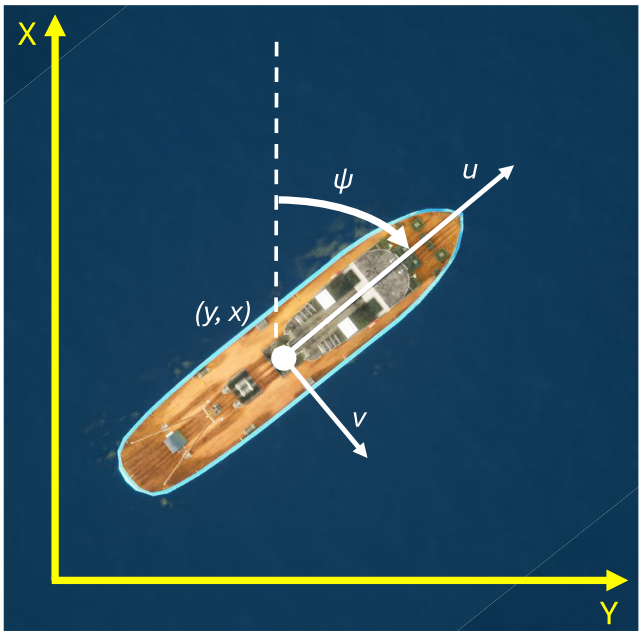


FIGURE 2. Visualization of the Fossen vessel model used in the simulation. The vessel has three degrees of freedom: surge, sway, and yaw.

simulator with high visual fidelity, serving the needs of autonomous navigation research, with particular attention to inland waterways and ports. Our platform prioritizes ease of use for deep learning and robotics research applications. Furthermore, the simulator can generate comprehensive, diverse datasets essential for training and validating autonomy solutions.

III. DYNAMICS SIMULATION

Since the main focus of this paper is inland waterway navigation, we opt for a 3DoF model, ignoring the heave, roll and pitch of the vessel. The 3DoF maneuvering model is the standard formulation for inland vessels, where sheltered, calm water conditions make vertical dynamics negligible [3]. As a result, wave excitation forces, vessel responses in the vertical plane, and effects such as propeller-hull interactions are not captured. Extending the simulator to 6DoF is architecturally straightforward, but was not pursued because 6DoF models introduce substantially more hydrodynamic parameters, making the system underdetermined without extensive experimental data for identification. As a result, published 6DoF parameter sets for inland vessel types are rarely available in the literature. Researchers requiring 6DoF dynamics can extend the model by providing the additional mass, damping, and Coriolis matrices for heave, roll, and pitch. The impact on sim-to-real transfer is limited, as the same Fossen 3DoF model is widely used for real-world maritime control system design [15]. Specifically, we implement the 3DoF maneuvering model (see Fig. 2), as presented by [3]. This model allows us to express the surge, sway and rotational velocity of the vessel, based on the force inputs and hydrodynamical constants that define the mass, damping and Coriolis and centripetal effects. Throughout this work, we will be using bold notation for vectors (e.g. $\boldsymbol{x}$) and the dot notation for the derivatives of these vectors (e.g. $\dot{\boldsymbol{x}}$). Additionally, matrices are denoted as capitalized bold symbols (e.g. $\boldsymbol{M}$).

A. KINEMATICS

We describe the vessel's position with its positional coordinates and heading (in the Earth-fixed frame), $\boldsymbol{\eta} = [x, y, \psi]$. Secondly, we include the surge, sway and rotational velocity using $\boldsymbol{\nu} = [u, v, r]$ in the body-fixed frame. The kinematics of these vectors are described by (1).

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}_{rot}(\boldsymbol{\psi})\boldsymbol{\nu}$$
$$\boldsymbol{R}_{rot}(\boldsymbol{\psi}) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Furthermore, we allow for currents, which are described using a velocity vector $\dot{\boldsymbol{\eta}}_c$ in the Earth-fixed frame. These can be translated to a local-frame current velocity vector $\boldsymbol{\nu}_c$ via the (orthonormal) rotation matrix:

$$\boldsymbol{\nu}_c = \boldsymbol{R}_{rot}(\boldsymbol{\psi})^T \dot{\boldsymbol{\eta}}_c \tag{2}$$
$$\dot{\boldsymbol{\eta}}_c = [V_c \cdot cos(\beta_c), V_c \cdot sin(\beta_c), 0]^T \tag{3}$$

where V_c describes the velocity of the current (in m/s) and β_c denotes the heading of the current in the Earth-fixed frame (in radians). Finally, the relative velocity vector $\boldsymbol{\nu}_r$ is defined as the difference between the velocity of the ego vessel and the velocity of the current.

$$\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c \tag{4}$$

B. DYNAMICS

The dynamics are specified by five vessel-dependent 3×3 matrices. $\boldsymbol{M}_{RB}$ and $\boldsymbol{M}_A$ are the rigid-body and hydrodynamical mass matrices, defining the (added) mass of the vessel, the centers of gravity and the inertia around the yaw axis. $\boldsymbol{C}_{RB}(\boldsymbol{\nu})$ and $\boldsymbol{C}_A(\boldsymbol{\nu}_r)$ define the rigid body and hydrodynamical Coriolis (and centripetal) matrices. Finally, $\boldsymbol{D}(\boldsymbol{\nu}_r)$ represents the damping force acting on the vessel. Importantly, $\boldsymbol{D}(\boldsymbol{\nu}_r)$ is allowed to contain both linear and non-linear damping elements, such as u^2, v^2 and r^2. Equation (5) fully describes the dynamics, subject to an external control vector $\boldsymbol{\tau}$ and wind $\boldsymbol{\tau}_{wind}$.

$$\boldsymbol{M}_{RB}\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{M}_A\dot{\boldsymbol{\nu}}_r + \boldsymbol{C}_A(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \boldsymbol{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r = \boldsymbol{\tau} + \boldsymbol{\tau}_{wind} \tag{5}$$

Many practical models ignore currents (i.e. $\boldsymbol{\nu}_c = \boldsymbol{0}$, where $\boldsymbol{0}$ denotes a vector containing only zeros) [29], [30], allowing a simplification of the dynamics by defining $\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A$ and $\boldsymbol{C} = \boldsymbol{C}_{RB} + \boldsymbol{C}_A$. This simplified model (ignoring currents) is described in (6).

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} + \boldsymbol{\tau}_{wind} \tag{6}$$

As we believe that currents and wind are important factors to consider in autonomous navigation research, we include these currents, under the assumption that they are irrotational and (approximately) constant. These assumptions are commonly made when modeling vessels, since they allow for practical parameter estimation on real-world vessels. More details on this are provided in Section III-D.

C. CONTROL FORCES

The simulator allows to place an arbitrary amount of thrusters. Each of them is controlled by setting a thrust force F_i (in Newton) and the desired angle θ_i (where i is the index of the thruster). Additionally, the position of each thruster can be configured using d_{x_i} and d_{y_i}, which represent the position of the thruster with regard to the center of gravity (COG). For example, a stern thruster can be created by placing it at the back of the vessel and limiting the minimal and maximal turning angle. Similarly, a bow thruster can be created by adding a thruster at the front of the vessel, and fixing its angle. We model the force of the thrusters on the vessel as follows:

$$\boldsymbol{\tau} = \begin{bmatrix} \cos(\theta_1) & \dots & \cos(\theta_n) \\ \sin(\theta_1) & \dots & \sin(\theta_n) \\ d_{x_1}\sin(\theta_1) - d_{y_1}\cos(\theta_1) & \dots & d_{x_n}\sin(\theta_n) - d_{y_n}\cos(\theta_n) \end{bmatrix} \times \boldsymbol{f} \quad (7)$$

where $\boldsymbol{f} = [F_1, F_2, \dots, F_n]^T$ represents the vector containing all thruster forces. Similarly, $\boldsymbol{\theta} = [\theta_1, \theta_2, \dots, \theta_n]$ represents all thruster angles (in the local frame). The control input vector $\boldsymbol{\tau}$ contains the surge, sway and yaw components that are added to the system.

D. CURRENTS

Equation (5) already includes the effect of currents, since it is parameterized on both $\boldsymbol{\nu}$ and $\boldsymbol{\nu_r}$. However, separate parameter estimation for the rigid-body matrices (such as $\boldsymbol{M_{RB}}$) and the hydrodynamical matrices (such as $\boldsymbol{M_A}$) is impractical and uncommon in research, since it makes real-world parameter estimation impractical. Therefore, we follow the common assumption that currents are irrotational and constant over short time spans [3], [31], [32]. The irrotational property refers to the absence of a sway component in the current, as denoted in (3). The constant property assumes that there is negligible acceleration of the current in the Earth-fixed frame, i.e., $\ddot{\boldsymbol{\eta}}_c \approx \mathbf{0}$. Under these ocean current assumptions, it is shown that, when we parametrize the $\boldsymbol{C_{RB}}$ matrix independent of surge and sway, the dynamics can be described as follows:

$$\boldsymbol{M}\dot{\boldsymbol{\nu}}_r + \boldsymbol{C}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \boldsymbol{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r = \boldsymbol{\tau} + \boldsymbol{\tau}_{wind}, \quad (8)$$

where $\boldsymbol{M}$ and $\boldsymbol{C}$ are defined as in Section III-B. This is a practical model, since it only requires parameter estimation for a singular mass, Coriolis and damping matrix. It is trivial to compute the absolute velocity from the relative velocity using (4).

E. IMPLEMENTED MODELS

ASVSim allows to easily add specific damping and Coriolis models to compute $\boldsymbol{C}(\boldsymbol{\nu_r})$ and $\boldsymbol{D}(\boldsymbol{\nu_r})$. Four models are already implemented and available in the simulator:

- MilliAmpere Ferry, an autonomous passenger ferry at the Norwegian University of Science and Technology (NTNU) [33],
- Qiuxin No.5, a small research vessel at the Wuhan University of Technology [34],
- CyberShip II, a 1:70 scale replica of a supply ship, created for towing tank experiments [35].
- A full scale Mariner class cargo vessel [36].

For the full scale vessel, we also provide a generic method to include shallow-water effects. We base ourselves on the (corrected) model by Clarke [37], appropriate when the draft of the vessel is at most 80% of the waterway depth. As an illustrative example of the simulated effect of currents on a vessel, we include Fig. 3. This figure shows the recorded trajectories of the MilliAmpere ferry under no, low and moderate current velocities. As expected, currents with a higher velocity make the ferry move more from a straight path, and introduces torque on the vessel. Additionally, the current in the example decreases the speed of the vessel, as its impact angle has a component that is opposite to the thrust force direction.

IV. RADAR SIMULATION

Marine radar units are still dominant when it comes to localization, navigation, and collision avoidance in maritime environments. Although cameras and LiDAR systems are increasing in popularity, they cannot compete with the range that can be achieved with radar [38]. Also, because of the familiarity between radar and the shipping industry, it makes sense to include a radar simulation in this simulation-based research. Cosys-Airsim [9] has an accurate radar simulation based on ray-tracing techniques; however, this is based on a frequency modulated continuous wave (FMCW) radar unit originally introduced by [39], which is inherently different from the single-frequency pulse-echo systems commonly used in maritime and inland settings. For this reason, this research mimics a marine radar by using a modified version of the original AirSim [40] LiDAR implementation. The simulated LiDAR provides, at a predetermined frame rate that can be set similar to the rotation speed of the radar unit, a dense 3D point cloud of the surroundings covering a field-of-view of 360 degrees azimuth and a predetermined elevation. To mimic the performance of a maritime radar such as the Furuno DRS12A, as described in [41], the elevation was set to 20 degrees with a rational angle of 36 RPM. This unit has a range of 69 NM. In this scenario, a maximum range of 5 km (2.7 NM) was chosen. This range is sufficient because the depth of measurement rarely exceeds this value due to occlusion by the buildings and infrastructure present in inland waterways. Furthermore, objects located outside of this 5 km range are deemed not relevant in this use-case and are in turn omitted to limit computing requirements. This simulated approach also offers the added benefit that the dense LiDAR point cloud contains the maximum number of reflections,

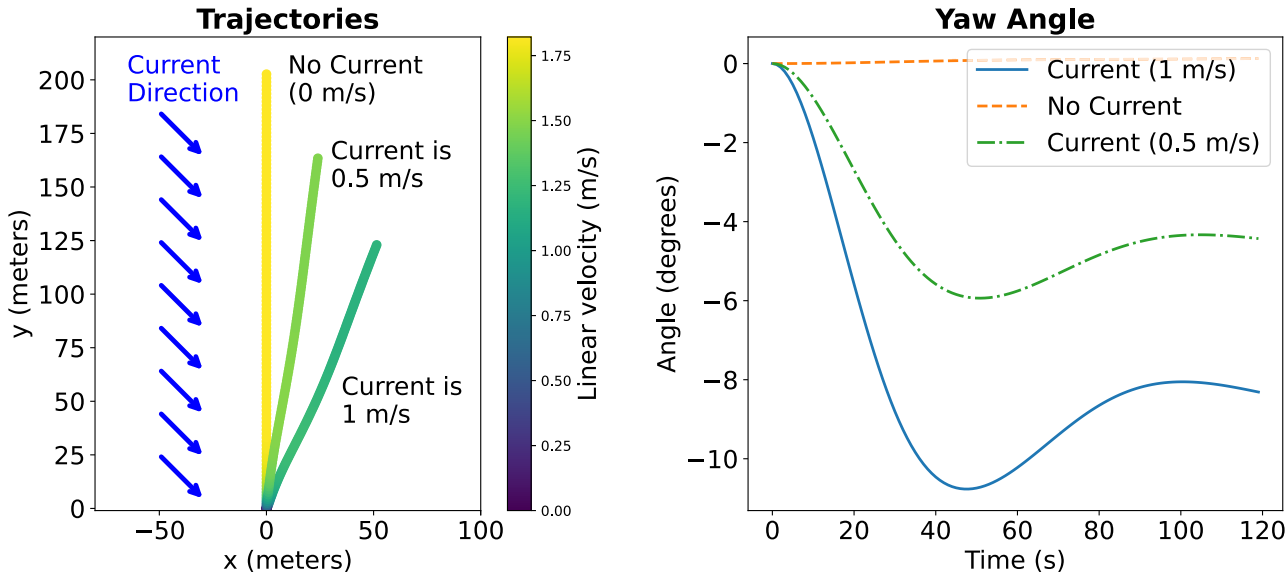


FIGURE 3. Sailing the MilliAmpere Ferry with thrust force at half ahead (250 N) with a straight thruster. Currents are simulated with an impact angle of 135° (top left). Left image: the recorded trajectories, starting from $x = 0$, $y = 0$. Right image: the rotation around the yaw axis, induced by the current.

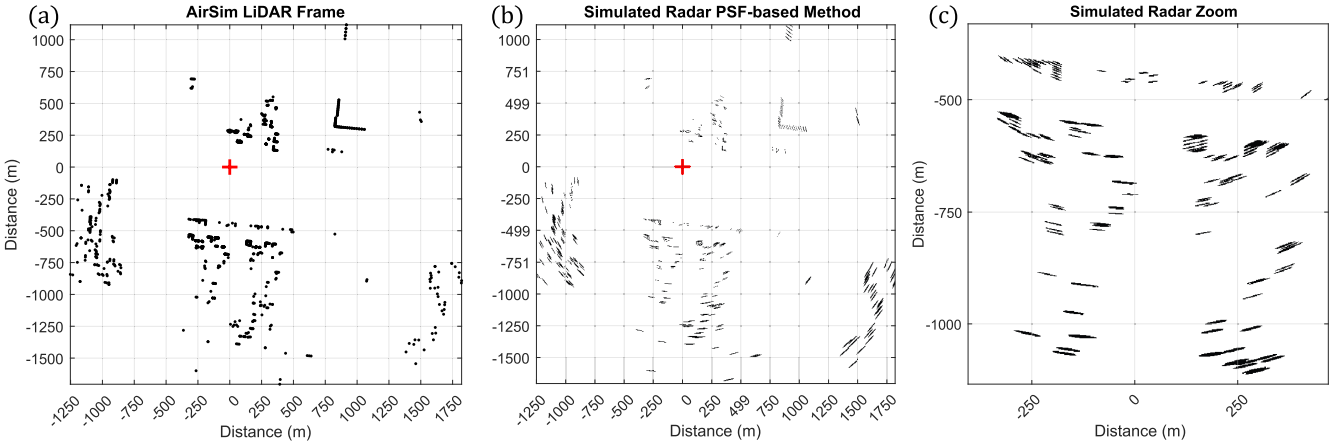


FIGURE 4. (a) The raw LiDAR point cloud obtained in the simulator. (b) The emulated radar data obtained using the PSF-based method (c) A subset of the center image, displaying the PSF varying over range and angle.

which allows us to generate datasets with varying levels of sparsity, mimicking scenarios with reduced vision.

A. RADAR DETECTION APPROXIMATION AND VISUALIZATION

Fig. 4 illustrates the process used to mimic maritime radar data, starting from the LiDAR sensor. An example LiDAR frame is shown in Fig. 4a. Each simulated reflection obtained by the light pulses results in a set of cartesian coordinates. To mimic the typical visualization found on maritime radar displays, these LiDAR detection coordinates should be rasterized into an image, and should be convolved with the scanning radar PSF that can be expected at the locations of these points, as illustrated in Fig. 4b. Fig. 4c shows a close up of the same scene, highlighting the shape and orientation of the radar PSFs. The shape and orientation of this PSF is dependent on the properties of the radar, the range, and angle with respect to the location of the radar.

Starting from a set of input reflector points $\mathbf{p}$, obtained from the LiDAR simulation, the locations of the nearest and furthest points, $\mathbf{p}_{\min}$ and $\mathbf{p}_{\max}$, with range $\Delta\mathbf{p} = \mathbf{p}_{\max} - \mathbf{p}_{\min}$, and image size G, the normalized point coordinates in the new image for each point $\mathbf{p}'_i$ are:

$$\mathbf{p}'_i = \left\lfloor \left(\frac{\mathbf{p}_i - \mathbf{p}_{\min}}{\Delta\mathbf{p}} \right) \cdot G \right\rfloor \tag{9}$$

The direction vector $\mathbf{d}_i$ from radar position $\boldsymbol{p_r}$ to normalized point $\mathbf{p}'_i$ is:

$$\mathbf{d}_i = \mathbf{p}'_i - \boldsymbol{p_r} \tag{10}$$

The range r_i is the Euclidean norm of $\mathbf{d}_i$:

$$r_i = \|\mathbf{d}_i\|_2 \tag{11}$$

As the radar PSF has its rotation with respect to the (centralized) location of the radar, the angle θ_i can be found as:

$$\theta_i = \arctan 2(d_{i,y}, d_{i,x}) \tag{12}$$

where $d_{i,x}$ and $d_{i,y}$ are the x and y components of $\mathbf{d}_i$, respectively. The basic shape of the radar PSF will be depicted as an ellipsoid [42], of which the semi-major axis a_i and semi-minor axis b_i are:

$$a_i = r_i \tan\left(\frac{\alpha}{2}\right) \frac{G}{\Delta p_x} \tag{13}$$

$$b_i = r_i \tan\left(\frac{\beta}{2}\right) \frac{G}{\Delta p_y} \tag{14}$$

where α is the horizontal beam width of the radar unit in radians, β a scalar which is manually set to obtain the correct range resolution. Δp_x and Δp_y are the x and y components of $\Delta\mathbf{p}$, respectively. To fill-in the correct pixels in the new image, it is not possible to use a standard convolution method as our PSF varies for every different point in the image. Therefore, we use the ellipse indicator function $E_i(x, y)$ is:

$$E_i(x, y) = \begin{cases} 1 & \text{if } \dfrac{x_r^2}{a_i^2} + \dfrac{y_r^2}{b_i^2} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where:

$$x_c = x - p'_{i,x} \tag{16}$$

$$y_c = y - p'_{i,y} \tag{17}$$

$$x_r = x_c \cos(\theta_i) + y_c \sin(\theta_i) \tag{18}$$

$$y_r = -x_c \sin(\theta_i) + y_c \cos(\theta_i) \tag{19}$$

and $p'_{i,x}$ and $p'_{i,y}$ are the x and y components of $\mathbf{p}'_i$, respectively. The radar PSF $P(x, y)$ is then the binary image that is obtained when taking the maximum of all ellipse indicator functions:

$$P(x, y) = \max_i E_i(x, y) \tag{20}$$

B. SCOPE AND LIMITATIONS

The PSF-based radar approximation faithfully emulates the geometric characteristics of a pulse-echo marine radar: the ellipsoidal resolution cell shape follows directly from the radar's azimuth beam width and range resolution [42], and the spatially-varying PSF correctly models how resolution degrades with range. Shadowing of objects behind solid structures is inherently captured by the underlying ray-casting engine.

However, several phenomena present in real marine radar are not modeled. Sea clutter, the backscatter from the water surface, is a noise source in marine radar typically modeled using K-distribution or Rayleigh statistics [43]. Our simulation does not generate sea surface returns, meaning the radar image appears cleaner than real-world recordings. We note that sea clutter is primarily a concern in open-ocean conditions and is less prevalent in sheltered inland waterways, which are the focus of this work. Multipath reflections, sidelobe returns, and radar cross-section (RCS) variations between different materials are also not captured, as the LiDAR-based approach treats all surfaces as ideal reflectors. These limitations make the current radar simulation most suitable for evaluating detection and navigation algorithms in terms of geometric accuracy, while quantitative radar signal fidelity requires validation against real marine radar recordings, which we identify as future work.

FIGURE 5. Overview of the example environments currently included in ASVSim.

V. AIRSIM FOR SURFACE VEHICLES

ASVSim is an extension of Cosys-AirSim [9], a simulator primarily for autonomous cars, drones and robotics research. It is an extension of the original AirSim framework [40], initially developed by Microsoft Research to provide a realistic simulation environment for training and testing autonomous systems. Based on Unreal Engine 5 (UE5) [26], the simulator provides high-quality graphics and physics simulation. Additionally, Matlab and Python APIs are provided to allow simulation-based research without requiring knowledge of the simulator's internals. ASVSim

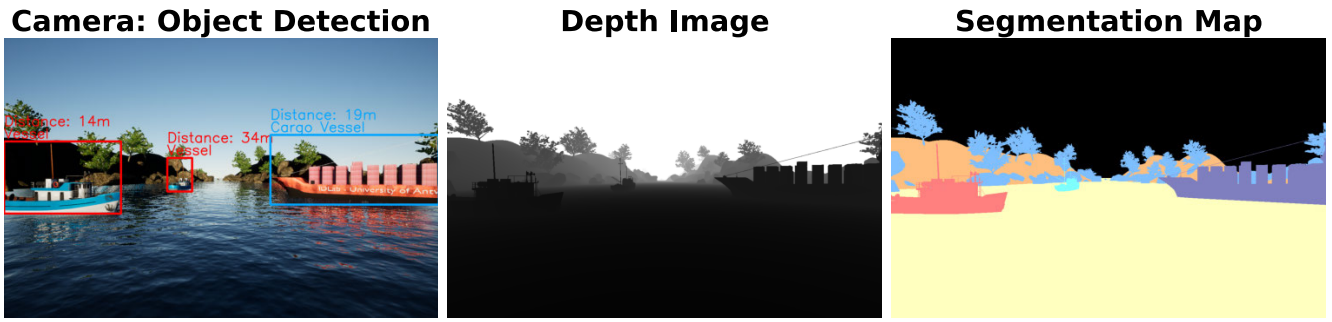


FIGURE 6. Example output from generated dataset. From left to right: bounding boxes generated automatically for vessels in the environment, depth camera measurements in grayscale (0 to 255m), segmentation map.

also supports procedural generation (PCG) of environments (See Sec. V-B). Because UE5 is used as a basis, we can leverage the blueprints system to easily change the simulation environments without writing code. With this visual scripting language, researchers can easily modify various aspects of the simulation, such as the objects in the waterway. Adjustments can include changing weather or time of day, altering the size, speed, orientation, and behavior of vessels, as well as changing camera perspectives to suit specific scenarios. Furthermore, it is straightforward to manipulate the environment to include new static and dynamic obstacles or change the layout. Researchers can also integrate custom 3D models or pre-existing assets from the Fab,[2] allowing for the creation of realistic and diverse environments tailored to research needs. In the first version of ASVSim, we have included three different basic example environments, a lake environment, a port environment, and a canal environment. These environments are shown in Fig. 5. Since ASVSim is built on UE5 [26], the hardware requirements follow those of the engine. The minimum requirements are a quad-core CPU, 16 GB RAM, and a GPU with 4 GB VRAM. The recommended specifications are a quad-core CPU (2.5 GHz or higher), 32 GB RAM, and a GPU with 8 GB VRAM. ASVSim supports Windows, macOS, and Linux. ASVSim also supports a headless mode, which disables rendering and allows the simulator to run without a display. This is particularly useful for training RL agents or collecting non-visual data, as it significantly reduces computational overhead.

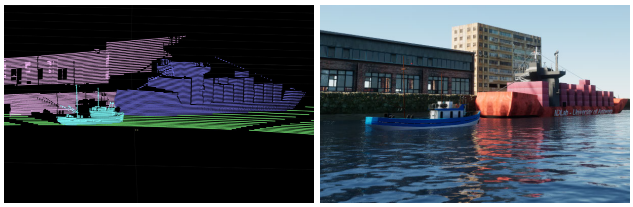
ASVSim extends upon the work of [9] (see Fig. 1) by adding vessel dynamics (see Sec. III), including the modeling of disturbances such as currents and wind. We provide multiple vessel models, based on existing research vessels, together with inland waterway environments. Furthermore, we include radar simulation, as explained in Sec. IV. Finally, we extended the Python/Matlab API to include functionalities for controlling the vessel, changing disturbances, and retrieving sensor data (see Sec. V-C). We also provide documentation[3] on the simulator and how to install, run and create your own projects. In Section V-A, we highlight the annotated dataset generation functionality and in Section V-C we demonstrate how to control the simulator, both using the API.

A. DATASET GENERATION

Vision-based perception is essential for autonomous shipping systems in constrained inland waterway and port scenarios. Although computer vision techniques have been extensively applied in areas such as manufacturing and autonomous driving [44], [45], progress in the maritime domain has been limited by the relative lack of annotated inland waterway vision datasets [27], [28]. Recent research [20] has shown that synthetic camera data generated from UE5 can significantly enhance the performance of object detection models on real-world data. To address this gap in the ocean surface robotics community, we enable dataset generation for object detection, depth estimation, and image segmentation with their corresponding ground-truth labels.

ASVSim currently includes vessel blueprints for a large container vessel and a fisher boat, along with photorealistic water rendering. Researchers can import their own 3D models (e.g., custom vessel types, port infrastructure) or use assets from the Fab marketplace. Additionally, plugins such as Cesium for Unreal[4] can be used to stream real-world photorealistic 3D environments (e.g., Google Photorealistic 3D Tiles), enabling the creation of geographically accurate waterway and port scenarios without manual environment modeling. Integration of Cesium for Unreal into ASVSim is currently a work in progress. Ground-truth annotations are generated automatically by ASVSim through UE5's built-in semantic segmentation: each object class in the scene is assigned a unique color in the segmentation map, from which pixel-wise labels, bounding boxes, and instance masks are derived programmatically via provided Python scripts. The semantic class definitions are configured by the user; in Section VI-A we use three classes (sky, water, and obstacles), but researchers can define their own taxonomy to suit their application. This eliminates the need for manual annotation and ensures consistency across generated datasets.

[2]https://www.fab.com/
[3]https://bavolesy.github.io/idlab-asvsim-docs/
[4]https://cesium.com/platform/cesium-for-unreal/



(a) (b)

FIGURE 7. (a) Generated point cloud, (b) corresponding camera image.

Moreover, as ASVSim provides access to various sensors, such as LiDAR, radar, and depth camera, these datasets can also be used for sensor fusion tasks, which are increasingly important for advancing autonomy. For example, the depth camera captures precise distance measurements for each pixel in the image, which can be used to develop monocular depth estimation models [46]. These models are crucial for scenarios where only RGB cameras are available, enabling the estimation of distances to objects in the scene based solely on visual input [47]. Furthermore, combining depth estimation from RGB cameras with traditional radar systems allows for the detection of smaller objects that might be missed by radar alone due to low sparsity or quality [48]. This enhanced optical precision is particularly important in IWT, where the presence of small obstacles, such as buoys, poses significant challenges for safe and reliable autonomous navigation. Furthermore, the simulator generates ground-truth labels for LiDAR point clouds, supporting dataset creation for 3D LiDAR-based object detection research, relevant for precise maneuvering such as docking or tugging. Fig. 6 demonstrates the generated dataset, showing vessels detected by the semantic camera with depth camera measurements and automatically generated bounding boxes based on the semantic color mapping, along with the distance to these vessels. These images were collected using the *simGetImages()* command from the Python API. Additionally we generated a point cloud (see Fig. 7) using the available LiDAR sensor and the Matlab API, via the command shown in Listing 1. This point cloud also automatically provides an instance segmentation ground truth. For more information on the LiDAR sensor, the reader is referred to [49].

B. PROCEDURAL ENVIRONMENT GENERATION

Recently, RL for autonomous navigation has gained significant interest. However, its success is often limited by the diversity of scenarios encountered during training. When trained in only a few environments, RL tends to struggle with generalizing to unseen situations. A common strategy to address this limitation is domain randomization [50]. By randomizing the environments and obstacles during training, RL may generalize better to unseen scenarios. Using the API, we can procedurally generate randomized port-like environments. When creating an environment, parameters such as the number of segments, random seed, width range, and the angle range between segments can be specified.

FIGURE 8. Example port-like waterways generated in ASVSim using the Python API. This environment can be used for training of RL agents for autonomous vessel navigation.

```
client = AirSimClient();
% Get LiDAR pointcloud p and
    semantic labels l
[p, l, ~, ~] = client.getLidarData('
    Lidar1', true, '');
% Define PID with (Kp, Ki, Kd)
pid = SimplePID(1.5, 1.0, 0.2);
desired_speed = 0.51  % in m/s
% Control loop
while true
    [gnssData, ~, ~] = client.
        getGpsData("Gps",'');
    v = gnssData.velocity;
    speed = sqrt(v(1).^2 + v(2).^2)
    thrust = pid.compute(
        desired_speed, speed)
    client.setVesselControls(thrust,
         0.5, '');
    pause(0.1);
end
```

LISTING 1. Example on thrust control with a PID using the Matlab API.

In Fig. 8, some examples of port-like environments can be seen. These can be generated at runtime, allowing for flexible RL training.

C. CONTROLLING VESSELS

Besides receiving the point cloud, Listing 1, provides a minimal example of how to control a vessel using the Matlab API. First, a connection to the running simulator is established. After the connection has been set up, measured state information can be received from a simulated Global Navigation Satellite System (GNSS) sensor. This includes information such as the current location, and velocities on the surge and sway axes. It is also possible to receive the acceleration vector of the vessel, using the available Inertial Measurement Unit (IMU) (allowing for Bayesian state estimation methods). More information about the ego vessel and its environment, such as wind speed and direction, the current time of day and the weather conditions are all available via the API. All sensor data, such as camera, LiDAR, radar and sonar sensors can also be read out.

For an exhaustive summary of all API availability, we refer the reader to the documentation. With this sensor information, it is possible to manually control the vessel or use autonomous navigation algorithms to control all thrusters of the vessel. In the example, we set the thrust to 0.8 and the angle to 0.5 (both are normalized from 0 to 1, where an angle of 0.5 is equal to 0° and 0 and 1 represent the maximum angle of the thruster in both directions). This will make the vessel move forward in the surge direction with 80% of its maximum thrust. Additionally, non-controllable vessels can also be spawned, which are useful for testing the performance of navigational algorithms when other vessels are operating in the proximity of the ego vessel. Listing 1 furthermore displays a control loop that employs a proportional-integral-derivative (PID) controller to control the normalized thrust force on the (single-thruster) vessel. In this example case the PID is set to reach and hold a surge velocity of 1 knot.

VI. EXPERIMENTS

To further demonstrate the potential of ASVSim for autonomy research, we include three experiments. As we also include the source code for these experiments, they serve as a starting ground for future research. First, in Sec. VI-A, we evaluate an existing waterway segmentation model on synthetic data. Secondly, in Sec. VI-B, we perform two path planning experiments. The first experiment demonstrates a navigation approach, based on the Dynamic Window Approach (DWA) [51], to navigate the MilliAmpere ferry [52] in a restricted waterway. The second experiment trains an RL agent to navigate the ferry through procedurally generated port channels with both static and dynamic obstacles.

A. WATERWAY SEGMENTATION

In addition to the ability of collecting data (See Section V-A), we investigate if an existing computer vision model (trained on real data) also works on the simulated camera images. For this, we perform a limited experiment that employs the WaSR-T [53] segmentation model in ASVSim. Importantly, we use a model that was pre-trained exclusively on real-world data and perform no additional training or fine-tuning. We only evaluate the model's performance on synthetic ASVSim images. The WaSR-T model is specifically trained for waterway and obstacle segmentation in an inland context and hence closely aligns with the setting of this work. For the experiment, a trajectory is collected in a simulated inland waterway, using a camera mounted at the bow of the ego vessel (milliAmpere ferry). The recorded video (downsampled to 2 frames per second) is then segmented by the deep learning model. Interestingly, WaSR-T succeeds in the segmentation task, even on simulated ASVSim data, as demonstrated by the results in Table 2 and further supported by the example in Fig. 9. The model achieves a very high test score on segmenting the water class (all close to 0.99). Additionally, the model almost fully succeeds in differentiating between the sky and obstacles, however, the model sometimes does not capture all minor details, such as strips of air that are visible through windows of buildings. This good performance of WaSR-T on the synthetic data demonstrates the potential of ASVSim for preliminary validation of computer vision models during research, and suggests that the simulator could indeed be used for efficient pre-training of deep learning models. However, we leave a more elaborate investigation of these aspects for future work, as results are expected to be highly dependent on how visually realistic the UE5 environments are made.

TABLE 2. Average test metrics of the WaSR-T model [53] during a simulated test trajectory. Intersection over Union (IoU), Precision, and Recall for Water, Air, and Obstacle classes are shown. Precision and recall are calculated by considering the segmentation task as a pixel-wise classification problem.

Metric	Water	Air	Other
IoU	0.9916	0.9859	0.9418
Precision	0.9971	0.9973	0.9806
Recall	0.9945	0.9885	0.9598

B. PATH PLANNING

Within the field of autonomous navigation, path planning is the process of determining a feasible path for an object, in this case a vessel, to navigate from a starting point to a goal location. The general problem of path planning can be divided into two main categories: global and local path planning. Global path planning algorithms find a static path from the starting position to a goal, whilst local path planning algorithms aim to navigate safely between two waypoints on that global path. The determined path must avoid collisions with obstacles. For inland shipping, this becomes increasingly complex as the vessel must take into account various constraints, such as fuel consumption or regulatory and physical constraints. ASVSim has functionality for researchers to implement and test both traditional navigation methods and RL algorithms for path planning. We demonstrate this functionality in Sections VI-B1 and VI-B2.

1) AUTONOMOUS NAVIGATION WITH DWA

As ASVSim is focused on IWT navigation, we consider a local path planning experiment, assuming that a global path is provided. It is important to distinguish between global and local path planning methods: commonly cited methods such as A^* [54] and RRT^* [55] are global planners that compute a complete, static path from start to goal, but do not account for vehicle dynamics during execution. For the local planning layer considered here, dynamically-feasible methods are more appropriate. Alternative local planners include Model Predictive Control (MPC) [29] and sampling-based approaches such as Model Predictive Path Integral (MPPI) [56], which can handle nonlinear dynamics and constraints. These methods typically require online optimization or Monte Carlo rollouts, incurring higher computational overhead which can hinder real-time operation. ASVSim can support any of these planners through

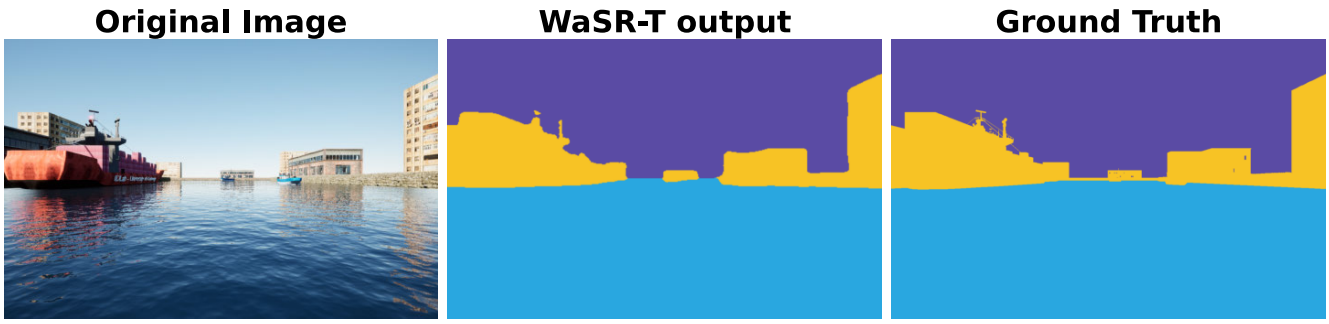


FIGURE 9. A representative example output of the WaSR-T model [53]. From left to right: the original simulated image, the WaSR-T output, and the ground-truth segmentation map. Blue denotes the waterway, purple denotes sky, yellow denotes all obstacles.

its API. For this experiment, DWA [51] was selected for its lower computational complexity, its established use in USV navigation and collision avoidance [5], [57], and because it directly encodes the vessel's dynamic constraints through the concept of a dynamic window and integrates naturally with the radar-based obstacle map and vessel state (heading, velocity) that ASVSim provides natively via its sensors. The approach works by creating a window of possible linear and angular velocities whilst taking into account the current dynamics of the vessel. All admissible velocity pairs are then evaluated with a predefined cost function over a trajectory with a finite horizon. To calculate admissible trajectories, the dynamics model of the vessel is used. The cost function that was used in the experiment is displayed in (21). To implement DWA, we opted for the PythonRobotics library [58]. We integrated the algorithm with the simulator and tested it on a channel-navigation scenario with a number of obstacles. The trajectory begins at (0, 0) m and proceeds through 11 global waypoints to approximately (580, 240) m. A waypoint is considered reached when the vessel is within 10 m of the goal, after which the next goal is activated. A radar (see Sec. IV) was used to detect the edges of the channel and other obstacles (such as moored vessels). The radar scan is then used to create an obstacle map, which in turn is used to calculate the cost function for each trajectory in the dynamic window. The trajectory with the lowest cost is then chosen and executed. The cost function is defined as follows:

$$C_{total} = C_{speed} + C_{obstacle} + C_{goal}, \tag{21}$$

where C_{speed} is the cost related to the speed with which the vessel travels, $C_{obstacle}$ is the cost of how close the vessel is to obstacles, and C_{goal} is the cost of how close the vessel's heading is to the desired goal heading. These costs are calculated as follows:

$$\begin{aligned} C_{goal} &= G_{goal} \cdot \arctan\left(\frac{\sin(\psi_{goal} - \psi_{vessel})}{\cos(\psi_{goal} - \psi_{vessel})}\right) \\ C_{obstacle} &= G_{obstacle} \cdot \begin{cases} \dfrac{1}{d_{\min}} & \text{if } d_{\min} > d_{threshold} \\ +\infty & \text{otherwise} \end{cases} \\ C_{speed} &= G_{speed} \cdot (v_{\max} - v_{vessel}), \end{aligned} \tag{22}$$

with G_{goal}, $G_{obstacle}$, and G_{speed} being constants to weigh the importance of each cost. $d_{\min}$ represents the minimum distance to obstacles along the trajectory, while $d_{threshold}$ denotes the vessel's dimensions. A collision is assumed when $d_{\min} < d_{threshold}$, and the cost is set to $+\infty$. ψ_{goal} is the desired heading to the goal, ψ_{vessel} is the heading of the vessel, and $v_{\max}$ is the maximum velocity of the vessel. After manual tuning of the cost function and configuring the parameters, the vessel navigated the path without collisions. This scenario is visualized, along with the corresponding radar detection, in Fig. 10. Although DWA navigates the channel successfully, it must be stated that the high computational cost is a significant downside of the methodology. This downside can be important for long-horizon problems such as vessel navigation, especially in busy IWT scenarios, where the situation can change significantly over a short period of time.

To demonstrate that DWA successfully navigates the channel, Fig. 11 reports the cross-track and heading errors [59] over time. Sharp increases in cross-track error occur immediately after a waypoint switch (indicated by the dotted lines), which is expected as we do not enforce arrival at the exact center of a waypoint with a particular orientation. These pose errors lead to temporary deviations, but DWA consistently demonstrates its ability to correct these errors as it navigates toward the next waypoint, shown by the decreasing cross-track error after each switch. We notice similar trends in the heading error, but with some oversteering. When DWA attempts to correct the heading, the vessel's angular momentum causes it to frequently overshoot the target orientation. This overshooting is visible in the graph where after a waypoint switch, the absolute value of the heading error decreases, crosses zero and then increases again. Increasing the planning horizon and the resolution of the dynamic window would likely reduce heading overshoot, and using a more accurate dynamics model for DWA instead of the linearized model we used could further mitigate oversteering. However, both approaches come with an increase in computation time. Note also that the ideal path does not account for obstacles and the vessel's maneuvers occasionally require unexpected turns, which further increase the cross-track and heading errors. The along-track error [59]

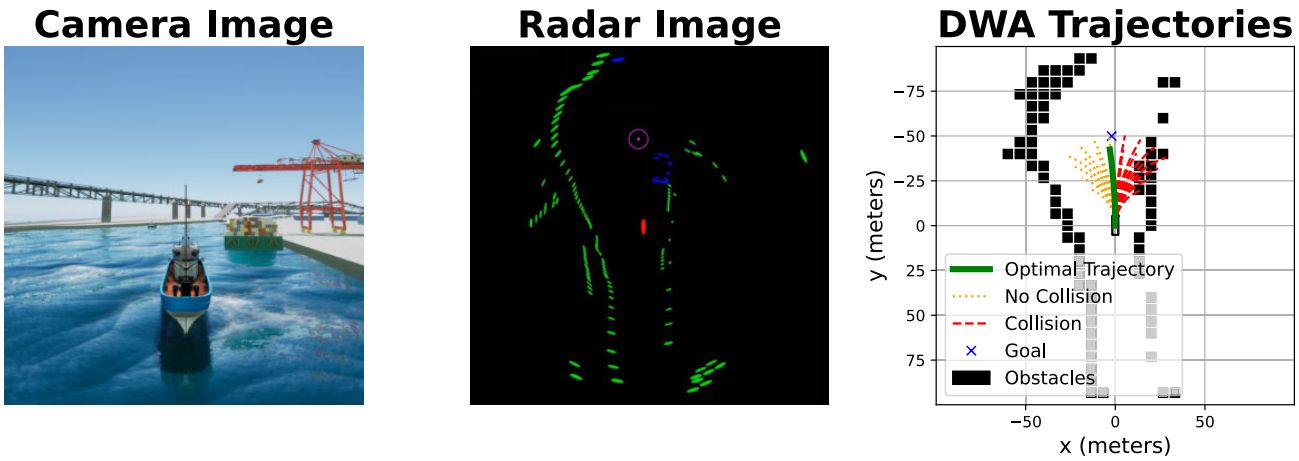


FIGURE 10. Overview of the path planning experiment with DWA to navigate a vessel through a channel. From left to right: Camera image, radar image (with goal as purple circle), and simulated DWA trajectories.

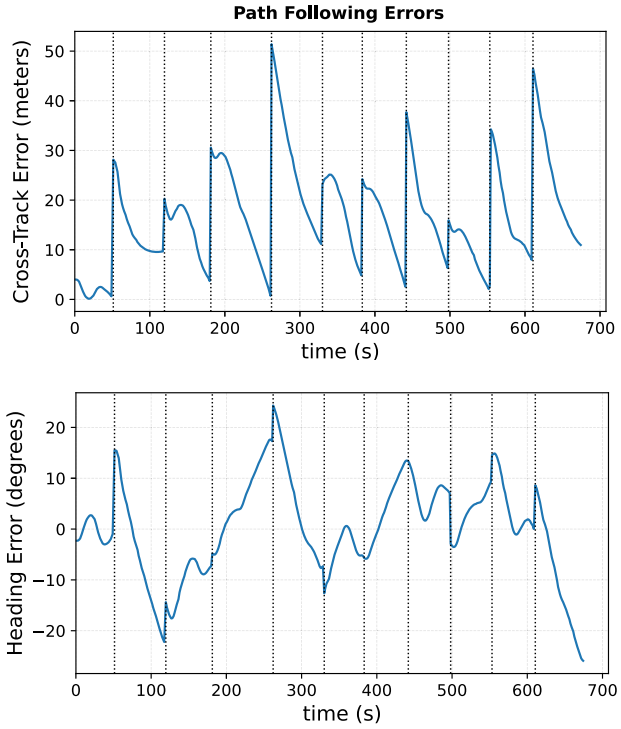


FIGURE 11. Path following errors [59] for the DWA experiment. The dotted lines represent the timesteps in which a new point on the global path is set. Top: Cross-track error, Bottom: Heading error.

is not shown on the figure, as it is not relevant for this specific scenario.

2) REINFORCEMENT LEARNING

As mentioned in Section VI-B1, a key limitation of DWA is its computational complexity, which scales poorly with the resolution of possible trajectories and trajectory length, with a median computation time of approximately 1.3 s per planning step in our experiments (see Appendix A-B). Another significant disadvantage of traditional control methods such as DWA is the need to manually adjust various parameters, in addition to the cost function, such as window size, time horizon, and velocity resolutions. To address these limitations, recent research has shown promising results using RL for path planning in autonomous shipping [4], [8]. In this section, we demonstrate how ASVSim can be used to train RL agents for autonomous navigation in shipping applications.

RL [60] is used to solve sequential decision-making problems where an agent learns an optimal strategy through interaction with an environment. These problems are modeled as Markov Decision Processes (MDPs). An MDP can be defined as a tuple (S, A, P, R, γ), where S is the set of states, A is the set of actions, P is the transition probability function, which defines the probability of transitioning from one state to another given an action, R is the reward function, and γ is the discount factor, which balances immediate rewards with future returns. The agent's goal is to learn an optimal policy that maps states to actions to maximize the cumulative reward across the horizon H. The definition of the optimal policy can be seen in (23).

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \left[\mathbb{E}_{a\sim\pi} \left(\sum_{t=0}^{H} \gamma^t R(s_t, a_t) \right) \right] \tag{23}$$

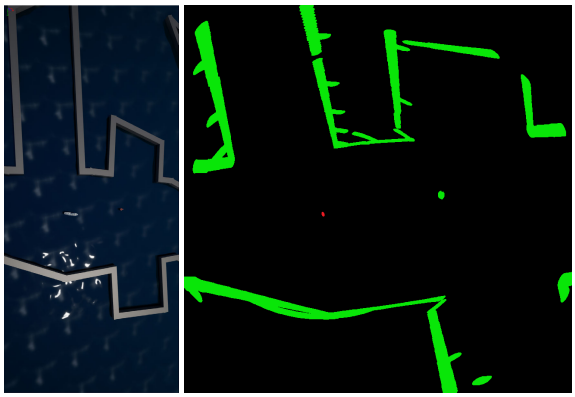
Through continuous interaction with the environment, the agent learns an optimal policy by exploring different actions and updating its strategy based on received rewards. This continuous interaction can be problematic for existing systems, since learning policies on a real vessel can be time-consuming, costly and dangerous. Furthermore, for autonomous shipping solutions to achieve commercial viability, they must demonstrate robust generalization across different vessels and maritime environments. Our previous research [8] has demonstrated that RL-based approaches maintain strong performance across vessels with varying physical properties, such as mass, without requiring

retraining or extensive manual recalibration. This represents a significant advantage over traditional control methods, which typically require substantial domain expertise and manual tuning for each new vessel configuration. The success of RL algorithms heavily depends on training with diverse, representative data. ASVSim addresses this need by enabling the creation of varied training scenarios with dynamic obstacle configurations, different weather and water conditions, and various port and waterway layouts. This rich simulation environment allows for training of agents across a wide range of IWT scenarios.

To leverage the PCG system described in Section V-B, we train an RL agent to navigate the procedurally generated port channels. The task is to steer the vessel from the starting position through the channel to a goal location. The terrain is regenerated every 10 episodes using a new random seed, producing a different port layout with varying channel geometry and border shapes each time. The `getGoal` API function is used to retrieve the goal coordinates for a given section of the generated port. By querying `getGoal` for each section from 1 up to the total number of generated sections, we obtain both intermediate waypoints and the final goal, which the agent must navigate to sequentially. The number of waypoints thus depends on the length of the generated environment. Additionally, at the start of every episode, static obstacles (buoys) and dynamic obstacles (vessels) are randomly spawned along the path between the vessel and the goal. Static buoys remain fixed throughout the episode, while dynamic vessels are assigned a random linear velocity and heading. This combination of terrain randomization, static buoys, and moving vessel traffic exposes the agent to a wide distribution of navigation scenarios during training, encouraging the learning of generalizable policies rather than the memorization of a single fixed environment. We included this experiment in the code release, along with the packaged environment, a trained model checkpoint, and an evaluation script, which can be used as a starting point for research towards RL agents in ASVSim.

In our experiment, at every timestep, the agent receives a 54-dimensional observation vector and outputs a continuous action, which is executed in the environment. The agent then receives the next observation and a scalar reward. This process repeats until the agent reaches the final goal, collides with an obstacle, or a maximum of 800 timesteps is exceeded. When an episode ends, the vessel is reset to its starting position and new obstacles are spawned. A waypoint is considered reached when the vessel is within 10 m of it, after which the next waypoint becomes the active navigation target.

The observation space consists of waypoint information $\boldsymbol{d}_{wp}$ (2D displacement vectors and scalar distances to the previous, current, and next waypoint), heading information $\boldsymbol{h}_{\psi}$ (heading error to the current waypoint, and vessel heading encoded as $\sin\psi$, $\cos\psi$), the linear velocity $\boldsymbol{v}$, linear and angular acceleration $\dot{\boldsymbol{v}}$, previous actions a_{t-1}, and LiDAR observations $\boldsymbol{d}_{lidar} \in \mathbb{R}^{36}$. The LiDAR sensor



(a) (b)

FIGURE 12. (a) Top-down simulator view of a procedurally generated port channel: the vessel navigates toward the goal through a channel with spawned obstacles (buoys and vessels). (b) The corresponding LiDAR point cloud as perceived by the agent, with detections shown in green and the vessel position in red.

produces 3600 range measurements which are min-pooled into 36 sectors of 10° each. The full observation vector is:

$$\text{obs}_t = \left[\boldsymbol{d}_{wp},\ \boldsymbol{h}_{\psi},\ \boldsymbol{v},\ \dot{\boldsymbol{v}},\ a_{t-1},\ \boldsymbol{d}_{lidar}\right] \tag{24}$$

The vessel used in this experiment is the milliAmpere ferry, whose dynamics are described in Section III. The agent controls only the rear thruster, with a continuous 2-dimensional action space for thrust force (normalized to $[0, 0.7]$) and thruster angle (normalized to $[0.48, 0.52]$), with the same controls as in Section V-C. Both ranges are constrained to limit ineffective and unstable actions.

The reward function is designed to encourage steady progress toward the current waypoint while penalizing collisions and idle behavior. At each timestep, the agent receives a progress reward based on the change in Euclidean distance to the active waypoint, combined with a constant time penalty:

$$R(s_t, a_t) = \underbrace{(d_{t-1} - d_t)}_{\text{progress}} - \underbrace{0.1}_{\text{time penalty}} \tag{25}$$

where d_t is the Euclidean distance to the current active waypoint at timestep t. When an intermediate waypoint is reached, the navigation target advances to the next waypoint and the distance tracking resets accordingly. Terminal rewards are assigned at the end of an episode:

$$R_{terminal} = \begin{cases} +500 & \text{if final goal reached} \\ -100 & \text{if collision} \end{cases} \tag{26}$$

The algorithm used for training is CrossQ [61], a sample-efficient off-policy actor-critic method, as implemented in the SB3-Contrib extension of Stable Baselines3 [62]. Observations are normalized using running statistics to stabilize training. The full set of hyperparameters is listed in Table 4 in Appendix B.

Fig. 13 shows the training curves over 5000 episodes from a single training run. We define the *success rate* as the

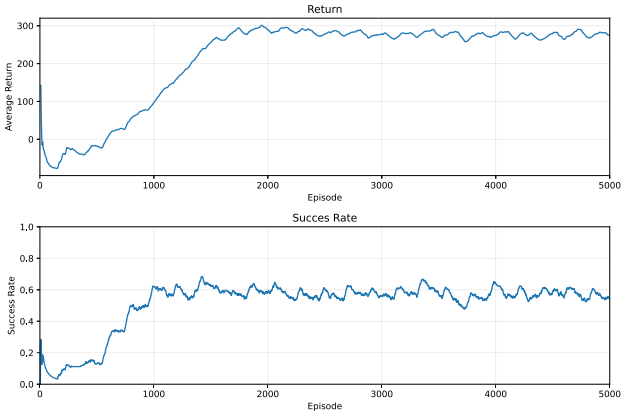


FIGURE 13. Training curves for the CrossQ agent over 5000 episodes in procedurally generated port environments. Top: average return. Bottom: success rate (rolling average).

TABLE 3. DWA planner configuration parameters. Note: these values were manually tuned for the DWA search space and do not represent the physical limits of the MilliAmpere ferry.

Parameter	Symbol	Value	Unit
Velocity limits			
Maximum speed	$v_{\max}$	25.0	m/s
Minimum speed	$v_{\min}$	1.0	m/s
Maximum yaw rate	$\dot{\psi}_{\max}$	30	deg/s
Acceleration limits			
Maximum acceleration	$a_{\max}$	20.0	m/s^2
Maximum yaw acceleration	$\ddot{\psi}_{\max}$	30	deg/s^2
Sampling resolution			
Velocity resolution	Δv	0.3	m/s
Yaw rate resolution	$\Delta\dot{\psi}$	1.0	deg/s
Prediction			
Time step	Δt	0.1	s
Prediction horizon	T_p	4.0	s
Cost weights			
Goal direction weight	G_{goal}	0.5	–
Speed weight	G_{speed}	0.2	–
Obstacle weight	G_{obstacle}	1.0	–
Vessel geometry (bounding box)			
Vessel width	W	4	m
Vessel length	L	8	m
Goal check radius	r_{goal}	10	m

fraction of episodes in which the agent reaches the final goal without collision, and the *return* as the cumulative reward collected over an episode. The average return increases steadily during the first 2000 episodes and then stabilizes. The success rate rises from near zero to approximately 60%, where it plateaus. The remaining failures are primarily due to collisions with randomly spawned obstacles that appear in close proximity to or directly in front of the vessel. Since the agent only controls the rear thruster and can only move forward, it has limited maneuverability to avoid obstacles in these configurations. We note that expanding the action space to include additional thrusters (e.g., bow thruster) or allowing reverse thrust would increase the agent's ability to avoid such collisions. Further optimization of the reward function and training hyperparameters may also improve performance,

TABLE 4. CrossQ training and environment configuration parameters.

Parameter	Symbol	Value	Unit
Training			
Learning rate	α	3×10^{-4}	–
Discount factor	γ	0.99	–
Batch size	B	256	–
Replay buffer size	$\lvert\mathcal{D}\rvert$	5×10^5	–
Learning starts	–	5000	steps
Train frequency	–	1	steps
Total timesteps	–	2.5×10^6	steps
Network architecture	–	MLP, 2×512, ReLU	–
Environment			
Terrain regen interval	–	10	episodes
Static obstacles	–	4	–
Dynamic obstacles	–	2	–
Control frequency	–	4	Hz
Max timesteps	$T_{\max}$	800	steps
Waypoint reach radius	r_{wp}	10	m
LiDAR sensor			
Channels	–	8	–
Measurements per cycle	–	450	–
Range	–	1000	m
Horizontal FOV	–	$[-180, 180]$	deg
Vertical FOV	–	$[-10, -2]$	deg
Action space			
Thrust range	–	$[0, 0.7]$	–
Thruster angle range	–	$[0.48, 0.52]$	–
Reward			
Progress weight	–	1.0	–
Time penalty	–	0.1	–
Goal reward	R_{goal}	$+500$	–
Collision penalty	$R_{\text{collision}}$	-100	–

TABLE 5. ASVSim performance during RL training on different hardware configurations and default graphics.

GPU	RAM	CPU	FPS
NVIDIA RTX 2060 Super	64 GB DDR4	Intel i5-9600KF	60
NVIDIA T550	32 GB DDR5	Intel i7-1260P	40
NVIDIA RTX 3060 (Laptop)	16 GB DDR5	Intel i7-12700H	80

which we leave to future work. An example of the PCG environment used during training is shown in Fig. 12.

To evaluate the generalization of the trained policy, we tested the final checkpoint on 25 previously unseen PCG environments (10 episodes each, 250 episodes total) with random obstacle configurations every episode not encountered during training. The agent achieved a success rate of 57.2% and a collision rate of 42.8%. This is consistent with the training success rate of approximately 60%, indicating that the policy generalizes to unseen environments without significant performance degradation. The failure mode remains the same: collisions with obstacles that are spawned in close proximity to or directly ahead of the vessel, where the limited action space (rear thruster only, forward motion) leaves insufficient room to maneuver.

VII. DISCUSSION AND FUTURE WORK

Our results demonstrate that the proposed open-source simulator provides a useful basis for further research towards autonomous vessel navigation. Sec. VI-A has demonstrated that, in the tested simulation environment, a computer vision model that works on real-world data, can also work in ASVSim. This implies the direct potential many research

directions such as investigating the impact of weather conditions on the computer vision model or comparing multiple models in a wide variety of situations. However, the question if we could train a model purely in the simulator and then deploy it in reality remains unanswered and is left for future work. Investigating this would require creating a large variety of simulation environments. In addition to the computer vision experiment, Sec. VI-B1 and Sec. VI-B2 demonstrate how the simulator can be used to train and test path planning algorithms. This allows for a decrease in time to deployment of novel algorithms, compared to only performing real-world experimentation. A limiting factor remains that several assumptions and simplifications have been made (such as the use of a 3DoF model and the radar limitations discussed in Section IV), leaving a necessity for real-world validation before deployment. Accordingly, ASVSim is currently best suited for prototyping and training algorithms in inland and port scenarios, with real-vessel transfer remaining an active direction for future work. Lastly, the Python and Matlab API, combined with the provided example experiments ensure that research within ASVSim is readily accessible to the research community.

There are several directions for future development that would further enhance the simulator's capabilities. Our research group has recently conducted experimental trials with a USV equipped with various sensors such as a depth camera, LiDAR, and IMU. A key direction for future work involves performing system identification, similar to the work presented in [33], on this USV to precisely determine the parameters corresponding to its dynamics model. This will enable us to create a digital model of our USV in ASVSim and to simulate its dynamics. Such precise modeling may allow for the development of navigation algorithms and training of RL agents in simulation that can be effectively transferred to the physical vessel with minimal performance degradation. Furthermore, we plan to create more detailed and expansive inland waterway environments to generate realistic datasets and to train RL agents to autonomously navigate in these environments. We also aim to add more vessel models for IWT applications, including 6-degree-of-freedom (6DoF) models. Lastly, another possible direction for future work is to develop more sensor models, including multi-beam echosounders and radar-based positioning systems, such as radar beacons (RACONs). By releasing our work in an open-source manner, we invite the broader research community to collaborate, contribute, and extend its capabilities. Through collaboration, we hope to accelerate innovation in inland autonomous navigation research.

VIII. CONCLUSION

In this paper, we presented ASVSim, an open-source simulation framework specifically designed for autonomous shipping research. This simulator addresses a current gap in the research domain, where no open-source simulators with high visual fidelity are available. By combining vessel dynamics models with relevant sensor simulations, we provide an extensive platform for developing and evaluating autonomous technologies for inland shipping applications. The experiments conducted on waterway segmentation and path planning demonstrate ASVSim's effectiveness in generating realistic sensor data and modeling vessel dynamics in inland waterway environments.

ACKNOWLEDGMENT

During the preparation of this work the authors used Claude Sonnet 4 in order to help with latex commands and proofread the work. After using this tool, they reviewed and edited the content as needed and take full responsibility for the content of the published article. *(Bavo Lesy and Siemen Herremans are co-first authors.)*

APPENDIX A
DWA IMPLEMENTATION DETAILS

This section provides the implementation details for the DWA experiment described in Section VI-B1. The implementation is based on the PythonRobotics library [58], which uses a linear motion model to forward-simulate candidate trajectories. This approximation is valid for sufficiently small time steps ($\Delta t = 0.1\,\text{s}$ in our configuration), where the linearization error remains negligible.

Table 3 lists all DWA parameters used in the experiments.

A. SEARCH SPACE

The dynamic window is bounded by both the kinematic limits of the vessel and the reachable velocities from the current state. In the worst case (full window), the number of candidate trajectories is given by:

$$N_{\text{candidates}} = \left\lfloor \frac{v_{\max} - v_{\min}}{\Delta v} \right\rfloor \times \left\lfloor \frac{2\,\dot{\psi}_{\max}}{\Delta \dot{\psi}} \right\rfloor = 80 \times 60 = 4{,}800 \tag{27}$$

Each trajectory consists of $T_p/\Delta t = 40$ predicted states, yielding up to 192,000 state evaluations per planning step. In practice, the acceleration constraints reduce the admissible window, particularly near the velocity and yaw rate boundaries.

B. COMPUTATION TIME

In our experiments (single-threaded Python with NumPy on an Intel Core i7-1260P, 32 GB DDR5, Windows 10), the median per-step planning time was approximately 1.3 s, with a typical range of 0.65–1.5 s. Peak times exceeding 2 s occurred when the vessel operated near the velocity or yaw rate boundaries, where the full admissible window is explored. A control frequency of 1 Hz is commonly used for autonomous vessel navigation, making the current single-threaded implementation not feasible for real-time operation. However, since the trajectory evaluations are independent, GPU acceleration or vectorized batch computation could reduce planning times sufficiently for real-time deployment. Alternatively, coarsening the sampling

resolution or shortening the prediction horizon would directly reduce the number of candidate evaluations at the cost of solution quality.

APPENDIX B
RL IMPLEMENTATION DETAILS

This section provides the hyperparameters for the RL experiment described in Section VI-B2. Table 4 lists all training, environment, sensor, and reward parameters. The simulation runs in real-time with no speed-up or parallelization; the total wall-clock time for training was approximately 42 hours.

A. RUNTIME PERFORMANCE

Table 5 reports the performance of ASVSim during the RL experiment (Section VI-B2), with the RL training process running concurrently on the same machine. Measurements were taken with the LiDAR sensor active (450 measurements per cycle, 8 channels, 3,600 points per second) at default graphics quality.

REFERENCES

[1] Y. Gu, J. C. Goez, M. Guajardo, and S. W. Wallace, "Autonomous vessels: State of the art and potential opportunities in logistics," *Int. Trans. Oper. Res.*, vol. 28, no. 4, pp. 1706–1739, Jul. 2021.
[2] Z. Pietrzykowski, P. Wołejsza, Ł. Nozdrzykowski, P. Borkowski, P. Banaś, J. Magaj, J. Chomski, M. Mąka, S. Mielniczuk, A. Pańka, P. Hatłas-Sowińska, E. Kulbiej, and M. Nozdrzykowska, "The autonomous navigation system of a sea-going vessel," *Ocean Eng.*, vol. 261, Oct. 2022, Art. no. 112104.
[3] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Hoboken, NJ, USA: Wiley, 2021.
[4] A. Zhang, W. Wang, W. Bi, and Z. Huang, "A path planning method based on deep reinforcement learning for AUV in complex marine environment," *Ocean Eng.*, vol. 313, Dec. 2024, Art. no. 119354.
[5] E. Serigstad, B.-O.-H. Eriksen, and M. Breivik, "Hybrid collision avoidance for autonomous surface vehicles," *IFAC-PapersOnLine*, vol. 51, no. 29, pp. 1–7, 2018.
[6] Z. Jiang, L. Su, and Y. Sun, "YOLOv7-ship: A lightweight algorithm for ship object detection in complex marine environments," *J. Mar. Sci. Eng.*, vol. 12, no. 1, p. 190, Jan. 2024.
[7] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning," *IEEE Access*, vol. 7, pp. 165262–165278, 2019.
[8] B. Lesy, S. Mercelis, and A. Anwar, "Robust offline reinforcement learning for autonomous vessel navigation," *J. Phys., Conf. Ser.*, vol. 3123, no. 1, Oct. 2025, Art. no. 012006.
[9] W. Jansen, E. Verreycken, A. Schenck, J.-E. Blanquart, C. Verhulst, N. Huebel, and J. Steckel, "COSYS-AIRSIM: A real-time simulation framework expanded for complex industrial applications," in *Proc. Annu. Modeling Simulation Conf. (ANNSIM)*, May 2023, pp. 37–48.
[10] (2025). *K-Sim Navigation*. [Online]. Available: https://marsim.kongsbergdigital.com/products/k-sim/k-sim-navigation/
[11] (2025). *Nautis Simulator*. [Online]. Available: https://vstepsimulation.com/nautis-simulator/
[12] (2025). *Wärtsilä R&D Simulator*. [Online]. Available: https://www.wartsila.com/marine/products/simulation-and-training/navigational-simulators/r-and-d-simulator
[13] (2025). *Ashss Simulator*. [Online]. Available: https://www.simtech.spb.ru/en/ashss
[14] J. Leudet, F. Christophe, T. Mikkonen, and T. Männistö, "AILiveSim: An extensible virtual environment for training autonomous vehicles," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2019, pp. 479–488.
[15] T. Perez, Ø. N. Smogeli, T. I. Fossen, and A. J. Sørensen, "An overview of the marine systems simulator (MSS): A simulink toolbox for marine control systems," *Model., Identificat. Control*, vol. 27, no. 4, pp. 259–275, 2006.
[16] M. R. Benjamin, H. Schmidt, P. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with MOOS-IvP," *J. Field Robot.*, vol. 27, no. 6, pp. 834–875, 2010.
[17] B. Bingham, C. Agüero, M. McCarrin, J. Klamo, J. Malia, K. Allen, T. Lum, M. Rawson, and R. Waqar, "Toward maritime robotic simulation in gazebo," in *Proc. OCEANS MTS/IEEE SEATTLE*, Mali, Oct. 2019, pp. 1–10.
[18] (2016). *European Ship Simulator*. [Online]. Available: https://store.steampowered.com/app/299250/EuropeanShipSimulator/
[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. Conf. Robot Learn.*, 2017, pp. 1–16. [Online]. Available: https://proceedings.mlr.press/v78/dosovitskiy17a.html
[20] T. Bu, X. Zhang, C. Mertz, and J. M. Dolan, "CARLA simulated data for rare road object detection," in *Proc. IEEE Int. Intell. Transp. Syst. Conf. (ITSC)*, Sep. 2021, pp. 2794–2801.
[21] M. Grimaldi, P. Cieślak, E. Ochoa, V. Bharti, H. Rajani, I. Carlucho, M. Koskinopoulou, Y. R. Petillot, and N. Gracias, "Stonefish: Supporting machine learning research in marine robotics," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2025, pp. 1–7.
[22] J. Song, H. Ma, O. Bagoren, A. Sethuraman, Y. Zhang, and K. A. Skinner, "OceanSim: A GPU-accelerated underwater robot perception simulation framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2025, pp. 1526–1533.
[23] C. A. Ortiz-Toro, C. Cerrada-Collado, D. Moreno-Salinas, D. Chaos-García, K. L. García-Suárez, P. Otero-Roth, J. M. Vidal-Pérez, M. Á. Luque-Nieto, A. I. Vázquez, J. J. Fraile-Ardanuy, V. Negro-Valdecantos, E. Jiménez-Yguacel, J. Aranda-Almansa, S. Zazo-Bello, P. J. Zufiria, L. Magdalena-Layos, J. Parras-Moral, and A. Gutiérrez, "Exploring UUV development with NauSim: An open-source simulation platform," in *Proc. Global Conf. Wireless Opt. Technol. (GCWOT)*, Sep. 2024, pp. 1–7.
[24] A. Amer, O. Álvarez-Tuñón, H. İ. Uğurlu, J. Le Fevre Sejersen, Y. Brodskiy, and E. Kayacan, "UNav-sim: A visually realistic underwater robotics simulator and synthetic data-generation framework," in *Proc. 21st Int. Conf. Adv. Robot. (ICAR)*, Dec. 2023, pp. 570–576.
[25] E. Potokar, S. Ashford, M. Kaess, and J. G. Mangelson, "HoloOcean: An underwater robotics simulator," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 3040–3046.
[26] (2025). *Unreal Engine*. [Online]. Available: https://www.unrealengine.com/
[27] L. Su, Y. Chen, H. Song, and W. Li, "A survey of maritime vision datasets," *Multimedia Tools Appl.*, vol. 82, no. 19, pp. 28873–28893, Aug. 2023.
[28] L. Trinh, S. Mercelis, and A. Anwar, "A comprehensive review of datasets and deep learning techniques for vision in unmanned surface vehicles," *Ocean Eng.*, vol. 334, Aug. 2025, Art. no. 121501.
[29] H. Zheng, R. R. Negenborn, and G. Lodewijks, "Trajectory tracking of autonomous vessels using model predictive control," *IFAC Proc. Volumes*, vol. 47, no. 3, pp. 8812–8818, 2014.
[30] R. Kristiansen, H. L. Ørke, and J. T. Gravdahl, "Modelling and simulation of interaction forces in tugboat-assisted docking of large marine vessels," *Simul. Model. Pract. Theory*, vol. 130, Jan. 2024, Art. no. 102866.
[31] H. K. Yoon and K. P. Rhee, "Identification of hydrodynamic coefficients in ship maneuvering equations of motion by estimation-before-modeling technique," *Ocean Eng.*, vol. 30, no. 18, pp. 2379–2404, Dec. 2003.
[32] T. P. DeRensis, "A robust linear dynamic positioning controller for a marine surface vehicle," M.S. thesis, Univ. Rhode Island, Kingston, RI, USA, 2013.
[33] A. A. Pedersen, "Optimization based system identification for the milliampere ferry," Master's thesis, 2019.
A. A. Pedersen, "Optimization based system identification for the milliAmpere ferry," M.S. thesis, Dept. Eng. Cybern., Univ. Sci. Technol. (NTNU), Trondheim, Norway, 2019.
[34] X. You, X. Yan, J. Liu, S. Li, and Y. Liu, "A novel model-based parameters estimation combining local optimization and global optimization of nonlinear ship models with physical experiment dataset," 2023, *arXiv:2312.05644*.
[35] R. Skjetne, Ø. Smogeli, and T. I. Fossen, "Modeling, identification, and adaptive maneuvering of CyberShip II: A complete design with experiments," *IFAC Proc. Volumes*, vol. 37, no. 10, pp. 203–208, Jul. 2004.
[36] M. S. Chislett and J. Strom-Tejsen, "Planar motion mechanism tests and full-scale steering and manoeuvring predictions for a mariner class vessel," *Int. Shipbuilding Prog.*, vol. 12, no. 129, pp. 201–224, May 1965.

[37] D. R. Clarke, "The shallow water effect on linear derivatives," *IFAC Proc. Volumes*, vol. 30, no. 22, pp. 117–122, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017465001

[38] B. Siciliano, O. Khatib, and T. Kröger, *Springer Handbook of Robotics*, vol. 200. Cham, Switzerland: Springer, 2008.

[39] G. Schouten, W. Jansen, and J. Steckel, "Simulation of pulse-echo radar for vehicle control and SLAM," *Sensors*, vol. 21, no. 2, p. 523, Jan. 2021.

[40] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: high-fidelity visual and physical simulation for autonomous vehicles," in *Field Service Robot.*, 2017, pp. 621–635.

[41] (2025). *Drs12a Product Page*. [Online]. Available: https://www.furuno.fr/en/lang--en--art--DRS12A-NXT--DRS12A-NXT.html

[42] M. A. Richards, J. Scheer, W. A. Holm, and W. L. Melvin, *Principles of Modern Radar*. Raleigh, NC, USA: SciTech Publishing, 2010.

[43] K. D. Ward, R. J. A. Tough, and S. Watts, *Sea Clutter: Scattering, the K Distribution and Radar Performance*. London, U.K.: IET, 2013.

[44] L. Zhou, L. Zhang, and N. Konz, "Computer vision techniques in manufacturing," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 1, pp. 105–117, Jan. 2023.

[45] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.

[46] J. Spencer et al., "The second monocular depth estimation challenge," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, Jun. 2023, pp. 3064–3076.

[47] W.-Y. Lee, L. Jovanov, and W. Philips, "Semantic-guided radar-vision fusion for depth estimation and object detection," in *Proc. Brit. Mach. Vis. Conf.*, 2021, p. 13.

[48] T. Ophoff, K. Van Beeck, and T. Goedemé, "Exploring RGB+depth fusion for real-time object detection," *Sensors*, vol. 19, no. 4, p. 866, Feb. 2019.

[49] W. Jansen, N. Huebel, and J. Steckel, "Physical LiDAR simulation in real-time engine," in *Proc. IEEE Sensors*, Oct. 2022, pp. 1–4.

[50] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 23–30.

[51] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[52] E. F. Brekke, E. Eide, B.-O.-H. Eriksen, E. F. Wilthil, M. Breivik, E. Skjellaug, Ø. K. Helgesen, A. M. Lekkas, A. B. Martinsen, E. H. Thyri, T. Torben, E. Veitch, O. A. Alsos, and T. A. Johansen, "MilliAmpere: An autonomous ferry prototype," *J. Phys., Conf. Ser.*, vol. 2311, no. 1, Jul. 2022, Art. no. 012029.

[53] L. Žust and M. Kristan, "Temporal context for robust maritime obstacle detection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 6340–6346.

[54] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Feb. 1968.

[55] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[56] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 1433–1440.

[57] H.-G. Kim, S.-J. Yun, Y.-H. Choi, J.-K. Ryu, and J.-H. Suh, "Collision avoidance algorithm based on COLREGs for unmanned surface vehicle," *J. Mar. Sci. Eng.*, vol. 9, no. 8, p. 863, Aug. 2021.

[58] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "PythonRobotics: A Python code collection of robotics algorithms," 2018, *arXiv:1808.10703*.

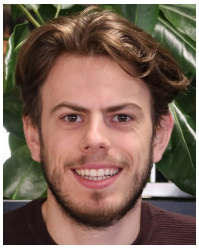
[59] A. M. Lekkas and T. I. Fossen, "Minimization of cross-track and along-track errors for path tracking of marine underactuated vehicles," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2014, pp. 3004–3010.

[60] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.

[61] A. Bhatt, D. Palenicek, B. Belousov, M. Argus, A. Amiranashvili, T. Brox, and J. Peters, "CrossQ: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024, pp. 1–19.

[62] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

BAVO LESY received the bachelor's and master's degrees in applied electronics and ICT engineering from the University of Antwerp, in 2022 and 2023, respectively. Since 2023, he has been a Ph.D. Researcher with the IDLab Research Group, IMEC, University of Antwerp. His research interests include (safe) reinforcement learning, autonomous vehicles, and representation learning.



SIEMEN HERREMANS received the bachelor's and master's degrees in applied electronics and ICT engineering from the University of Antwerp, in 2020 and 2021, respectively. Since 2022, he has been a Ph.D. Researcher with the IDLab Research Group, IMEC, University of Antwerp. His research interests include robust (model-based) reinforcement learning, autonomous navigation, and deep learning.

ROBIN KERSTENS received the B.Sc. degree in applied engineering electronics and ICT from the University of Antwerp, Antwerp, Belgium, in 2014, with a thesis focus on radar signal processing, the M.Sc.Eng. degree in electronics and ICT engineering technology from the University of Antwerp, with a thesis focus on sonar beamforming, and the Ph.D. degree in advanced array signal processing for in-air sonar. After a short spell in the automotive industry designing pressure sensors, he completed his Ph.D. degree. His research interests include in-air sonar, signal processing, radar, wave propagation, acoustics, machine learning, and algorithm development.

JAN STECKEL (Member, IEEE) received the degree in electronic engineering from University College "Karel de Grote," Hoboken, Belgium, in 2007, and the Ph.D. degree from the Active Perception Laboratory, University of Antwerp, Antwerp, Belgium, in 2012, with a dissertation titled "Array processing for In-Air Sonar Systems-Drawing Inspirations From Biology". During this period, he developed state-of-the-art sonar sensors, both biomimetic and sensor-array based. During his postdoctoral period, he was an active member of the Center for Care Technology, University of Antwerp, where he was in charge of various healthcare-related projects concerning novel sensor technologies. Furthermore, he pursued industrial exploitation of the patented 3-D array sonar sensor, which was developed in collaboration during the Ph.D. degree. In 2015, he became a Tenure Track Professor with the Cosys-Laboratory, University of Antwerp, where he researches sensors, sensor arrays, and signal processing algorithms using an embedded, constrained systems approach.

WALTER DAEMS (Senior Member, IEEE) received the B.Sc. degree in electronics from Katholieke Industriële Hogeschool Antwerp, Hoboken, Belgium, in 1994, and the M.Sc. and Ph.D. degrees in electrical engineering from Katholieke Universiteit Leuven, Leuven, Belgium, in 1996 and 2002, respectively. After a Postdoctoral Fellowship at the Fund for Scientific Research, Flanders, Belgium, he cofounded Kimotion Technologies, Inc., Geneva, Switzerland. In 2006, he joined University College "Karel de Grote," Hoboken, where he was appointed as the Vice Dean of Academic Affairs. In 2013, he was one of the main founders of the Faculty of Applied Engineering, University of Antwerp, Antwerp, Belgium. Since then, he has been a Professor with the Faculty of Applied Engineering. Until 2023, he was the Chairperson of the Faculty's Educational Board and the Vice Dean for Education of the Faculty. He is a member of staff of Cosys-Laboratory, Antwerp. His research interests include signal and sensor processing and the implementation and automatic generation of these systems in embedded technology for applications in industry and (health) care.

SIEGFRIED MERCELIS received the master's degree in music production from the Royal Conservatory of Ghent, Belgium, in 2008, the Master of Science degree in applied engineering (electronics and ICT) from Karel de Grote University College, in 2012, and the Ph.D. degree in applied engineering from the University of Antwerp, Belgium, in 2016. From 2012 to 2016, he was with Van den Berghe Research and Development under a Baekeland Ph.D. mandate, working on the optimization and parallelization of real-time media applications. He is currently an Assistant Professor with the IDLab Research Group, IMEC, University of Antwerp, where he leads the Adaptive Intelligence Research Program, focusing on bridging academic AI research and industrial applications in domains, such as process control, autonomous shipping, logistics, and mobility. He has authored over 100 peer-reviewed publications. He received the VIK Award for his master's thesis on parallel data structures.

ALI ANWAR (Member, IEEE) received the Ph.D. degree in control science and engineering from Harbin Institute of Technology, Harbin, China, in 2019. Since 2020, he has been a Principal Research Fellow with IMEC Research Group, IDLab, University of Antwerp, Belgium, where he is currently leads a team on context-aware control systems. Since, 2024, he has been a Principal Investigator for context-aware control systems with the Adapt-I Program of IDLab. His research interests include autonomous vessel navigation, safe and robust reinforcement learning, cooperative perception, and generative modeling in computer vision. He is with the IEEE Industrial Electronics and Systems, Man and Cybernetics Society, where he is part of the technical committees on motion control, and control, robotics, and mechatronics. He is a Reviewer in journals, including IEEE Transactions on Industrial Electronics, IEEE Transactions on Industrial Informatics, IEEE/ASME Transactions on Mechatronics, and IEEE Access. Moreover, he also reviews for conferences, such as CVPR, IECON, ICRA, and IROS.