AI

MDPI

Article

From Camera Image to Active Target Tracking: Modelling, Encoding and Metrical Analysis for Unmanned Underwater Vehicles †

Samuel Appleby *, Giacomo Bergami and Gary Ushaw

School of Computing, Faculty of Science, Agriculture and Engineering, Newcastle University, Newcastle Upon Tyne NE4 5TG, UK; giacomo.bergami@newcastle.ac.uk (G.B.); gary.ushaw@newcastle.ac.uk (G.U.)

* Correspondence: s.appleby3@newcastle.ac.uk

† This paper is an extended version of our paper published in the 5th Conference on Games, Boston, MA, USA, 21–24 August 2023. Appleby, S.; Crane, K.; Bergami, G; McGough, S. SWiMM DEEPeR: A Simulated Underwater Environment for Tracking Marine Mammals Using Deep Reinforcement Learning and BlueROV2.

Abstract: Marine mammal monitoring, a growing field of research, is critical to cetacean conservation. Traditional 'tagging' attaches sensors such as GPS to such animals, though these are intrusive and susceptible to infection and, ultimately, death. A less intrusive approach exploits UUV commanded by a human operator above ground. The development of AI for autonomous underwater vehicle navigation models training environments in simulation, providing visual and physical fidelity suitable for sim-to-real transfer. Previous solutions, including UVMS and L2D, provide only satisfactory results, due to poor environment generalisation while sensors including sonar create environmental disturbances. Though rich in features, image data suffer from high dimensionality, providing a state space too great for many machine learning tasks. Underwater environments, susceptible to image noise, further complicate this issue. We propose SWiMM$_{2.0}$, coupling a Unity simulation modelling of a BLUEROV UUV with a DRL backend. A pre-processing step exploits a state-of-the-art CMVAE, reducing dimensionality while minimising data loss. Sim-to-real generalisation is validated by prior research. Custom behaviour metrics, unbiased to the naked eye and unprecedented in current ROV simulators, link our objectives ensuring successful ROV behaviour while tracking targets. Our experiments show that SAC maximises the former, achieving near-perfect behaviour while exploiting image data alone.

Keywords: deep learning; reinforcement learning; active target tracking; computer vision; sim-to-real; unity; simulation

Academic Editors: Di Yuan and Xiu Shu

Received: 24 February 2025
Revised: 22 March 2025
Accepted: 31 March 2025
Published: 7 April 2025

Citation: Appleby, S.; Bergami, G.; Ushaw, G. From Camera Image to Active Target Tracking: Modelling, Encoding and Metrical Analysis for Unmanned Underwater Vehicles. *AI* **2025**, *6*, 71. https://doi.org/10.3390/ai6040071

1. Introduction

Marine megafauna conservation has seen increased research focus over recent decades, primarily due to water pollution, with projects varying in methodology, execution and scale. One branch of projects involves monitoring, profiling, and rescuing endangered marine megafauna (large marine mammals) species, including whales, dolphins, and turtles. Traditionally, mammals are 'tagged' with satellite-linked transmitters, either embedded or attached to the skin [1]. These enable GPS positioning. However, such approaches are well known to cause mental and physical discomfort. Small wounds caused by such applications are susceptible to infection, which can lead to disease and death. Furthermore, many aquatic populations are prone to disturbance caused by such techniques [2].

UUVs provide a less intrusive approach to monitoring marine mammals. UUVs have a wide range of applications, from inspecting marine infrastructures for the energy industry to detecting naval mines [3]. By providing increased coverage and access to previously inaccessible ocean parts, UUVs provide a valuable research tool to various scientific fields, including biology, ecology, geology, meteorology and medicine. These unmanned vehicles can be sub-categorised into ROVs and AUVs. The former are typically characterised by an umbilical tether that sends sensory information and receives power and commands from a human operator. Hundreds of metres long, this physical connection is an additional expense that can be both hard to manage and limiting to the range and freedom of movement of the vehicle [4]. An AUV, extends the former by providing autonomous control, making them more practical. Communication with a land or vessel-based server can minimise system recovery or human-in-the-loop intervention. This is important given that underwater communication methods (e.g., acoustic waves) suffer from high latency and frequent data loss [5]. Mapping sensory information to a control system presents a challenging problem. The configuration and extent of such tools are vast, combined with various algorithmic approaches and implementations.

BLUEROV, a versatile UUVs that can explore low to mid-range underwater environments, is at the forefront of cost-effective UUV research. The default configuration provides the chassis, housing, four thrusters, onboard camera and Rasberry Pi4. Powered by a tether connecting the BLUEROV to another machine, the rover sends sensory information (e.g., camera stream) to the ArduSub software housed on the receiving platform. A joystick controller provides commands to the onboard Pixhawk, which are converted to motor signals. We aim to extend the BLUEROV from an UUV to an AUV, responding to visual stimuli via a DRL algorithm. The DRL model's outputted actions emulate joystick controls, thus eliminating the need for an operator.

Given that state-of-the-art simulations vary in their complexity, versatility and usability, we draw attention to our requirements, which can be promptly satisfied via a Game Engine:

Real-time Physics. A real-time dynamically changing environment modelling the physical world where game data can be facilitated as input features.

High-fidelity Rendering. To aid generalisation, an accurate model of the Sony IMX322/323 image sensor (https://datasheetspdf.com/pdf-file/938855/Sony/IMX322LQJ-C/1, accessed on 21 March 2025) is housed on the ROV.

Game World Manipulation. Game state manipulation on the order of milliseconds.

1.1. Motivations

BLUEROV has previously been modelled for sim-to-real transfer [6–9] (Section 2). While these applications' requirements vary, none combine accurate camera and BLUEROV modelling with an integrated RL pipeline. To the best of our knowledge, we, for the first time, enable autonomous BLUEROV control by exploiting *image data alone* through DRL.

Other solutions [4] exploit heavier hardware to enrich the feature space. However, the former do not support the control communication protocol required during training and such components are usually expensive, have high power consumption and are subject to interfere with the environment. Many also lack manoeuvrability while we require an ROV responsive to rapid target movement. We investigate if a DRL algorithm can generalise a commercial game engine environment (Unity), thereby extending the UUV to an AUV. Unity, conjoining rendering, physics and control have proven capable of bridging the gap in robotics between simulation and reality [10]. By exploiting camera data alone, we also aim to minimise environmental disturbance. For solutions also exploiting image data for

DRL sim-to-real transfer [9,11], the former operates in above-ground conditions while few if any operate in dynamically changing environments presented by active target tracking.

Given that even low-resolution images define an extensive feature space, we require compression methods to achieve dimensionality reduction suitable as input to our DRL pipeline. Additionally, an architecture capable of achieving generalisation fit for sim-to-real transfer, particularly in noisy environments such as those found underwater. The CMVAE exploited in our solution reduces the dimensionality of our 64×64 images by orders of magnitudes, with image reconstructions and state space extractors also proving that the critical features are learned.

A recent tool benchmarks marine robotic simulations [12]: HoloOcean [13]; Dave [14] and Stonefish [15]. These simulators provide the meshes for UUVs, including BLUEROV. While the latter two focus on robotic simulation, HoloOcean includes a Gym-like interface supporting RL with a game binary. However, none of the previous models include a physical camera emulation of the Sony IMX322/323 image sensor and high-quality 3D mesh of a dynamic target. Furthermore, this tool was not explicitly designed to evaluate UUV behaviour by exploiting model behaviour metrics, which are here proposed for the first time (Section 4.4.4).

1.2. Objectives

Our previous solution, $\text{SWiMM}_{1.0}$ [16], successfully trained an SAC DRL agent yet failed to meet some of the previous requirements. Firstly, the target policy was delayed and slow to react (https://youtu.be/uLHhKhIprKA, accessed on 21 March 2025); the target often deviated from the camera's central vision and the BLUEROV would stray too far/near the target. Furthermore, training times were high: the CMVAE/DRL taking over 1/2 days. These limitations motivated us to formulate the following criteria the desired system should abide:

1. **Target Visibility.** A stable and clear video feed is vital for providing meaningful feedback to marine conservation experts. Therefore, the target object should deviate little from the camera's central field of view.
2. **Target Distance.** Many marine populations are sensitive to disturbances caused by robotic monitoring. BLUEROV's restricted target distance helps to avoid potential stress of discomfort that may be caused by a AUV with a stricter range threshold. In addition, we advocate that freedom of movement is encouraged, capturing more of the environment and providing a greater environmental context.
3. **Collision Avoidance.** The rover should avoid collisions with the target, preventing stress or panic in the animal and avoiding damage to the ROV, resulting in challenging autonomous control recovery.
4. **Smooth Control.** DRL algorithms applied in real-time environments are particularly susceptible to *jitter*, as the target policy fails to infer velocity and aims only to achieve the *current* optimum.
5. **Pipeline Requirements.** Sim-to-real transfer should minimise training time and reduce the costs and danger associated with real-world training. Our training pipeline should (1) minimise training time, (2) minimise hardware demand, and (3) exploit modern state-of-the-art solutions.

1.3. Methodology and Pipeline (Figure 1)

Figure 1 visualises the $\text{SWiMM}_{2.0}$ (https://github.com/SamuelAppleby/SWiMM_DEEPeR/releases/tag/MDPI_FINAL, accessed on 21 March 2025) training and inference pipeline. For the former, the Unity game engine acts as our data generator and real-world simulator (https://github.com/SamuelAppleby/SiM_DEEPeR/releases/tag/

MDPI_FINAL, accessed on 21 March 2025). *Static* image datasets are generated using physically accurate camera configurations. These datasets train and evaluate our CMVAE with an encoder achieving a dimensionality reduction of 1.23×10^3, a more manageable observation for our DRL network.

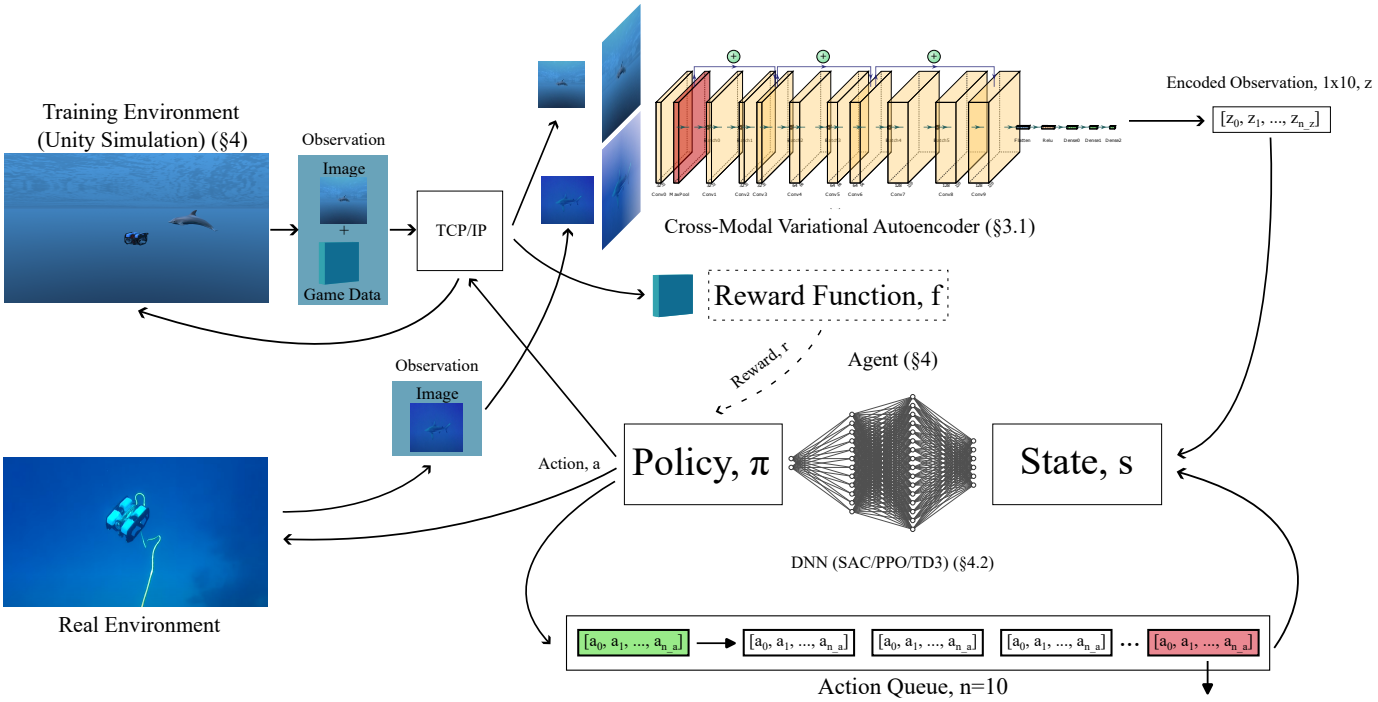

Figure 1. SWiMM$_{2.0}$ pipeline. The sample taken from the real environment captures a shark located off the west coast of the Big Island, Hawaii, from a GoPro camera attached to BLUEROV, permission of Blue Robotics.

During DRL training, the client (Figure 1: 'Training Environment (Unity Simulation)') combines in-game image renders with game state information (rover and target position, orientation and forward vectors). The former are bundled into one *observation* (Figure 1: 'Observation') and sent across a TCP/IP connection (Figure 1: 'TCP/IP'). The CMVAE on the receiving pipeline encodes the received images (Figure 1: Cross-Modal Variational Autoencoder), obtaining a much more compressed state representation z (Figure 1: Encoded Observation, z). Then, previous 10 actions (Figure 1: 'Action Queue') are concatenated with z, providing the state to the DRL network (Figure 1: 'State' s). Meanwhile, by exploiting game data, we compute discrepancies between current versus optimum target distance and rover heading, and a reward is calculated r (Figure 1: 'Reward function' $\mathcal{F}$, Reward r). The new state, s, and reward, r, are processed by the DRL algorithm and the target policy π dictates the next action, a (Figure 1: 'DNN' (SAC/PPO/TD3)', 'Policy' π). a, is both appended to the action queue and also returned to the Unity client across the TCP/IP connection, where the former is translated into forces providing the simulated thrust. The trained target policy π requires only image data, given that game data were exploited only for reward computation to aid the optimisation process. Thus, in real-world inference (Figure 1: 'Real Environment'), image frames alone, obtained from the on-board BLUEROV camera stream, provide all the necessary data to retrieve the next action a.

1.4. Improvements from SWiMM$_{1.0}$

1.4.1. Target Visibility

Firstly, SWiMM$_{1.0}$'s azimuth penalisation was ill-defined. An upgraded reward function logarithmically penalises heading discrepancies, resulting in a much stricter policy

(per our objective). Second, search tasks represent different challenges to active target tracking. SWiMM$_{1.0}$'s training environment allowed for the target to leave the field of view, and, particularly during the early stages of training, the target was not in the camera's view. An image containing no target cannot be optimised for active target tracking without extending the observation space. SWiMM$_{2.0}$ exploits camera frustum culling calculations within the simulation, which is included within the episode termination criteria during DRL training. Should the target not be visible, the episode is terminated, preventing the DRL algorithm from exploring unnecessary states.

1.4.2. Target Distance

SWiMM$_{1.0}$ overly penalised distance discrepancies resulting in erratic behaviour causing stress or discomfort to the animal. SWiMM$_{2.0}$ relaxed this penalty, granting the BLUEROV greater freedom from the optimum distance while reducing power consumption.

1.4.3. Smooth Control

Strict azimuth penalties resulted in control jitter not exhibited by SWiMM$_{1.0}$ (in the absence of strict penalties). By extending the observation space (Section '*Observation Space*'), our agent can infer velocity, optimising strict penalisation criteria while eliminating jitter.

1.4.4. System Compatibility and Enhancements

SWiMM$_{1.0}$ exploited old versions of the Tensorflow v1.14.0, OpenAI Gym and Stable Baselines libraries, absent of optimisations and improvements contained in the modern implementations. SWiMM$_{2.0}$ updates the entire pipeline. For the encoding step, Tensorflow 2 provides modern state-of-the-art Keras network implementations for our CMVAE, achieving improved image and state reconstruction values. In the best case, target distance (d) MAE is reduced by 69.6% (Section 4.3.2), and low image reconstruction MAE enables a more accurate observation as input to the DRL network. For training, Stable Baselines 3 (stable-baselines3 = 2.2.1) exploits the latest state-of-the-art DRL algorithms, providing enhanced algorithm performance and GPU support. This is coupled with OpenAI's Gymnasium environments, which focus more on DRL research. Two additional state-of-the-art algorithms, PPO [17] and TD3 [18], are also investigated as a more extensive coverage over industry-standard DRL algorithms.

1.4.5. Training Time

SWiMM$_{1.0}$ suffered from high training times, with the entire pipeline requiring over 3 days to train. Firstly, we explore the effectiveness of other DRL algorithms, including both on- and off-policy, which may optimise a target policy achieving faster convergence and greater accuracy. SWiMM$_{1.0}$ exploited SAC alone, while we also include PPO and TD3. Second, SWiMM$_{1.0}$ was constrained by CPU training. By exploiting GPU acceleration with CUDA, SWiMM$_{2.0}$ achieves similar accuracies in much lower training times than SWiMM$_{2.0}$. For CMVAE training, a 21.5$\times$ speed-up was achieved. Concerning both pipelines, training time achieves a 3.97$\times$ speed-up, with our best model attaining an improved average episodic reward from 2.12×10^3 to 2.40×10^3.

1.4.6. Benchmarking Correctness

Firstly, SWiMM$_{1.0}$ (incorrectly) considered training rewards to gauge model performance. In ML applications, behaviour should be judged on unforeseen data. SWiMM$_{2.0}$ exploits our custom evaluation callbacks, resetting the environment during training and testing the agent on a new instance of the simulation, thus providing more realistic rewards. Second, SWiMM$_{1.0}$ used cosine similarity for our image similarity experiments, which is agnostic brightness or contrast. In its place, SWiMM$_{2.0}$ uses pixel-wise MAE. For our

resizing similarity experiments (Section 4.2), $\text{SWiMM}_{2.0}$ achieves MAE of 1.09, and for the resolution scaling experiments, an MAE of 2.03. Both results are comparable to those from $\text{SWiMM}_{1.0}$.

This paper is organised as follows: Section 2 provides literature targeting similar problems and background design, Section 3.1 explains the learning architectures and training mechanisms, Section 3.2 details the simulation environment, including communication with the RL engine, Section 4 showcases our experiments, benchmarks and findings, and finally, Section 6 provides future work directions.

2. Related Work

Our model differs from traditional methods used for DRL target tracking, which often focus on vehicle racing or waypoint following [19]. As erratic or aggressive control—particularly those that produce acoustics [20]—are known to increase marine mammal stress, an emphasis on smooth movement is necessary. This would not be achievable with racing or tasking algorithms, whose main objective is quickly reaching a destination.

2.1. Unity

Game engines are systems with a well-defined separation of core software components required for a game (multidimensional graphics, physics simulation, audio, etc.) [21]. Traditional game engines were written in-house, but the growth of the games industry and the popularity of 'modding' games encouraged the development of standalone engines that could be outsourced to video game companies. Standalone engines are designed as extensible and can be exploited for the foundation of various games. But, given the realistic and physically accurate simulation environments provided by modern game engines, their application has extended into the domain of test beds for algorithms including, naturally, RL algorithms operating under real-time environments. The most recent projects [10,13] utilise commercial game engines, in line with a growing trend in DRL. In game engines, a *transform*, $\mathcal{T}$, is a core game object component. Given a point $p = (x, y, z)$:

Position. A *translation matrix* moves p to p' by v: $\begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$

Rotation. A *rotation matrix* rotates p to p': $\begin{bmatrix} a & b & c \\ d & e & f \\ h & i & j \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$, where $\{ a, \cdots, f \}$ depend on the rotated dimension.

Forward. The forward vector, **f** defines the z-facing direction vector (derivable from rotation).

All of the position, rotation, and forward vectors are stored in $\mathcal{T}$. Firstly, PhysX (Nvidia (https://developer.nvidia.com/physx-sdk, accessed on 21 March 2025)), integrated with Unity, is an accurate physical modelling SDK successfully used by many state-of-the-art applications, particularly in robotics.

Second, DirectX, Metal, OpenGL, and Vulkan are all graphics APIs supported by Unity, enabling cross-platform state-of-the-art visual rendering. Camera components also support physical camera specifications, allowing for a **direct** replication of the Sony IMX322/323 image sensor.

Third, C# scripting support allows for game world manipulation on both rendering and physical frame updates, allowing for almost instant application of DRL commands and camera renders to be returned across a TCP/IP connection.

ML-Agents [22], an open-source toolkit (provided by Unity), enables developers to directly train ML agents within Unity. While such a tool may be beneficial for the development of game AI, our project requires heavy pipeline customisation and extensions that are not easily achieved through ML-Agents. Our custom pipeline consists of 2 parts: the first managing the simulation (client) and the second handling all RL and network optimisations (server), communicating across a TCP/IP connection.

Recently, sim-to-real has seen a surge in popularity [19] and many applications successfully have exploited sim-to-real transfer across various platforms and industries [23]. As a world-leading UUV in robotics, the BLUEROV has been a lead candidate for sim-to-real research. Von Benzon et al. provide a detailed mathematical model of BLUEROV in Matlab/Simulink [6], while Yang et al. exploit the same simulation for optimising rover control [7]. Walker et al. provide experimental validation of wave-induced disturbances [8]. Other applications include digital twin modelling the BLUEROV in the Gazebo simulator [9]. Comparisons of our approach against other simulations are shown in Table 1.

Table 1. Comparison between solutions exploiting different robots for autonomous control.

	Benzon et al. [6]	**Yang et al. [7]**	**Walker et al. [8]**	**Skaldebo et al. [9]**	**Viitala et al. [11]**	**SWiMM$_{2.0}$**
Engine	Matlab/Simulink	Matlab/Simulink	Matlab	Gazebo	Unity	Unity
Purpose	Physical modelling	Dynamic positioning control	Wave-induced disturbances	Underwater vehicle manipulator system	Image-based track following	Image-based marine megafauna monitoring
Robot	BLUEROV	BLUEROV	BLUEROV	BLUEROV	Donkey Car	BLUEROV
Sim-to-Real	✓	✓	✓	✓	✓	✓
Physical Camera Modelling	✗	✗	✗	✓	✓	✓
RL Support	✗	✗	✗	✗	✓	✓

A very recent solution in [12] provides a framework, URoBench, analysing RL robotics simulators including HoloOcean [13], Dave [14], and Stonefish [15]. Of these, HoloOcean, a solution exploiting a game engine similar to SWIM, provides the greatest similarity to SWIM, though it does not support dynamic target tracking. URoBench evaluates the training of RL algorithms across a variety of AUV (including BLUEROV) through system resource demand (CPU, GPU, and primary memory) and RTF (a measure between simulation training versus real-world training efficiency). While UroBench provides a robust framework for analysing the demand and efficiency of robotics RL simulators, it does not assess the resulting agent performance. SWIM's primary requirement is the optimisation of a DRL behaviour policy for smooth autonomous marine mammal tracking achieving high reward, the success of which can be validated with our custom metrics (Section 4.4.4). Neither of the former is considered by UroBench.

2.2. Data

Computer vision tasks concerning marine cetaceans span several objectives. Classification tasks are highly researched, for example, identifying specimens from images or categorising species from audio data. For any ML task, a sufficient and robust dataset is required to both enable successful convergence and achieve suitable generalisation. Specifically for dolphins, dorsal fin data such as Risso's dolphin dataset [24] can be exploited to identify individuals [25], while acoustic data from cetacean echolocation clicks [26] can be exploited to identify click types [27].

While classification tasks can be solved in a static environment, active target tracking extends the former, as dynamic environments have a temporal perspective and must respond to model behaviour, which also must be considered in optimisation loss. Concretely, tackling a challenge such as active target tacking requires a reaction from the environment

in response to an action chosen by the model. Additionally, the training objective must be extensive enough to cover a sufficient range of world states, including both dolphin pose and rover pose, lest the model generalisation suffers.

Datasets such as [28] provide images obtained from GoPro Hero 3 and GoPro Hero 4 video feeds, capturing bottlenose dolphins in the North Sea. Yet, their use in active target tracking is limited: a DRL model requires a response to its chosen action, static datasets do not provide this and are more suitable to ML problems. Furthermore, a ML model must witness an extensive collection of target/rover poses to generalise well, often requiring hundreds of thousands of samples; in particular, the dataset provided by Trotter et al. [28] contains only 2201 poses, without a guarantee of sufficient state distribution. Gathering a sufficient and varied amount of training data on underwater megafauna is extremely costly, from both a time and economic standpoint. Furthermore, ensuring a suitable range of aquatic movement is captured defines a difficult research task whereby the gathering of such data would cause further stress to such animals. Finally, the CMVAE exploited in our pre-processing step (Section 2.3.1) requires, in addition to image data, the target distance, azimuth, and target yaw such that the former features are embedded into the resulting latent space. The prior datasets lack such information, necessitating our requirement for a data generation step providing the former which can be retrieved directly via simulation.

The prior reasons motivate our attempt at sim-to-real transfer, whereby we can enforce that a sufficient variety and magnitude of poses are captured. Our pre-processing training image dataset comprises of 3×10^5 images across equally distributed target and rover poses, thus ensuring an accurate encoding for future environment states witnessed in DRL.

2.3. ML

2.3.1. Feature Embedding

Stable Baselines default to a small CNN to limit the number of network parameters and, therefore, search space for the policy optimisation. However, this can lead to reduced feature learning. By decoupling feature learning from policy learning, the former can be allocated a neural network extensive enough for learning rich representations, whilst the latter can be allocated a network small enough to succeed with the credit assignment problem—learning the value of states and actions through experience. Furthermore, a lower dimension observation creates a smaller observation space, and an abstract representation aids generalisation, producing a more robust policy that can better survive the domain transfer challenge of sim-to-real.

Feature learning uses a cross-modal variant of a VAE [29]. The VAE takes the raw image as input and performs non-linear dimensionality reduction down to a specified number of dimensions (features). The decoder then tries to recover the input by decompressing the encoded feature vector.

CMVAE was first introduced by Spurr et al. [30], jointly encoding data from multiple modalities. Bonatti et al. [31] acknowledged that, by jointly encoding this information with image data, we could aid the control policy pipeline given we know the important state data *a priori*, while also guaranteeing adequate ambient noise reduction (Section 4.4.6). While the authors were able to learn a successful imitation control policy given a dynamically moving drone and static gates, we further investigate the CMVAE's capability operating under active target tracking conditions containing both an active target and observer. The authors exploited a supervised imitation pipeline enabling drones to navigate through a series of static gates by training in simulation, where expert trajectories were defined. We investigate if the former expert trajectories can instead be *learned* by a DRL network operating under semi-supervised conditions with reward alone being the prime motivator. We use a CMVAE solution consisting of 3 networks:

Encoder, q_{img}**.** Dronet [32], an 8-layer residual network, defines our encoder architecture, Figure 2. q_{img} reduces a high-dimensional image to a latent space, z.

Decoder, p_{img}**.** p_{img} reconstructs an image from z, consisting of six transposed convolutional layers, exploiting the ReLU activation function and batch normalisation before each convolution, providing the image reconstruction loss against the original image.

State Decoder, p_s**.** A set of auxiliary networks [31] (one per feature: d; θ and ψ) performs pose regression—predicting d, θ and ψ. Each network consists of 2 dense layers. During training, each relevant feature from z is passed through p_s, and losses between the ground truth and predicted feature values influence the gradient updates of q_{img}, disentangling the learnt feature space and ensuring the first three features are task-relevant.

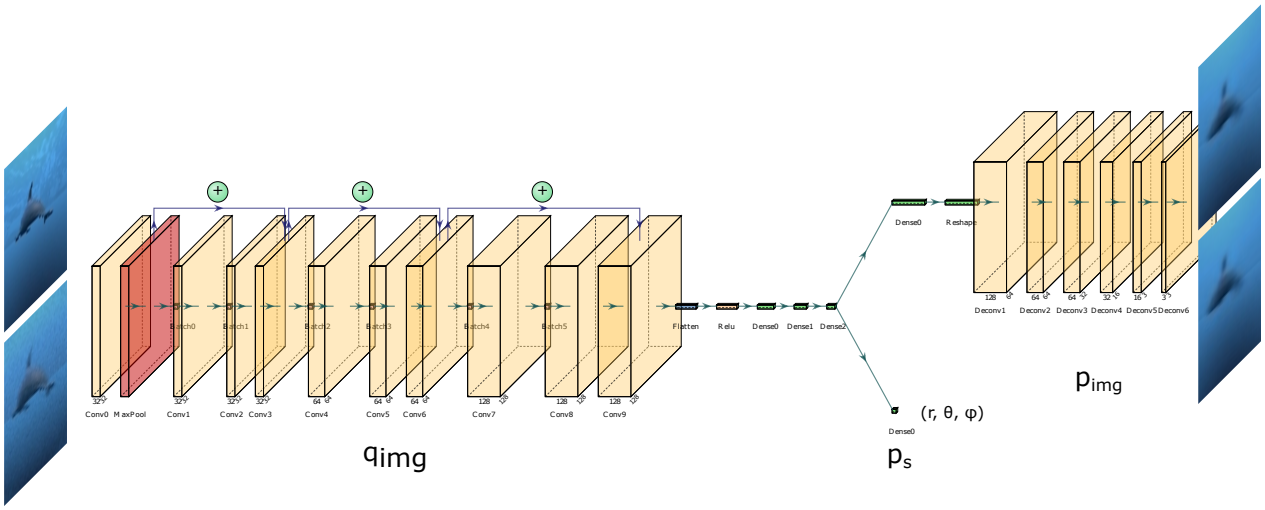

Figure 2. Showcasting the usage of the Dronet [32] 8-layer residual network architecture as part of our pipeline, q_{img}.

The new compressed output from q_{img} greatly reduces the observation space for the DRL algorithm, aiding the DRL optimisation process. While these authors exploited imitation learning, we aim to investigate if the same gains can be gained by DRL. Thus, this paper uses the multimodal solution for better compressing the image while deriving the main object position in the environment while exploiting RL-based approaches to learn better which is the best way to navigate the environment.

2.3.2. Three-Dimensional Object Tracking

Images provide a rich data representation containing a multitude of features, and thus image feature extraction is a heavily researched field across a variety of disciplines. Concerning robotics, techniques such as [33] estimate object location in space by exploiting corner flow detection using image data from the surrounding environment. Given this information, the robot's displacement can be derived. The author's experiments demonstrate the solution's capability in environments including reconstructing vehicle trajectories from a static camera, as well as identifying the objects' displacements when the camera is moving and the obstacle has a fixed location. Our solution's camera is housed on a system with a dynamically changing position also combined with a dynamic target, where knowing the object's relative position is as important as knowing the target rotation as well as reconstructing the tracked objects' azimuth position. Thus, features critical for target tracking such as θ and ψ cannot be accurately gauged from the relative pose alone as per CMVAE, as [33] require at least two poses for deriving such information.

State-of-the-art classification methods such as [34] exploit video feed data to predict target actions, the former extending VARG techniques. While such information could help

to enrich the feature space for better categorizing the target's action through discretisation, our DRL control policy represents a continuous classification task, mapping continuous features to continuous actions. Discretising such an environment (whether simulation or reality) would provide a less accurate control policy in comparison with the CMVAE, providing continuous feature mappings z while ensuring that a subset of learned features are objective-critical (r, θ, ψ), with remaining features providing additional contextual information from which we can immediately transfer as prompt operations to be transferred to UUV for target tracking.

More complex environments, such as those faced by autonomous driving in real urban scenarios, require a more holistic understanding of the environment, often boiling down to ethical problems [35]. Accurate environmental perception is then critical, as the objective is not only concerned with learning a sufficient control policy [36] but also considers complex decision-making processes, including obeying road laws, collision detection, and avoidance. The focus of SWIM is optimising a successful control policy alone, as underwater target tracking does not face the complications that the former environments encompass, requiring an extended feature space than that currently provided by the CMVAE.

2.3.3. RL

RL is a branch of ML lying between supervised and unsupervised approaches. The *behaviour* policy dictates the actions the agent will take based on environment observations (unsupervised), while the *target* policy is optimised through reward values computed using these observations, guiding the model towards some optimum state (supervised). RL excels in real-time dynamically changing environments such as autonomous vehicle navigation, where objectives can be well constructed by exploiting environmental observations (e.g., distance from the centre of the road). Environment *resets* return the environment and agent to an initial state, thus terminating the episode. Q-learning and SARSA are examples of RL algorithms, relying on tabular or approximation methods in the optimisation process.

$\text{SWiMM}_{1.0}$ exploits Gymnasium, a fork of OpenAI's Gym (https://github.com/openai/gym, accessed on 21 March 2025); the industry-standard API for enabling integration of RL applications and libraries. While providing some defaults, Gymnasium also allows for the integration of custom RL environments, requiring only the implementation of the *step* and *reset* templates. The former provides the action for the previous observation, while the latter reinitialises the environment at the user's discretion.

We consider a standard RL setup consisting of an agent interacting with an environment to learn a behavioural policy π. At time step t, the agent receives an observation o_t of the state s_t, and uses π to predict action a_t. The agent receives feedback as a bounded value—the reward r_t. The **agent policy** is shaped by the DRL algorithm whereby the final objective is to achieve high reward. In the RL framework, we have 2 policies:

Behaviour Policy: π_b enforces the agent's action for the current state in the current timestep s_t.

Target Policy: π is updated based on the rewards from the current action. It is used to 'learn' the best action based on future rewards. These policies are often *greedy*.

RL Algorithms can be on-policy or off-policy:

On-policy. the behaviour policy is the *same* as the target policy, $\pi = \pi_b$. Therefore, the action proposed by the target policy (often a greedy policy estimating future rewards) is chosen.

Off-policy. the behaviour policy can be *different* to the target policy, $\pi \neq \pi_b$. For example, our behaviour policy may be random, while the target policy may follow a ϵ-greedy approach.

Off-policy algorithms are more sample-efficient than on-policy ones, as they can exploit previous experiences stored in a *replay buffer* to optimise their network. In contrast, on-policy algorithms typically discard previous experiences but can optimise much faster. We investigate this trade-off by exploiting the following state-of-the-art RL algorithms:

SAC (off-policy) [37]: This is an actor-critic approach optimizing both policy and value network—a model estimating the expected return of a given state or state-action pair. It maximises a trade-off between expected cumulative reward and entropy (a measure of policy stochasticity), encouraging exploration by penalizing overly deterministic policies while maximizing expected rewards.

PPO (on-policy) [17]: This provides a policy gradient solution improving training stability and efficiency by using a clipped objective function to limit the extent of policy updates.

TD3 (off-policy) [18]: This combines aspects from Q-learning and policy gradients and has been exploited in advancing RL applications in continuous control tasks, such as robotic manipulation and autonomous driving.

Traditional RL methods struggle with large or continuous state and action spaces. DRL extends traditional RL by exploiting neural networks as function approximators to estimate value functions, Q-values, and policies. This allows the algorithm to generalise across large, complex state and action spaces. Pixel data from even low-resolution images provides a high-dimension feature vector, and throttle/steering values define a continuous action space with infinite outputs. Traditional RL methods would suffer in both memory and time under such high-dimension spaces (e.g., Q-learning combinatorial explosion). In addition, RL suffers as the exploration space grows due to the exponential increase in possible state configurations. DRL algorithms provide efficient exploitation techniques to visit the relevant unexplored states.

Given visual discrepancies between simulation and reality, we exploit a subset of DRL algorithms enabling us to generalise and explore the environment. DRL has seen high success rates in its ability to map perceptual data to image control [38].

3. Materials and Methods

3.1. Methodology

Firstly, the environment is *reset* (Section 2.3.3), randomly placing and rotating the observer (rover) in the water body. The target (dolphin) is placed at a distance d and azimuth θ and assigned randomised animation parameters. The rover performs active target tracking by sensing the environment and moving the actuators. The observation/action cycle continues until either a rollout is completed (the network weights are then trained) or the maximum step threshold is reached. As our current solution leverages state-of-the-art CVMAE solutions for compressing and denoise images while also retaining object tracking information, we refer to Section 2.3.1 for further information in this regard.

3.1.1. Deep Reinforcement Learning Pipeline

Observation Space

$SWiMM_{1.0}$ exploited only image data, whereby the agent could not gauge its speed. To help aid the agent in velocity inference (regardless of the observation space selection), $SWiMM_{2.0}$ provides a history of past behaviour through *action concatenation*. The observation space is extended by concatenating the previous 10 actions (*ActPrev*) to the encoded image. By providing the most recent actions as additional agent observations identifying its current state alongside the perceived image, the agent can then be influenced by any previous continual changes that may suggest a velocity. Overshooting was significantly

reduced, and motion jitter (from a stricter θ policy, Section '*Reward Function*, $\mathcal{F}$') was eliminated. Table 2 details the implementation differences between the observation differences from SWiMM$_{1.0}$ and SWiMM$_{2.0}$.

Table 2. Observation space extension between SWiMM$_{1.0}$ and SWiMM$_{2.0}$. $\mathcal{T}_i^{pos}/\mathcal{T}_i^{rot}$ denote the position/rotation of the object on the i'th axis. W/H denote the width/height of a given image render.

Observation Type	Dimensionality	SWiMM$_{1.0}$	SWiMM$_{2.0}$
Ground Truth	12	rov/target $[\mathcal{T}_x^{pos}, \mathcal{T}_y^{pos}, \mathcal{T}_z^{pos}, \mathcal{T}_x^{rot}, \mathcal{T}_y^{rot}, \mathcal{T}_z^{rot}]$	SWiMM$_{1.0}$ + *ActPrev*
Raw Image	$W \cdot H \cdot 3$	$[Img_0^0.RGB, Img_0^1.RGB, \ldots, Img_H^{W-1}.RGB, Img_H^W.RGB]$	
Encoded Image	n_z	$[z_0, z_1, \ldots, z_{n_z}]$	

Action Space

We use a 5-part continuous action space. The first four actions correspond with the vehicle's four degrees of freedom given a default thruster configuration $\{ x, y, z, y' \} \in [-1, 1]$. x, y, z represent linear force along the AUV's axis (i.e., lateral thrust or sway, vertical thrust or heave, and forward thrust or surge), and y' represents proportional angular force around the AUV's y axis (i.e., yaw). The BLUEROV also has a series of dive modes *mode* normally toggled using buttons on the controller:

$mode_{man}$**:** No feedback stabilisation, heading holding or depth holding.

$mode_{stab}$**:** Roll is stabilised and yaw when not commanded to turn.

$mode_{depth}$**:** As $mode_{stab}$ but also will maintain heave unless commanded otherwise.

In our simulation, we enable $mode_{depth}$ by default, as stable rover depth and roll would limit the state space, thus providing additional support for the optimisation of z, y'.

Curriculum learning is a common technique in autonomous vehicle navigation [39] to break down complex environments. Once good model behaviour has been achieved in the more primitive environments, complexity can be added in an iterative approach, allowing for a targeted focus on each additional feature. We apply the same motivation here. From an observational perspective, our training environment consists of only one target object, the dolphin, and one rover, both moving underwater. From an action perspective, we reduce the dimensions to 2, with π outputting values for z and y' only. The target's movement is restricted to the xz-plane. We transform the Unity simulation into an RL agent environment by providing observation and action space implementations.

Reward Function, $\mathcal{F}$

We provide the maximum reward followed by penalties; an agent is 'rewarded' by reducing these penalties. For any two game objects, a, b, with translation vectors $\mathbf{v}_a$ and $\mathbf{v}_b$, the heading ($\vec{h} = \mathbf{v}_a - \mathbf{v}_b$) between them is required for computing azimuth error penalty between the rover and the dolphin.

The target has freedom of movement on the xz-plane. The target can also move on the y-axis, though we save this for future work. On this plane, we compute the *radial* distance between the rover and target, d:

$$d = \sqrt{\vec{h}.x^2 + \vec{h}.z^2} \tag{1}$$

Likewise, θ defines the angular difference in the y component on the same plane, taking the dot product between the normalised heading vector and the rover's forward vector:

$$\theta = \arccos\left(\hat{\mathbf{h}} \cdot \mathbf{f}_x\right) \tag{2}$$

In SWiMM$_{1.0}$, our reward function penalises the agent for the difference between the current and optimum heading/distance with equal weighting (Table 3). *opt_d* is the optimal distance between AUV and dolphin, *max_d* is the maximum distance the AUV can deviate from the **optimal** distance before the episode is terminated. Our freedom of movement has a range: $[opt\_d \pm max\_d]$. However, note the following:

1. **Distance Strictness, pen$_d$.** In SWiMM$_{1.0}$, pen$_d$ was too great (scaling penalty quadratically), over-biasing the agent to achieve optimum distance while also causing greater changes to azimuth, thus leading to poor responsiveness. Instead, we should allow greater freedom of movement from *opt_d* as the video feedback (and, therefore, the penalty for azimuth increases) is affected more by the ROV's heading than the raw distance. This also has the added benefit of minimising azimuth fluctuations. We replace the quadratic penalty with an exponential one; a more lenient behaviour.
2. **Poor Responsiveness, pen$_\theta$.** To enforce a stricter heading penalty, we provide a logarithmic increase, rather than a linear one as in SWiMM$_{1.0}$. For the latter, subsequent experiments reveal unresponsiveness to significant azimuth changes concerning the original maximum possible azimuth of 180°, which were not considering the camera's horizontal field-of-view cam_{hfov}:
$$\alpha = \frac{cam_{hfov}}{2} = \arctan\left(\frac{cam_w}{2 \cdot f}\right) \tag{3}$$
where cam_w stands for camera sensor's width. This new penalty encourages a much more responsive agent.
3. **Smooth Control.** An agent penalised heavily for small changes from the optimum will *jitter*, whereby the action will continuously flip between negative and positive values. The challenge, then, is discovering an agent that can maintain optimum heading and distance while eliminating jitter caused by overshooting at the optimums. Preliminary investigations applying a smoothness penalty resulted in an agent being overly conscientious of maintaining smooth control. While the resulting policy almost eliminated jitter, sharp control changes were lacking, and the target was not tracked effectively. We opted for a different mechanism by extending the observation space to include the previous n actions (*action* stacking, distinct from *frame* stacking, Section '*Observation Space*'), which successfully eliminated jitter/overshooting. This is detailed in the following section, '*Smoothness Error Implementation*'.

Concerning $\mathcal{F}$, high rewards are achieved by keeping the target within strict bounds of the centre of vision while maintaining a suitable distance from the target but allowing some freedom of movement.

Table 3. Reward penalty for azimuth for SWiMM$_{1.0}$ and SWiMM$_{2.0}$. pen$_d$ denotes the distance penalty and pen$_\theta$ denotes the Azimuth Penalty with range $[0, 1]$. r_t denotes the step reward. Thus $r_t \in [-1, 1]$. k denotes the penalisation weighting; a lower value will penalise the feature less. For our experiments (Section 4), we found that $k = 1$ was an appropriate value. *opt_d* and *max_d* are set to 6 and 4 metres respectively, which was chosen as a suitable target range.

Function	SWiMM$_{1.0}$	SWiMM$_{2.0}$
1. pen$_d$	$\frac{(d - opt\_d)^2}{max\_d^2}$	$\frac{e^{(k \cdot (d - opt\_d))} - 1}{e^{(k \cdot max\_d)} - 1}$
2. pen$_\theta$	$\frac{\lvert\theta\rvert}{180}$	$\frac{\ln((k \cdot \lvert\theta\rvert) + 1)}{\ln((k \cdot \alpha) + 1)}$
r_t	$1 - (\text{pen}_d + \text{pen}_\theta)$	

Smoothness Error Implementation

A consistent video feed necessitates stable and smooth control, minimising power usage and environmental disturbance.

The smoothness errors are calculated by performing a normalised weighted absolute difference between the current action and all the previous from *ActPrev*. The weighting list, $\mathcal{W}$, diminishes for actions further in the past and is defined as follows:

$$\mathcal{W} = [0.25, 0.20, 0.15, 0.10, 0.08, 0.06, 0.05, 0.04, 0.04, 0.03]$$

0.03 corresponds to the weight of the occurring 10 steps before being considered less important than the last action, 0.25. The normalisation divides the respective errors by the maximum error possible, $|1 - (-1)| = 2$, given that our continuous actions lie in the range $[-1, 1]$. An action index, k, is used to select the control of interest, where, given our current set of actions, act, act$[k]$ selects the value for the k'th action. In our setup, act$[0]$ denotes the surge smoothness, while act$[1]$ denotes the yaw smoothness. $\mathcal{S}_k$ denotes the smoothness for the k'th index (action) of interest. For our action space, $\mathcal{S}_0/\mathcal{S}_1$ indicate the smoothness errors for surge/yaw, which, for clarity, are renamed $\mathcal{S}_\mathcal{D}/\mathcal{S}_\mathcal{A}$. At any step, s, the intermediate smoothness error is as follows:

$$\mathcal{S}_k = \frac{1}{2} \cdot \sum_{i=1}^{n} (|(\text{act}[k] - \mathit{ActPrev}((n-i)+1)[k])|) \cdot \mathcal{W}_i \tag{4}$$

where $n = |\mathit{ActPrev}|$ (in our case, $n = 10$). For example, let us suppose we are at step $s = 4$, with a history of the previous 3 actions: $\mathit{ActPrev} = [(-1, 0.5), (1, 0), (0.4, 0.2)]$. A new action is decided: $\mathit{act} = (0.5, 0.2)$. The first element $\mathit{act}[k = 0] = 0.5$, indexes the surge output (50% of maximum thrust) while the second $\mathit{act}[k = 1] = 0.2$ selects the yaw (20% of maximum thrust). We return the following intermediate smoothness errors:

$$\mathcal{S}_\mathcal{A} = \frac{(|0.2 - 0.2| \cdot 0.25) + (|0.2 - 0| \cdot 0.2) + (|0.2 - 0.5| \cdot 0.15)}{2} = \frac{8.5 \times 10^{-2}}{2} = 4.25 \times 10^{-2} \tag{5}$$

$$\mathcal{S}_\mathcal{D} = \frac{(|0.5 - 0.4| \cdot 0.25) + (|0.5 - 1| \cdot 0.2) + (|0.5 - (-1)| \cdot 0.15)}{2} = \frac{3.5 \times 10^{-1}}{2} = 1.75 \times 10^{-1} \tag{6}$$

An episode with $\mathcal{S}_\mathcal{A}/\mathcal{S}_\mathcal{D} = 0$ indicates **perfect** smoothness (i.e., all actions are identical), while $\mathcal{S}_\mathcal{A}/\mathcal{S}_\mathcal{D} \to 1$ indicates high action variance. Unlike traditional error metrics, achieving 0 smoothness error is not desired, as that is only achievable with identical action commands. Instead, a low $\mathcal{S}_k$ combined with low $\mathcal{A}/\mathcal{D}$ values indicate good behaviour with minimal jittering.

Episode Termination

Good practice in RL literature [40] is the termination of episodes upon the satisfaction of certain criteria. The target is susceptible (particularly during early training) to instances where the target leaves the camera view. As a preventative measure, we extend the set of episode termination criteria, Table 4.

Only observations containing the target are included in the state space. Otherwise, our problem extends beyond the scope of target monitoring (in the absence of a perfect information environment) to also include target search, requiring different methodologies.

Table 4. All episode terminating criteria. Criteria marked with *CALLBACK* are processed after completing each step.

Parameter	Definition	Value	SWiMM$_{1.0}$	SWiMM$_{2.0}$
Distance Threshold	A distance from the optimum no greater than the threshold should occur.	4	✓	✓

Table 4. *Cont.*

Parameter	Definition	Value	SWiMM$_{1.0}$	SWiMM$_{2.0}$
Target Visibility	The target should always be within view of the camera.	N/A	✗	✓
Collision Avoidance	A collision with the target must be avoided.	N/A	✓	✓
Time Limit	Episodes, $\mathcal{E}$, can run for, at most, s steps, $\lvert\mathcal{E}\rvert \leq s$.	3×10^3	✓	✓

3.2. Simulation Engine

Inaccurate environmental representation can harm sim-to-real transfer, including physics, graphics, and hardware. Figure 3 displays a still from the simulation.

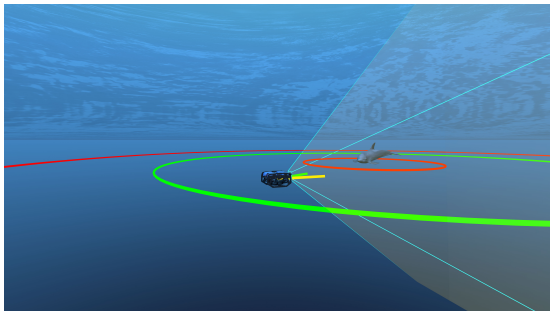

Figure 3. Debug information can be visualised. The red (green) rings represent the optimum (out-of-bounds) ranges from the target. The green (yellow) headings display the rover's optimum (current) forward direction vector. The yellow container (bounded by the cyan lines) is the onboard camera's view frustum, calculated using physical camera properties.

3.2.1. Environment Wrappers

Unity ml-agents Section 2, provides a *Gym* (**not** *Gymnasium*) wrapper, allowing the integration of a DRL pipeline on top of a Unity simulation. We do not exploit ml-agents for several reasons. Firstly, a bespoke underwater environment for continuous action spaces is unlike any of the simulations provided by ml-agents, which mainly consider discrete actions in above-ground scenarios. Second, we require an encoding pipeline as a pre-processing step to the DRL integration, for which ml-agents does not provide native support. Third, we want to exploit the latest state-of-the-art DRL algorithms and environments available (using Gymnasium and Stable Baselines 3). By using ML agents, we limit the tools we can use.

Gymnasium's environment wrappers provide additional control over the training process, computing metrics detailing the optimising process, or allowing manipulation of the training cycle. We exploit the *TimeLimit*, with a maximum episodic length of 3×10^3, and *Monitor*, providing logging information evaluating agent performance. By limiting the maximum episode length, the gains are two-fold:

1. Episodes lasting longer than the threshold length indicate that the target policy has already been optimised for the current sequence of states. It is better to reset the environment to a set of new (potentially unobserved) states than to continue the current episode, continually receiving states already in the replay/rollout buffer.
2. Resetting the environment results in state configurations that are unlikely to be observed once an episode has begun. This includes the starting and early states, where the ROV/target may have greatly varying velocities than those exhibited mid-episode.

3.2.2. Callbacks

RL solutions are often brittle and susceptible to gradient escape or early convergence. Good practice in such applications would be agent evaluation, monitoring accuracy concerning the network weights at the time of evaluation. Stable Baselines *callbacks* are triggered at specific intervals during training. Default callbacks can, for instance, inject evaluation episodes during training, while custom callbacks enable bespoke implementations. SWiMM$_{2.0}$ includes a custom evaluation callback with the following improvements from Stable Baselines:

Rollout-Based Evaluation. Regarding agent performance, we advocate evaluation only after model optimisation. Evaluation on an episode threshold means several evaluations can be made for identical networks, which is not optimal.

Step and Episode Threshold. Granted that continuous learning tasks are more concerned with step reward against the conditions at the end of an episode, we also allow step-based evaluation frequencies to support continuous learning tasks.

Minimum Steps for Evaluation. DRL networks, and therefore the target policy, are more susceptible to large changes in behaviour during the early stages of training (particularly with dynamic learning rates that anneal with time). Given that this (inexperienced) behaviour has been trained only on a limited number of samples, we provide a new criterion, a minimum training step threshold, that must be reached before evaluation can begin.

3.2.3. Physics

PhysX, a state-of-the-art physics engine developed by Nvidia, emulates physical pseudo-realism. SWiMM$_{2.0}$ manipulates object movement using PhysX forces. An internal physics step processes all physical properties. Physics updates at a constant rate, while rendering is varied.

Buoyancy

PhysX allows us to implement realistic buoyancy, discussed in our previous work [16]. As its set-up is irrelevant to the specific AI pipeline and, for brevity, we refer to the reader to the former for further details.

3.2.4. Reproducibility

Ensuring reproducible results represents a challenging task:

Firstly, Unity runs the physics and rendering systems on separate threads. Each has its own time: *fixedDeltaTime* (fixed) is the frequency at which all physical calculations are performed, while *deltaTime* (variable) also considers rendering time, updating the graphics based on the *current* physical world state. Therefore, there is always a desynchronisation between the physics engine and graphics. While these discrepancies are often discrete, they have a significant impact, as observations rely on the image data and the objects' physical attributes. In addition, DRL algorithm optimisation may be particularly susceptible to even small changes, generating very different models for identical training environments.

Second, a TCP connection introduces variable delay, exaggerated by large network messages required by image encodings. How can we remove this influence?

Our *freeze* pipeline, detailed in Section '*Action Inference*', guarantees replicable results.

To enable greater control over the physics processing of the simulation, we propose our *freeze* pipeline setting, providing a deterministic environment, that can **guarantee** reproducible results.

Freeze Pipeline

1. **First.** Native Unity physics is disabled.
2. **Second.** We perform a physics update for the rover/target. For the former, actions from the DRL network provide the emulated joystick controls, while the latter integrates the forces as determined by its state machine.
3. **Third.** We await the end of the current rendering step. At this point, physics and graphics are synchronised and the resulting image accurately represents the physical characteristics.
4. **Fourth**. Game data are retrieved, including the positions and rotations of the rover/-target. Both the game data and image render are encoded into an observation, which is then sent back across the TCP/IP socket.
5. **Fifth**. Repeat 2–4 until commanded by the network otherwise.

Algorithm 1 provides the implementation details for simulation management during such a freeze cycle.

The benefits of such a mechanism are two-fold. First, we eliminate any discrepancies resulting from the desynchronisation between physics and rendering by employing a custom physics step. Second, as the entire engine is frozen between observations, we remove the latency resulting from a TCP connection.

Algorithm 1: Freeze Pipeline Implementation

```
Class SimulationManager:
    Function Start():
        Unity.Physics.Disable();
    Function NetworkReceive(action):
        ROV.OnActionReceived(action);
    Function NetworkSend(observation):
        tcpConnection.Send(observation);
Class ROV:
    Function OnActionReceived(action):
        thrusters.SetForces(action);
        FixedUpdate() ;                          /* Process ROV physics */
        for obj in physicsObjects do
            obj.FixedUpdate();
        Unity.Physics.Simulate(Time.fixedDeltaTime);
        img ← yield return StartCoroutine(ImageRenderer.SendImageData());
        gameData ← RetrieveGameData() ;      /* Retrieve rov/target game
         state information */
        obs ← Encode([img, gameData]);
        SimulationManager.NetworkSend(obs);
Class ImageRenderer:
    Function SendImageData():
        yield return new WaitForEndOfFrame();
        renderTex ← camera.Render() ;    /* Rendering details redacted for
         brevity purposes */
        return renderTex;
```

Network

Traditional RL solutions couple their RL environment and simulation into one executable enabling direct communication. By exploiting Unity, we require a communication mechanism between the simulation and the learning architecture. As an extension from $SWiMM_{1.0}$, hosting all operations on one connection, we exploit 2 separate TCP network sockets: the first for training and the latter for evaluation. We decouple the read/write thread, enabling greater communication flexibility and higher throughput.

BLUEROV Modelling

BLUEROV includes six thrusters providing four degrees of freedom: surge; sway, heave and yaw. Our goal is the optimisation of two commands: surge and yaw, using image data alone. Our experimental section showcases that these actions suffice to provide full control of UUV when tracking marine wildlife.

Mesh

We use the standard 3D model mesh of BLUEROV (Figure 4) as provided by Blue Robotics.

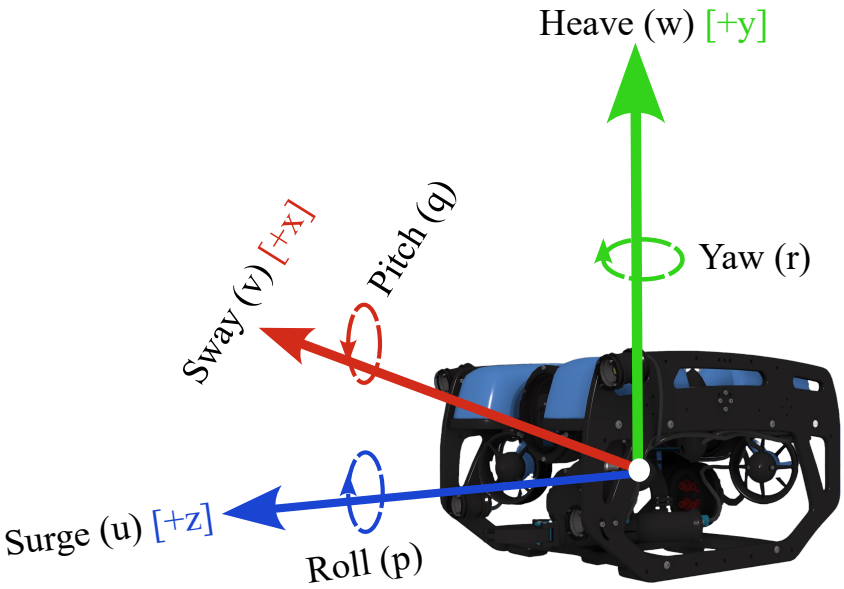

Figure 4. BLUEROV, consisting of 1.67407×10^5 tris. Designed by 3D Molier International also illustrating the 6 degrees of freedom exhibited by aquatic vehicles.

Unity provides physical camera tools including sensor size, aperture, and focal length, enabling accurate modelling of the Sony IMX322/323 image sensor (https://datasheetspdf.com/pdf-file/938855/Sony/IMX322LQJ-C/1, accessed on 21 March 2025) used in BLUEROV's Low-Light HD USB camera (https://bluerobotics.com/store/sensors-sonars-cameras/cameras/cam-usb-low-light-r1/, accessed on 21 March 2025).

Controls

The traditional control method for BLUEROV is a joystick controller sending input commands through a wired tether to the Rasberry Pi4 to a series of thrusters housed on the model's chassis (Figure 5). Thus, emulating such inputs in Unity is easily transferable, as game engines share such a control scheme.

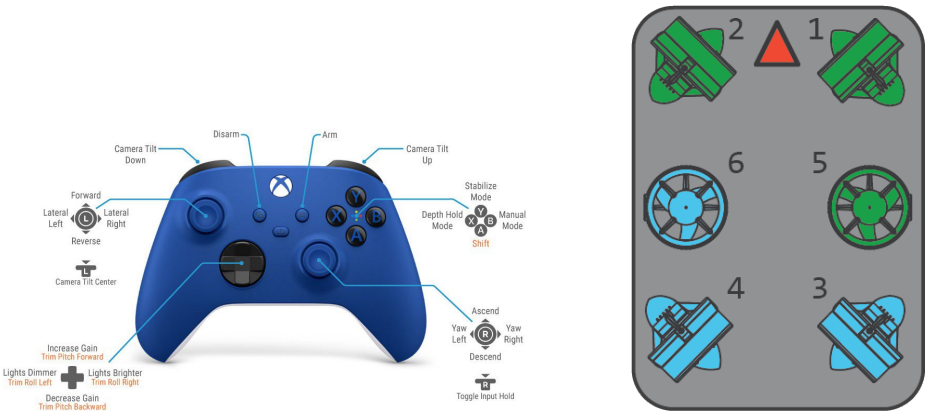

Figure 5. BLUEROV specifications (https://bluerobotics.com/learn/bluerov2-assembly/, accessed on 21 March 2025). (**Left**) Xbox joystick controller for BlueROV2. (**Right**) BLUEROV's (standard) thruster configuration.

Action Inference

Unity has a variable frame rate, and the ML pipeline takes an arbitrary time to process each observation while sending data through TCP sockets between the two introduces further delay. An arbitrary number of frames may occur between each action/observation, so how should the agent behave in these intermittent frames?

Our pipeline provided 3 'action inference' implementations, separated into the Unity physics default behaviour (△), and the custom physics handling (⋆, Section 3.2.3):

△ *maintain*. Every action is applied for all simulation ticks in between actions.

△ *onReceive*. Each action is enforced for *one* simulation tick only. The observer then waits for the next command.

⋆ *freeze*. Each action is enforced for *one* simulation tick only. Native physics behaviour is disabled. Upon receiving an action, our custom physics step is exploited, guaranteeing simulation consistency. Detailed implementation is described in Section 3.2.4.

3.2.5. Target Modelling

Building on $\text{SWiMM}_{1.0}$ [16], we improved the realism of the target model. Marine megafauna directional behaviour includes the following [41]:

- **Undulatory Locomotion**. Sinusoidal movement along the bodies and tails provides forward movement.
- **Rolling**. Enables more efficient water navigation.
- **Yawing**. Bending and flexing of bodies changes direction.

Additional animations including banking are provided in $\text{SWiMM}_{2.0}$. Figure 6 showcases some such animations at their extents.

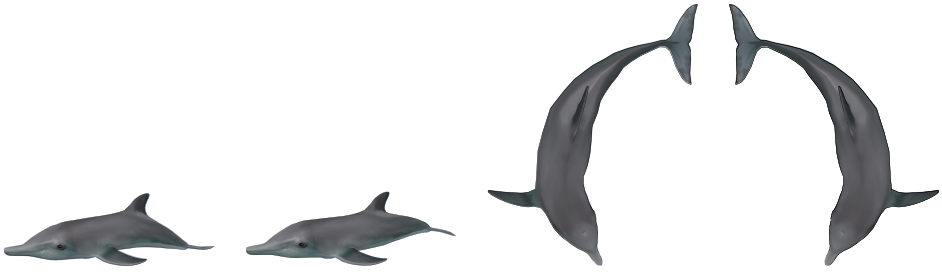

Figure 6. Common bottlenose dolphin *Tursiops truncatus* model, consisting of 3.524×10^3 tris. It includes a rigged skeleton and realistic animation patterns. Designed by Junnichi Suko (https://junichistamesi.wixsite.com/my-personal-site), accessed on 21 March 2025.

4. Results

We exploit a Razer Blade Pro laptop running Ubuntu 20.04/Windows 11: Intel Core i7-10875H CPU @ 2.30–5.10 GHz, 64 GB DDR4 2933 MHz RAM, 265 GB disk space. Due to space limitations, hyperparameters used in our experiments, such as arguments and environment configuration files are provided in our data repository (https://doi.org/10.17605/OSF.IO/7KS2C). The main packages include: Stable Baselines 3 (2.2.1); OpenAI Gymnasium (0.29.1) and Tensorflow 2 (2.10.1). GPU acceleration (CUDA v11.8, CuDNN v8) for all compatible pipeline sections, including TensorFlow and PyTorch v1.13.0. Saved models are denoted as: $\mathcal{M}_{ts}^{seed}$, where $\mathcal{M}$ is the model type, *seed* represents the seed and *ts* is the weight-freezing timestamp.

The SWiMM$_{2.0}$ pipeline is influenced by stochastic behaviours:

DRL network weight initialisation. Network weight initialisation is well known to impact a network's optimisation, evidenced by our DRL experiments.

CMVAE sampling strategy. Values sampled by our decoder are obtained from normal random distributions.

DRL action determinism. A multilayer perceptron policy for each SAC/PPO/TD3 represent *non-deterministic* policies.

Simulation Randomisation. Unity exploits random number generators, determining the environment's initial and reset state (positions, orientations, etc.).

While seeding can provide similar results, ensuring identical ones requires setting the pipeline to *freeze* (Section '*Action Inference*').

$IMAGE_{\text{TRAIN}}$/$IMAGE_{\text{TEST}}$ denote our CMVAE training/test sets (90/10 split, 2.7×10^5 and 3×10^4 samples, respectively). For these, rover/target position and rotation and, for the latter, animation frames (for both the previous and new poses) ensure uniformly distributed states. A large sample size provides a more robust state-space coverage. $IMAGE_{\text{INTERPOLATE}}$ contains 6 samples, whereby the target $d/\theta/\psi$ fields are set between their minimum and maximum extents with all other fields being constant.

$IMAGE_{\text{SIMILARITY}}$ (1×10^3 samples) denotes our image similarity dataset, undergoing the same randomisation process as the previous with additional resolutions.

4.1. Data Generation Setup

As an extension of our previous solution, randomised fields are limited to ranges within our episode termination thresholds (Section '*Episode Termination*'). We exploit early episode termination during the DRL training cycle (Section '*Episode Termination*', which includes distance thresholds) and therefore restrict the Euclidean distance to within the threshold only. Table 5 details the data generation for the image sampling regarding image specifications while Table 6 details the environment configuration.

Table 5. The data amounts required for the training and benchmarking of the CMVAE. *n* denotes the number of samples contained in each set. By exploiting different seeds, we can guarantee a mutual exclusion between training, validation, and test data.

Set	*n*	Resolutions (W × H)	Seed
$IMAGE_{\text{SIMILARITY}}$	1×10^3	$\{ (1920, 1080), (640, 360), (64, 64) \}$	2
$IMAGE_{\text{TRAIN}}$	2.7×10^5	$\{ (64, 64) \}$	3
$IMAGE_{\text{TEST}}$	3×10^4	$\{ (64, 64) \}$	5
$IMAGE_{\text{INTERPOLATE}}$	6	$\{ (1920, 1080) \}$	7

Table 6. Rover telemetry feature randomisation and target positioning and animation limits. Distance is relative to the rover. Animation is merged in a blend tree. $x/y/z$ values are relative to global coordinates $(0, 0, 0)$. Data was sampled from a normal distribution, [Seed = 2].

	Range (Unity *x*/*y*/*z*)
Rover	
Position	$([-30, 30], [-4, 4], [-30, 30])$
Rotation	$([0, 0], [-180, 180], [0, 0])$
Target	
Euclidean Distance	$[2, 10]$
Rotation	$([0, 0], [-180, 180], [0, 0])$
Forward Animation Weight	$[1, 1]$
Turn Animation Weight	$[-1.3, 1.3]$
Normalised Animation State	$[0, 1]$

4.2. Image Similarity

Emulating a 1920×1080 video stream introduces the following problems:

1. **Texture Rendering.** The cost of rendering high-resolution images is high, from both a memory and time perspective, in particular for sample-hungry DRL algorithms.
2. **Message Latency.** Latency defines the ratio between message size and throughput of our TCP connection. High-resolution image vectors necessitate large buffer size requirements, introducing a significant delay between observation and control commands.

Our CMVAE receives 64×64 images. Therefore, we assess reducing rendering computation and network load by exploiting the low-dimensional input resolution directly. Thus we asked the following:

1. Can we exploit image scaling techniques in Unity to reduce network traffic latency?
2. If above, can we render our input resolution directly, reducing rendering load?

One image is defined as $\mathbb{R}^W \times \mathbb{R}^H \times \mathbb{R}^C$-dimensional tensor img, where $W/H/C$, respectively, denote the image's width, height, and number of RGB colour channels. We require $C = 3$, as colour features help better highlight the target object. A pixel, p_w^h, at a width, w and height h, is denoted as a tuple $p_w^h = (r, g, b) \mid \forall(c) \in p,\ \ 0 \leq c \leq 255$.

SWiMM$_{1.0}$ exploited cosine similarity, which, while indicating the shape differences between images, is insensitive to scaling factors:

$$Similarity_{SWiMM_{1.0}} = 1 - \frac{\sum_{i=1}^{W}\sum_{j=1}^{H} p_i^j \cdot {p'}_i^j}{\sqrt{\sum_{i=1}^{W}\sum_{j=1}^{H}(p_i^j)^2} \cdot \sqrt{\sum_{i=1}^{W}\sum_{j=1}^{H}({p'}_i^j)^2}} \tag{7}$$

We opt for using MAE, considering absolute pixel-wise differences, a more appropriate metric for comparing the 'same' image:

$$Similarity_{SWiMM_{2.0}} = \frac{1}{W \cdot H}\sum_{i=1}^{W}\sum_{j=1}^{H}|p_i^j - {p'}_i^j| \tag{8}$$

We first determine that the choice of resizing algorithm (client versus server) produces almost negligible differences (in the worst case, MAE = 1.09). We then observed that rendering in high resolution and down-scaling versus rendering in low resolution directly achieves very low data loss (MAE = 2.03). The gains of our proposed optimisations far outweigh minimal data losses. Throughout the rest of this paper, $W/H/C = 64/64/3$.

4.3. CMVAE Training

Our preparatory experiments train our CMVAE, providing a lower-dimensional input for the latter DRL network. We exploited the network architectures proposed by Bonatti et al. [31]. We enhance $SWiMM_{1.0}$ as follows:

Early Stopping. To prevent any unnecessary training providing no benefit to models having already found a maximum optimum [42], we monitor validation losses concerning a window size ω. Where no monitoring exists, we denote this as $\omega = \infty$, otherwise, an integer n indicates the patience value.

State-of-the-art DNN Libraries. $SWiMM_{1.0}$ exploited old DNN packages, including Tensorflow v1.14.0, and downgraded the implementation of [31]. $SWiMM_{2.0}$ exploits modern networks from Tensorflow 2 including the modern implementation from Bonatti et al. [31], enabling improved performance and accuracy.

4.3.1. Training

$SWiMM_{1.0}$ required 100 epochs. As optimisation may occur far before this number, we dissect our training into 2 batches. The first batch ($\omega = \infty$) operates under the same conditions as $SWiMM_{1.0}$. The second batch ($\omega = 5$) monitors validation losses using a sliding window for the previous 5 iterations requiring a minimum validation loss reduction of $\geq$1%. We exploit $IMAGE_{\text{TRAIN}}$ with five seeded training runs. The 'best' network weights are saved for the lowest validation loss concerning image and state data reconstruction. Our preliminary experiments, detailed in Supplement S1, justify our early stopping approach. Table 7 defines the configuration exploited during training of the variational autoencoder and Figure 7 displays the results of our second batch of experiments, $\omega = 5$.

The 'best' model's weights (lowest validation loss, Figure 7), are saved: $CMVAE_{23}^{47}$/ $CMVAE_{33}^{53}$/$CMVAE_{23}^{59}$/$CMVAE_{29}^{61}$/$CMVAE_{25}^{67}$.

Table 7. CMVAE training configurations.

Parameter	Definition	Value	
		$w = 5$	$w = \infty$
	cmvae_training_config		
train_dir	Training samples directory (see Table 5)	$IMAGE_{\text{TRAIN}}$	$IMAGE_{\text{TRAIN}}$
batch_size	Size of splices for training data	32	32
learning_rate	Gradient learning rate	1×10^4	1×10^4
load_during_training	Should all images undergo splicing on the fly	True	True
max_size	Limit on number of samples	NULL	NULL
epochs	Maximum number of epochs to run	100	100
window_size	Epochs to analyse for early stopping criteria	5	NULL
loss_threshold	Fractional gain to reach within window_size	1×10^{-2}	1×10^{-2}
	cmvae_global_config		
use_cpu_only	Prevent GPU acceleration	False	False
n_z	Latent space size	10	10
img_res	Expected Resolution of Input Data	[64, 64, 3]	[64, 64, 3]
latent_space_constraints	Disentangle latent space constraints [31]	True	True
deterministic	Tensorflow determinism	True	True

Table 7. *Cont.*

Parameter	Definition	Value	
		$w = 5$	$w = \infty$
	env_config		
...			
seed	Seed exploited for stochastic operations	$\{11, 13, 17, 19, 23\}$	$\{47, 53, 59, 61, 67\}$
...			

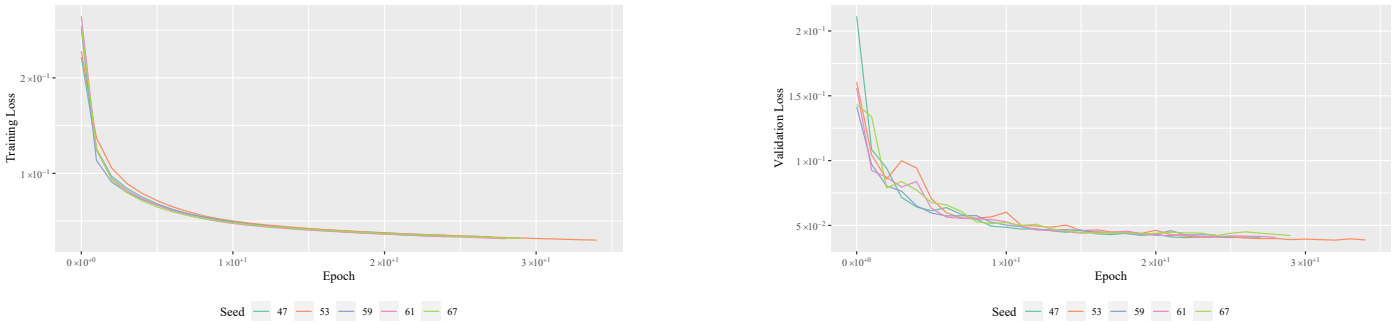

Figure 7. Training (**left**) and validation (**right**) losses, $\omega = 5$.

4.3.2. Inference

The 'best' models generated from Section 4.3.1 are evaluated. In total, 1×10^3 images and state data are sampled from $IMAGE_{\text{TEST}}$ for 5 different seeds.

Firstly, we want to investigate how MAE varies with training time (Figure 8). A network pass-through is performed: encoding, decoding, and returning the image and state reconstructions along with the latent space vector z. Given the ground truth values of the state data and images versus those reconstructed, we record the MAE for each image/d/θ/ψ. These values are normalised considering their maximum possible errors (255/4/360/360, respectively). d and ψ exhibited the worst reconstructions, though these values are still low, $MAE < 1.00 \times 10^{-1}$. Conversely, image and θ, had exceptional reconstructions, often achieving a MAE of 1%. Marginal accuracy gains are achieved for models where $\omega = \infty$, while training times suffer a great increase of almost 4×. Given that there will always be a trade-off between training time and optimisation, we select CMVAE_{33}^{53} as our CMVAE of choice, coupling low MAE with low training times.

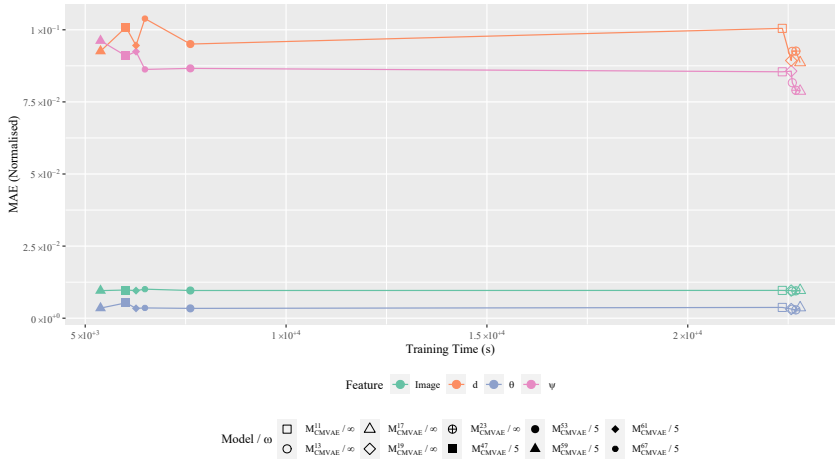

Figure 8. MAE (normalised) versus training times across 4 features: image, d, θ and ψ. Both models with/without early termination ($\omega = \infty/\omega = 5$) are evaluated.

Second, we investigate how accurately CMVAE_{33}^{53} can infer cross-modal features against the former CMVAE from $\text{SWiMM}_{1.0}$ and, from the original solution, Airsim [31], by comparing the raw MAE values obtained from the previous experiment (Table 8). $\text{SWiMM}_{1.0}$ outperformed Airsim with θ but not d and was unable to infer ψ accurately, while $\text{SWiMM}_{2.0}$ improved across all fields, in particular d, achieving a loss reduction of 3.30×. These gains result primarily from the more optimised datasets (Section 4.1). Regarding ψ, while $\text{SWiMM}_{2.0}$ makes considerable gains from $\text{SWiMM}_{1.0}$ (even with the addition of extended animations), Airsim still achieves the lowest loss. While Airsim simulated square gates, symmetrical and uniform in all dimensions, SWIM models dolphin meshes, introducing a more complex image to infer a correct state. Table 9 defines the inference configuration.

Table 8. MAE when predicting target distance (d), azimuth (θ), and yaw (ψ) in SWiMM DEEPeR and AirSim [31]. Red boxes indicate worst performance while bold font indicates the best (lowest) values per feature.

	Env		
	Airsim	**$\text{SWiMM}_{1.0}$**	**$\text{SWiMM}_{2.0}$**
d	$3.9 \times 10^{-1} \pm 2.30 \times 10^{-2}$	$1.25 \times 10^{0} \pm 8.00 \times 10^{-3}$	$\mathbf{3.80 \times 10^{-1} \pm 1.08 \times 10^{-2}}$
$\theta[^\circ]$	$2.6 \times 10^{0} \pm 2.30 \times 10^{-1}$	$1.41 \times 10^{0} \pm 1.2 \times 10^{-2}$	$\mathbf{1.23 \times 10^{0} \pm 3.07 \times 10^{-2}}$
$\psi[^\circ]$	$\mathbf{1.00 \times 10^{1} \pm 7.50 \times 10^{-1}}$	$5.00 \times 10^{1} \pm 3.29 \times 10^{-1}$	$3.12 \times 10^{1} \pm 1.26 \times 10^{0}$

Table 9. CMVAE evaluation configurations.

Parameter	**Definition**	**Value**	
		$w = 5$	$w = \infty$
cmvae_inference_config			
test_dir	Testing samples directory (see Table 5)	$IMAGE_{\text{TEST}}$	$IMAGE_{\text{TEST}}$
weights_path	Model's network weights path	$\text{Path}(\mathcal{M}_{CMVAE}^{s \mid s \in \{11,13,17,19,23\}})$	$\text{Path}(\mathcal{M}_{CMVAE}^{s \mid s \in \{47,53,59,61,67\}})$
cmvae_global_config, see Table 7			
env_config			
...			
seed	Seed exploited for stochastic operations	$\{29, 31, 37, 41, 43\}$	$\{71, 73, 79, 83, 89\}$
...			

Third, to benchmark the effectiveness of the image reconstruction, we randomly sampled 8 images from $IMAGE_{\text{TEST}}$. One pass-through of CMVAE_{33}^{53} returned the reconstructed image, encoded into z, and decoded into the desired dimensions. The results are shown in Figure 9. The target's shape, size, and colour are always maintained during image reconstruction.

Fourth, given 2 images containing the target in distinct locations and rotations, can an interpolation between the z values produce a smooth transition from one image to the other? To do so, we exploited $IMAGE_{\text{INTERPOLATE}}$. This consists of images containing all mix/max values for r; θ and ψ, keeping constant the other features. Then, given the latent space z from the minimum and maximum, we artificially generate a new z, interpolating some degree between the two. This is then passed back through the decoder to reconstruct the artificial image. Figure 10 demonstrates that the Dronet [32] architecture has correctly optimised its network.

Fifth, can we extract meaningful behaviour through feature-wise interpolation between the minimum and maximum values for each feature encoded in z ($dims|z| = 10$)?

By exploiting the constrained model proposed by [31], the first 3 features of z are forced to encode to the relative pose, as shown by Figure 11. While the first 3 rows reveal the first three features are task-relevant, the latter 7 features are more difficult to interpret from a visual perspective.

Figure 9. Reconstruction pairs from 8 arbitrary images (original raw images are on the left followed by their decoded counterpart on the right).

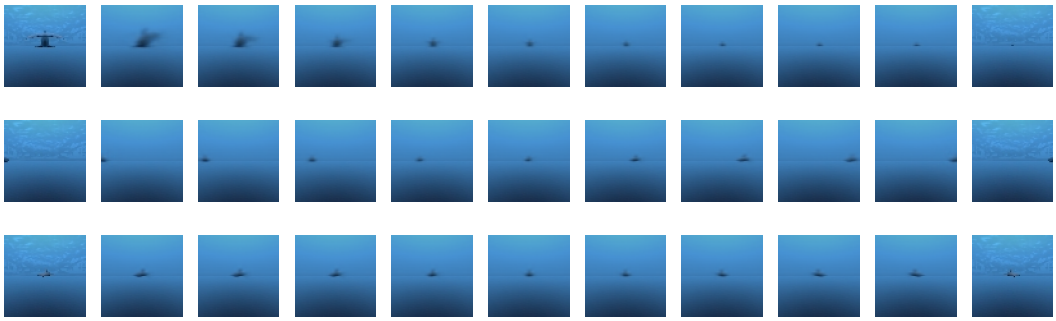

Figure 10. Six images are taken in the simulation (the far left/right of each row): the min/max values for each r (Top)/θ (Middle)/ψ (Bottom) (keeping constant the other 2 features). These images are encoded to generate their feature vectors. The resulting z vectors are then linearly interpolated between, where each newly generated vector is decoded and reconstructed as a new image (the middle 9 images).

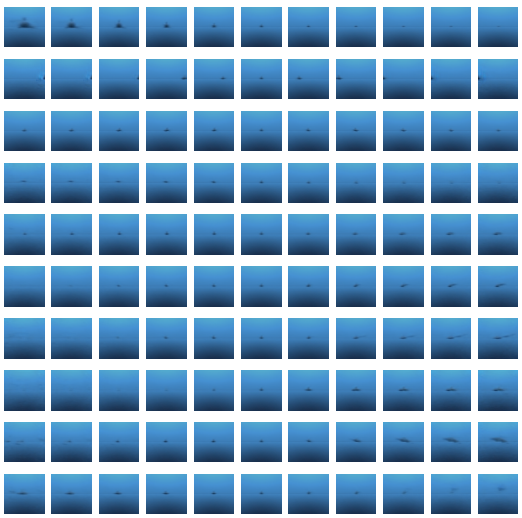

Figure 11. Interpolation results between min/max values for each feature of z. The first 3 rows ($r/\theta/\psi$) demonstrate a successful disentanglement of the learnt feature space.

Last, given 2 random images from $IMAGE_{\text{TEST}}$, can an interpolation between the z values generated by both provide a meaningful translation from one to the other? An interpolation between the 2 z values generates a sequence of 10 intermediate results. These values are processed through the image decoder, shown by Figure 12. The intermediate images demonstrate that the CMVAE has successfully learnt the relevant features.

Figure 12. Interpolated z values are decoded from 2 randomly-sample images from $IMAGE_{\text{TEST}}$.

4.4. Reinforcement Learning

$\text{SWiMM}_{2.0}$ exploits the SAC/PPO/TD3 algorithms provided by Stable Baselines with PyTorch GPU acceleration. We now also consider the best SAC model from $\text{SWiMM}_{1.0}$, denoted as $\text{SAC}_{\text{SWiMMv1.0}}$/$\text{SAC}_{\text{SWiMMv2.0}}$.

Table 10 defines the configuration exploited during training of the each DRL algorithms.

Table 10. DRL Training Configurations.

Parameter	Definition	Value
	cmvae_inference_config	
...		
weights_path	Path to the 'best' model's network weights	$\mathcal{M}^{53}_{CMVAE}$
...		
	cmvae_global_config, see Table 7	
	env_config	
obs	Observation type	'cmvae'
n_envs	Number of environments to run in parallel	1
img_res	Resolution of incoming images	$[64, 64, 3]$
debug_logs	Should network logs be generated	FALSE
seed	Seed exploited for stochastic operations	$\{ 97, 101, 103, 107, 109 \}$
algorithm	DRL algorithm to train with	{ 'sac', 'ppo', 'td3' }
render	Mode of rendering the Unity environment	'human'
pre_trained_model_path	Model to either: train on-top of or to use for inference	~
	env_wrapper	
	stable_baselines3.common.monitor.Monitor	
allow_early_resets	Are we allowing early episode termination	TRUE
	gymnasium.wrappers.time_limit.TimeLimit	
max_episode_steps	Allow at most max_episode_steps per training episode	3×10^3
	callbacks	
	gym_underwater.callbacks.SwimEvalCallback	
eval_inference_freq	How many episodes/steps to run the evaluation for	[2, 'episode']
eval_freq	Training episodes required per evaluation sequence	10
min_train_steps	Minimum number of training steps required to begin evaluation	5×10^5
deterministic	Should the actor use deterministic or stochastic actions	TRUE
verbose	Output information and metrics	1
	gym_underwater.callbacks.SwimCallback	

4.4.1. Training

Firstly, 5 training runs are performed for each of SAC/PPO/TD3 for 1×10^6 steps (distinct from the maximum episode length of 3×10^3 steps) and seeds $s \in \{97, 101, 103, 107, 109\}$. For monitoring, a window size, *stats_window_size* $= 10$, records statistical data for the previous 10 episodes. We maintain a memory of the previous 10 actions for *ActPrev* (Section '*Observation Space*'). Network architectures are denoted as net_arch $= [hl_0, hl_1, \ldots, hl_n]$, where for any i, net_arch$i$ indicates the number of neurons at hidden layer i. Both SAC/ TD3 exploit a 2-layered DNN (net_arch$_{SAC} = [64, 64]$, net_arch$_{TD3} = [400, 300]$) while PPO exploits a single layer (net_arch$_{PPO} = [256]$). In place of hyperparameter tuning, we exploit well-tested configurations (https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/hyperparams, accessed on 21 March 2025) previously demonstrating successful results amongst similar environments. SAC/PPO/TD3's hyperparameters are provided in Supplement S2, Tables S1–S3. We include the *Monitor* and *TimeLimit* environment wrappers. Results are shown by Figure 13.

Concerning mean episodic reward, SAC far outperforms both PPO and TD3, across **all** iterations. SAC, in most cases, achieves high episodic rewards ($>2 \times 10^3$, best case 2.5×10^3) within a fraction of the time. PPO performs poorly across all instances, achieving consistently low mean episodic rewards ($<2.5 \times 10^2$). Even in the latter stages of training, episodes are terminated almost immediately due to exceeding the distance threshold or the target leaving the camera's view, as the target policy either steers the rover away from the target or provides the incorrect turning direction. TD3 displays the most volatile behaviour, with most iterations (seeds 97, 101, and 109) behaving similarly to PPO, with the remaining (seeds 103 and 107) achieving much higher rewards (5×10^2).

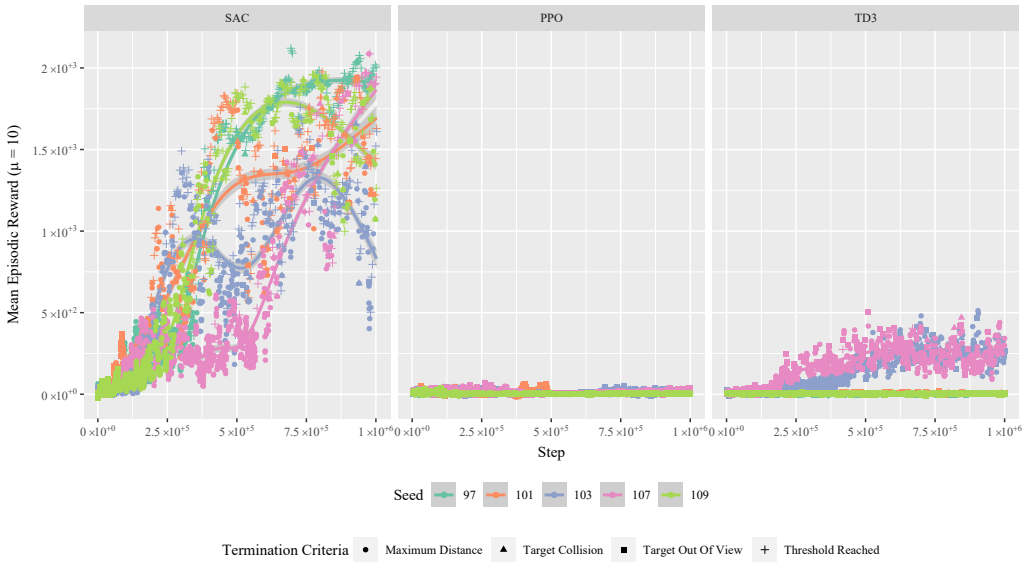

Figure 13. Mean episodic training reward for SAC/PPO/TD3, with *stats_window_size* $= 10$. Every terminated episode is denoted by a point, where the point shape denotes the cause.

4.4.2. Evaluation

We inject evaluations halfway through training, at 5×10^5 training steps, performing 2 evaluation episodes every 10 training episodes onwards. Significantly, given that the target is guaranteed to spawn in a reasonable location to the rover on each episode reset (where step reward is guaranteed to be positive), achieving a negative episodic reward indicates extremely poor behaviour, where the agent has failed to track the target completely.

To better gauge performance in reality, our evaluation episodes lack early termination; they continue until the time limit is reached (3×10^3 steps). The minimum/maximum step

reward ($[-1, 1]$) therefore bounds the minimum/maximum episode reward to be in the range $[-3 \times 10^{-3}, 3 \times 10^{3}]$. Figure 14 displays the evaluation results while Table 11 details the timestamps and optimum values found per algorithm per seed.

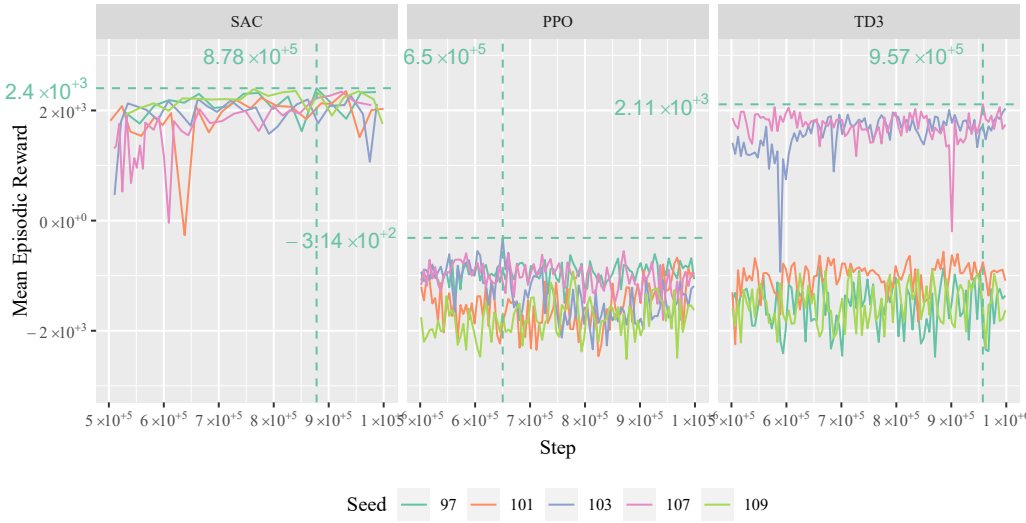

Figure 14. Mean episodic reward per evaluation SAC/PPO/TD3. Evaluation episodes begin at 5×10^5 steps. The vertical/horizontal lines display the step/reward where the highest mean episodic reward was recorded, determining the models exploited for the inference Section 4.4.5.

Table 11. Evaluation results denoting the timestamps and values of the best evaluations per algorithm per seed: (Top) SAC, (Middle) PPO, (Bottom) TD3. Each algorithm has its best model listed (also displayed in bold): $\text{SAC}^{97}_{8.78\times10^5}/\text{PPO}^{103}_{6.50\times10^5}/\text{TD3}^{107}_{9.57\times10^5}$.

	Seed	Timestamp	Value
SAC	97	8.78×10^5	$\mathbf{2.40 \times 10^3}$
	101	9.31×10^5	2.35×10^3
	103	9.39×10^5	2.28×10^3
	107	9.23×10^5	2.34×10^3
	109	7.63×10^5	2.39×10^3
PPO	97	9.27×10^5	-6.07×10^2
	101	9.68×10^5	-6.73×10^2
	103	6.50×10^5	$\mathbf{-3.14 \times 10^2}$
	107	8.70×10^5	-5.53×10^2
	109	7.78×10^5	-9.24×10^2
TD3	97	6.60×10^5	-8.36×10^2
	101	8.86×10^5	-5.66×10^2
	103	9.98×10^5	2.04×10^3
	107	9.57×10^5	$\mathbf{2.11 \times 10^3}$
	109	7.17×10^5	-8.67×10^2

Training/evaluation episodes share performance. SAC achieves consistently high rewards and the highest reward, while PPO suffers, with even the best model achieving negative rewards; a near-random control policy. TD3, displays large fluctuations in behaviour. Seeds 97, 101 and 109 are comparable with the poor performance from PPO, while seeds 103 and 107 achieve performance similar to SAC. However, even in these

instances, SAC outperforms in **all** instances. The best models (model (reward)) are as follows: $SAC^{97}_{8.78\times10^5}$ (2.4×10^3), $PPO^{103}_{6.50\times10^5}$ (-3.14×10^2) and $TD3^{107}_{9.57\times10^5}$ (2.11×10^3).

4.4.3. Training Times

We investigate how training times vary among model performance across algorithms. First, we consider the factors influencing running time.

Training Time Influencing Factors

RL FPS (a measure of the average number of steps (and therefore frames) processed per second) is a commonly used metric to analyse performance, yet running time and FPS are measures of the same variable, *time*. With each algorithm bound by the maximum training timesteps (1×10^6), and given that the rest of the pipeline for each of SAC/PPO/TD3 remains constant, we would like to highlight how the FPS may be influenced:

Model Performance. Well-behaving models regularly achieve the maximum episode length, while a poorly behaving one activates some of our episode termination criteria (Table 4). The latter, activating early episode termination, results in lower FPS, due to the high resource demand of environment reset.

Network Optimisation Frequency. While SAC performs network updates per episode, PPO/TD3 perform theirs every 512 step. Therefore, the latter exhibits an (almost) constant number of updates per run, while the former performs an indeterminable number of updates. Well-performing models will suffer from the latter due to the more regular (and unnecessary) updating of network weights.

Network Architecture. SAC/PPO/TD3 exploit different network architectures (Section 4.4.1), in both the number of hidden layers and neurons per layer. More complex networks have a higher resource demand.

Algorithmic Implementation. TD3 exploits twinned critics while SAC and PPO exploit only one, doubling the demand on critic network optimisation.

Training Time Benchmarks

Figure 15 displays the results. Concerning total training time, SAC (obtaining a mean running time of 5.82×10^4 s) demonstrates a significant gain over PPO/TD3 (8.44×10^4/7.58×10^4), a 1.45/1.30× speedup. All algorithms demonstrate little volatility in total training time across all instances with *every* SAC/TD3 instance achieving lower times than TD3/PPO. We also record the time required to discover the best model (higher alpha bars), given that these are the models exploited for our latter experiments. For this, SAC (5.06×10^4 s) again outperforms PPO/TD3 (6.50×10^4/5.88×10^4), a 1.28/1.16× speedup. However, there is much greater volatility in these values per algorithm than for total training time. For SAC, in the worst case (seed/time = $107/5.73 \times 10^4$) is considerably slower than the best cases for PPO($103/4.20 \times 10^4$)/TD3($97/3.96 \times 10^4$). Still, the mean running time of 5.82×10^4 s obtained by SAC is a significant improvement over the training time of 9.38×10^4 s for the SAC model in $SWiMM_{1.0}$ (albeit for double the number of total timesteps, 2×10^6).

SAC (5.82×10^4 s) demonstrates a 1.45/1.30× training time speedup over PPO/TD3 (also a significant improvement over $SAC_{SWiMMv1.0}$ (9.38×10^4 s)). Best model discovery exhibits a higher variance, though SAC still achieves a 1.28/1.16× speedup against PPO/TD3.

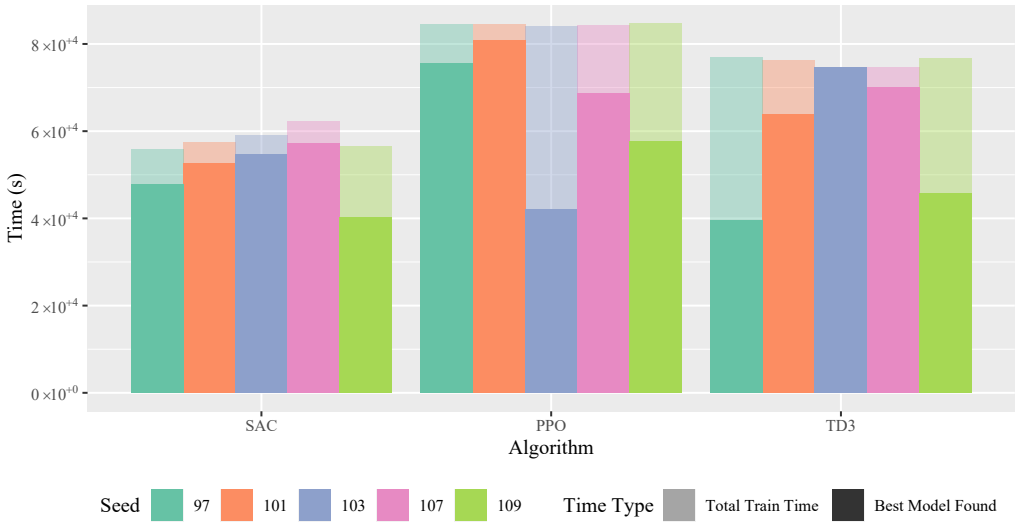

Figure 15. Mean total training time for each of SAC/PPO/TD3. The best model timestamps reached are visualised with lower alpha values.

4.4.4. Model Behaviour Metrics

Reward values do not accurately estimate objectives. Learned target policy is often judged with the naked eye, which is also prone to bias. We have no grounded quantifiable means for model-to-model analysis to compare the learned target policies.

Our custom behaviour error metrics (Table 12, one per our initial objectives (Section 1.2)) directly correlate with intended agent behaviour, allowing for quantitative comparison between the best SAC model discovered in our previous pipeline, $\text{SWiMM}_{1.0}$. First, $\mathcal{D}/\mathcal{A}$ provide a notion of the rover's control accuracy, stating the mean distance/angle away from their optimum. Second, the more serious of these errors are 'safety conditions'; errors either significantly harming the BLUEROV/target, i.e., a collision between rover and target ($\mathcal{C}'$), or rendering the rover in a comprising state that no longer guarantees reliable autonomous control (such as exceeding the maximum distance threshold $\mathcal{D}'$, or losing sight of the target entirely $\mathcal{A}'$). The remaining metrics $\mathcal{S}_\mathcal{D}$ and $\mathcal{S}_\mathcal{A}$ are concerned with the stability of surge and yaw thrust, with high values indicating high amounts of jitter.

Table 12. Metric definitions for final model analysis. All metrics are normalised against their largest respective error.

Objective (Section 1.2)	Symbol	Safety Condition	Formula (Normalised)
Objective 2	$\mathcal{D}$	✗	Mean d error
Objective 1	$\mathcal{A}$	✗	Mean θ error
Objective 2	$\mathcal{D}'$	✓	Distance threshold exceeded (max 1 per episode)
Objective 1	$\mathcal{A}'$	✓	Target visibility lost (max 1 per episode)
Objective 3	$\mathcal{C}'$	✓	Total number of collisions (max 1 per episode)
Objective 4	$\mathcal{S}_\mathcal{D}$	✗	Mean distance smoothness error
Objective 4	$\mathcal{S}_\mathcal{A}$	✗	Mean yaw smoothness error

$\mathcal{D}/\mathcal{A}$ are normalised between α/max_d. $\mathcal{D}'/\mathcal{A}'/\mathcal{C}'$ are normalised against the total number of evaluations episodes, 100. $\mathcal{S}_\mathcal{D}/\mathcal{S}_\mathcal{A}$ are detailed in Section '*Smoothness Error Implementation*'.

4.4.5. Inference

The purpose of the following experiments is two-fold. Firstly, we provide a direct quantitative benchmark of our objectives (Section 1.2). Second, rewards validate the correctness of our reward function. We also consider $\text{SAC}_{\text{SWiMMv1.0}}$. We exploit the same callbacks and setup as our evaluation experiments, with an increased number of episodes to 20 for a further 5 seeds, $s \in \{113, 127, 131, 137, 139\}$. We also provide videos for each: $\text{SAC}_{\text{SWiMMv1.0}}$ (https://youtu.be/Ere3rlZWChw, accessed on 21 March 2025)/$\text{SAC}_{\text{SWiMMv2.0}}$ (https://youtu.be/T3EWD9sNQ_o, accessed on 21 March 2025)/PPO (https://youtu.be/ZnuBpzF9l8, accessed on 21 March 2025)/TD3 (https://youtu.be/OL-aF9csEPY, accessed on 21 March 2025). Results are displayed in Figure 16.

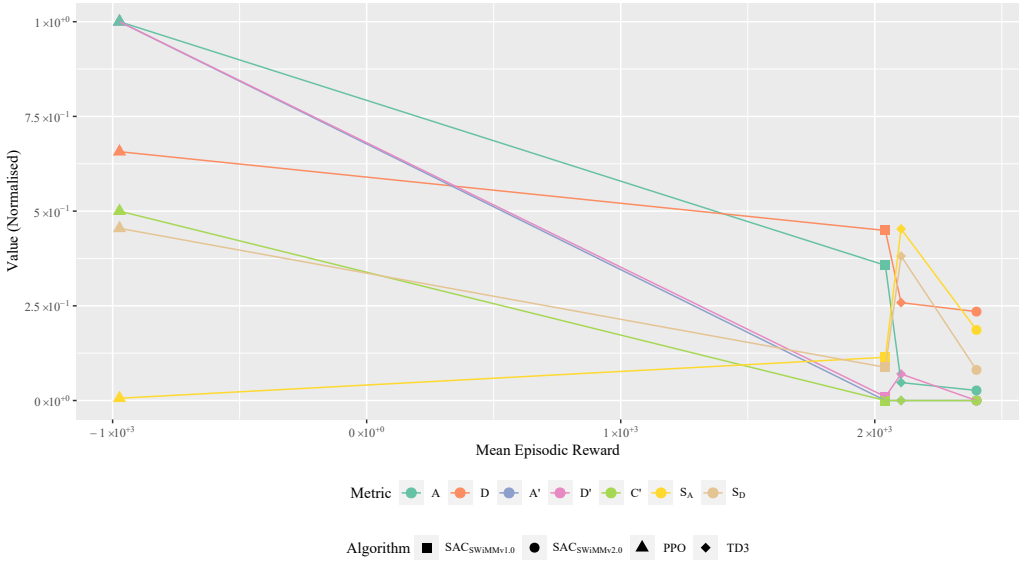

Figure 16. Inference results per algorithm. Twenty episodes are run per seed. Some points may overlap (e.g., $\mathcal{D}'/\mathcal{A}'/\mathcal{C}' = 0$) for $\text{SAC}_{\text{SWiMMv2.0}}$).

Episodic reward has a direct negative correlation with metric errors. PPO achieves the lowest rewards of -9.72×10^2 and suffers the highest errors for almost all metrics; $\mathcal{S}_\mathcal{A} = 6.04 \times 10^{-3}$ indicates smooth (but erroneous) behaviour. Distance smoothness errors $\mathcal{S}_\mathcal{D} = 4.55 \times 10^{-1}$, indicate high surge volatility. Across 100% of episodes, the sight of the target is lost, and the maximum distance threshold is exceeded, and in 50% of episodes, a collision occurs. $\mathcal{A} = 1.0$, indicates the highest error, indicating the target is rarely within view. $\mathcal{D} = 6.57 \times 10^{-1}$ as the rover fails to maintain the optimum distance.

$\text{SAC}_{\text{SWiMMv1.0}}$ achieves much higher rewards (2.04×10^3). Against the former, **all** (aside $\mathcal{S}_\mathcal{A}$) errors were reduced. Significantly, $\mathcal{A}'$ and $\mathcal{C}'$ were never violated and $\mathcal{D}'$ was violated only once. $\mathcal{A}$ was more than halved to 3.57×10^{-1} while distance errors were also reduced to $\mathcal{D} = 4.49 \times 10^{-1}$. Smooth surge control was achieved, $\mathcal{S}_\mathcal{D} = 8.77 \times 10^{-2}$.

TD3 makes slight reward gains to 2.10×10^3, corresponding with significant reductions in $\mathcal{D}/\mathcal{A}$; the latter being drastically reduced by almost an order of magnitude to 4.74×10^{-2}. A sharp increase in $\mathcal{S}_\mathcal{A} = 4.53 \times 10^{-1}/\mathcal{S}_\mathcal{D} = 3.81 \times 10^{-1}$, indicates the rover cannot achieve smooth control. $\mathcal{D}'$ rose to 7×10^{-2}, aligning with more abrupt control.

$\text{SAC}_{\text{SWiMMv2.0}}$ achieved the highest reward of 2.40×10^3. $\mathcal{D}/\mathcal{A}$ were further reduced, with $\mathcal{A} = 2.67 \times 10^{-2}$, an azimuth error of just over one degree and $\mathcal{D} = 2.35 \times 10^{-1}$, a value less than one metre. $\mathcal{D}'$ was reduced to 0. Notably, $\mathcal{S}_\mathcal{A} = 1.86 \times 10^{-1}$ lowers into a much more stable range and $\mathcal{S}_\mathcal{D} = 8.08 \times 10^{-2}$, the lowest for the metric. Table 13 defines the configuration exploited during inference of the each DRL algorithms.

Table 13. DRL Inference Configurations.

Parameter	Definition	Value
	cmvae_inference_config, see Table 10	
	cmvae_global_config, see Table 7	
	env_config	
obs	Observation type	'cmvae'
n_envs	Number of environments to run in parallel	1
img_res	Resolution of incoming images	$[64, 64, 3]$
debug_logs	Should network logs be generated	FALSE
seed	Seed exploited for stochastic operations	$\{ 113, 127, 131, 137, 139 \}$
algorithm	DRL algorithm to train with	{ 'sac', 'ppo', 'td3' }
render	Mode of rendering the Unity environment	'human'
pre_trained_model_path	Model to either: train on-top of or to use for inference	$\text{Path}(\mathcal{M}_{DRL}^{s \mid s \in \{ 97,101,103,107,109 \}})$
	env_wrapper, see Table 10	
	callbacks	
	gym_underwater.callbacks.SwimEvalCallback	
eval_inference_freq	How many episodes/steps to run the evaluation for	[20, 'episode']
eval_freq	Training episodes required per evaluation sequence	10
min_train_steps	Minimum number of training steps required to begin evaluation	5×10^5
deterministic	Should the actor use deterministic or stochastic actions	TRUE
verbose	Output information and metrics	1
	gym_underwater.callbacks.SwimCallback	

4.4.6. Model Robusteness to Noise

The previous experiments demonstrated that SAC can learn a successful behaviour policy satisfying our objective desiderata. While the previous environment operated under clear conditions, more realistic environments are susceptible to noise, such as water clarity, optic distortion, and blurred focus. Then, to address the robustness of the proposed methodology under noisy conditions, we perform 2 separate experiments: one for the image pre-processing and CMVAE, and the other concerning DRL working on noisy data.

To simulate such noise, we extend our base pipeline to use Unity's 'Post-Processing Stack v2'. Where applicable, we also simulate the real-world specification of the Sony IMX322/323 image sensor. The following post-processing effects are included into our pipeline:

Enhanced Fog: Unity fog overlays colour onto pixels concerning their distance from the camera. Our enhanced fog exploits the squared exponential distance. This enables a much greater emphasis on a murky underwater effect in contrast with $\text{SWiMM}_{1.0}$, which used exponential with lesser weight only.

Grain: Grain exploits a gradient coherent-noise function to simulate irregularities in film tape. In our simulation, we exploit grain to emulate stochastic underwater particles particularly those causing reflection.

Depth of Field: This simulates the focusing properties of a camera lens, enabling physically accurate Bokeh depth of field effects.

Lens Distortion: Warps pixel data to make objects bend inward (pincushion distortion) or outward (barrel distortion). We provide barrel distortion to emulate underwater fisheye lens.

Motion Blur: When objects move faster than a camera's exposure time, they are blurred. We provide a camera-based motion blur.

While for small resolution rendering, such as the 64×64 image renders we provide, such noise may be difficult to visualise with the naked eye (Figure 17), but could significantly affect the optimisation process. Under certain conditions, the distortion becomes much more apparent as showcased by Figure 18.

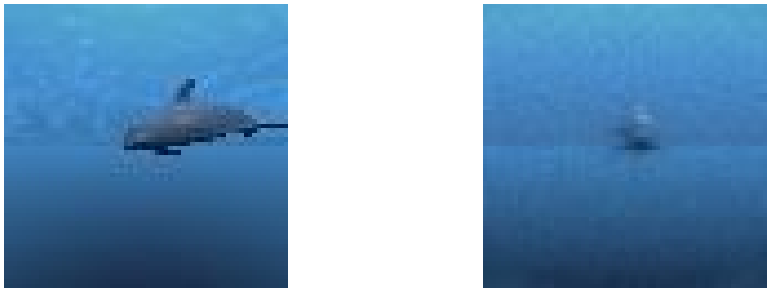

Figure 17. A 64×64 random image sample taken from both a noiseless (**left**) and noise world (**right**).

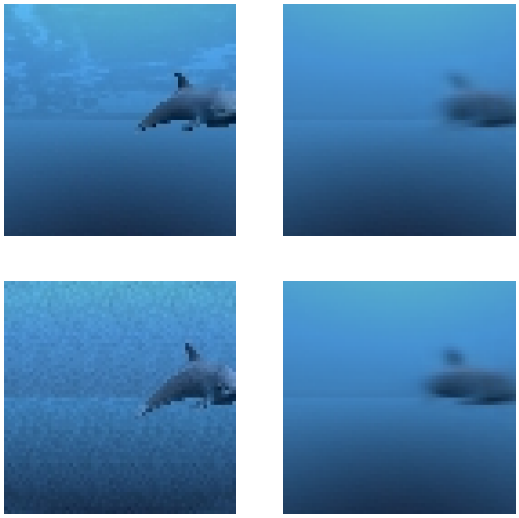

Figure 18. Showcasing the ability of p_{img} above (and q_{img}, below) to reconstruct originally fed images (on the left) from the latent space: the results on the right remark that reconstructed images are fairly similar after either noise and noiseless conditions.

To aid sim-to-real generalisation, the study from [31] introduced distance noise in simulation, proven to emulate such effects in reality successfully. Additionally, real image encodings were demonstrated to have significant similarity with a simulation representation when reconstructed through the image decoder, thus affirming the CMVAE's ability to generalise real-world data.

We perform a similar experiment, investigating if our previously trained encoder CMVAE_{33}^{53} (untrained on noise) can retain the critical information (target position and rotation) without being influenced by noise. First, exploiting the data generation pipeline from Section 4.1, we generate 1×10^3 images for the noiseless environment using stochastic environment randomisation. The same is repeated for the noisy environment containing the effects previously described. Every image is rendered under identical environment conditions by exploiting the same seed, 149. Thus, we generate image pairs from the noiseless and noisy environments. Then, a pass-through of each pair through q_{img} generates

the compressed vectors $z_{noiseless}$ and z_{noisy}. Next, each $z_{noiseless}/z_{noisy}$ is passed through the image decoder p_{img}, generating the image reconstructions across both vectors. Finally, we compare these reconstructed images to evaluate the extent of influence noise has on the CMVAE. Similarly to our image generation step, we generate 1×10^3 noisy images through stochastic simulation as previously discussed (Section 4.1). Then, we compute error metrics across all reconstruction pairs, including both image reconstruction and state prediction.

Table 14 demonstrates that the image reconstructions suffer a 9.91×10^{-1} MAE. While significantly higher than errors previously obtained for noiseless environments (Figure 8), the majority of critical information is still retained. This is visualised by Figure 18, showing an example reconstruction from a noiseless/noisy image pair included in the prior dataset.

Table 14. Error statistics from 1×10^3 image and state reconstructions from noiseless and noisy environments.

Feature	MAE	Standard Error	Max Error
Image	9.91×10^{-1}	1.60×10^{-1}	1.19×10^{2}
r	2.19×10^{-1}	4.80×10^{-2}	5.23×10^{0}
θ	1.04×10^{1}	2.59×10^{-1}	4.08×10^{1}
ϕ	7.23×10^{1}	1.92×10^{0}	2.86×10^{2}

Finally, with the prior experiments showcase CMVAE_{33}^{53}'s generalisation capabilities, we perform a new DRL training run using $\text{SAC}_{\text{SWiMMv2.0}}$, with alternating noiseless/noisy episodes ($\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$), using the same seed generating $\text{SAC}_{\text{SWiMMv2.0}}$ (97). We investigate whether $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$ can learn a suitable behaviour policy with the added complexity of the target objective. For the following experiments, 'Noiseless' denotes environments containing no noise at all (the prior results), while 'Noisy' denotes training run with alteration noisy episodes.

Figure 19 demonstrates that the noisy environment negatively affects the rover by decreasing the maximum reward achievable. In the best case, the noisy environment cannot still obtain a mean episodic reward greater than 7.5×10^2. While the noisy environment is susceptible to reward fluctuations, the noiseless environment reaches its plateau at 7.5×10^5 steps. Still, training reward is better than both PPO and TD3 on noiseless environments (Figure 13).

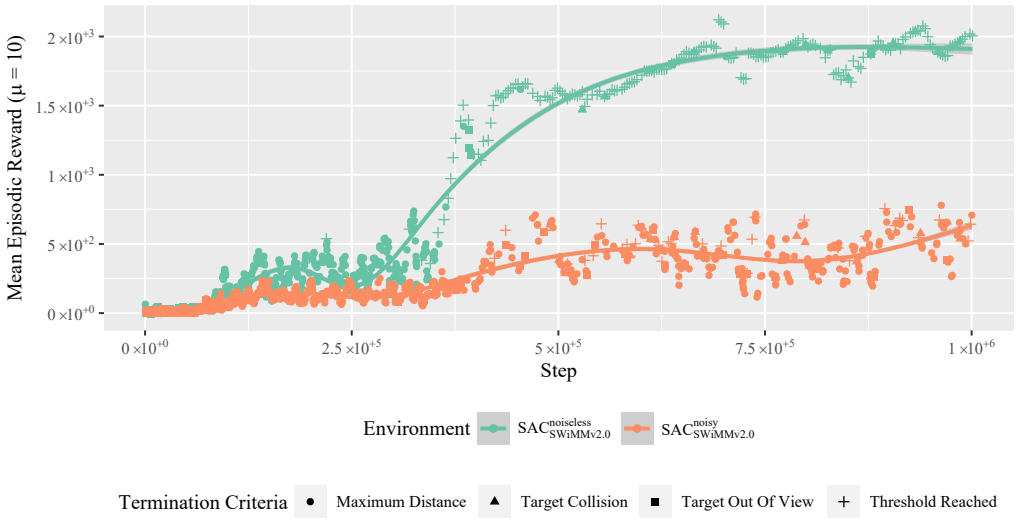

Figure 19. Mean episodic reward using $\text{SAC}_{\text{SWiMMv2.0}}$ across noiseless and noisy environments.

As before, evaluation episodes are injected into training. We investigate how the inference of the best model compares with the previous. Figure 20 shows that $SAC^{noisy}_{SWiMMv2.0}$ obtains the best model (8.07×10^5) at fewer steps than $SAC^{noiseless}_{SWiMMv2.0}$ (8.78×10^5) at the cost of lower mean episodic reward (1.99×10^3 versus 2.40×10^3). This performance far exceeds that of PPO and is comparable with the best TD3, both without noise interference (Figure 14).

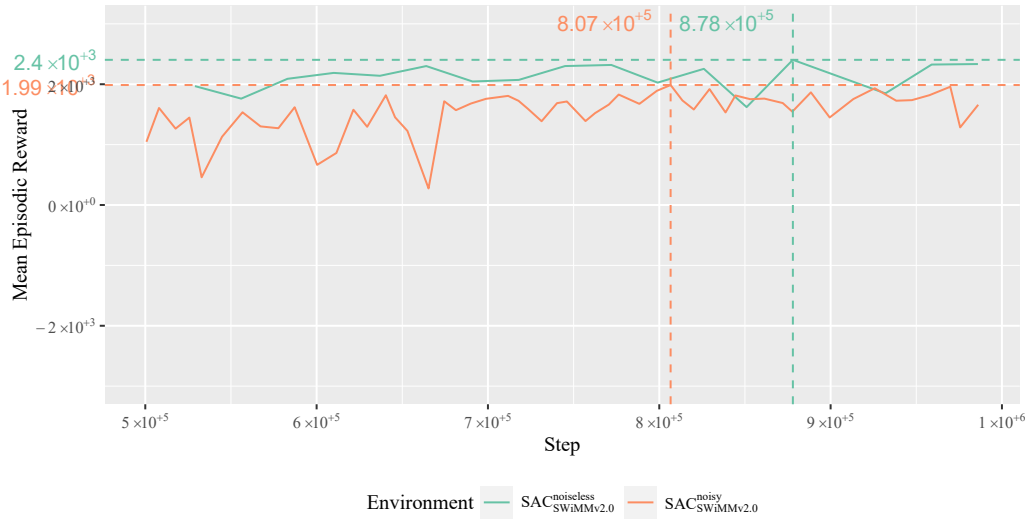

Figure 20. Mean episodic reward using $SAC_{SWiMMv2.0}$ across noiseless and noisy environments during inference.

Next, we compare how the new environment impacts training time. Figure 21 remarks that noise slows down the training, taking 6.49×10^4 s against 5.58×10^4 in a noiseless environment. The dissimilarity between noisy and noiseless episodes confuses the optimisation task, thus resulting in a slower improvement in behaviour policy, therefore requiring more episode resets and optimisation updates. However, this is still a reasonable outcome, as it outperforms both training times for PPO and TD3 on noiseless environments. The best models are found at similar timestamps (4.93×10^4 versus 4.78×10^4).

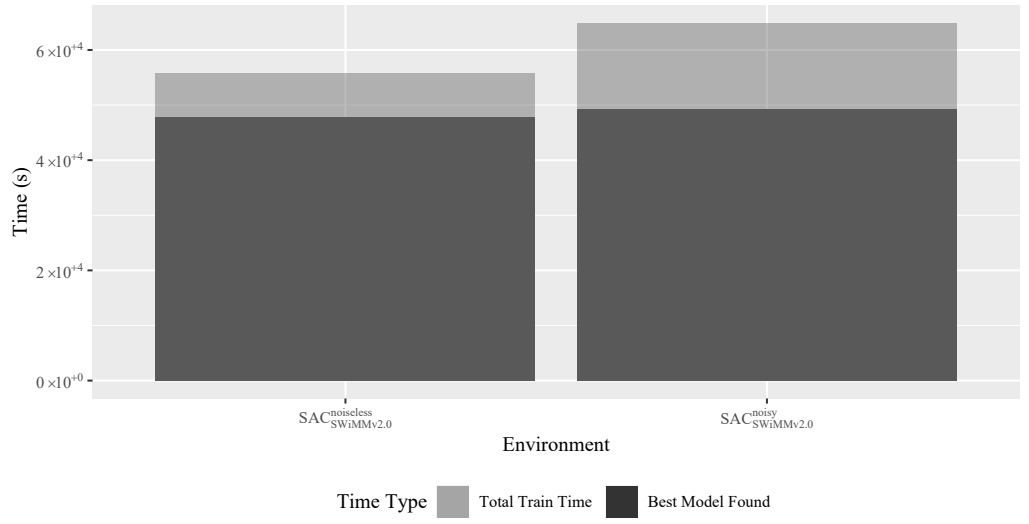

Figure 21. Mean episodic reward using $SAC_{SWiMMv2.0}$ across noiseless and noisy environments.

Finally, we perform our final model metric experiments from Section 4.4.4 to estimate the model fulfilment of our training objectives (Figure 22). Both $\mathcal{D}$ and $\mathcal{A}$ achieve higher

errors for $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$ (3.22×10^{-1} and 6.49×10^{-2}, respectively). $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$ achieves lower azimuth smoothness errors $\mathcal{S}_{\mathcal{A}} = 1.26 \times 10^{-1}$ with comparable distance smoothness errors $\mathcal{S}_{\mathcal{D}} = 9.51 \times 10^{-2}$. Notably, $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$ manages to avoid triggering any safety violations, also achieving $\mathcal{D}'/\mathcal{A}'/\mathcal{C}' = 0$ as with $\text{SAC}^{\text{noiseless}}_{\text{SWiMMv2.0}}$. A video (https://youtu.be/OL-aF9csEPY, accessed on 21 March 2025) showcasing $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$ inference on noiseless and noisy environments is also available.

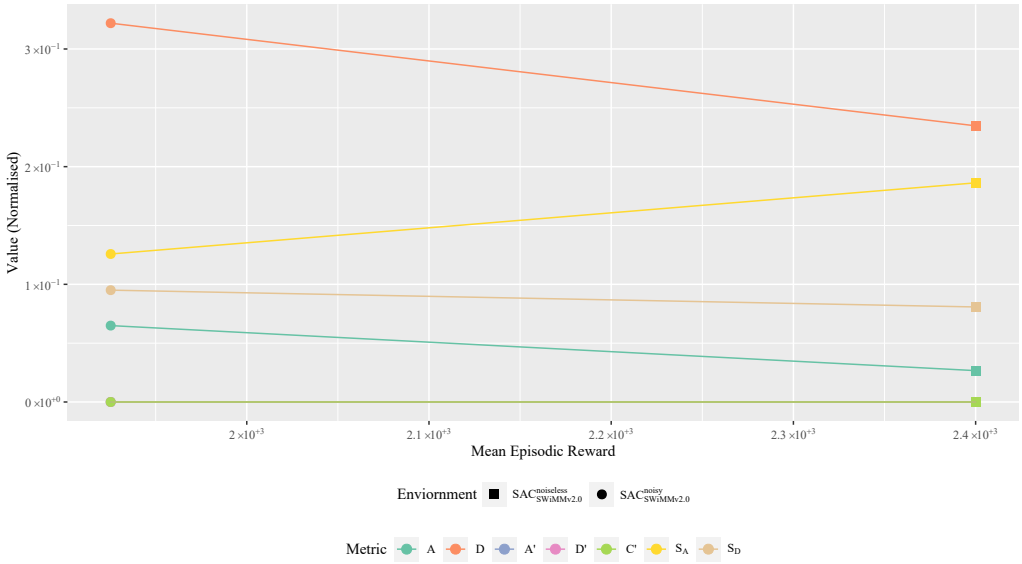

Figure 22. Model metric comparison versus inference rewards across $\text{SAC}^{\text{noiseless}}_{\text{SWiMMv2.0}}$ and $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$. $\mathcal{D}'/\mathcal{A}'/\mathcal{C}' = 0$ for both $\text{SAC}^{\text{noiseless}}_{\text{SWiMMv2.0}}$ and $\text{SAC}^{\text{noisy}}_{\text{SWiMMv2.0}}$.

These experiments showcase CMVAE^{53}_{33}'s robustness to the environmental noise of interest while maintaining meaningful encodings. By this, SAC has also learned an aptly successful behaviour policy with a comparable training time to noiseless environments. Most importantly, the lack of violations of safety conditions validates the model's robustness to dynamically changing environments. An additional experiment also tested the ability of the DRL algorithm to generalise under conditions not witnessed during training time. These are provided in our discussion section, Section 5.4.1).

5. Discussion

To conclude, $\text{SAC}_{\text{SWiMMv2.0}}$ is the superior model concerning our metrics. No safety conditions were breached, avoiding any 'dangerous' behaviour and all other errors were minimal. $\text{SAC}_{\text{SWiMMv2.0}}$, versus $\text{SAC}_{\text{SWiMMv1.0}}$, displays a significant reduction in $\mathcal{A}$ of over 1 magnitude and a significant improvement in $\mathcal{D}$. Interestingly, TD3 is the only algorithm that displayed the ability to *recover* from a *compromising* state; a state which differs greatly from those exhibited during training. This demonstrates a successful DRL generalisation that can still provide meaningful behaviour. TD3, while displaying high accuracy, displays volatile control. $\text{SAC}_{\text{SWiMMv2.0}}$ combines high accuracy with smooth control, capable of good behaviour while minimising action jitter. A clear negative correlation couples high reward with low metric errors, thus verifying a well-defined reward function.

5.1. CMVAE/DRL Training Decoupling

Differently from related work [11], which conjoin the training of a feature extractor and a DRL network, we propose their decoupling. Firstly, this streamlines the DRL training (no longer requiring a feature extractor) and second, enables the performance monitoring of each training phase in isolation. Our results demonstrate that such a decoupling allows

for an optimised CMVAE training process unhindered by running concurrently with a game engine.

5.2. Headless Mode Optimisations

Unity has the added benefit of being able to be run in batch mode. In this configuration, no graphics are rendered on the running system, but game logic is executed nonetheless. The removal of graphical rendering enables a much higher update rate. Normally, such a setting can be enabled when running an application as, for example, a headless server. However, we investigate the possibility of exploiting such a mode to provide further speed-up to training.

However, given that our application relies heavily on camera rendering, running in a headless mode threatens the reliance on graphical outputs. This was evidenced by our preliminary experiments, which achieved a 30–40% speedup from an identical run with graphical rendering, where the training process could not be optimised.

5.3. Seeding and Reproducibility

For our model inference (Section 4.4.5) experiments, 2 seeds are used, as Stable Baselines exploits the original model seed also to initialise the random number generators for PyTorch and Gymnasium operations. To provide a more robust exploration of the search/state spaces, a second seed initialises all other random number generators (also exploited in Unity). For example, a Stable Baselines model generated using seed 1 will, on loading warm-up, *also* initialise PyTorch/Gymnasium random number generator with 1, but the rest of SWiMM$_{2.0}$'s pipeline can also exploit another. For these experiments, the seeds displayed indicate the latter.

5.4. Limitations

5.4.1. DRL Robustness

Given that a sim-to-real pipeline's ability to generalise in reality is critical for the success of a learned target policy, we aim to investigate further SWiMM$_{2.0}$'s ability to adapt in unforeseen circumstances **without retraining** of both the CMVAE pre-processing pipeline and the latter DRL. Our previous generalisation experiment (Section 4.4.6) demonstrated CMVAE's [31] robustness to our ambiental noise of interest, as this provided accurate image reconstruction and could be used to re-train a new DRL network.

First, we perform inference directly on $\text{SAC}^{\text{noiseless}}_{\text{SWiMMv2.0}}$ in a **noisy** environment. The resulting behaviour shows exceptional behaviour (comparable with Section 4.4.5) in noiseless environments but poor behaviour otherwise, frequently violating safety conditions and achieving low reward values. Our current pipeline fails to generate a SAC model robust to unforeseen noisy environments.

This result was surprising, given that our prior experiments proved the CMVAE architecture was robust to noise, whereby the image reconstructions appear to 'denoise' the noisy images, resulting in highly similar outputs against a noiseless environment. This result motivated the investigation of the output z from CMVAE^{53}_{33} under such conditions, given that the compressed latent space is the solitary feature provided to the DRL network.

For the 1×10^3 samples exploited in Section 4.4.6, we also retain the encoded latent vectors z for both noiseless and noisy compressions, $z_{noiseless}, z_{noisy} \in \mathbb{R}^{1000\times 10}$. Then, for each feature $z_i \in z$, we calculate the MAE, SE and ME across also samples.

Table 15 displays our findings. The largest errors z_0, z_1, z_2 indicate that the cross-modal features suffer the worst from noise. While these values appear small, each feature in z is normalised $z \in [-1, 1]$, thus even small variations per feature result in a significant change in state. Thus, even minor discrepancies between $z_{noiseless}$ and z_{noisy} yield extremely different input feature vectors for the DRL algorithm. Then, for noisy episodes, z_{noisy}

represents an unexplored state space the optimisation objective has not witnessed, thus explaining the poor behaviour policy.

Table 15. Similarity errors for each feature of z across 1×10^3 samples taken from noiseless and noisy environments.

$z_i \in z$	MAE	SE	ME
0	7.75×10^{-3}	1.59×10^{-4}	2.04×10^{-2}
1	2.32×10^{-3}	6.09×10^{-5}	9.87×10^{-3}
2	8.34×10^{-3}	2.25×10^{-4}	4.10×10^{-2}
3	1.02×10^{-3}	2.38×10^{-5}	4.70×10^{-3}
4	1.11×10^{-3}	2.77×10^{-5}	5.31×10^{-3}
5	9.00×10^{-4}	2.15×10^{-5}	3.70×10^{-3}
6	1.28×10^{-3}	3.21×10^{-5}	8.53×10^{-3}
7	8.21×10^{-4}	2.10×10^{-5}	3.83×10^{-3}
8	1.24×10^{-3}	2.58×10^{-5}	4.93×10^{-3}
9	1.53×10^{-3}	3.53×10^{-5}	6.90×10^{-3}

5.4.2. Noise Type Volatility

During the investigation of noise inclusion in the simulated world, we found great DRL performance variation depending on the types of the former. Concretely, both the noiseless and noisy worlds contain fog, with the latter under a different and much more exaggerated function. Previously, the simulated environment existed as an infinitely spanning space, whereby the majority of the image renders concerns either the underside of the water's surface plane or the skybox.

One mechanism we exploited to exaggerate the fog was including quad surfaces attached to the camera rendered into the far plane. These quads, although untextured, provide a mesh for post-processing fog effects to be rendered into, thus enabling a much stronger influence on rendered pixels.

Figure 23 displays an example image render from an environment using the former effect in isolation. We found that even a SAC algorithm failed to achieve any reasonable optimisation when using the former. To determine the causality, we generated 1×10^3 images (seed = 151) using **only** fog quad effects. Then, we investigated the image reconstructions from CMVAE_{33}^{53} from the newly generated dataset.

Figure 24 shows 8 image renders and their reconstructed counterparts. The reconstructed images yield great dissimilarities with the original renders, thus indicating that the CMVAE is unable to generalise this environment. In such an environment, while the shape and size of the target are maintained, the **colour** is significantly affected. Then, given that almost all pixels in the image renders are more heavily influenced by this effect than others. The CMVAE is unable to extract any meaningful features at all, thus providing a meaningless encoding as input to the DRL optimisation objective.

Figure 23. An image render taken from an environment using fog and fog quad renders only.

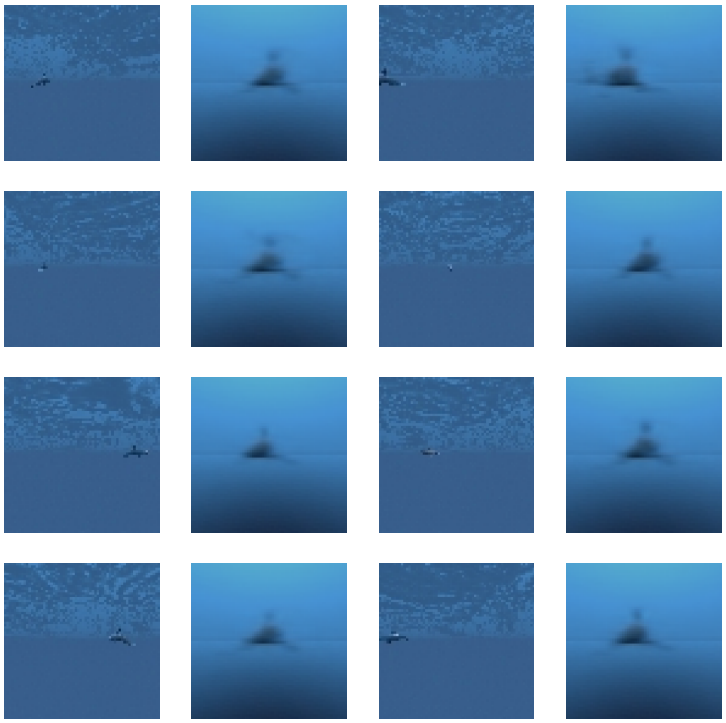

Figure 24. 8 image renders are taken from training episodes in a simulation using distance quad renders. The resulting reconstructions from CMVAE_{33}^{53} are shown in the right column.

5.5. Additional Sensors

Although image data provides a rich format for which a successful behaviour policy can be learned, the inclusion of additional sensors, including sonar, could enhance the state space and benefit the optimisation objective. This is particularly true when some sensors become unreliable, such as the target escaping the camera's field of view.

However, while such sensors may also be emulated in simulation (including sonar [13]), many are a disturbance to wildlife, especially the former, given that many species of cetacean rely on sonar as their primary communication mechanism. Thus, we aim to investigate a lightweight approach that has the added benefit of minimising environmental disturbance.

6. Conclusions and Future Work

SWiMM$_{2.0}$ couples a commercial game engine with DRL to optimise control policy by emulating joystick commands to BLUEROV. Our trained CMVAE can successfully reconstruct images and achieve orders of magnitudes of dimensionality reduction, heavily optimising DRL training. Our experiments demonstrate that SAC achieves exceptional model behaviour concerning our newly defined custom metrics, able to consistently and accurately maintain the target within a strict azimuth range while allowing suitable freedom of movement. We also couple strict control optimisation while minimising jitter, a criterion not previously considered. By transferring our learned agents onto the physical device(s); we are confident that the trained networks can generalise sufficiently for sim-to-real transfer, enabling active target tracking of marine megafauna.

The next step for our project is the transfer of the best DRL models to BLUEROV's onboard hardware. Firstly, has our CMVAE achieved sufficient generalisation such that it can interpret the new images obtained from the real environment and encode them in a meaningful way? Then, can the DRL network generalise these new encodings? Last, can these action commands provide behaviour similar to simulation? A pass-through of the encoder/DRL network(s) is expected to be demanding on the more limited hardware. However, model post-processing has proven drastically to reduce the sizes of such networks while still maintaining their accuracy [43].

The existing environment consists of a single target object. Given that cetaceans often exist in pods (groups of individuals), we shall investigate how to track single/multiple targets containing any number of distractor objects in addition to the noise already experimented with. This would likely require modifications to the current feature space and thus extending the CMVAE.

Previous literature [38,44] exploits frame stacking for velocity inference in real-time simulations. Though our preliminary experiments found this detrimental to SAC behaviour, we shall investigate any possible optimisations and fine-tuning.

We will extend autonomous control to three dimensions to further facilitate curriculum learning, where this is constrained by our attempt to achieve realism to emulate impulses replacing original joystick inputs directly. Instead, we can actuate thrusters directly at the cost of a more complex action space search. Yet, the former can still be modelled in a commercial game engine.

Our simulation defaults the rover's diving mode to $mode_{depth}$ (Section '*Action Space*'). Further motivated by the desire for 3-dimensional control, being able to toggle such a mode via the behaviour policy could further enhance the control response. For the former, we aim to provide an additional control command m to toggle $mode_{depth}$. Thus, supporting both 3-dimension action space with diving modes extends the action space to $\{ x, y, z, y', m \}$, with a behaviour policy optimised for x, y, y', m.

We previously (Section 4.4.6) demonstrated the ability of the CMVAE architecture from [31] to generalise the environment and be robust to noise. Also, the limitations of the current pipeline were identified when (1) performing inference on a noisy world not previously witnessed by a DRL algorithm (Section 5.4.1) and (2) volatility between visual effects and CMVAE encodings. To address both of these generalisation objectives, an extra preprocessing step, including both object detection and denoising algorithms prior to image encoding, would strengthen the pipeline's ability to deal with noise including distractor objects and visual effects.

Finally, due to time, we could not thoroughly investigate hyperparameter tuning. One notable result from our experiments is the extremely poor behaviour of PPO. While a single-layered DNN may not provide the correct approximation, closing experiments proved that even an identical network to SAC/TD3 did not aid target policy optimisation.

Still, PPO is the industry-standard DRL algorithm. A more thorough hyperparameter search may discover a greater array of behaviours.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/ai6040071/s1. , Our supplementary works detail the preliminary experiments for our CMVAE, inspiring the training environment demonstrated in this paper. Additionally, all hyperamaterers are provided for the main algorithms exploited in our experiments.

Author Contributions: Conceptualisation, S.A. and G.B.; methodology, G.B.; software, S.A.; validation, S.A. and G.B.; formal analysis, S.A.; investigation, S.A.; resources, G.B. and G.U.; data curation, S.A.; writing—original draft preparation, S.A. and G.B.; writing—review and editing, S.A and G.B.; visualisation, S.A. and G.B.; supervision, G.B. and G.U.; project administration, G.B. and G.U.; funding acquisition, G.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Newcastle University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset associated with the presented experiments was available online the 21 March 2025: https://doi.org/10.17605/OSF.IO/7KS2C.

Acknowledgments: We acknowledge Stephen McGough and Kirsten Crane for providing their expertise during the start of the project.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AUV	autonomous underwater vehicle
BLUEROV	BlueROV2
CMVAE	cross-modal variational autoencoder
CNN	convolutional neural network
DNN	deep neural network
DRL	deep reinforcement learning
FPS	frames per second
L2D	learning to drive
MAE	mean absolute error
ME	max error
ML	machine learning
PPO	proximal policy optimisation
RL	reinforcement learning
ROV	remote-operated vehicle
RTF	real-time factor
SAC	soft actor critic
SE	standard error
SWIM	SWiMM DEEPeR
TD3	twin delayed deep deterministic policy gradients
UUV	unmanned underwater vehicle
UVMS	underwater vehicle manipulator system
VAE	variational autoencoder
VARG	video action recongition networks

Nomenclature

Notation	Description	Unit
$\text{SWiMM}_{2.0}$	SWIM v2.0	
$\text{SWiMM}_{1.0}$	SWIM v1.0	
z	latent space vector	$\rightarrow$
s	state	
r	reward	$\mathbb{R}$
$\mathcal{F}$	reward function	
π	target policy	
d	distance	$\mathbb{R}_{\geq 0}$
$\mathcal{T}$	transform	$\mathcal{T}$
q_{img}	image encoder	
p_{img}	image decoder	
p_s	state decoder	
θ	azimuth	$\mathbb{R}$
ψ	target rotation	$\mathbb{R}$
o	observation	$\rightarrow$
a	action	$\rightarrow$
π_b	behaviour policy	
ϵ	exploration probability	$\mathbb{R}_{\geq 0}$
$ActPrev$	list of the previous 10 actions	$[\vec{a_0}, \ldots, \vec{a_{n-1}}]$
W	image width	$\mathbb{N}$
H	image height	$\mathbb{N}$
n_z	dimension of z	$\mathbb{N}$
$mode$	BLUEROV dive mode	$\{ mode_{man}, mode_{stab}, mode_{depth} \}$
opt_d	optimum distance	$\mathbb{R}_{\geq 0}$
max_d	maximum distance	$\mathbb{R}_{\geq 0}$
pen_d	distance penalty	$\mathbb{R}_{\geq 0}$
pen_θ	azimuth penalty	$\mathbb{R}_{\geq 0}$
cam_{hfov}	camera horizontal field-of-view	$\mathbb{R}_{\geq 0}$
cam_w	camera sensor width	$\mathbb{R}_{\geq 0}$
f	focal length	$\mathbb{R}_{\geq 0}$
α	half the camera horizontal field-of-view	$\mathbb{R}_{\geq 0}$
$\triangle$	Unity-based physics	
$\star$	custom physics	
$\mathcal{M}_{ts}^{seed}$	a model saved with a seed, *seed*, saved at a timestamp *ts*	
$IMAGE_{\text{TRAIN}}$	cmvae training set	$\{ img_1, \ldots, img_{2.7\times10^5} \}$
$IMAGE_{\text{TEST}}$	cmvae test set	$\{ img_1, \ldots, img_{3\times10^4} \}$
$IMAGE_{\text{INTERPOLATE}}$	image interpolation set	$\{ img_1, \ldots, img_6 \}$
$IMAGE_{\text{SIMILARITY}}$	image validation set	$\{ img_1, \ldots, img_{1\times10^3} \}$
C	number of colour channels	$\mathbb{N}$
p_w^h	pixel at a width w and height h	(r, g, b)
ω	retain the previous ω episodes for early stopping	$\mathbb{N}$
CMVAE_{33}^{53}	best trained CMVAE	
$\text{SAC}_{\text{SWiMMv1.0}}$	previous best SAC model	
$\text{SAC}_{\text{SWiMMv2.0}}$	current best SAC model	
$stats\_window\_size$	retain the previous $stats\_window\_size$ episodes for monitoring	$\mathbb{N}$
net_arch	network architecture	$[hl_0, \ldots, hl_{n-1}] \quad \mid \quad hl_i \in \mathbb{N} \quad \forall i \in \{0, \ldots, n-1\}$
$\mathcal{D}$	mean distance error	$\mathbb{R}_{\geq 0}$
$\mathcal{A}$	mean azimuth error	$\mathbb{R}_{\geq 0}$
$\mathcal{C}'$	target collision count	$\mathbb{R}_{\geq 0}$
$\mathcal{D}'$	distance threshold exceeded count	$\mathbb{R}_{\geq 0}$
$\mathcal{A}'$	target visibility count	$\mathbb{R}_{\geq 0}$
$\mathcal{S}_\mathcal{D}$	surge smoothness error	$\mathbb{R}_{\geq 0}$
$\mathcal{S}_\mathcal{A}$	yaw smoothness error	$\mathbb{R}_{\geq 0}$
$\text{SAC}_{\text{SWiMMv2.0}}^{\text{noisy}}$	$\text{SAC}_{\text{SWiMMv2.0}}$ model trained on a noisy environment	
$\text{SAC}_{\text{SWiMMv2.0}}^{\text{noiseless}}$	$\text{SAC}_{\text{SWiMMv2.0}}$ model trained on a noisy environment	

References

1. Wells, R.; Early, G.; Gannon, J.; Lingenfelser, R.; Sweeney, P. *Tagging and Tracking of Rough-Toothed Dolphins (Steno bredanensis) from the March 2005 Mass Stranding in the Florida Keys*; NOAA Technical Memorandum; National Oceanic and Atmospheric Administration (NOAA): Washington, DC, USA, 2008. Available online: https://repository.library.noaa.gov/view/noaa/8656 (accessed on 21 March 2025).
2. Sequeira, A.M.M.; Heupel, M.R.; Lea, M.A.; Eguíluz, V.M.; Duarte, C.M.; Meekan, M.G.; Thums, M.; Calich, H.J.; Carmichael, R.H.; Costa, D.P.; et al. The importance of sample size in marine megafauna tagging studies. *Ecol. Appl.* **2019**, *29*, e01947. [CrossRef] [PubMed]
3. Inzartsev, A.V. *Underwater Vehicles*; IntechOpen: Rijeka, Croatia, 2009.
4. Wynn, R.B.; Huvenne, V.A.; Le Bas, T.P.; Murton, B.J.; Connelly, D.P.; Bett, B.J.; Ruhl, H.A.; Morris, K.J.; Peakall, J.; Parsons, D.R.; et al. Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience. *Mar. Geol.* **2014**, *352*, 451–468. [CrossRef]
5. Liu, Y.; Anderlini, E.; Wang, S.; Ma, S.; Ding, Z. Ocean Explorations Using Autonomy: Technologies, Strategies and Applications. In *Offshore Robotics*; Springer: Singapore, 2022; pp. 35–58.
6. von Benzon, M.; Sørensen, F.F.; Uth, E.; Jouffroy, J.; Liniger, J.; Pedersen, S. An Open-Source Benchmark Simulator: Control of a BlueROV2 Underwater Robot. *J. Mar. Sci. Eng.* **2022**, *10*, 1898. [CrossRef]
7. Yang, X.; Xing, Y. Tuning for robust and optimal dynamic positioning control in BlueROV2. *IOP Conf. Ser. Mater. Sci. Eng.* **2021**, *1201*, 012015. [CrossRef]
8. Walker, K.; Gabl, R.; Aracri, S.; Cao, Y.; Stokes, A.; Kiprakis, A.; Giorgio Serchi, F. Experimental Validation of Wave Induced Disturbances for Predictive Station Keeping of a Remotely Operated Vehicle. *IEEE Robot. Autom. Lett.* **2021**, *6*, 5421–5428. [CrossRef]
9. Skaldebø, M.; Schjølberg, I.; Haugaløkken, B.O. Underwater Vehicle Manipulator System (UVMS) with BlueROV2 and SeaArm-2 Manipulator. In *International Conference on Offshore Mechanics and Arctic Engineering*; American Society of Mechanical Engineers: New York, NY, USA, 2022; Volume 85901, p. V05BT06A022. [CrossRef]
10. Katara, P.; Khanna, M.; Nagar, H.; Panaiyappan, A. Open Source Simulator for Unmanned Underwater Vehicles using ROS and Unity3D. In Proceedings of the 2019 IEEE Underwater Technology, Kaohsiung, Taiwan, 16–19 April 2019.
11. Viitala, A.; Boney, R.; Zhao, Y.; Ilin, A.; Kannala, J. Learning to Drive (L2D) as a Low-Cost Benchmark for Real-World Reinforcement Learning. In Proceedings of the 2021 20th International Conference on Advanced Robotics (ICAR), Ljubljana, Slovenia, 6–10 December 2021; pp. 275–281. [CrossRef]
12. Huang, Z.; Buchholz, M.; Grimaldi, M.; Yu, H.; Carlucho, I.; Petillot, Y.R. URoBench: Comparative Analyses of Underwater Robotics Simulators from Reinforcement Learning Perspective. In Proceedings of the OCEANS 2024-Singapore, Singapore, 15–18 April 2024; pp. 1–8. [CrossRef]
13. Potokar, E.; Ashford, S.; Kaess, M.; Mangelson, J. HoloOcean: An Underwater Robotics Simulator. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022.
14. Zhang, M.; Choi, W.S.; Herman, J.; Davis, D.; Vogt, C.; Mccarrin, M.; Vijay, Y.; Dutia, D.; Lew, W.; Peters, S.; et al. DAVE Aquatic Virtual Environment: Toward a General Underwater Robotics Simulator. In Proceedings of the 2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV), Singapore, 19–21 September 2022; pp. 1–8. [CrossRef]
15. Cieślak, P. Stonefish: An Advanced Open-Source Simulation Tool Designed for Marine Robotics, With a ROS Interface. In Proceedings of the OCEANS 2019-Marseille, Marseille, France, 17–20 June 2019; pp. 1–6. [CrossRef]
16. Appleby, S.; Crane, K.; Bergami, G.; McGough, A.S. SWiMM DEEPeR: A Simulated Underwater Environment for Tracking Marine Mammals Using Deep Reinforcement Learning and BlueROV2. In Proceedings of the 2023 IEEE Conference on Games (CoG), Boston, MA, USA, 21–24 August 2023; pp. 1–8. [CrossRef]
17. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
18. Fujimoto, S.; van Hoof, H.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the International Conference on Machine Learning, Macau, China, 26–28 February 2018.
19. Kaufmann, E.; Bauersfeld, L.; Loquercio, A.; Müller, M.; Koltun, V.; Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature* **2023**, *620*, 982–987. [CrossRef]
20. Rolland, R.M.; Parks, S.E.; Hunt, K.E.; Castellote, M.; Corkeron, P.J.; Nowacek, D.P.; Wasser, S.K.; Kraus, S.D. Evidence that ship noise increases stress in right whales. *Proc. R. Soc. B Biol. Sci.* **2012**, *279*, 2363–2368. [CrossRef]
21. Gregory, J. *Game Engine Architecture*, 3rd ed.; A K Peters/CRC Press: Boca Raton, FL, USA, 2018.
22. Juliani, A.; Berges, V.P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; et al. Unity: A general platform for intelligent agents. *arXiv* **2020**, arXiv:1809.02627.
23. Zhao, W.; Queralta, J.P.; Westerlund, T. Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In Proceedings of the 2020 IEEE symposium series on computational intelligence, Canberra, Australia, 1–4 December 2020; pp. 737–744.

24. Maglietta, R.; Fanizza, C.; Cherubini, C.; Bellomo, S.; Carlucci, R.; Dimauro, G. Risso's Dolphin Dataset. *IEEE Dataport* **2023**. [CrossRef]
25. Maglietta, R.; Bussola, A.; Carlucci, R.; Fanizza, C.; Dimauro, G. ARIANNA: A novel deep learning-based system for fin contours analysis in individual recognition of dolphins. *Intell. Syst. Appl.* **2023**, *18*, 200207. [CrossRef]
26. Zhivomirov, H.; Nedelchev, I.; Dimitrov, G. Dolphins Underwater Sounds Database. *TEM J.* **2020**, *9*, 1426. [CrossRef]
27. Frasier, K.E. A machine learning pipeline for classification of cetacean echolocation clicks in large underwater acoustic datasets. *PLoS Comput. Biol.* **2021**, *17*, e1009613. [CrossRef] [PubMed]
28. Trotter, C.; Atkinson, G.; Sharpe, M.; Richardson, K.; McGough, A.S.; Wright, N.; Burville, B.; Berggren, P. NDD20: A large-scale few-shot dolphin dataset for coarse and fine-grained categorisation. *arXiv* **2020**, arXiv:2005.13359.
29. Kingma, D.P.; Welling, M. An introduction to variational autoencoders. *arXiv* **2019**, arXiv:1906.02691.
30. Spurr, A.; Song, J.; Park, S.; Hilliges, O. Cross-Modal Deep Variational Hand Pose Estimation. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 15–20 June 2018; pp. 89–98. [CrossRef]
31. Bonatti, R.; Madaan, R.; Vineet, V.; Scherer, S.; Kapoor, A. Learning visuomotor policies for aerial navigation using cross-modal representations. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 1637–1644.
32. Loquercio, A.; Maqueda, A.I.; Del-Blanco, C.R.; Scaramuzza, D. Dronet: Learning to fly by driving. *IEEE Robot. Autom. Lett.* **2018**, *3*, 1088–1095. [CrossRef]
33. Li, J.; Wang, B.; Ma, H.; Gao, L.; Fu, H. Visual Feature Extraction and Tracking Method Based on Corner Flow Detection. *IECE Trans. Intell. Syst.* **2024**, *1*, 3–9. [CrossRef]
34. Wang, F.; Yi, S. Spatio-temporal Feature Soft Correlation Concatenation Aggregation Structure for Video Action Recognition Networks. *IECE Trans. Sens. Commun. Control* **2024**, *1*, 60–71. [CrossRef]
35. Sidorenko, G.; Thunberg, J.; Vinel, A. Cooperation for Ethical Autonomous Driving. In Proceedings of the 2024 20th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Paris, France, 21–23 October 2024; pp. 391–395. [CrossRef]
36. Abro, G.E.M.; Ali, Z.A.; Rajput, S. Innovations in 3D Object Detection: A Comprehensive Review of Methods, Sensor Fusion, and Future Directions. *IECE Trans. Sens. Commun. Control* **2024**, *1*, 3–29. [CrossRef]
37. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv* **2018**, arXiv:1801.01290.
38. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
39. Anzalone, L.; Barra, P.; Barra, S.; Castiglione, A.; Nappi, M. An End-to-End Curriculum Learning Approach for Autonomous Driving Scenarios. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 19817–19826. [CrossRef]
40. Lapan, M. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods to Practical Problems of Chatbots, Robotics, Discrete Optimization, Web Automation, and More*; Packt Publishing: Birmingham, UK, 2020.
41. Hays, G.C.; Ferreira, L.C.; Sequeira, A.M.; Meekan, M.G.; Duarte, C.M.; Bailey, H.; Bailleul, F.; Bowen, W.D.; Caley, M.J.; Costa, D.P.; et al. Key Questions in Marine Megafauna Movement Ecology. *Trends Ecol. Evol.* **2016**, *31*, 463–475. [CrossRef] [PubMed]
42. Yao, Y.; Rosasco, L.; Caponnetto, A. On Early Stopping in Gradient Descent Learning. *Constr. Approx.* **2007**, *26*, 289–315. [CrossRef]
43. Yao, S.; Zhao, Y.; Shao, H.; Liu, S.; Liu, D.; Su, L.; Abdelzaher, T. FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. In Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, SenSys '18, New York, NY, USA, 4–7 November 2018; pp. 278–291. [CrossRef]
44. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602