2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Scaling Multi-Agent Reinforcement Learning for Underwater Acoustic Tracking via Autonomous Vehicles

Matteo Gallici, Ivan Masmitja, and Mario Martín

***Abstract*— Autonomous vehicles (AVs) offer a cost-effective solution for scientific missions such as underwater tracking. Reinforcement learning (RL) has emerged as a powerful method for controlling AVs, but scaling to fleets (essential for multi-target tracking or rapidly moving targets) is challenging. Multi-Agent RL (MARL) is notoriously sample-inefficient, and while high-fidelity simulators like Gazebo's LRAUV provide up to 100× faster-than-real-time single-robot simulations, they offer little speedup in multi-vehicle scenarios, making MARL training impractical. Yet, high-fidelity simulation is crucial to test complex policies and close the sim-to-real gap. To address these limitations, we develop a GPU-accelerated environment that achieves up to 30,000× speedup over Gazebo while preserving its dynamics. This enables fast, end-to-end GPU training and seamless transfer to Gazebo for evaluation. We also introduce a Transformer-based architecture (*TransfMAPPO*) that learns policies invariant to fleet size and number of targets, enabling curriculum learning to train larger fleets on increasingly complex scenarios. After large-scale GPU training, we perform extensive evaluations in Gazebo, showing our method maintains tracking errors below 5m even with multiple fast-moving targets.**

I. INTRODUCTION

Underwater tracking (UT) is essential to advance marine research missions, such as tracking marine species and oceanographic phenomena [1, 2], monitoring large-scale data collection using autonomous underwater vehicles (AUV) [3], and managing marine protected areas [4]. However, underwater communication systems face challenges such as low reliability and bandwidth, as GPS is ineffective [5]. Acoustic tracking using AUVs or autonomous surface vehicles (ASVs) offers a promising solution, improving efficiency and reducing costs compared to traditional fixed equipment [2, 3]. Yet, challenges like unreliable communication, complex environment dynamics, and energy constraints necessitate advanced motion planning to enhance AVs' tracking capabilities.

Reinforcement learning (RL) enables AVs to learn optimal navigation strategies through trial and error, providing dynamic responses to environmental conditions compared to pre-programmed approaches [6]. Preliminary work by [7] demonstrated the potential of using RL to train an ASV policy for tracking an underwater target using range-only acoustic data. More recent studies [8, 9] have shown that Multi-Agent Reinforcement Learning (MARL) can enable

M. G. is with the KEMLG Research Group, Universitat Politècnica de Catalunya Barcelona, Spain. gallici@cs.upc.edu

I. M. is with the Instituto de Ciencias del Mar, Consejo Superior de Investigaciones Científicas, Barcelona, Spain. masmitja@icm.csic.es

M. M. is with the KEMLG Research Group, Universitat Politècnica de Catalunya (UPC), and with the HPAI group at Barcelona Supercomputing Center (BSC), Barcelona, Spain. mmartin@cs.upc.edu

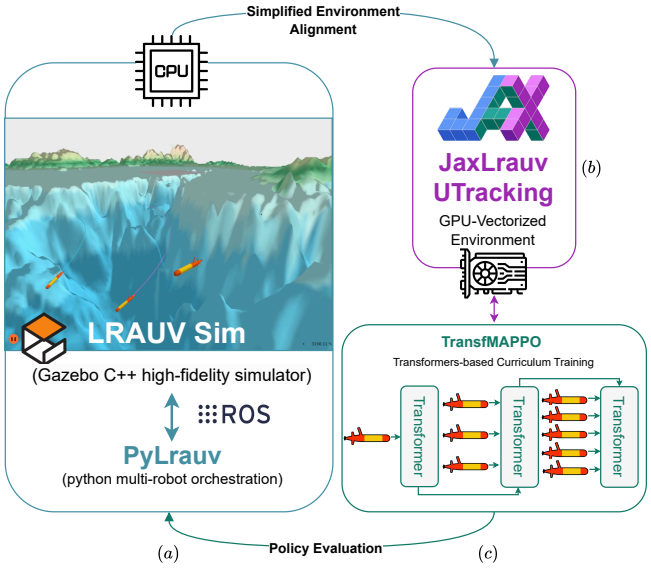


Fig. 1: Overview of our training and evaluation pipeline. **PyLrauv** (*a*) is a new Python package to control multiple robots in the C++ high-fidelity LRAUV simulator. **JaxLrauv** (*b*) is a GPU-accelerated, simplified environment that supports massive parallelization while preserving the dynamics of the LRAUV simulator. **TransfMAPPO** (*c*) employs transformers to train progressively larger fleets of vehicles to coordinate via curriculum learning. The final policies trained in JaxLrauv with TransfMAPPO are then evaluated in the realistic LRAUV simulator prior to real-world deployment.

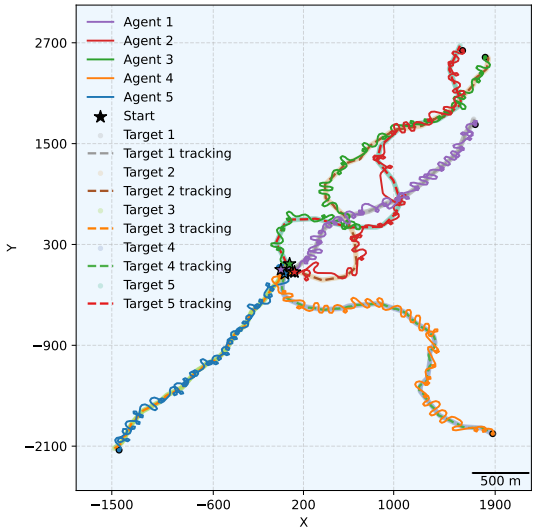


Fig. 2: Five agents trained in the GPU simplified environment follow in Gazebo simulator five fast targets over several kms. Video: https://mttga.github.io/posts/pylrauv/images/5v5.gif

cooperative tracking by sharing data collected simultaneously from different locations. However, these MARL studies were conducted in very simple, abstract environments. Deploying such a complex system in the sea requires extensive preliminary testing. Simulators such as Gazebo's Long-Range AUV simulator (LRAUV Sim) [10], which models hydrodynamics, acoustic communication, and marine sensors of real vehicles, are crucial tools for this phase. Unfortunately, LRAUV Sim provides only about a 10× faster-than-real-time speedup in multi-robot simulations, and its C++-based control complicates integration with RL methods, making direct application of MARL impractical in this simulator.

To advance multi-agent systems for underwater tracking, we therefore make the following contributions:

- **Fast GPU Training and High-Fidelity Testing Pipeline**: We propose a new pipeline for scaling MARL in underwater tracking scenarios, composed of two interconnected frameworks (see Figure 1). The first, *PyLrauv*, is an open-source Python package built on ROS2 [11] that provides a Gym-like interface for controlling LRAUV Sim. PyLrauv enables seamless control and observation of a variable number of agents and targets in the Gazebo high-fidelity backend, and also supports direct deployment on real vehicles. The second, *JaxLrauv*, is a simplified environment implemented on top of JaxMARL [12], a popular GPU-accelerated MARL framework. JaxLrauv achieves up to a 30,000× speedup over Gazebo, enabling training of effective policies in minutes, while preserving full compatibility and transferability with Gazebo and real vehicles through direct equivalence with the PyLrauv controller.
- **Agent-Target Invariant Policies with Transformers and Curriculum Learning**: By integrating Transformers into a novel variant of MAPPO [13] (*TransfMAPPO*), we train policies that are invariant to the number of agents and targets. This enables a curriculum learning (CL) pipeline that progressively trains larger fleets in increasingly complex multi-target scenarios. TransfMAPPO learns robust cooperative policies where traditional from-scratch approaches such as MAPPO fail. Thanks to the combination of fast training and curriculum learning, we obtain a general policy capable of tracking up to 5 simultaneous targets with only 5 vehicles (f.i. Figure 2), whereas state-of-the-art require up to 12 vehicles to track 4 targets.

II. RELATED WORK

Early research on underwater tracking (UT) with autonomous vehicles has primarily relied on traditional control strategies for AUVs [5, 14]. A promising new direction was introduced by [7], who incorporated RL to improve navigation and tracking robustness. However, their approach was limited to a single vehicle tracking fixed or slowly moving targets. More recent studies [9, 8] have explored the use of MARL to train cooperative policies for tracking moving targets. These methods, however, were evaluated in highly abstract and simplified environments. Our proposed pipeline, by contrast, is designed with implementation considerations in mind and aims to ensure that learned policies remain robust in more realistic scenarios. In particular,[9] demonstrated cooperative behaviour for a fixed configuration of two vehicles tracking a single target. Similarly to our approach,[8] employed an attention mechanism. However, their policies were trained separately for fixed team configurations (e.g., 4 agents tracking 2 targets, 6 agents tracking 3 targets, or 12 agents tracking 4 targets), and did not aim to achieve invariance with respect to team size. In contrast, our method explicitly targets generalisation across varying numbers of agents and targets, and we demonstrate robust tracking of 5 targets using 5 vehicles within a single scalable framework. Neither [9] nor [8] report wall-clock training times or provide official codebases, which makes a direct comparison of computational efficiency infeasible. More importantly, their primary objective differs from ours: they focus on learning coordination policies for fixed team sizes, whereas our work emphasises scalability and curriculum-based training across increasing team sizes.

Our TransfMAPPO architecture is inspired by TransfQMix [15], but adopts PPO instead of Q-Learning as the RL backbone due to its natural compatibility with parallelized environments. Moreover, TransfMAPPO replaces the centralized hypernetwork with a centralized critic, simplifying the model structure. Several other methods have explored scaling coordination in MARL using attention mechanisms, Graph Neural Networks (GNNs), or Transformers [16, 17, 18, 19], but a detailed comparison with these approaches is beyond the scope of this work. The most similar work to ours that combines transformers with MAPPO is Multi-Agent Transformer [20]. However, in that work transformers are used to model the multi-agent problem as a sequential decision process, whereas in our case we employ them to improve simultaneous coordination. Related to our approach, [21, 22] employ curriculum learning to progressively train policies that adapt to larger team sizes, but these methods have not been evaluated in complex robotic scenarios.

III. BACKGROUND

A. Underwater Localization with Autonomous Vehicles

Underwater acoustic tracking with autonomous vehicles is a member of the family of Range Only Single Beacon (ROSB) localization techniques [23, 24], which use range-only acoustic observations gathered over time to locate a target. For static targets, the trilateration problem can be linearised by least squares (LS) algorithms, which assume static conditions and incur errors when targets move [25]. For dynamic scenarios, particle filters (PF) use Bayesian estimation, representing potential target states with weighted particles updated via motion models and resampled based on measurement likelihoods [26, 27].

B. Multi-Agent Reinforcement Learning

Cooperative multi-agent tasks are formalized as decentralized partially observable Markov decision processes (Dec-POMDP), defined by the tuple $G = \langle S, \mathbf{U}, P, r, Z, O, n, \gamma \rangle$, where S is the global state space, $\mathbf{U} = U^n$ is the joint action

space for n agents (each selecting actions $u^a \in U$), $P(s' \mid s, \mathbf{u})$ is the state transition function, $r(s, \mathbf{u})$ is the shared reward function, Z is the observation space for each agent, $O(s, a)$ defines the observation $z^a \in Z$ for agent a, and $\gamma \in [0, 1)$ is the discount factor. At each timestep, agent a receives a partial observation $z^a = O(s, a)$ and maintains an action-observation history $\tau^a = (z_0^a, u_0^a, \ldots, z_t^a) \in \mathcal{T}$. The agent's policy $\pi^a(u^a \mid \tau^a) : \mathcal{T} \times U \rightarrow [0, 1]$ maps this history to a distribution over actions. The agents collectively aim to maximize the expected discounted return $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$. A key paradigm in MARL is centralized training with decentralized execution (CTDE) [28], where agents leverage centralized information during training but act independently during execution. This allows the use of a centralized critic that estimates the value function $V(S)$ using the global state S, while policies π^a remain decentralized, depending only on local histories τ^a.

MAPPO [13] extends Proximal Policy Optimization (PPO) [29] to multi-agent settings. Each agent's policy (actor) π_θ^a and centralized value function (critic) V_ϕ are optimized jointly. The actors are updated using the standard PPO clipped surrogate objective, while a single critic estimates the value function of the global environment's state. During execution, only the actors are used, enabling decentralized control.

IV. METHOD

A. Fast GPU training to High-fidelity testing Pipeline

1) PyLrauv: PyLrauv provides a Python interface to LRAUV Sim by leveraging ROS 2 [11]. This is achieved by integrating a set of additional communication handlers into the official Gazebo C++ library, `ros_gz`[1]. The integration allows control of the LRAUV robots with Python using `rclpy` [2]. We have developed a set of Python controllers to send commands to the LRAUV vehicles in Gazebo and observe their published states. Since the actual LRAUV vehicles use the same technology, the controllers provided within PyLrauv can also be used to control real LRAUV robots.

2) PyLrauv UTracking Environment: `UTrackingEnv` is a Gym-like environment managing agent and target commands, communication, tracking, and RL tasks like observation building and reward calculations. Agents communicate via the Gazebo `LRAUV Acoustic Comms Plugin` and collect target ranges using the `LRAUV Range Bearing Plugin`. At the start of an episode, agents are spawned on the sea surface and targets at random depths, ensuring a minimum distance between each entity of 50 m and maximum of 200 m. Both agents and targets are LRAUV vehicles. Each step lasts 30 seconds in our experiments, at the beginning of which agents listen for range signals from targets, and at the end broadcast their locations and observations to other agents (communication phase). Each agent maintains its own tracking model (LS or PF) for each target, updated with all the information received. This ensures a genuine decentralization of the system. Targets are tracked in 2D space, as depth is typically known [7].

a) Observation Space: The agents receive their partial observation of the environment at the end of the step phase, which includes the relative distances to all other entities in the 3D space. For other agents, these distances are calculated using the information they have received from them (if available), and for targets, the distances are calculated from the tracking predictions generated by their own tracking modules. [7] incorporates the range observations into observation vectors, which may be helpful, but may also cause agents to overfit to training dynamics and overlook tracking information. To prevent this, we limit observations to tracking-derived values, ensuring agents rely on tracking data and its inherent errors. The environment also returns the global state of the environment, which includes the true 3D position of each entity, its velocity, and its direction.

b) Reward Function: We define two possible reward functions. The **tracking** reward is designed to reduce the global tracking error. Let e_i denote the tracking error for target i, $\epsilon_{\min}$ the ideal error of the system, $n\epsilon_{\max}$ the maximum rewarding error, and define the reward for that target as an exponential decay function:

$$r_i^{tracking} = \begin{cases} 1, & e_i < \epsilon_{\min}, \\ \exp\left(-\dfrac{2t}{1-t}\right), & \epsilon_{\min} \le e_i \le \epsilon_{\max}, \\ 0, & e_i > \epsilon_{\max}, \end{cases} \tag{1}$$

with $t = (e_i - \epsilon_{\min})/(\epsilon_{\max} - \epsilon_{\min})$. We set $\epsilon_{\min} = 10\,\text{m}$ and $\epsilon_{\max} = 50\,\text{m}$. The exponential decay encourages precise tracking, distinguishing more clearly states where the tracking was closer to the ideal tracking error. The **follow** reward is based on the proper distribution of agents with respect to the targets. This reward encourages a perfect distribution across targets for the agents, which is known to be the best policy when N agents need to follow (rapid enough) N targets. For target i, if the minimum distance between it and any agent, d_i, is less than specified threshold, $d_{\min}$, then the target is considered successfully followed. The reward is computed as

$$r_i^{follow} = \mathbb{I}\{d_i \le d_{\min}\}, \tag{2}$$

with the threshold set in our experiments to 50 m (and 100 m for fast targets). In both cases, the global reward to the system is given by $r = \frac{1}{N}\sum_{i=1}^{N} r_i$, where N is the number of targets to track, i.e. the reward is normalized by the number of targets to be always in the interval $[0, 1]$, so that learned value functions can be transferred across scenarios with different numbers of targets. In addition, a crash penalty is always applied if the distance between any two agents falls below a minimum valid distance, d_{safe}, i.e., $r = -1 \quad \text{if} \quad \min_{i \neq j}\{d_{ij}\} < d_{safe}$.

c) Actions Space: We keep the agents' velocity constant and allow them to control only the rudder. This simplifies the action space of the multi-agent system and is equivalent to controlling the vehicle's heading. This approach is robust and can be transferred to real robots by using low-level velocity and rudder controllers to ensure that the vehicle reaches a specified heading. The rudder is discretised into five values

[1] https://github.com/gazebosim/ros_gz

[2] https://github.com/ros2/rclpy

corresponding to equidistant angles ranging from -0.24 to 0.24 radians. This is equivalent to a heading change between -1.45 and 1.45 radians when the velocity is 1 m/s and the time step is 30 seconds, as in our experiments. Agents can adjust the rudder only to one of the two adjacent angles, thereby preventing abrupt changes in direction.

d) Termination: The episode ends when the maximum number of steps is reached.

Why is UTracking a difficult environment? UTracking poses a challenging benchmark due to partial observability, where agents can only detect targets within 450 m using noisy acoustic signals and share data through intermittent communication limited to 1500 m. The environment is highly stochastic, with sensor noise, communication dropouts, environmental perturbations, and unpredictable target motions. Effective tracking requires near-perfect coordination among agents, especially when following multiple fast-moving targets, as poor coordination can quickly lead to target loss. Finally, the long horizons of hour-scale missions (thousands of steps) demand learning policies that handle delayed consequences and remain robust to error accumulation in non-stationary multi-agent settings.

3) JaxLrauv UTracking: LRAUV Sim simulates the dynamics of an LRAUV robot at the millisecond level. However, in practice, LRAUV vehicles take actions only every few seconds to minutes, while the tracking missions of interest span several hours. For this reason, it is beneficial to simulate the environment state at a much larger time scale. Formally, we are interested in modeling the new position in the 2D plane $\mathbf{p}_{t+1} \in \mathbb{R}^2$ after a time step δ_t, given the current robot position $\mathbf{p}_t$, absolute speed v, and rudder angle γ. Instead of re-building an accurate physical model (already provided by Gazebo), we approximate the change in the robot's heading, δ_ψ, in order to use a simplified trajectory model: $\mathbf{p}_{t+1} = \mathbf{p}_t + v\delta_t \left[\cos(\psi_t + \delta_\psi), \sin(\psi_t + \delta_\psi)\right]$. Fixing v and δ_t, we parametrize δ_ψ with respect to the rudder angle through a function $\theta(\gamma)$. To approximate $\theta(\gamma)$, we collect trajectories in PyLrauv and train a supervised model that predicts δ_ψ from $\theta(\gamma)$. We found that using a linear model for each fixed combination of v and δ_t, $\theta(\gamma)$ can be approximated with a mean absolute error below 0.015 radians and a global R^2 score of 0.99. Based on this, we build an ensemble of linear models for different combinations of v and δ_t. While such a simple model cannot capture long-term dependencies and diverges when used autoregressively for long trajectories, it performs well for shorter trajectories and is much faster than, for example, an LSTM or a handwritten physical system. To improve robustness, we introduce Gaussian noise with a standard deviation of 0.02 radians, forcing the trained agent to handle trajectory uncertainty. In addition, we equip the agents with recurrent mechanisms. We found that training in this simplified environment leads to policies that transfer directly to Gazebo.

The rest of the environment, including measurement noise, communication loss, partial observability, and collision handling, is implemented manually based on data collected in Gazebo. The final simplified environment is implemented in JAX, enabling an end-to-end GPU-based RL pipeline. To further improve training efficiency, we implement a Particle Filter in JAX, allowing the tracking phase to also benefit from GPU acceleration. By nesting vectorized functions, we update all particles across targets, agents, and parallel environments in a single pass, achieving both high speed and precise tracking during training. The same vectorized PF algorithm is also integrated into PyLrauv for consistency. The speedups achieved across different agent–target configurations are reported in Table I. These gains make it possible to train a single-target tracking model that can be directly deployed in LRAUV Sim in as little as 10 minutes.

TABLE I: Speedups Obtained with the Accelerated Environment. Steps Per Second (SPS) and relative speedup.

Config	Pylrauv	JaxLrauv 1 Envs		JaxLrauv 128 Envs		JaxLrauv 1024 Envs	
	SPS	SPS	Speedup	SPS	Speedup	SPS	Speedup
1A,1T	2.7	1289	477	68508	25352	81686	30229
2A,2T	1.0	1068	1084	19712	20017	21534	21867
3A,3T	0.4	1053	2439	9128	21139	9786	22663
4A,4T	0.3	1020	3571	5180	18125	5450	19070
5A,5T	0.2	936	4751	3469	17607	3574	18142

B. Curriculum Learning Via Transformers

1) TransfMAPPO: The complexity of training multi-agent systems with RL typically scales with the number of agents involved. To address this, we introduce *TransfMAPPO*, inspired by TransfQMix [15]. Our approach preserves the core ideas of the latter but replaces DQN with PPO as the RL backbone. PPO is naturally suited to vectorised environments, and its clipping mechanism acts as an implicit KL regulariser, which is important for stabilising curriculum learning. To leverage Transformers, we formalise the problem as learning a latent coordination graph through self-attention, where only the vertices of the graph are explicitly defined. Specifically, the n vertices (or tokens) correspond to the agents and targets present in the environment, each represented by z features such as position, velocity, and orientation. This representation is permutation-invariant, allowing the Transformer to learn a coordination graph that generalises across different numbers of agents and targets—much like an LLM can learn semantic attention patterns over sentences of varying length. This structural property of the Transformer, namely its invariance to input ordering and flexibility with respect to input size, makes the network naturally transferable across team sizes, thereby enabling the construction of a curriculum with progressively increasing numbers of agents. We distinguish between two types of latent representations: (1) the *local latent graphs* which can be learned by each agent aggregating noisy and potentially incomplete information from onboard sensors and inter-agent communications, and (2) the *global latent graph* learnable using the absolute states of all agents and targets, which is available only during training in simulation. Accordingly, we use two Transformer networks Figure 3: the *TransformerAgents*, deployed in a decentralized fashion (one per vehicle), which learns the local coordination graph to control the agents using only their local observations; and the *TransformerCritic*, which learns a value function for the

joint action–state pair using the global information, and is employed for centralized training. The use of Transformers in both actor and critic improves coordination representation and yields policies and value functions that are invariant to the number of entities. This property enables the curriculum learning procedure described in subsubsection IV-B.2.

Following [30], we found that normalizing the entire network with LayerNormalization greatly improves stability and performances.

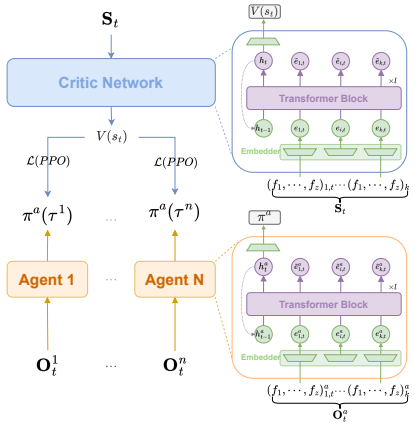


Fig. 3: In TransfMAPPO, both each agent (actor) and the centralized critic are implemented as Transformer networks. Each TransformerAgent observes the environment as a set of entity vectors describing itself, its teammates, and the targets. Each vector encodes the (possibly noisy or missing) relative 3D distance with respect to the observing agent, together with a one-hot identifier indicating whether the entity corresponds to itself, another agent, or a target. These vectors are embedded and treated as graph vertices; self-attention is applied to learn a latent local coordination graph. An additional recurrent embedding is processed jointly by the Transformer, and the final action distribution is produced by an MLP applied to the transformed recurrent embedding. The TransformerCritic follows the same architecture but is used only during training and receives the full global state (true positions, velocities, and orientations of all agents and targets). It also maintains a recurrent embedding, from which a final MLP outputs the value estimate $V(s)$.

2) Curriculum Learning: Taking advantage of our Transformer-based architecture, we employ curriculum learning to progressively learn Multi-Agent policies starting from a Single-Agent policy (see Figure 4). We begin with a large pre-training phase in which we train a single agent to maximize the **tracking** reward described in subsubsection IV-A.1 with respect to an erratic-moving fast target (0.6x of its speed). We train for 10^{10} timesteps, corresponding to 1.5 days of compute (that would take months in Gazebo). In this phase, we maintain a short horizon (128 steps) before resetting the environment, as it significantly aids learning. The resulting policy is able to track a fast target with an error below 35m, but it fails to generalize to longer horizons. We therefore fine-tune the policy in progressively longer episodes (256, 512, and 1024 steps) for 10^8 timesteps, until we observe that the policy can perfectly track a fast target for more than 10k steps, corresponding to more than 3 days of real-time tracking.

From this horizon-invariant base model, we transition to a multi-agent phase, where we fine-tune the policy for cooperative tasks. The actor parameters are preserved from the single-agent phase and shared across all agents; adding a new agent simply corresponds to instantiating another copy of the same TransformerAgent with identical weights. In particular, we consider two scenarios: (1) n agents tracking n targets, using the **follow** reward, and (2) n agents tracking one very fast target (0.8x of their velocity), maximising the same **tracking** reward used in the single-agent phase. When transitioning from the single-agent to the multi-agent setting, we reset the *critic parameters once*, since the value function must now estimate joint returns under coordination and credit assignment effects that were absent in the single-agent case. We then maintain separate multi-agent fine-tuning branches (one per scenario), without further parameter resets as the team size increases. In both cases, the episode horizon remains fixed during this phase, and we fine-tune the model to cooperate with progressively more agents (5 and 3 at maximum, respectively) for a maximum of 2^9 timesteps.

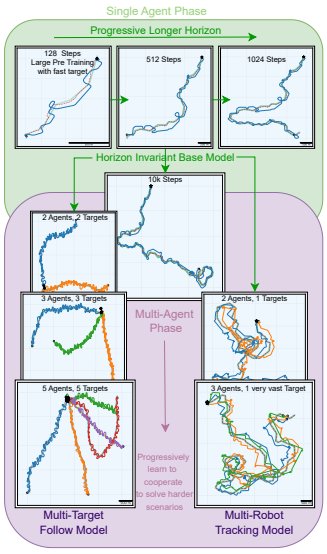


Fig. 4: Overview of our Curriculum Learning Procedure.

V. EXPERIMENTS

All the training was performed on a single H100 GPU of MareNostrum 5 while the Gazebo simulator experiments run on a Intel Sapphire Rapids 8460Y CPU. All the episode returns are normalized for the horizon length to be presented in the interval $[0, 1]$. All the training results are averaged across 5 seeds, except for the curriculum models. All the figures representing agents trajectories are produced with data collected in the Gazebo simulator.

A. Multi-Robot Tracking

In Figure 5, we show the training curve for the most complex multi-robot tracking task we consider, i.e., three agents tracking a very fast target (0.8× the agents' velocity). Specifically, we present results for training standard MAPPO (with an RNN) and TransfMAPPO from scratch, as well as the result of fine-tuning TransfMAPPO on this task following the curriculum procedure described in subsubsection IV-B.2. Note that training Transformers from scratch on this task is not more effective than using an RNN; however, the benefits

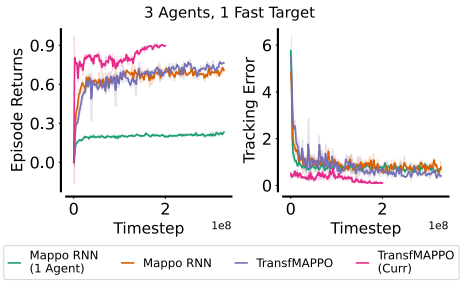


Fig. 5: Training multiple agents to track a very fast target.

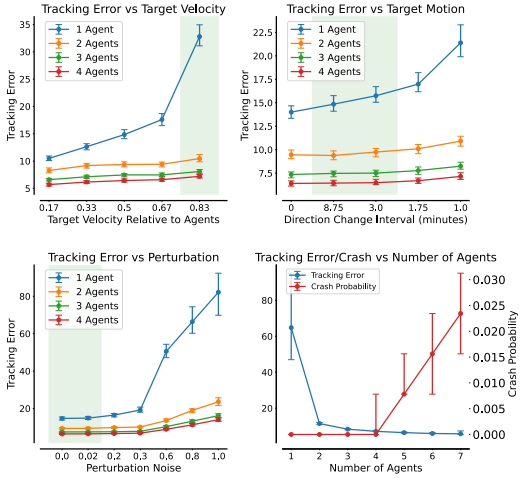


Fig. 6: Multi-Robot Tracking Evaluation. The highlighted area represents the values seen in training.

of curriculum learning are evident in terms of expected returns and reduced tracking error.

In Figure 6, we perform a thorough evaluation of the final multi-target follow model obtained through curriculum training. Each data point in the validation phase is averaged over 1000 episodes, each lasting 256 steps, with the target moving at 0.5× the agents' velocity (unless otherwise specified). Specifically, we assess how the tracking error is affected under three conditions: (1) when the target moves faster, (2) when its motion becomes more unpredictable (defined by the frequency of direction changes), and (3) when the agents' motion is perturbed (simulating external forces such as ocean currents). In the last sub-figure, we illustrate how the performance of the multi-agent system scales with the number of agents used to track the target. As expected, even using only two agents significantly improves the system's robustness compared to a single-agent approach. Robustness increases proportionally with the number of agents, both in terms of resistance to variations in target velocity, target motion (i.e., the interval of target direction changes), and perturbation noise (additional noise added to agents' trajectories). We also highlight that the tracking error decreases monotonically with the number of agents, with three agents representing a sweet spot: achieving negligible collision probability while maintaining optimal tracking performance.

B. Multi-Target Tracking

Similarly, our curriculum method achieves remarkable advantages compared to training MAPPO from scratch in the multi-target tracking scenario. In Figure 7, we present the training curves of training MAPPO from scratch to track moderately fast targets (up to 0.5x the agents' velocity). MAPPO can learn to follow barely half of the targets only in the 2-agent-2-target scenario, while our curriculum approach successfully follows more than 90% of the targets on average, even in the 5-agent-5-target scenario.

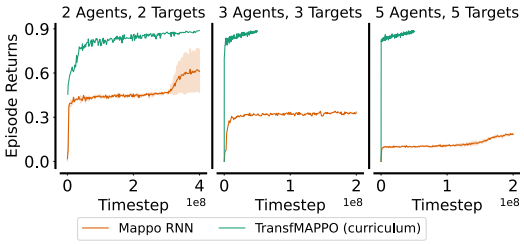


Fig. 7: Training Multiple Agents to Follow Fast Targets.

To further test the effectiveness of TransfMAPPO, we train it from scratch alongside MAPPO in a simpler configuration where targets move slower (0.3x the agents's speed) and in a more predictable manner (Figure 8). In this case, TransfMAPPO and MAPPO achieve similar scores with 2 agents, but MAPPO's performance drops significantly with 3 agents. Surprisingly, TransfMAPPO learns very effectively even from scratch in the 5-agent-5-target scenario, confirming the sample efficiency obtained by learning a coordination graph using Transformers.

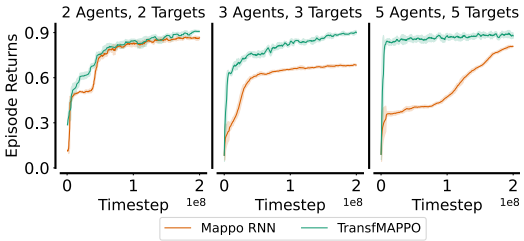


Fig. 8: Training Multiple Agents to Follow Slower Targets.

C. Results in Gazebo

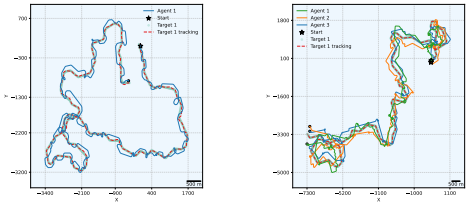
We conducted a series of final experiments in Gazebo to evaluate the performance of our trained agents in a high-fidelity simulation environment. We collected data over 50 episodes using the models obtained at the conclusion of the curriculum training. Specifically, we performed the following tests:

1) Tracking a slow target (0.3× the agent's velocity) for 300 steps using a single robot trained for 10 minutes in JaxLrauv.
2) Tracking a fast target (0.6× the agent's velocity) for 1000 steps (equivalent to over 8 hours of real-time tracking) using our horizon-invariant base model.
3) Tracking a very fast target (0.8× the agent's velocity) for 1000 steps using a multi-robot tracking fine-tuned model with 3 agents.
4) Tracking 3 targets simultaneously moving at 0.5× the agents' velocity using a multi-target model for 300 steps.
5) Tracking 5 targets simultaneously moving at 0.5× the agents' velocity using a multi-target model for 300 steps.

TABLE II: Evaluation in JaxLrauv and PyLrauv of the curriculum training policies (* except for first line which refers to 10 minutes training).

Configuration	Avg. Agent-Target Distance		Avg. Tracking Error		Probability of Collision (%)		Probability of Loosing a Target (%)	
	JaxLrauv	PyLrauv	JaxLrauv	PyLrauv	JaxLrauv	PyLrauv	JaxLrauv	PyLrauv
1 Agent, 1 Target (Slow)*	55.20 ± 30.11	59.10 ± 32.29	5.12 ± 3.1	5.89 ± 3.7	0.00	0.00	0.00	0.00
1 Agent, 1 Target (Fast)	110.23 ± 58.90	121.01 ± 62.11	17.40 ± 21.5	20.33 ± 28.38	0.00	0.00	5.00	7.14
3 Agents, 1 Target (Very Fast)	200.11 ± 85.70	217.64 ± 89.85	2.65 ± 5.30	3.03 ± 6.17	0.0	15.32	0.00	0.00
3 Agents, 3 Targets (Moderate)	42.15 ± 25.00	45.30 ± 26.11	4.85 ± 4.20	5.29 ± 4.88	0.5	5.00	3.50	5.00
5 Agents, 5 Targets (Moderate)	46.12 ± 28.75	49.24 ± 30.92	3.80 ± 4.00	4.25 ± 4.57	2.1	10.00	4.10	5.26

For the multi-target experiments, we reduced the number of steps to 300 due to the computational burden of simulating multiple robots in Gazebo. The results of these experiments are presented in Table II. For completeness, we also include in this table the results obtained in JaxLrauv with the same configurations, so that it is possible to appreciate how the learned policy behaves in a very similar way in both simulators, resulting in comparable evaluation measures. Our findings demonstrate that even with only 10 minutes of training, we were able to develop an effective model capable of maintaining target tracking without losing the target. The multi-robot tracking model achieved a promising average tracking error of 3m, even when following a very fast target. This performance highlights a clear advantage over single-agent trackers following slower targets. Notably, the average agent-target distance increased when using multiple robots, reflecting a broader distribution of robots around the target space. The multi-target models performed as expected, with a small 5% probability of losing a single landmark. However, the multi-agent models exhibited an increased probability of collisions. This issue is likely attributable to the fact that the models were trained to avoid collisions only every 30 seconds, which can result in intra-step collisions in Gazebo. This underscores a critical area for future work to enhance the systems' safety before deploying them in real-world marine missions. Finally, in Figures 9 to 12, we present some of the trajectories obtained during the final evaluation, together with zoomed-in views of the emergent coordination strategies learned.



(a) (b)

Fig. 9: (a): Single-agent tracking, target velocity 0.66× of agent velocity. Video. (b): Multi-agent tracking, target velocity 0.88x of agents' velocity. Video.

VI. CONCONCLUSIONS

We presented a method for scaling up Multi-Agent Reinforcement Learning (MARL) in the context of underwater tracking, demonstrating how it is possible to efficiently train MARL models purely on GPUs and deploy them in high-fidelity simulators, where MARL training would otherwise

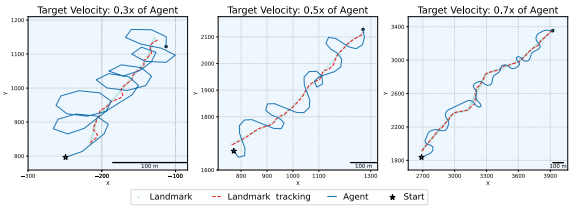


Fig. 10: Agents' trajectory according to target speed. Notice the change in the curvature of the agent's trajectory to optimally track targets moving at different speeds.

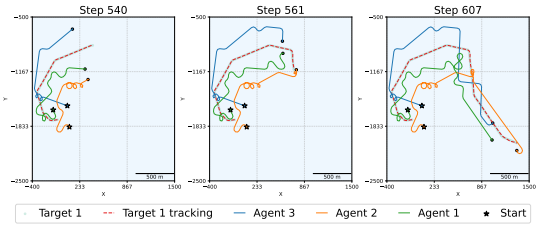


Fig. 11: Coordination when tracking a very fast target. Notice how agents react to a rapid change of direction of the target by waiting for one other in order not to lose communication and then rapidly moving together.

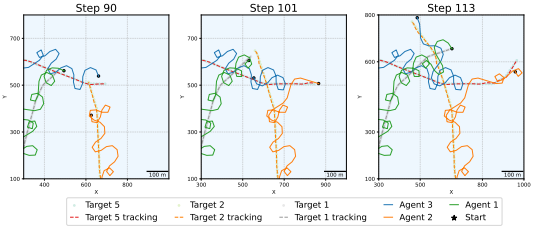


Fig. 12: Coordination in a multi-target setting. Notice how blue and orange agents resolve the "traffic" problem by making circles to wait for other agents' passing, and one target is dynamically reassigned from one agent to another.

be extremely expensive or infeasible. Our overall pipeline represents a promising approach for closing the sim-to-real gap in MARL. A natural next step for this work is to enhance the safety of the multi-agent system and test it in real-world sea missions.

ACKNOWLEDGMENT

This work acknowledges the Spanish Ministerio de Ciencia, Innovación y Universidades (TECTUGA: PID2024-161772OA-I00). This work is part of DIGI4ECO, European Union's Horizon Europe programme (No 101112883). I. M. received financial support from the MCIN/AEI/10.13039/501100011033 and FSE+ (No

RYC2022-038056-I). M. G. was partially funded by the FPI-UPC Santander Scholarship FPI-UPC_93. This work also acknowledges the 'Severo Ochoa Centre of Excellence' accreditation (CEX2024-001494-S) from AEI 10.13039/501100011033 and the computing resources provided by the Barcelona Supercomputing Center (BSC).

References

[1] Nikolaos D Zarokanellos et al. "Frontal dynamics in the Alboran sea: 1. Coherent 3D pathways at the Almeria-Oran front using underwater glider observations". In: *Journal of Geophysical Research: Oceans* 127.3 (2022), e2021JC017405.

[2] Ivan Masmitja et al. "Mobile robotic platforms for the acoustic tracking of deep-sea demersal fishery resources". In: *Science Robotics* 5.48 (2020), eabc3701.

[3] Yanwu Zhang et al. "A system of coordinated autonomous robots for Lagrangian studies of microbes in the oceanic deep chlorophyll maximum". In: *Science Robotics* 6.50 (2021), eabb9138.

[4] Maria Vigo et al. "Spatial ecology of Norway lobster Nephrops norvegicus in Mediterranean deep-water environments: implications for designing no-take marine reserves". In: *Marine Ecology Progress Series* 674 (2021), pp. 173–188.

[5] John Heidemann, Milica Stojanovic, and Michele Zorzi. "Underwater sensor networks: applications, advances and challenges". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 370.1958 (2012), pp. 158–175.

[6] David Cote et al. "Characterizing snow crab (Chionoecetes opilio) movements in the Sydney Bight (Nova Scotia, Canada): a collaborative approach using multiscale acoustic telemetry". In: *Canadian Journal of Fisheries and Aquatic Sciences* 76.2 (2019), pp. 334–346.

[7] I. Masmitja et al. "Dynamic robotic tracking of underwater targets using reinforcement learning". In: *Science Robotics* 8.80 (2023), eade7811.

[8] Shengbo Wang et al. "Multi-AUV cooperative underwater multi-target tracking based on dynamic-switching-enabled multi-agent reinforcement learning". In: *IEEE Transactions on Mobile Computing* (2024).

[9] Zhaoqi Yang et al. "Secure and cooperative target tracking via AUV swarm: A reinforcement learning approach". In: *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2021, pp. 1–6.

[10] Timothy R. Player et al. "From Concept to Field Tests: Accelerated Development of Multi-AUV Missions Using a High-Fidelity Faster-than-Real-Time Simulator". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3102–3108.

[11] Steven Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074.

[12] Alexander Rutherford et al. "JaxMARL: Multi-Agent RL Environments in JAX". In: *arXiv preprint arXiv:2311.10090* (2023).

[13] Chao Yu et al. "The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games". In: (2022). arXiv: 2103.01955 [cs.LG].

[14] João Almeida, Carlos Silvestre, and António Pascoal. "Synchronization of Multiagent Systems Using Event-Triggered and Self-Triggered Broadcasts". In: *IEEE Transactions on Automatic Control* 62.9 (2017).

[15] Matteo Gallici, Mario Martin, and Ivan Masmitja. "TransfQMix: Transformers for Leveraging the Graph Structure of Multi-Agent Reinforcement Learning Problems". In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '23. 2023, pp. 1679–1687.

[16] Siddharth Nayak et al. "Scalable multi-agent reinforcement learning through intelligent information aggregation". In: *International conference on machine learning*. PMLR. 2023, pp. 25817–25833.

[17] Anthony Goeckner et al. "Graph neural network-based multi-agent reinforcement learning for resilient distributed coordination of multi-robot systems". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024.

[18] Yuxin Cai et al. "Transformer-based multi-agent reinforcement learning for generalization of heterogeneous multi-robot cooperation". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024, pp. 13695–13702.

[19] Siyi Hu et al. "UPDeT: Universal Multi-agent RL via Policy Decoupling with Transformers". In: *International Conference on Learning Representations*. 2020.

[20] Muning Wen et al. "Multi-agent reinforcement learning is a sequence modeling problem". In: *Advances in Neural Information Processing Systems* 35 (2022).

[21] Tianle Zhang et al. "Automatic curriculum learning for large-scale cooperative multiagent systems". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 7.3 (2022), pp. 912–930.

[22] Weixun Wang et al. "From few to more: Large-scale dynamic multiagent curriculum learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 05. 2020, pp. 7293–7300.

[23] Edwin Olson, John J. Leonard, and Seth Teller. "Robust Range-Only Beacon Localization". In: *IEEE Journal of Oceanic Engineering* 31.4 (2006), pp. 949–958.

[24] Jake D. Quenzer and Kristi A. Morgansen. "Observability based control in range-only underwater vehicle localization". In: *2014 American Control Conference*.

[25] Alex Alcocer Penas. "Positioning and navigation systems for robotic underwater vehicles". In: *Doctor thesis, Instituto Superior Tcnico* (2009).

[26] Jake D Quenzer and Kristi A Morgansen. "Observability based control in range-only underwater vehicle localization". In: *2014 American control conference*. IEEE. 2014, pp. 4702–4707.

[27] Edwin Olson, John J Leonard, and Seth Teller. "Robust range-only beacon localization". In: *IEEE Journal of Oceanic Engineering* 31.4 (2006), pp. 949–958.

[28] Tabish Rashid et al. *QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*. 2018. arXiv: 1803.11485 [cs.LG].

[29] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[30] Matteo Gallici et al. "Simplifying Deep Temporal Difference Learning". In: *The International Conference on Learning Representations (ICLR)* (2025). eprint: 2407.04811 (cs.LG).