ATCI Practical 1

Pedro Agundez Fernandez

Abstract

This document outlines a Reinforcement Learning (RL) framework applied to two distinct autonomous system challenges: oil spill mapping and underwater landmark tracking. Both systems utilize Soft Actor-Critic (SAC) or Proximal Policy Optimization (PPO) agents to navigate simulated 2D environments. The actual implementations reside in separate software repositories, though they share significant architectural similarities and leverage common RL principles. The landmark tracking project, in particular, aims to replicate and build upon the methodologies presented in Masmitja et al. (2023) [3]. This document details the shared conceptual RL framework, including the configuration system philosophy, agent architectures, and training workflows. It then delves into the project-specific adaptations within each respective repository, covering their unique environment designs, state and action representations, reward structures, specialized components like landmark estimators, and project-specific experimental designs aimed at evaluating algorithmic performance and feature impact based on explicitly defined configurations.

Contents

1 Introduction

Autonomous systems navigating complex and dynamic environments represent a significant domain for Reinforcement Learning. This document describes the conceptual design and specific implementations of an RL framework applied to two such problems:

1. **Autonomous Oil Spill Mapping**: An RL-driven agent controls a simulated vehicle to efficiently survey an area and estimate the boundaries of an oil spill.

2. **Autonomous Underwater Landmark Tracking**: An RL-driven agent controls a simulated vehicle to actively maneuver and improve its estimation of a potentially moving underwater landmark's position using range-only acoustic measurements. This project is inspired by and seeks to replicate aspects of the work by Masmitja et al. (2023) [3] on range-only underwater target localization.

These projects were developed in separate software repositories: the Oil Spill Mapping project is available at `github.com/chernowi/atci-p1-oil-spill-mapping`, and the Landmark Tracking project at `github.com/chernowi/atci-p1-target-tracking`. Interactive web applications to visualize the trained agents are also provided: for oil spill mapping at `oilspill.streamlit.app`, and for landmark tracking at `asv-vis.streamlit.app`.

While developed independently, both projects leverage a common foundation of RL algorithms, specifically Soft Actor-Critic (SAC) [1] and Proximal Policy Optimization (PPO) [2], which are well-suited for continuous control tasks. In both scenarios, the agents operate at a constant speed defined in their respective configuration files; the RL policies learn to control the agent's yaw angle (or change in yaw angle) to achieve their objectives. This document details the shared conceptual framework components and the specific design choices made within each project's codebase to tailor these algorithms to the respective problem domains.

The core architectural philosophy across both repositories emphasizes modularity and configurability. Centralized configuration systems, managed through Pydantic models in project-specific `configs.py` files, allow for systematic experimentation. Despite the separate codebases, the structural approach to defining agents, environments, and training pipelines exhibits strong parallels. Each project, however, features a unique environment, distinct state and action space definitions, and specialized reward engineering to guide the learning process effectively within its own context.

2 Core Framework Concepts and Configuration Philosophy

A common set of tools and a unified configuration philosophy are employed across both projects, even though they are implemented in separate repositories. This facilitates understanding the underlying design patterns.

2.1 Centralized Configuration (Project-Specific `configs.py`)

In each project's repository, all major parameters for the agents, environments, training, and evaluation are managed through Pydantic models defined in a dedicated `configs.py` file. A `DefaultConfig` class (or a similarly named top-level configuration class) serves as the main container. This class holds specific configurations for SAC (`SACConfig`), PPO (`PPOConfig`), the world (`WorldConfig`), replay buffers (`ReplayBufferConfig`), training (`TrainingConfig`), evaluation (`EvaluationConfig`), visualization (`VisualizationConfig`), and other project-specific components (e.g., `MapperConfig` for Oil Spill, `LeastSquaresConfig` for Landmark Tracking).

Key global constants or default fields within each project's `WorldConfig` define the core dimensionality of observations and actions for that specific application:

- `CORE_STATE_DIM`: Dimension of the basic instantaneous state tuple. For the Oil Spill project, this is 8 (5 sensor readings + normalized X, Y, heading). For the Landmark Tracking project, it is 9 (agent_x, y, vx, vy, heading, est_landmark_x, y, depth, range), aligning with the type of observations used in range-only localization problems [3].
- `CORE_ACTION_DIM`: Dimension of the agent's action space. This is 1 for both projects, representing a normalized yaw change.
- `TRAJECTORY_REWARD_DIM`: Dimension of the reward component in the trajectory history, typically 1.

The configuration system in each repository supports creating variations of default settings, allowing for easy setup of hyperparameter sweeps or different experimental conditions. This is crucial for the experimental designs detailed later in Section 3.2 and Section 4.2.

3 Project 1: Autonomous Oil Spill Mapping

This project, developed in its own repository, focuses on training an agent to autonomously map an oil spill in a 2D simulated environment.

3.1 Environment: Oil Spill Mapping World (`world_oil_spill.py`)

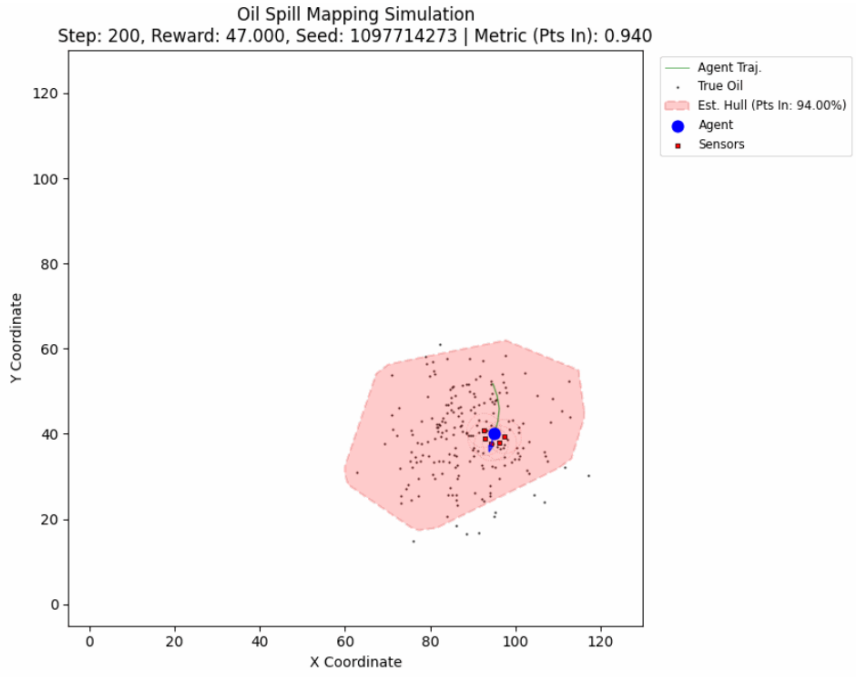


Figure 1: Simulated environment for the oil spill mapping task, showing an agent (blue circle with sensors) and an oil spill (red points).

The custom-built environment (from the Oil Spill project's repository) simulates the oil spill mapping task. `CORE_STATE_DIM` is 8, `CORE_ACTION_DIM` is 1.

3.1.1 Environment Setup and Dynamics

- Spill: `num_oil_points`, randomized cluster.

- Agent: Randomized start, constant speed (defined in configuration), `num_sensors`, `sensor_distance`, `sensor_radius`. The agent's movement is at a fixed speed, with the RL policy controlling its yaw.
- Mapper: Uses `MapperConfig`, estimates hull, `performance_metric` is oil point inclusion.
- Termination: `success_metric_threshold`, `terminate_out_of_bounds`, `max_steps`.

3.1.2 State Representation

- Basic State (8-dim): 5 sensor readings, normalized X, Y, normalized heading.
- Trajectory History (length 10): Sequence of (normalized basic state, action, raw reward). Feature dim 10.

3.1.3 Action Space

- Continuous, 1-dim: Normalized yaw change, scaled by `yaw_angle_range[1]`. The agent moves at a constant speed, and this action directly dictates the change in its heading.

3.1.4 Reward Engineering for Mapping

The reward function in the oil spill mapping project is a critical element, meticulously designed through experimentation to guide the agent towards efficient and accurate spill delineation. The default values for its components, as defined in the `WorldConfig` within `configs.py`, represent settings found to promote stable learning and effective behavior. The total reward at each step $t+1$, denoted R_{t+1}, is a sum of the following components:

- **Coverage Improvement (`metric_improvement`)** $R_{coverage}$: This is the primary driver for effective mapping. It rewards the agent for increasing the percentage of true oil points enclosed by its estimated spill boundary (the convex hull). Calculated as $S \times \max(0, M_{t+1} - M_t)$, where M is the performance metric (point inclusion percentage) and S is `world_config.metric_improvement_scale`. The default scale (`50.0`) was found to provide a strong, positive signal that directly reinforces actions leading to better map accuracy, ensuring this core objective is prioritized.
- **New Oil Detection Bonus (`new_oil_detection`)** R_{detect}: This component is intended to encourage exploration by rewarding the agent for new oil discoveries. The default value for `world_config.new_oil_detection_bonus` is `0.0`. This setting was experimentally determined to be optimal for stability; it was found that the primary coverage improvement reward, combined with the inherent need to explore to improve the metric, was sufficient. Adding an explicit bonus for new detections did not consistently improve performance and sometimes led to less stable learning patterns in the default configuration.
- **Step Penalty (`step_penalty`)** R_{step}: A penalty applied at each time step to encourage efficiency. The default value for (`world_config.step_penalty`) is (`0`). Through experimentation, it was observed that other reward components and the maximum episode step limit (`world_config.max_steps`) implicitly encouraged efficient policies. A non-zero step penalty was found to sometimes overly discourage exploration, especially in early stages, without a clear benefit to final mapping quality or stability in the default setup.
- **Uninitialized Mapper Penalty (`uninitialized_penalty`)** R_{uninit}: This penalizes the agent if its internal mapper has not yet gathered enough data (default: 3 oil-detecting sensor locations via `mapper_config.min_oil_points_for_estimate`) to form an initial spill estimate. The default value for `world_config.uninitialized_mapper_penalty` is

0. Similar to the step penalty, experiments showed that the drive to achieve coverage improvement was a strong enough incentive for the agent to quickly make initial oil detections. An explicit penalty here was not critical for stable learning progression in the default settings.

- **Out-of-Bounds Penalty** (`out_of_bounds_penalty`) R_{oob}: A significant penalty is applied if the agent moves outside the defined world boundaries. The default value for `world_config.out_of_bounds_penalty` is `20.0`. This substantial negative reward was found to be essential for stable learning, effectively teaching the agent to respect operational limits and avoid unproductive or unsafe trajectories.

- **Success Bonus** (`success_bonus`) $R_{success}$: A large positive reward is given when the performance metric first meets or exceeds the `world_config.success_metric_threshold` (default 95% coverage). The default value for `world_config.success_bonus` is `55.0`. This substantial bonus provides a strong terminal signal, reinforcing policies that achieve comprehensive mapping and contributing significantly to stable convergence towards the desired outcome.

The careful balancing of these components, particularly the emphasis on coverage improvement and significant terminal rewards/penalties through their default values, has been key to achieving robust learning for the oil spill mapping task. The formula for the total reward at step $t + 1$ is $R_{t+1} = R_{coverage} + R_{detect} + R_{step} + R_{uninit} + R_{oob} + R_{success}$.

3.2 Experimental Design and Research Questions (Oil Spill Mapping)

The `configs.py` file within the Oil Spill Mapping project's repository defines a set of configurations for experimentation, designed to address key research questions regarding agent performance and learning characteristics.

3.2.1 Research Questions (Oil Spill Mapping)

This experimental setup aims to answer:

1. **Algorithm Comparison**: How do SAC and PPO (MLP versions) compare in terms of mapping efficiency (coverage vs. steps), sample efficiency, and final performance, as reflected by the accumulated rewards? (Comparing `"default_sac_mlp"` vs. `"default_ppo_mlp"`).

2. **Impact of Recurrence**: Does incorporating RNNs improve the ability of SAC and PPO agents to learn effective mapping strategies, potentially by better utilizing temporal patterns in sensor readings, leading to higher rewards? (Comparing MLP vs. RNN versions for both SAC and PPO).

3. **Benefit of PER for SAC**: Does PER accelerate learning or lead to better final mapping performance (higher rewards) for the SAC MLP agent? (Comparing `"default_sac_mlp"` with `"sac_mlp_per"`).

4. **Sensitivity to Hyperparameters**: How do variations in learning rates, network sizes, discount factors, and other key hyperparameters affect the reward accumulation of SAC MLP, PPO MLP, and SAC RNN agents in the mapping task?

3.2.2 Base Configurations (Oil Spill Mapping)

The `CONFIGS` dictionary in the Oil Spill Mapping `configs.py` defines several foundational experimental setups to address the research questions:

- `"default_sac_mlp"`: Serves as the baseline SAC agent with an MLP architecture and standard parameters for the oil spill mapping environment. This is also aliased as `"default_mapping"`.
- `"default_ppo_mlp"`: Baseline PPO agent with an MLP architecture.
- `"default_sac_rnn"`: SAC agent employing an RNN (LSTM by default, `sac.rnn_hidden_size = 68`) to process trajectory history.
- `"default_ppo_rnn"`: PPO agent employing an RNN (GRU by default, `ppo.rnn_hidden_size = 64`).

3.2.3 Hyperparameter Variations (Oil Spill Mapping)

The configuration file systematically generates variations for key hyperparameters of SAC MLP, PPO MLP, and SAC RNN agents to investigate their sensitivity (RQ4).

SAC MLP Hyperparameter Variations: Based on `"default_sac_mlp"`:

- **Learning Rates**: Actor LR (`sac.actor_lr`: 1e-5, 1e-4), Critic LR (`sac.critic_lr`: 1e-5, 1e-4).
- **Discount Factor**: Gamma (`sac.gamma`: 0.95, 0.999).
- **Target Update Rate**: Tau (`sac.tau`: 0.001, 0.01).
- **Network Size**: Hidden Dims (`sac.hidden_dims`: [64,64], [256,256]).

These result in named configurations like `"sac_mlp_actor_lr_low"`, `"sac_mlp_hidden_dims_large"`, etc.

PPO MLP Hyperparameter Variations: Based on `"default_ppo_mlp"`:

- **Learning Rates**: Actor LR (`ppo.actor_lr`: 1e-5, 1e-4).
- **GAE Lambda**: (`ppo.gae_lambda`: 0.90, 0.99).
- **Policy Clip**: (`ppo.policy_clip`: 0.1, 0.3).
- **Entropy Coefficient**: (`ppo.entropy_coef`: 0.005, 0.5).
- **Network Size**: Hidden Dim (`ppo.hidden_dim`: 128, 512).

These result in named configurations like `"ppo_mlp_actor_lr_high"`, `"ppo_mlp_policy_clip_low"`, etc.

SAC RNN Hyperparameter Variations: Based on `"default_sac_rnn"`:

- **RNN Hidden Size**: (`sac.rnn_hidden_size`: 32, 128).

Resulting in configurations like `"sac_rnn_rnn_hidden_size_small"`.

SAC MLP with Prioritized Experience Replay (PER):

- `"sac_mlp_per"`: Based on `"default_sac_mlp"` with `sac.use_per = True`, used to investigate RQ3.

3.3 Oil Spill Mapping Results

This section presents results from the oil spill mapping experiments, focusing on the Average Return and Performance Metric to assess agent performance. To maintain the brevity of this report and focus on core comparisons, the detailed exploration of hyperparameter sensitivity (RQ4), while conducted, is only briefly exemplified; not all individual plots for these variations are presented.

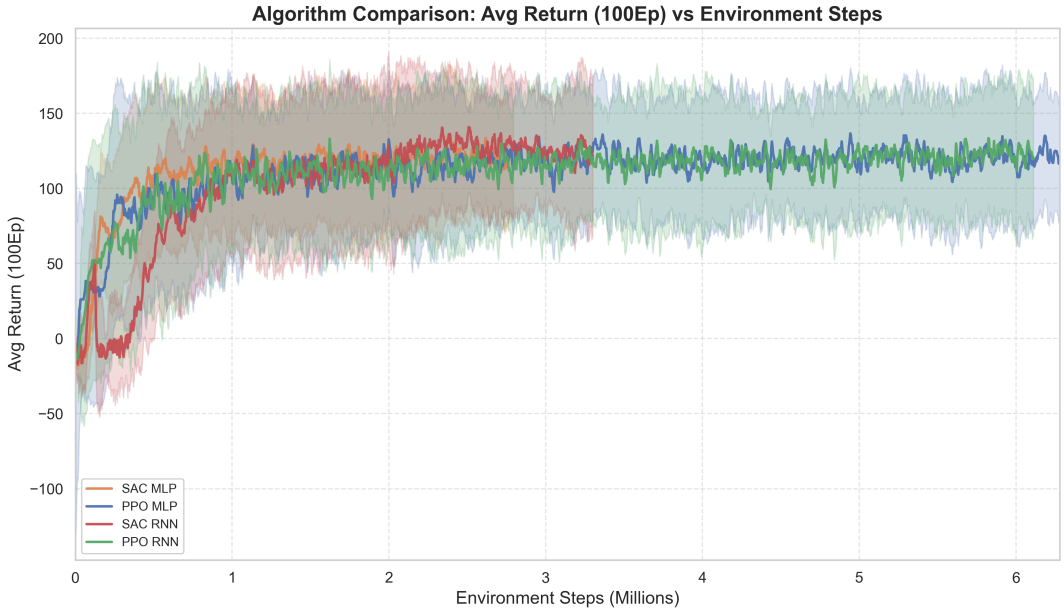


Figure 2: Oil Spill Mapping: Algorithm Comparison - Average Return vs. Environment Steps for SAC MLP, PPO MLP, SAC RNN, and PPO RNN. This addresses RQ1 and RQ2.

Figure 2 compares the reward accumulation profiles for different algorithms and architectures in the oil spill mapping task.

- **Algorithm Comparison (RQ1) and Impact of Recurrence (RQ2)**: The learning curves for SAC MLP (orange), PPO MLP (blue), SAC RNN (red), and PPO RNN (green) demonstrate broadly similar overall performance characteristics in this challenging oil spill mapping environment. All agents show an initial phase of learning where average returns increase, eventually approaching a plateau. While there are visual variations in their learning trajectories—for instance, SAC MLP and PPO MLP exhibit closely matched profiles, and SAC RNN shows an initial dip before recovering—their final performance levels, considering the inherent stochasticity of the training process and the visual overlap in their reward bands in later stages, do not indicate a definitive superior algorithm among these four configurations. PPO RNN (green), while appearing to achieve somewhat lower returns in this particular run, still demonstrates a learning trend, and its performance, when viewed in the context of the overall complexity and single-run data, contributes to the general observation of comparable efficacy. Given that only a single run is presented for each configuration, definitive conclusions about subtle performance differences or the specific impact of recurrence are tentative. The primary takeaway is that all tested approaches (SAC/PPO, MLP/RNN) learn to perform the task, reaching broadly comparable levels of proficiency.

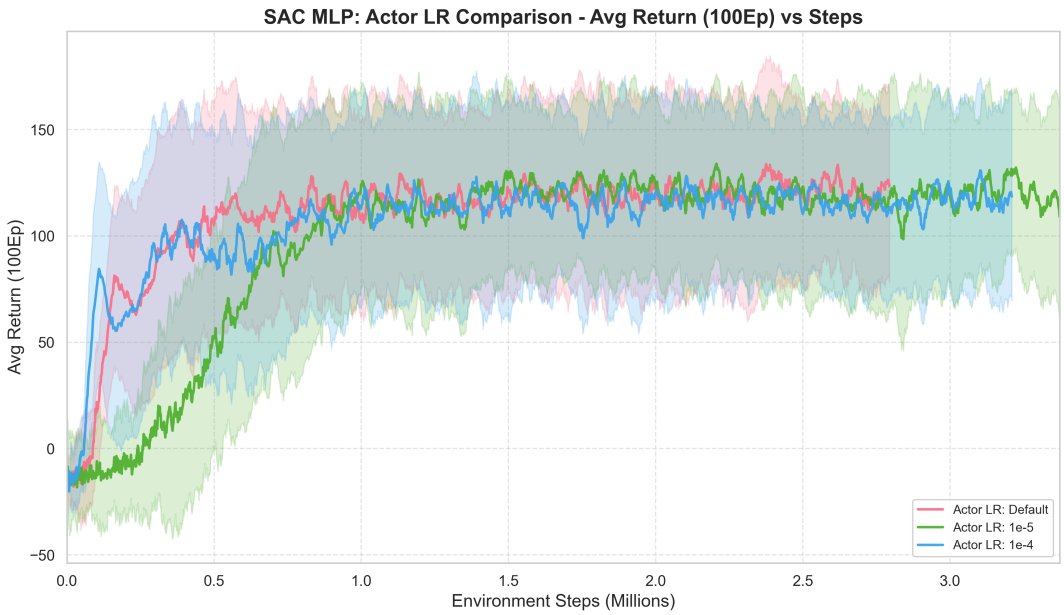


Figure 3: Oil Spill Mapping: SAC MLP Actor Learning Rate Comparison - Average Return vs. Environment Steps. This illustrates hyperparameter sensitivity for SAC MLP (RQ4) and provides insights relevant to the impact of changes similar to those explored for PER.

Figure 3 demonstrates the sensitivity of SAC MLP to the actor learning rate.

- **Sensitivity to Hyperparameters (SAC MLP, RQ4)**: SAC MLP's performance is sensitive to the actor learning rate. The default learning rate (pink, "Actor LR: Default") exhibits good performance, converging to a relatively high reward. Reducing the learning rate to 1e-5 (blue, "Actor LR: 1e-5") leads to slightly slower initial learning but converges to a similar, or potentially slightly higher and more stable, return compared to the default. A learning rate of 1e-4 (green, "Actor LR: 1e-4") results in significantly hampered learning progress, with performance plateauing at a much lower reward level. This example illustrates the importance of hyperparameter choices, which was observed across various agents and parameters, reinforcing that such tuning is crucial for achieving optimal performance.

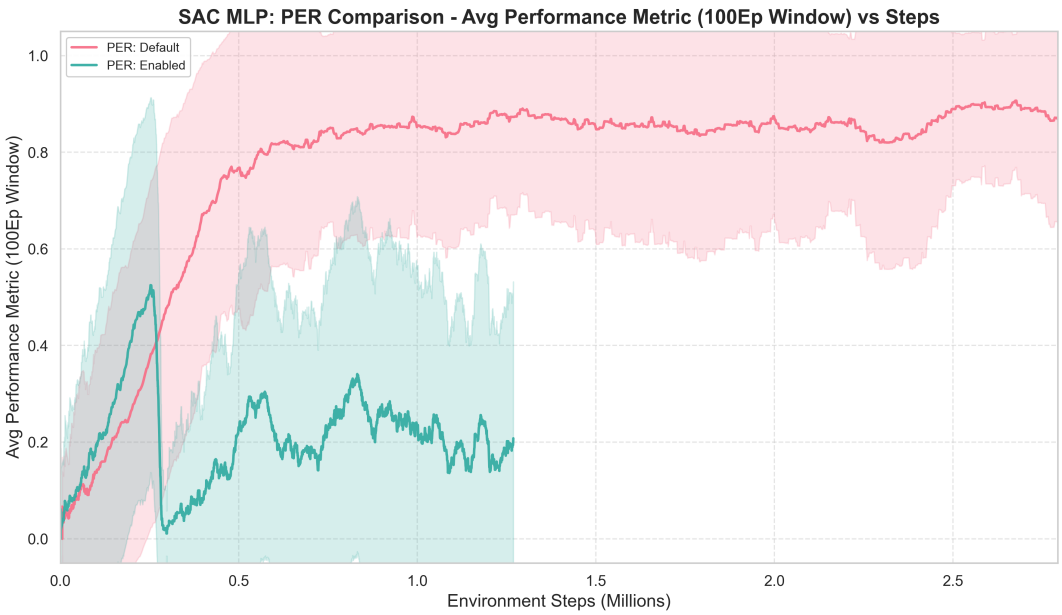


Figure 4: Oil Spill Mapping: SAC MLP - Prioritized Experience Replay (PER) vs. Default Comparison. Average Performance Metric (100Ep Window) vs. Environment Steps. This plot investigates the benefit of PER for the SAC MLP agent (RQ3).

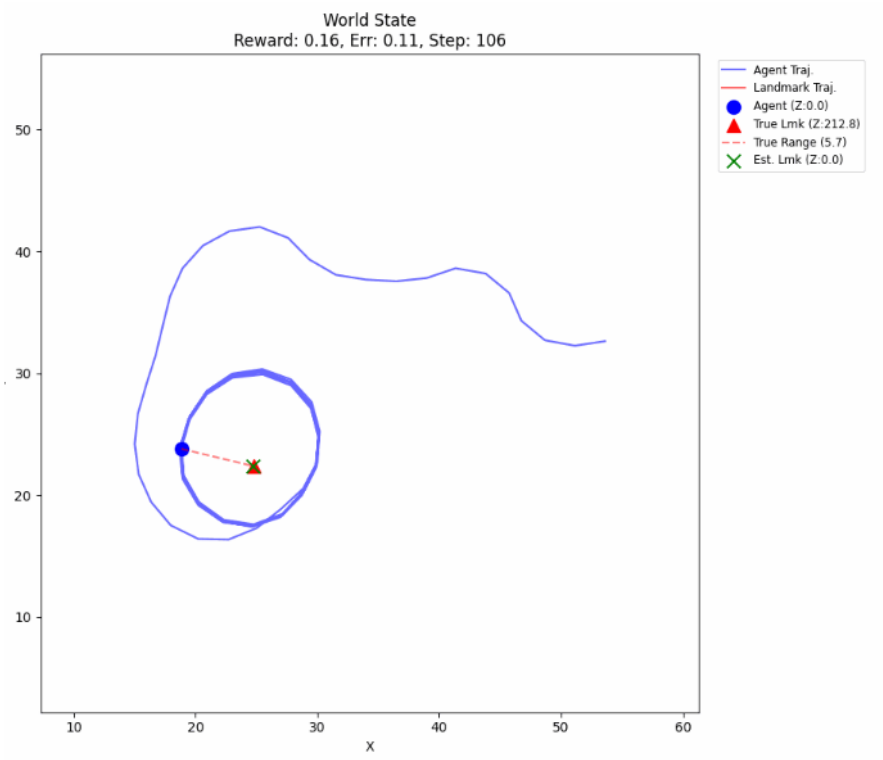
Figure 4 illustrates the effect of enabling Prioritized Experience Replay (PER) for the SAC MLP agent in the oil spill mapping environment, using the average performance metric (oil point inclusion percentage) over a 100-episode window.

- **Benefit of PER for SAC (RQ3)**: The SAC MLP agent with default experience replay (pink, "PER: Default") shows a consistent learning curve, achieving a high performance metric (around 0.8-0.9) and maintaining it. In contrast, the SAC MLP agent with PER enabled (teal, "PER: Enabled") initially learns and improves its performance up to approximately 0.4 million environment steps, reaching a metric of about 0.5. However, beyond this point, its performance dramatically degrades, falling to a low level (around 0.2) and failing to recover. This significant drop in performance for the PER-enabled agent, potentially indicative of issues such as catastrophic forgetting, was observed in this single experimental run. Due to only one run being performed for this configuration, this observed degradation cannot be conclusively linked solely to the use of PER without further investigation with multiple random seeds.

Overall, for the oil spill mapping task, the tested agent configurations (SAC MLP, PPO MLP, SAC RNN, PPO RNN without PER) demonstrate comparable efficacy in achieving effective mapping strategies, as indicated by their broadly similar reward accumulations and performance metrics. Definitive statements about the superiority of one specific architecture or algorithm (excluding the PER case) are difficult to make based on these single-run results. All agents are sensitive to hyperparameter choices, especially learning rates, as illustrated by the SAC MLP example. The preliminary result for PER suggests it might be detrimental in this context, but, as noted, further investigation is required.

4 Project 2: Autonomous Underwater Landmark Tracking

This project, developed in its own separate repository, focuses on training an agent to autonomously navigate and improve its estimate of a landmark's position in a 2D environment using



noisy range-only measurements. This work is directly inspired by the challenges and methodologies for range-only underwater target localization presented by Masmitja et al. (2023) [3], whose original research and codebase can be found at `github.com/imasmitja/RLforUTracking`. Our project aims to replicate and potentially extend their approach using deep reinforcement learning for path planning.

4.1 Environment: Landmark Tracking World (`world.py`)

The environment (from the Landmark Tracking project's repository) simulates an agent (e.g., an ASV) tracking a landmark (e.g., a benthic rover or tagged animal) using range-only measurements. `CORE_STATE_DIM` is 9, `CORE_ACTION_DIM` is 1. The setup is analogous to the single-tracker, single-target scenario considered in [3].

4.1.1 Environment Setup and Dynamics

- World: Defined bounds (`world_x_bounds`, etc.), state normalization. Agent operates in 2D.
- Agent: Constant speed (`agent_speed` from configuration), randomized start. Kinematics model is simplified as in [3]. The agent operates at this constant speed, and the RL policy output dictates the change in yaw angle.
- Landmark: Target position (`q` in [3]) can be static or mobile, randomized start. Target depth is assumed known by the agent for planar range projection.
- Observation: Noisy range (`d_t` in [3]), modeled with `range_measurement_base_noise`, `range_measurement_distance_factor`, and `new_measurement_probability`.
- Estimator: `LeastSquaresConfig` (default) via `world_config.estimator_config`. The LS estimator mirrors the simple, computationally efficient method used in [3] for online position updates.

- Termination: `success_threshold` on estimation error, `collision_threshold`, `max_steps`.

4.1.2 State Representation

The state representation includes agent kinematics, estimated target position, and the current range measurement, analogous to the information available to the agent in [3]'s problem formulation (denoted as o_t in their paper, which includes $p_t, v_t, \tilde{d}_t, d_{pt}$).

- Basic State (9-dim): Agent pos/vel/hdg, est. landmark pos/depth, current range. All normalized.
- Trajectory History (length 10): Sequence of (normalized basic state, action, raw reward). Feature dim 11.

4.1.3 Action Space

- Continuous, 1-dim: Normalized yaw change $\Delta\psi$, scaled by `yaw_angle_range[1]`. The agent maintains a constant speed, and this action directly influences its heading for navigation.

4.1.4 Reward Engineering for Landmark Tracking

The reward function is designed to guide the agent to optimize its trajectory for accurate target localization, similar to the objective in [3]. It combines elements related to estimation accuracy and agent behavior.

- Estimation Error: Log-based reward for low error (e_q in [3]), scaled.
- Distance to Landmark: Small penalty for true distance ($\hat{d}$ in [3]'s reward r_d).
- Terminal Conditions: Implicit success bonus, collision penalty. (A terminal reward $r_{terminal}$ is also described in [3]).

4.2 Experimental Design and Research Questions (Landmark Tracking)

The `configs.py` file within the Landmark Tracking project's repository defines its specific set of base configurations and variations, tailored to explore the research questions below. The problem formalization, including the environment setup, agent model, and measurement model, draws from the scenario described in [3].

4.2.1 Research Questions (Landmark Tracking)

This experimental setup aims to answer:

1. **Algorithm Comparison**: How do SAC and PPO (MLP versions) compare in terms of tracking accuracy (estimation error), success rate, and sample efficiency when applied to the range-only localization problem similar to [3]?
2. **Impact of Recurrence**: Does using RNNs in SAC and PPO improve performance, especially for tracking potentially mobile landmarks or when dealing with sequences of noisy range measurements?
3. **Benefit of PER for SAC**: Does PER lead to faster convergence or better final tracking accuracy for the SAC MLP agent in this specific localization context?

4. **Sensitivity to Hyperparameters**: How robust are SAC MLP and PPO MLP agents to changes in their core hyperparameters for the tracking task?

5. **Effect of Observation Quality**: How significantly does sensor noise and measurement frequency impact the agent's ability to learn effective tracking policies, reflecting challenges in real-world underwater acoustic environments?

4.2.2 Base Configurations (Landmark Tracking)

The `CONFIGS` dictionary in the Landmark Tracking `configs.py` includes configurations designed to address these research questions:

- `"default"`: Serves as the baseline, an SAC MLP agent with standard parameters for the Landmark Tracking environment. The default estimator is Least Squares, similar to the approach used for target position estimation in [3].
- **Signal Quality Variations** (for RQ5): Based on the SAC MLP agent, these configurations explore scenarios with different levels of observation noise and measurement availability, which are critical factors in underwater acoustic localization.
  - `"default_poor_signal"`: Increased range measurement noise and reduced measurement probability.
  - `"default_good_signal"`: Reduced range measurement noise and increased measurement probability.
- **Algorithm and Architecture Variations** (for RQ1, RQ2, RQ3):
  - `"sac_rnn"`: SAC agent employing an RNN (LSTM by default, `sac.rnn_hidden_size = 128`). This explores if temporal information processing aids in path planning under noisy conditions.
  - `"sac_per"`: SAC MLP agent augmented with Prioritized Experience Replay.
  - `"ppo_mlp"`: PPO agent with an MLP architecture.
  - `"ppo_rnn"`: PPO agent employing an RNN (LSTM by default, `ppo.rnn_hidden_size = 128`).

4.2.3 Hyperparameter Variations (Landmark Tracking)

Variations for SAC MLP and PPO MLP agents are defined to assess sensitivity (RQ4).

SAC MLP Hyperparameter Variations: Based on `"default"`:

- Learning Rates (Actor/Critic), Discount Factor, Target Update Rate, Network Size, Initial Alpha.

PPO MLP Hyperparameter Variations: Based on `"ppo_mlp"`:

- Learning Rates (Actor/Critic), GAE Lambda, Policy Clip, Entropy Coefficient, Network Size, N Epochs.

4.3 Landmark Tracking Results

This section presents and analyzes selected results from the landmark tracking experiments. To maintain the brevity of this report, the focus is on comparing PPO and SAC performance and the impact of key architectural choices and environmental factors; a detailed account of all hyperparameter sweep outcomes (RQ4) is not provided, though sensitivity was observed and is exemplified. The primary metric discussed is the Average Absolute End-of-Episode Error, which reflects the agent's tracking accuracy.

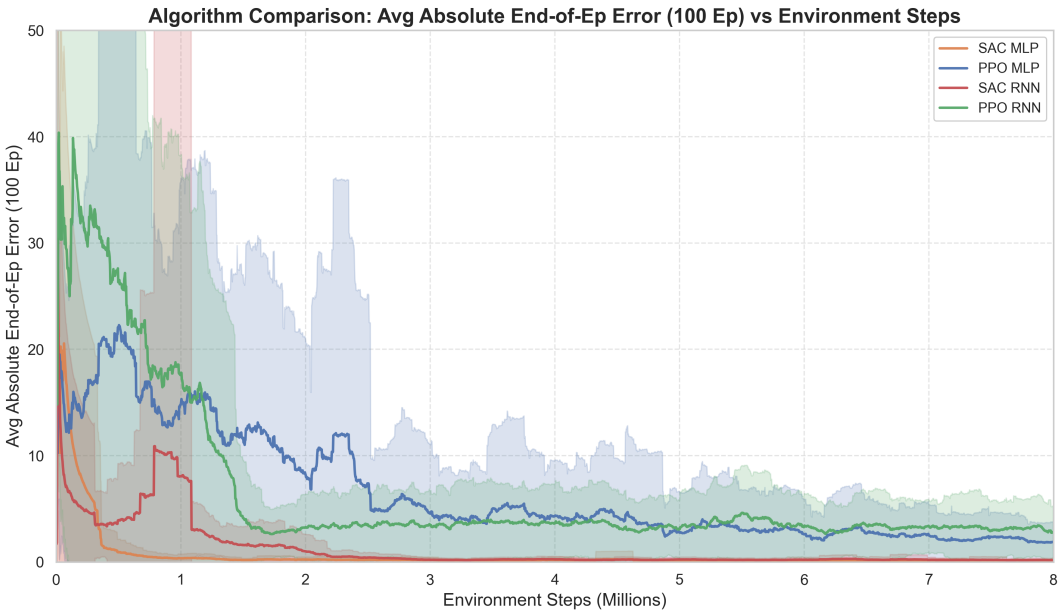


Figure 5: Landmark Tracking: Algorithm Comparison - Average Absolute End-of-Episode Error vs. Environment Steps for SAC MLP, PPO MLP, SAC RNN, and PPO RNN. This plot addresses the performance of different RL algorithms and the impact of recurrent architectures (RQ1, RQ2).

Figure 5 compares the learning curves of SAC and PPO agents, with both MLP and RNN (LSTM) architectures, for the landmark tracking task.

- **SAC MLP vs. PPO MLP (RQ1)**: SAC MLP (orange) demonstrates significantly superior performance compared to PPO MLP (blue). While still converging very slowly after 8 million steps, this plot shows how PPO is, by design, less sample efficient than SAC.
- **Impact of Recurrence (RQ2)**: The use of RNNs shows similar results for both SAC and PPO, both are able to converge similarly with and without the use of Recurrent Networks. Indicating that for this environment they are not needed.

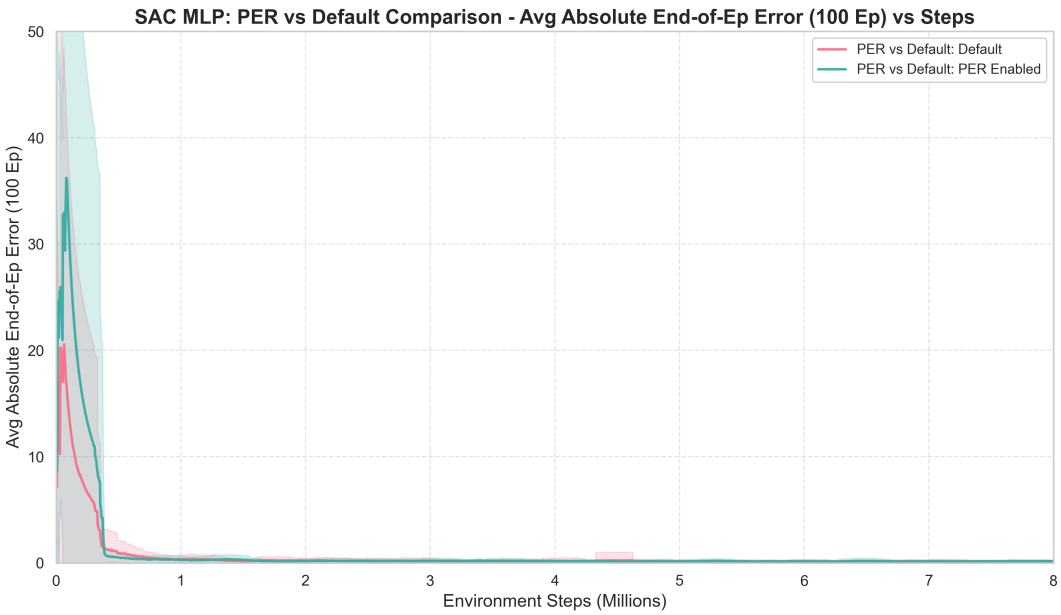


Figure 6: Landmark Tracking: SAC MLP - Prioritized Experience Replay (PER) vs. Default Comparison. Average Absolute End-of-Episode Error vs. Environment Steps. This plot investigates the benefit of PER for the SAC MLP agent (RQ3).

Figure 6 evaluates the impact of Prioritized Experience Replay (PER) on the SAC MLP agent's performance in landmark tracking.

- **Benefit of PER for SAC (RQ3)**: The results show that enabling PER (teal) provides no significant advantage over the default SAC MLP agent (pink) using uniform replay. While there might be a marginal initial speed-up with PER in the very early phase of training (before 0.5 million steps), both configurations converge to similar error levels with comparable learning speeds and final performance. This suggests that for this specific landmark tracking problem and SAC MLP setup, standard experience replay is sufficient, and PER does not unlock substantial gains.

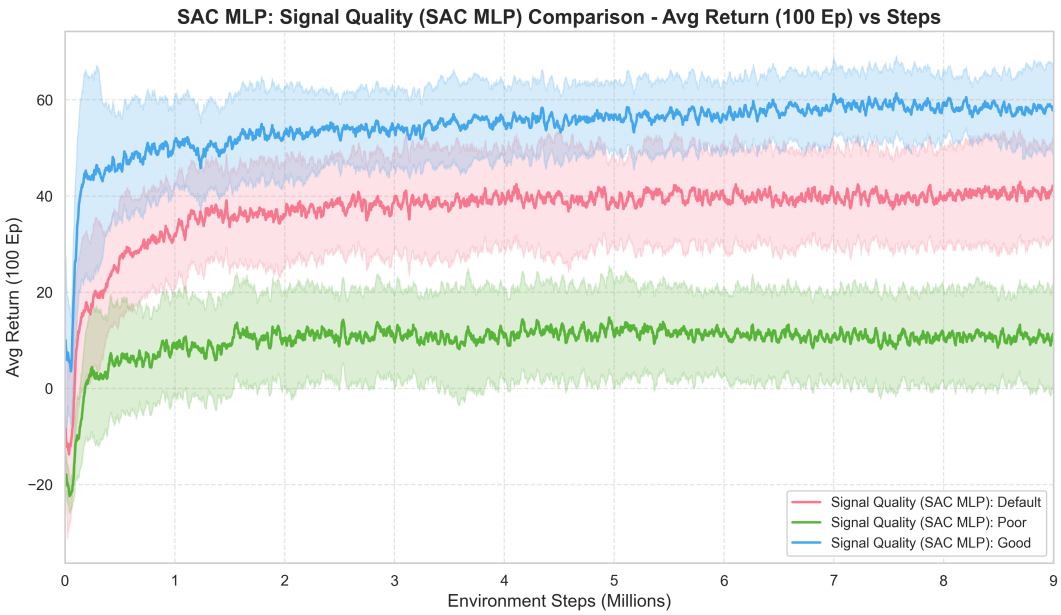


Figure 7: Landmark Tracking: SAC MLP - Signal Quality Comparison. Average Return vs. Environment Steps under Good, Default, and Poor signal conditions. This plot addresses the effect of observation quality (RQ5).

Figure 7 illustrates the crucial role of observation quality on the SAC MLP agent's learning and achieved rewards.

- **Effect of Observation Quality (RQ5)**: The signal quality, representing sensor noise and measurement frequency, profoundly affects performance as measured by accumulated reward. The "Good" signal quality (blue, "Signal Quality (SAC MLP): Good") allows the agent to achieve the highest average returns (around 50-60 units). The "Default" signal quality (pink, "Signal Quality (SAC MLP): Default") results in intermediate average returns (around 30-40 units). Critically, "Poor" signal quality (green, "Signal Quality (SAC MLP): Poor") leads to the lowest average returns (around 10-20 units), with slower learning progress. This underscores that better sensory input (lower noise, higher measurement frequency) enables the agent to learn policies that yield higher rewards, likely corresponding to more effective tracking behavior and thus lower tracking error, although error is not directly plotted here.

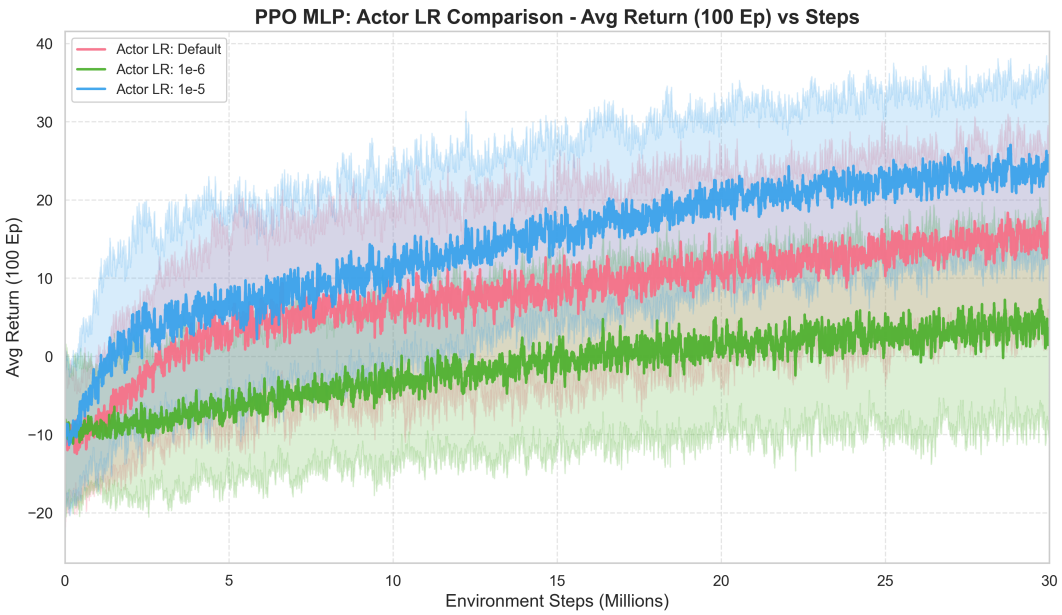


Figure 8: Landmark Tracking: PPO MLP - Actor Learning Rate Comparison. Average Return vs. Environment Steps for different actor learning rates. This plot provides an example of hyperparameter sensitivity (RQ4).

Figure 8 demonstrates the sensitivity of the PPO MLP agent to a key hyperparameter, the actor learning rate, in the landmark tracking context, measured by average return.

- **Sensitivity to Hyperparameters (RQ4)**: The choice of actor learning rate significantly impacts PPO MLP's performance in terms of accumulated reward. A very low learning rate (1e-6, green, "Actor LR: 1e-6") results in extremely slow learning and achieves very low rewards (plateauing near 0-5 units). The default learning rate (pink, "Actor LR: Default") leads to faster initial improvement but plateaus at a moderate reward level (around 15-20 units) with some instability. An intermediate learning rate (1e-5, blue, "Actor LR: 1e-5") strikes the best balance, enabling stable convergence to the highest average reward (around 30-35 units) among the tested LRs for PPO MLP. This highlights the importance of careful hyperparameter tuning for achieving optimal performance with PPO. Similar sensitivity was observed for other hyperparameters across both SAC and PPO agents.

In summary, the landmark tracking experiments indicate that SAC (especially SAC RNN) and PPO MLP (with tuned hyperparameters) can effectively learn policies for this active perception task. Recurrence can be highly beneficial (as seen with SAC RNN achieving the lowest error) but is not universally advantageous (PPO RNN performed poorly). PER did not offer significant gains for SAC MLP in terms of final tracking error. Critically, performance (both in terms of tracking error and accumulated reward) is heavily dependent on observation quality and careful hyperparameter selection. These findings provide valuable insights for deploying RL agents in challenging underwater localization scenarios.

5 Qualitative Results

Analyzing the evaluation videos generated for the oil spill and tracker projects, both seem to perform according to what is expected given their reward functions. The qualitative behavior of the trained models can be further observed through the interactive web applications and project repositories previously detailed in the Introduction.

In the case of the tracker, the agent usually does not perform perfect circles around the landmark, a strategy identified as optimal for maintaining observability in range-only underwater target localization by Masmitja et al. (2023) [3], even though the estimated landmark position appears to be very close to the real one. This behavior is likely due to the specific modelization of signal noise and/or signal loss within the simulation.

As for the oil spill mapper, there is a slight bias observed in the agent, which tends to move towards the opposite end of the world whenever it starts at a location close to the corners. This is most definitely due to the logic for the initialization. In order to generate consistently hard episodes, the initial random locations of the oil spill and the agent were constrained to be separated by a distance defined as a hyperparameter. This results in the oil spill often being generated on the opposite side of the world whenever the agent initializes in a corner, potentially influencing the agent's initial exploratory pattern.

6 Future Work

6.1 Oil Spill Mapping

Several avenues exist for extending the capabilities of the oil spill mapping agent:

- **Multiple Concurrent Spills**: Adapt the system to detect and map more than one oil spill simultaneously within the environment.
- **Non-Convex Spill Shapes**: Enhance the mapping algorithms to accurately represent oil spills with non-convex geometries, moving beyond simple convex hull estimations.
- **Multi-Agent Mapping**: Investigate the use of multiple collaborating agents for mapping tasks. This could involve strategies for decentralized data fusion or using clustering techniques to consolidate points detected by different agents into a unified map.

6.2 Landmark Tracker

For the landmark tracking project, future work could focus on:

- **Tracking Moving Landmarks**: Train and evaluate the agent's ability to track landmarks that are not static but exhibit their own motion patterns.
- **Particle Filter Integration**: Study the effects of using the implemented Particle Filter instead of the Least Squares method evaluated here.
- **Signal Noise/Loss Study**: Conduct a detailed study on the effects of signal noise modeling and signal loss probability. This involves analyzing how different noise characteristics and frequencies of measurement availability impact the agent's learned policies and tracking performance, building on the observation from Section 5.

6.3 General Framework Enhancements

In general, to improve reusability and facilitate further research:

- **Shared RL Algorithm Package**: Adapt the implementations of the RL algorithms (SAC, PPO, etc.) into a common, reusable package. This would streamline the application of these algorithms to new problems and ensure consistency across projects.

7 Conclusion

This project applied Reinforcement Learning, primarily comparing SAC and PPO, to two autonomous system challenges: oil spill mapping and underwater landmark tracking.

For the **oil spill mapping project**, SAC and PPO variants (MLP/RNN, excluding PER) demonstrated broadly comparable learning and final performance in mapping effectiveness, as measured by reward accumulation and performance metrics. However, all agents showed sensitivity to hyperparameter tuning. A single run using SAC MLP with Prioritized Experience Replay (PER) exhibited a significant performance degradation after initial learning, potentially due to catastrophic forgetting; as this was a single run, this degradation cannot be conclusively linked solely to PER, and more runs are needed for definitive conclusions. Qualitatively, the mapping agent performed as expected given the defined reward function, although sometimes showed a bias in initial exploration paths due to the episode initialization logic.

For the **autonomous underwater landmark tracking project**, SAC agents, generally outperformed PPO in terms of achieving lower final landmark estimation errors. Performance for all agents was critically dependent on observation quality (signal noise and frequency) and hyperparameter settings. PER did not offer significant gains for SAC MLP in this task. Qualitatively, even though the tracking agent did not always adopt theoretically optimal circular paths, its behavior shows good convergence, with policies that result most of the time in low estimation error.

Overall, both SAC and PPO proved capable of learning complex behaviors in these distinct environments, with their relative efficacy and the impact of architectural choices varying by task. The findings underscore the importance of task-specific tuning and robust experimentation, noting that hyperparameter analysis was kept brief in this report for conciseness.

References

References

[1] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv preprint arXiv:1801.01290*.

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

[3] Masmitja, I., Martin, M., Katija, K., Gomariz, S., & Navarro, J. (2023). A reinforcement learning path planning approach for range-only underwater target localization with autonomous vehicles. *arXiv preprint arXiv:2301.06863*.