Pyquaticus: A Sim-to-Real Pipeline for Learning in Multi-Agent Maritime Strategy Games

Peter Crowley[1,2⋆], Brendan Long[2⋆], Andrew Schoer[2], Zachary Serlin[2], Makai Mann[2], Tyler Gonsalves[2], John Kliem[3], and Calin Belta[4]

[1] Boston University, Boston, MA 02215, USA,
[2] MIT Lincoln Laboratory, Lexington, MA 02421, USA,
[3] U.S. Naval Research Laboratory, Washington, DC 20375, USA,
[4] University of Maryland, College Park, MD 20742, USA

Abstract. While reinforcement learning (RL) has been used to solve complex strategic and adversarial games in simulation, its application to games played in real-world maritime environments is lacking. This can be partially attributed to the scarcity of learning-compatible maritime simulators with an easy-to-use pipeline to deploy learned policies on hardware. To address the lack of such a tool, we introduce Pyquaticus, a Python-based simulator and pipeline for training and deploying RL policies for multi-agent maritime strategy games. Pyquaticus conforms to RL Gymnasium standards, and handles communication with MOOS-IvP middleware to enable rapid training, testing, and deployment on MOOS-compliant vehicles. We demonstrate the Pyquaticus pipeline by deploying a learned policy for maritime capture-the-flag in simulation and on uncrewed surface vehicles.

Keywords: adversarial games · sim-to-real · marine robotics · maritime autonomy

1 Introduction

Reinforcement learning (RL) has shown a great ability to train agents to play strategic and adversarial games in simulated environments [11, 18, 1]. As marine robotics has improved, it is now possible to play these games in real-world maritime environments using uncrewed surface vehicles (USVs); one example of a complex, strategic, and adversarial game is multi-agent capture-the-flag [17].

⋆ Authors contributed equally.

However, the lack of an open-source, flexible, and easy-to-use sim-to-real pipeline makes it difficult to train and deploy RL policies on real maritime hardware. Past research analyzing common maritime simulators from an RL perspective [7] has found that some are able to provide high fidelity simulation and Gymnasium-like interfaces, but either did not have native support for sim-to-real pipelines using common robotic middleware [15, 3] or did have sim-to-real support at the expense of communication latency, making RL training more difficult [21]. Outside of the maritime field, DARPA's Alpha Dogfight [5] successfully transitioned policies from simulation [14] to physical aircraft [6], but did not establish a clear standard for sim-to-real pipelines that could be adapted for marine applications.

To facilitate advances in RL for the rapidly growing field of maritime autonomy, we introduce Pyquaticus, a flexible, modular, and scalable Python-based simulator and pipeline for training, testing, and deploying policies for multi-agent maritime strategy games. We validate the Pyquaticus pipeline with real-world maritime capture-the-flag (MCTF) experiments on USVs. The results of these experiments inform our approach to closing the sim-to-real gap through more accurate modeling of real-world dynamics and gameplay nuances, as well as the incorporation of observation history utilities.

2 Pyquaticus Simulator and Pipeline

Pyquaticus is a lightweight and flexible MCTF simulator with an easy-to-use pipeline for transitioning learned policies from simulation to real-world USVs. The MCTF playing field can be defined as a rectangle in Cartesian space, or alternatively can be anchored to a location on the world map with latitude and longitude bounds. In the latter case, obstacles are automatically generated based on topographical features from OpenStreetMap [12]. Additional obstacles can be specified by the user. To streamline RL integration, Pyquaticus implements the PettingZoo [19] parallel API, a Gymnasium-standard [20] interface for multi-agent RL. Example scripts for training an RL policy with RLLib [10] are included in the Pyquaticus GitHub repository[1]. Multiple observation space options are available, including agent-centric observations in local coordinate frames, lidar-like partial observations with semantic labels, and the global state in the world frame, all with built-in support for recording and returning histories. These observation settings are customizable with parameters such as lidar density and range, as well as history-specific options like length and sampling interval. The Pyquaticus environment supports both discrete and continuous action spaces. The discrete action space includes 17 actions, each representing a tuple of desired speed and relative bearing: one action with zero speed, and 16 actions generated by combining two speed levels (50% and 100% of an agent's maximum speed) with eight relative bearings spaced at 45° intervals, covering the range from $-135°$ to 180°. The continuous action space accepts any tuple consisting of a desired speed in the range $[0, \text{max speed}]$ m/s and a relative bearing in the range $[-180°, 180°]$.

[1] https://github.com/mit-ll-trusted-autonomy/pyquaticus

The Pyquaticus repository includes heuristic-based policies that function both as MCTF baselines, and opponents for RL agents to train against. These baseline policies are agnostic to the environment configuration and have defensive, offensive, and mixed behaviors at multiple difficulty levels. Wrappers are provided to use these baselines as opponents when training with RLLib. The repository also includes a waypoint-following policy that uses RRT* [8] for route planning in environments with obstacles. Users can create custom heuristic policies by extending the `BaseAgentPolicy` class. In addition to base policies, Pyquaticus includes several prebuilt agent dynamics models for simulating real-world maritime robots such as the Clearpath Heron and the SeaRobotics Surveyor. Users can implement custom agent dynamics by extending the base `Dynamics` class.

To enable rapid, distributed real-world deployment of policies learned in the Pyquaticus simulator, we developed `PyquaticusMoosBridge` (the bridge). An instance of the bridge runs on each robot's backseat computer, connecting a policy implemented in Python with MOOS-IvP middleware also running on the backseat computer. Specifically, the bridge uses PyMOOS [16] to subscribe and publish to the robot's MOOS Database (MOOSDB). The bridge registers for variables containing game state information in the MOOSDB and converts them into an observation in the Pyquaticus format. The policy uses this observation to choose an action, which is then published via the bridge to the robot's MOOSDB for execution. The `PyquaticusMoosBridge` class inherits from `PyQuaticusEnvBase`, a shared superclass of the full `PyQuaticusEnv` environment. `PyQuaticusEnvBase` includes common helper functions for handling observations and actions, and itself inherits from PettingZoo's `ParallelEnv` class. As a result, `PyquaticusMoosBridge` adheres to the PettingZoo API and shares the same core attributes as the full `PyQuaticusEnv`. Pyquaticus' consistent API usage and automated MOOS communication enable rapid deployment of Pyquaticus-trained policies on MOOS-compliant hardware. The Pyquaticus simulation environment and Pyquaticus-MOOS bridge create a sim-to-real pipeline that is easily adaptable to other multi-agent maritime strategy games beyond MCTF.

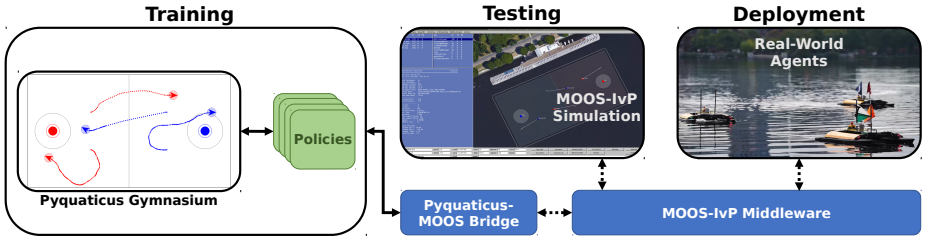


Fig. 1. The `PyquaticusMoosBridge` enables plug-and-play policy transfer from Pyquaticus to MOOS-IvP simulation and real-world MOOS-compatible USVs.

3 Experiments

To test our Pyquaticus pipeline, we first used the Pyquaticus PettingZoo environment to train a policy with Behavioral Cloning (BC) [13] for 2v2 MCTF. BC is an imitation learning algorithm that takes a set of human demonstrations of a task as input and uses supervised learning to train a neural network to model the demonstrator policy. As shown in Fig. 2, the MCTF playing field is set up such that a scrimmage line divides the field into 2 sides (one for each team). A flag is initialized on each side, and the goal of each team is to retrieve the opponent's flag while protecting the team's own flag from capture. The game is won by the team with the most flag captures at the end of a given amount of time. To grab an opponent's flag, an agent must come within a certain distance of the flag (the flag region). To defend its team's flag, an agent can tag an opponent when it comes within a certain distance of the opponent and both agents are on the tagging agent's own team's side. When tagged, an agent must navigate directly back to its team's flag region, and it cannot participate in other game behaviors until it arrives there. A tagging cooldown time is imposed so that an agent must wait a certain duration of time after executing a tag before it can tag again. In our experiments, we used a field size of 160m $\times$ 80m, a tagging and flag region radius of 10m, and a tagging cooldown time of 30s.

We used the Pyquaticus simulator to collect 50 demonstrations of 2v2 MCTF in which a human team played against a team of autonomous agents all running the `Heuristic_CTF_Agent` from the Pyquaticus repository's library of baselines. In these games, the agents were all set to the prebuilt Heron agent from Pyquaticus which models the dynamics of the Clearpath Robotics Heron USV and implements a speed and heading-error PID controller from MOOS-IvP software. After training on these demonstrations, the BC policy won 99/100 games against the `Heuristic_CTF_Agent` baseline in the Pyquaticus simulator. While we used BC as the learning paradigm in these experiments, the Pyquaticus pipeline is compatible with a wide variety of machine learning algorithms—particularly reinforcement learning—because it conforms to the Gymnasium-standard API.

We then deployed the learned and baseline policies in MOOS-IvP simulation, with each agent using an instance of the `PyquaticusMoosBridge` class to connect to its corresponding vehicle in the simulation. This allowed us to verify that the policies could properly send actions to and receive state information from MOOS-IvP middleware via the bridge, and therefore were deployable on hardware. Finally, we tested the policies against each other on a real-world MCTF testing platform on the Charles River, as shown in Fig. 2b and Fig. 2c. We ran a set of 2v2 MCTF games with Heron USVs which run MOOS-IvP and the `PyquaticusMoosBridge` locally on a backseat Raspberry Pi 4. Actions and commands sent from the bridge to the agent's MOOSDB are communicated via MOOS to the Heron's frontseat computer for execution on the vehicle. The vehicles use 2.4 GHz radios to communicate with a shoreside computer via a shoreside antenna. The shoreside computer processes GPS information from the vehicles, manages tag requests and flag grabs, and sends up-to-date game state information to each agent. We report results from these games in Table 1. For

other experiments using this pipeline, see [2, 4]. Further experiments are underway to validate RL policies trained in Pyquaticus on real-world USVs [9]. These tests involve 3v3 MCTF games using the SeaRobotics Surveyor vehicle, whose dynamics model is included in the Pyquaticus repository.

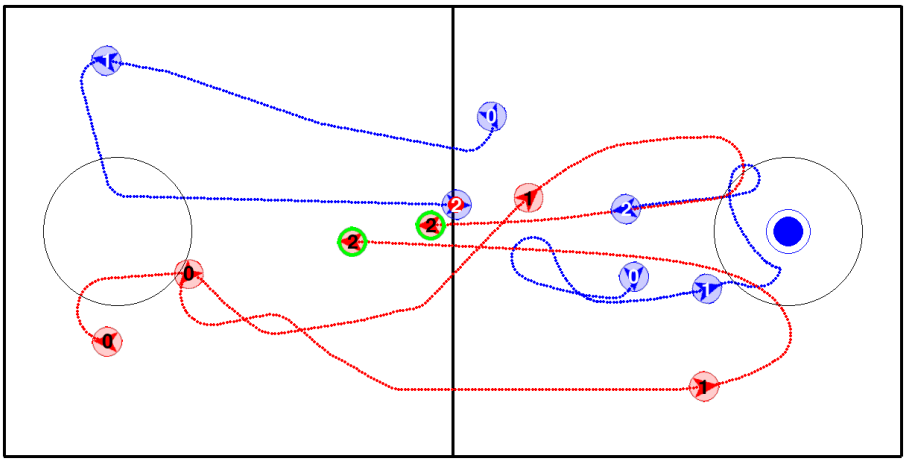
Table 1. Real-World MCTF Experimental Results. The game events are key actions performed by each team on the other (e.g., flag capture means the team captured the other team's flag). Starred numbers indicate the best performance in the event type for the particular game.

Game Number	**Game Duration** (seconds)	**Game Events**					
		Flag Captures		***Flag Grabs***		***Tags***	
		BC	Baseline	BC	Baseline	BC	Baseline
1	848	1	**2***	2	**4***	**6***	1
2	570	**2***	1	2	2	**5***	1
3	891	2	2	3	3	**6***	1

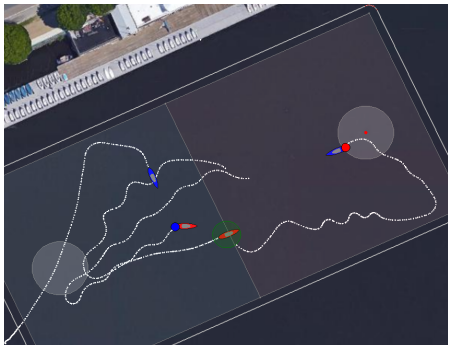
4 Main Experimental Insights

Sim-to-Real Gap: The strategy the BC policy learned from human demonstrations was to exploit the offensively aggressive nature of the heuristic baseline by luring both opponents onto offense and then trapping and tagging them with a single agent. Meanwhile, the second agent flanks the opponent's flag region and eventually grabs and captures the flag once the defenders are neutralized by its team member. As shown in Fig. 2, the Pyquaticus pipeline demonstrated effective sim-to-real transfer of these strategies, with each policy (BC and baseline) exhibiting comparable behavior between simulation and real-world deployments. As shown in table 1, the one win, one tie, one loss record of the BC policy on hardware demonstrates that it was able to compete with the heuristic baseline (red team), but was not as clearly superior as it was in simulation. We attribute the performance difference between simulation and hardware experiments to real-world disturbances, such as currents and wind, and mismatch between the simulated Heron model and the physical robot.

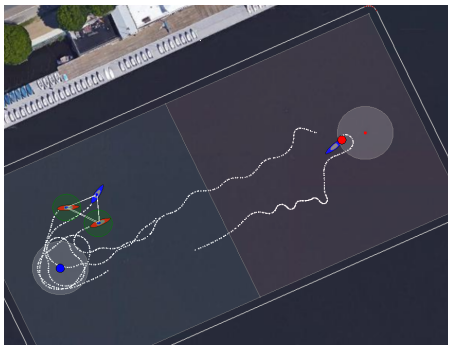
River Currents: On multiple occasions, we observed a BC agent getting pushed out-of-bounds by the current while driving along the edge of the field to go on offense. When an agent goes out-of-bounds, it is set to a tagged state and must return home before it can re-join the game. We suspect the BC agents (blue team) would have made more captures had this not been the case. When the demonstrations for BC were originally collected in Pyquaticus, the environment was configured such that agents would bounce off of the environment boundaries to keep them inside of the playing field. This meant that real-world agents spent more time in a tagged state for an out-of-bounds event than the simulated agents. To fix this inaccuracy in Pyquaticus for future experiments, we removed



(a) Policy execution in the Pyquaticus simulator



(b) Real-world policy execution 499 seconds into game 1



(c) Real-world policy execution 824 seconds into game 3

Fig. 2. Running the BC policy (blue team) against the heuristic baseline (red team) in the Pyquaticus simulator and on the real-world Charles River testing platform. In **(a)** each agent is shown at three different timesteps in the game. In **(a)**, **(b)**, and **(c)** the blue team has retrieved the red flag. In **(c)** the red team has retrieved the blue flag. A green ring or circle indicates a tagged agent.

the invisible walls keeping the agents in the playing field and added a drive-back-in-bounds behavior that is triggered when an agent goes out-of-bounds. To further address the drift problem, we have added the option to return history observations in Pyquaticus, which was not supported at the time of these experiments. Providing agents with historical game state information can better enable a learning agent to infer and adapt to disturbances. In the future, we plan to support both randomized and custom currents (modeled as vector fields) in

the Pyquaticus simulator, enabling agents to learn to adapt to these real-world disturbances.

Agent Dynamics: Another contributor to the sim-to-real gap that we discovered as a result of these experiments was a mismatch between the Pyquaticus Heron model and the real-world vehicle. As shown in Fig. 2b and Fig. 2c, the vehicles are able to achieve their desired paths, but the inefficient sinusoidal nature of the trajectories is indicative of chattering in the controller. While this may have been partially due to finite actuator bandwidth, it was largely caused by the BC policy oscillating between the extremes of its discretized action space in an attempt to maintain a desired heading. To address this issue for future experiments, we added support for the continuous action space, and updated Pyquaticus to more accurately model the Heron dynamics and PID controller by matching the latest versions of the MOOS-IvP `uSimMarine` and `pMarinePID` apps.

5 Conclusions and Future Work

The `PyquaticusMoosBridge` class was the key to our efficient deployment of a learned policy for MCTF on hardware. Specifically, it allowed us to test connection and communications with MOOS-IvP middleware in simulation ahead of time so that when we arrived at the testing site, we were able to successfully launch an experiment without the extensive period of hardware communications troubleshooting that is typical of robotics research. The plug-and-play nature of our Pyquaticus pipeline allows for rapid experimentation which in turn accelerates the process of closing the sim-to-real gap, and developing robust RL algorithms for multi-agent maritime strategy games.

In the future, we plan to further narrow the sim-to-real gap by enhancing simulation fidelity with features such as currents and communication limitations (e.g., delays and dropped messages). We also intend to decouple the MCTF game rules from the maritime simulator by providing a base PettingZoo maritime simulator class that can be wrapped for specific scenarios, with an MCTF wrapper included out of the box. This will enable flexible creation of new games and scenarios that share the common simulator and sim-to-real pipeline.

Acknowledgments. We thank Michael Benjamin for allowing us to use the Charles River testbed at the MIT Marine Autonomy Lab to run the hardware experiments included in this paper. Additionally, we thank Tyler Paine and Michael Benjamin for their help in running these experiments.

References

1. Arulkumaran, K., Cully, A., Togelius, J.: Alphastar: an evolutionary computation perspective. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '19, p. 314–315. ACM (2019). DOI 10.1145/3319619.3321894. URL http://dx.doi.org/10.1145/3319619.3321894

2. Beason, J., Novitzky, M., Kliem, J., Errico, T., Serlin, Z., Becker, K., Paine, T., Benjamin, M., Dasgupta, P., Crowley, P., et al.: Evaluating collaborative autonomy in opposed environments using maritime capture-the-flag competitions. arXiv preprint arXiv:2404.17038 (2024)
3. Cieślak, P.: Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface. In: OCEANS 2019 - Marseille, pp. 1–6 (2019). DOI 10.1109/OCEANSE.2019.8867434
4. Dasgupta, P., Kliem, J., Serlin, Z., Novitzky, M., Beason, J., Errico, T., Benjamin, M., Paine, T., Crowley, P., Lucas, S.M., Chao, J., Piotrowski, W.M., Dixit, G., Leprell, M., Richley, J., Tucker, C., Alamar, A., Ohto, K., Meo, J.: The first international maritime capture the flag competition: Lessons learned and future directions. In: The First MARW: Multi-Agent AI in the Real World Workshop at AAAI 2025 (2025). URL https://openreview.net/forum?id=4xlDyyY19L
5. DeMay, C.R., White, E.L., Dunham, W.D., Pino, J.A.: Alphadogfight trials: Bringing autonomy to air combat. In: Johns Hopkins APL Technical Digest, vol. 36 (2022). URL https://secwww.jhuapl.edu/techdigest/Content/techdigest/pdf/V36-N02/36-02-DeMay.pdf
6. Harper, J.: Pentagon takes ai dogfighting to next level in real-world flight tests against human f-16 pilot (2024). URL https://defensescoop.com/2024/04/17/darpa-ace-ai-dogfighting-flight-tests-f16/
7. Huang, Z., Buchholz, M., Grimaldi, M., Yu, H., Carlucho, I., Petillot, Y.: Urobench: Comparative analyses of underwater robotics simulators from reinforcement learning perspective. In: OCEANS 2024 - Singapore. IEEE, United States (2024). DOI 10.1109/oceans51537.2024.10682284
8. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. CoRR **abs/1105.1186** (2011). URL http://arxiv.org/abs/1105.1186
9. Kliem, J., Dasgupta, P., Crowley, P., Long, B., Cleaveland, M., Novitzky, M., Serlin, Z., Benjamin, M., Pottrill, E., Beason, J., Hubczenko, D.: Maritime capture the flag competition at aamas 2025 (2025). URL https://mctf2025.com
10. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., Stoica, I.: Ray rllib: A composable and scalable reinforcement learning library. CoRR **abs/1712.09381** (2017). URL http://arxiv.org/abs/1712.09381
11. 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 **518**(7540), 529–533 (2015)
12. Openstreetmap (2025). URL https://www.openstreetmap.org
13. Pomerleau, D.A.: Efficient training of artificial neural networks for autonomous navigation. Neural computation **3**(1), 88–97 (1991)
14. Pope, A.P., Ide, J.S., Mićović, D., Diaz, H., Twedt, J.C., Alcedo, K., Walker, T.T., Rosenbluth, D., Ritholtz, L., Javorsek, D.: Hierarchical reinforcement learning for air combat at darpa's alphadogfight trials. IEEE Transactions on Artificial Intelligence **4**(6), 1371–1385 (2023). DOI 10.1109/TAI.2022.3222143
15. Potokar, E., Lay, K., Norman, K., Benham, D., Ashford, S., Peirce, R., Neilsen, T.B., Kaess, M., Mangelson, J.G.: Holoocean: A full-featured marine robotics simulator for perception and autonomy. IEEE Journal of Oceanic Engineering **49**(4), 1322–1336 (2024). DOI 10.1109/JOE.2024.3410290
16. Pymoos (2022). URL https://github.com/russkel/python-moos
17. Robinette, P., Novitzky, M., Fitzgerald, C., Benjamin, M.R., Schmidt, H.: Exploring human-robot trust during teaming in a real-world testbed. In: 2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 592–593 (2019). DOI 10.1109/HRI.2019.8673134

18. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
19. Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N., Lokesh, Y.: Pettingzoo: Gym for multi-agent reinforcement learning. URL https://github.com/Farama-Foundation/PettingZoo
20. Towers, M., Kwiatkowski, A., Terry, J.K., Balis, J.U., de Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Tan, H.J.S., Younis, O.G.: Gymnasium: A standard interface for reinforcement learning environments. URL https://github.com/Farama-Foundation/Gymnasium
21. Zhang, M.M., Choi, W.S., Herman, J., Davis, D., Vogt, C., McCarrin, M., Vijay, Y., Dutia, D., Lew, W., Peters, S., Bingham, B.: Dave aquatic virtual environment: Toward a general underwater robotics simulator. In: 2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV), pp. 1–8 (2022). DOI 10.1109/AUV53081.2022.9965808