ROS2MLAgents: A Framework for Enabling Seamless Transfer of Multi-Robot Decision-Making from Simulation to Reality

John Rogers*, Sriram Siva*, and Maggie Wigness

***Abstract*— This paper presents a framework that integrates multi-robot systems (MRS) and reinforcement learning (RL) to address the challenges of real-world deployment. We provide a framework for integrating the Robot Operating System 2 (ROS2) with Unity's ML-Agents toolkit. By building on top of the ROS2 ecosystem, our framework can take advantage of the numerous sensor drivers, state estimation algorithms, motion planning libraries, and other low-level capabilities that have already been developed and field-tested within the ROS2 community. This avoids having to reinvent several core functionalities for robots and enables the training and deployment of reinforcement learning policies for complex tasks on real-world robotic platforms. The proposed framework aims to enable quick and effective integration of existing ROS2 robots and sensors, and facilitate the quick transfer of learned policies from simulation to real-world robotic systems, addressing the challenging sim-to-real gap. Furthermore, the paper discusses techniques for continual real-world training to further refine the learned behaviors. Finally, we also provide experimental demonstrations to establish that using ROS2's perception abstraction, we are able to bridge the sim-to-real gap more effectively compared to using raw sensory observations.**

I. Introduction

In recent years, the integration of Multi-Robot Systems (MRS) and Reinforcement Learning (RL) has emerged as a burgeoning field of research with significant implications across various domains. This synergy holds particular promise in areas such as disaster response scenarios [1], [2], autonomous farming [3], planetary exploration [4], and warehouse manufacturing [5]. However, these operations present compounding challenges that hinder the use of robots in such scenarios. For example, in the particular case of disaster response, critical situations demand efficient coordination among multiple agents to navigate hazardous environments, locate survivors, deliver aid, and execute rescue missions. With the potential to revolutionize how we approach several humanitarian assistance and disaster response scenarios, adaptable multi-robot systems equipped with RL capabilities offer a powerful toolkit for addressing the complex challenges inherent in these scenarios.

Despite the promise of multi-agent systems, they also introduce significant challenges. Coordinating the actions of multiple autonomous agents in dynamic and uncertain environments requires sophisticated decision-making capabilities. RL offers a principled approach to address these challenges by enabling agents to learn optimal policies through interaction with their environment. By formulating the problem as a Markov Decision Process (MDP), RL algorithms can learn to make sequential decisions that maximize long-term rewards, taking into account the actions of other agents and the stochastic nature of the environment.

However, one of the key challenges in deploying RL-based solutions in real-world scenarios is the sim-to-real gap. Typically, RL algorithms are trained in simulation environments to support efficiency and safety during learning. However, these learned policies from simulation may not always transfer effectively to the physical world due to differences in dynamics, sensor noise, and environmental conditions. Usually, sim environments deal with agents of negligible dynamics, whereas in the real-world, agents need to learn how to deal with their dynamics in planning tasks. Bridging this gap requires techniques for transferring learned policies from simulation to reality, adapting them to account for discrepancies between the two domains, and ensuring robust performance in diverse and unpredictable environments. Addressing the sim-to-real challenge is crucial to the practical applicability and scalability of RL-based approaches in real-world scenarios.

In this paper, we present the *ROS2MLAgents* framework that bridges Unity's ML-Agents [6] with ROS2 [7], and offers several advantages for deploying learnable robotic tasks readily to real-world scenarios. To begin with, our interface is simulator-agnostic and works with any simulator that can interface with ROS2. Leveraging the ROS2 ecosystem provides support for a wide range of sensors, robot platforms, and numerous motion planning, perception, mapping and other libraries that can be used to learn more complex behaviors while also providing tools for quick deployment on physical robots. Specifically, using these libraries ROS2 provides abstract representations, such as costmaps or semantic segmentation, that have a narrower gap between simulated and real-world inputs compared to raw sensor data. At a high level, our interface takes as input abstract perception representations generated from ROS2 using raw sensory data, processes it, estimates the rewards from these observations and robot states, and feeds it to the reinforcement learning Python brain provided by Unity's MLAgents. Additionally, our framework enables continual real-world training to fine-tune policies learned initially in simulation, further bridging the sim-to-real gap. The framework also supports parallel training of multiple agents across simulated and real environments for efficient multi-agent learning. Through this ROS2MLAgents

*Equally contributing authors.

†Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-23-2-0169.

John Rogers, Sriram Siva and Maggie Wigness are with the US Army DEVCOM Army Research Laboratory, Adelphi, MD 20783, USA. Email: john.g.rogers59.civ@army.mil, sivasriram2112@gmail.com, maggie.b.wigness.civ@army.mil

framework, we aim to enable the safe and effective transfer of learned multi-agent policies from simulation to real-world robotic systems. We also present implementation details and experimental results validating the framework's efficacy in bridging the sim-to-real gap for complex, multi-agent robotic applications.

II. Related Work

A. Multi-Agent Reinforcement Learning (MARL)

MARL has emerged as a promising approach to address the challenges of decision-making in complex multi-agent environments. MARL algorithms aim to develop policies that enable agents to learn and adapt their behavior through interactions with the environment and other agents, without relying on explicit models or assumptions about the underlying dynamics.

Seminal works in MARL include [8], [9]. These works have laid the foundation for various MARL techniques, including value-based methods that estimate the expected future reward (value function) for each possible state and action combination, which can then be used to derive an optimal policy. Examples of value-based methods include independent Q-learning [10], friend-or-foe Q-learning [11] and extensions to the multi-agent setting [12]. There are also policy gradient-based methods that directly optimize the policy parameters to maximize the expected return, without explicitly representing the value function. Such methods include counterfactual multi-agent policy gradients (COMAPG) [13] and actor-critic algorithms [14]. Finally, there is the class of model-based methods that learn a model of the environment dynamics, which can then be used for planning or simulating the effects of different actions. Examples include interactive particle environments [15] and Dreamer [16].

B. Frameworks and Toolkits

To facilitate research and experimentation in the field of multi-agent systems, several frameworks and toolkits have been developed. *OpenAI's AI Gym* [17] is a widely-used toolkit for developing and comparing RL algorithms. While it primarily focuses on single-agent environments, it also includes a limited set of multi-agent environments, such as particle environments and classic game theory problems. Similarly, *PettingZoo* [18] is a Python library built on top of AI Gym, specifically designed for MARL. It provides a diverse set of multi-agent environments, ranging from classic games to more complex scenarios, and includes tools for evaluating and comparing MARL algorithms. In the past few years, *Unity's MLAgents* [6] has come up as a comprehensive platform for training and evaluating intelligent agents in various scenarios, including multi-agents settings. It provides a rich 3D simulation environment, built-in support for popular deep RL algorithms, and tools for designing and customizing agent behaviors. Unlike AI Gym and PettingZoo, which are primarily focused on providing benchmark environments, ML-Agents offers a more holistic solution for developing and deploying multi-agent systems, and has a strong emphasis on 3D simulations and integration with game based environments.

Additionally, PyBullet [19] and AirSim [20] are also popular simulators used for multi-agent reinforcement learning research. PyBullet is a Python wrapper on the Bullet physics engine, providing a robotics simulation environment for research and education. AirSim, on the other hand, is a high-fidelity simulator for drones and other vehicles, developed by Microsoft, and has been used for multi-agent reinforcement learning experiments.

C. Sim-to-Real Transfer

While simulations provide a controlled and safe environment for developing and testing multi-agent systems, transferring the learned policies to real-world scenarios remains a significant challenge. The sim-to-real gap, where simulated environments may not accurately capture the complexities of the real world, can lead to performance degradation or failure when deploying trained agents in real-world scenarios.

Methods have explored various techniques to address the sim-to-real transfer problem, such as domain randomization to randomize various aspects of the simulated environment like texture or lighting conditions [21]. Another method is adversarial training where perturbations or noise are injected into an agent's observations and actions during training [22]. One of the seminal works on physical robots involves transferring learned quadruped walking policies from Sim to Real [23]. Transfer learning has been used to leverage knowledge gained from simulated environments to initialize or guide the learning the process in real-world settings (i.e., fine-tuning pretrained models) [24], [25]. Hybrid approaches that combine elements of simulated worlds and real-world data have been introduced to refine policies learned in simulation [26].

III. Approach

In this section, we introduce the *ROS2MLAgents* interface, describe our framework for learning in a ROS2-based system in both simulation and on hardware, and finally explain how the trained policy from simulation is transferred to a physical platform.

A. The ROS2MLAgents Interface

The Unity ML-Agents toolkit [6] provides a framework for training intelligent agents within the Unity simulation environment. However, it is primarily designed to operate within the Unity ecosystem, making it challenging to integrate with other simulation platforms or real-world robotic systems. To address this limitation, we have developed the *ROS2MLAgents* interface, which serves as a bridge between the Unity ML-Agents Python brain and ROS2 [7].

ROS2 is a widely-adopted middleware framework for robotics applications, providing a modular architecture for managing sensors, actuators, and control algorithms across various hardware platforms. Our interface leverages the Protocol Buffers (protobuf) [27] message definitions used

by the Unity ML-Agents toolkit to communicate with the Python brain. By implementing a ROS2 node that understands and translates these protobuf messages, we decouple the ML-Agents brain from the Unity simulation environment, enabling seamless integration with other simulation platforms such as Unreal, IssacSim, Gazebo, or real-world robotic systems that support ROS2.

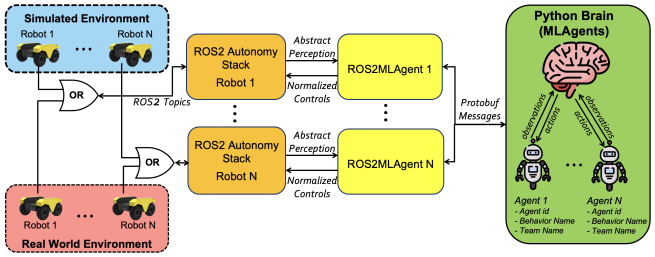


Fig. 1. A block diagram of the modules referred to in this paper. From left to right: for each agent, data is transported via ROS2 messaging from either a simulator (upper) or a physical robot (lower) to an autonomy stack of ROS2 nodes which produce higher level representations (costmaps and inter-robot range/bearing measurements in this paper). These representations are consumed by ROS2MLAgent nodes which communicate to the MLAgents Python brain via its protobuf interface. Control outputs are processed via the same pipeline from right to left.

Our ROS2MLAgents interface acts as a two-way communication channel. In one direction, it receives sensor data and observations, including camera images, GPS, range sensors, and LiDAR data, from the simulator or physical robot. In traditional RL research, such rich and high-dimensional sensor data are rarely used, as most benchmark environments rely on simplistic observations like low-dimensional state vectors or pixel-based renderings. However, in real-world robotic applications, robots are equipped with diverse sensors that provide complex data streams necessary to support safe and effective operation. Working with these complex sensor modalities presents challenges for RL algorithms. First, the high-dimensional nature of sensor data can overwhelm traditional deep learning architectures, leading to computational inefficiencies and scalability issues. Second, these sensing modalities often have different data distributions in simulation compared to real-world scenarios, making it difficult to directly transfer learned policies.

To address these challenges, our ROS2MLAgents interface incorporates a modular and extensible preprocessing pipeline that leverages the capabilities of ROS2 to process raw sensor data and extract normalized abstract representations with minimal distribution differences between simulation and real-world settings. For example, LiDAR point clouds can be converted into occupancy grid maps or elevation maps. These abstract representations are then encapsulated into protobuf messages that can be consumed by the ML-Agents brain. In the other direction of the communication channel, ROS2MLAgents translates the brain's action commands into ROS2 messages that can be executed by the appropriate actuators or controllers within the simulation or on the physical robot. A visualization of this two-way communication channel is depicted in Fig. 1. Note that unlike most simulation-based learning libraries, our ROS2MLAgents interface is capable of handling asynchronous sensory updates.

B. Training the Agent in Simulation

Building on the ROS2MlAgents interface, we now describe how to perform training of robot agents in simulation running ROS2.

1) Defining Agents and Teams: In the ROS2MLAgents framework, each agent is defined by its *id*, *behavior name* and *team name*. This allows for flexible configuration of both single- and multi-agent scenarios, where agents can be grouped into different teams with distinct objectives and policies. For example, multiple agents within the same team with unique IDs to learn the same behaviors, reducing training time. Additionally, we can have multiple agents in different teams to learn cooperative or adversarial behaviors. When defining agents, these behavior and team names are specified in the Agent Configuration.

2) Defining Model and Trainers: We can further simplify the process of defining the neural network architectures and training algorithms by leveraging a configuration-driven approach. Specifically, network architectures and hyperparameters for each agent behavior can be defined in a YAML configuration file. This allows for easy experimentation and modification of the neural network structure without the need to directly edit the code. The configuration file can specify parameters such as the input dimensions (e.g., size of observation space), network layers (e.g., number and size of fully connected or convolutional layers), Activation functions, normalization techniques, regularization methods, and other parameters. By centralizing these network definitions in a configuration file, the ROS2MLAgents framework enables rapid prototyping and exploration of different model architectures to find the most suitable one.

3) Trainer Configuration: Similarly, the training algorithm and its associated hyperparameters can also be specified in the YAML configuration file. The ROS2MLAgents framework supports a variety of reinforcement learning algorithms, including Proximal Policy Optimization (PPO) [28], Soft Actor-Critic (SAC) [14], and POsthumous Credit Assignment (POCA) [29]. The choice of the appropriate training algorithm and its hyperparameters (e.g., learning rate, batch size, discount factor) can have a significant impact on the performance and stability of the learned policies. By defining these parameters in the configuration file, users can easily experiment with different training approaches and find the most suitable one for their specific application. This configuration-driven approach to defining the network architecture and training algorithms greatly simplifies the development and deployment of reinforcement learning-based control policies in the ROS2MLAgents framework.

4) Saving Learned Models: During training in the simulation environment, the learned models for each agent behavior are periodically saved as checkpoint files as the rewards increase. These checkpoint files contain the weights and parameters of the neural networks that represent the agent's policy and value function. The ROS2MLAgents interface

provides functionality to save these checkpoint files, which can then be used to initialize the agents on the physical robot platform.

C. Sim-to-Real Transfer via ROS2MLAgents

Building on the ROS2MLAgents interface, we can now describe how to implement simulation-to-real-world transfer of learned policies with little to no additional setup overhead. There are two different approaches for transferring the trained model on real robots using the ROS2MLAgents framework:

- **Fine Tuning on Hardware**: In this method, the agents are initialized with the checkpoint files saved from the simulation training. This allows the agents to start with a good initial policy and continue learning in the real-world environment. The exploration of diverse policies is still enabled, allowing the agents to improve their expected rewards.
- **Inference Mode**: Alternatively, the agents can be run in a pure inference mode, without any additional training. In this case, the agents simply execute the policy learned during the simulation training, without exploring and modifying the model parameters. This approach can be useful when the simulation environment closely matches the real-world conditions, and no further fine-tuning is required.

The choice between these two methods depends on the specific application, the perception abstraction method used, and the available computational resources on the physical robot platform. The ROS2MLAgents framework supports both approaches, allowing for seamless integration of simulation-trained policies onto real robotic systems.

IV. Experiments

In this section, we evaluate the effectiveness of our ROS2MLAgents interface in bridging the sim-to-real gap for a catch-up game scenario with mobile robot agents.

A. Experimental Setup

We conduct a series of experiments involving a catch-up game scenario with mobile robot agents. In this game, one robot (the *chaser*) attempts to minimize the distance (i.e., catch up) between itself and the other robot (the *target*). This catch-up game is represented as a Partially Observable Markov Game (POMG). Formally, at time t, each robot agent receives local robot-centric observations $\mathbf{o}^t$ (e.g., laserscans or costmaps) and generates output actions $\mathbf{a}^t$, which include the linear and angular velocity of the robot. The robots also have information about the other robot's location via the state observations $\mathbf{s}^t$.

Accordingly, the reward for the *target* at any time step t is computed as:

$$R^{t,(target)}(\mathbf{o}_i^t, \mathbf{s}^t) = r_{dist}^{t,(target)} + r_{align}^{t,(target)} \tag{1}$$

where $r_{dist}^{t,(target)}$ represents the reward associated with progressing away from the chaser as $r_{dist}^{t,(target)} = d^t - d^{t-1}$. Here, d^t denotes the Euclidean distance between *chaser* and *target* at time t. The term $r_{align}^{t,(target)}$ incentivizes the *target* to maintain its heading away from the *chaser* goal. Mathematically, $r_{align}^{t,(target)} = 0.5 + {}^{target}\phi_{chaser}$, with ${}^{target}\phi_{chaser} \in [-1, 1]$ denoting the noramlized bearing angle from *target* to the *chaser*.

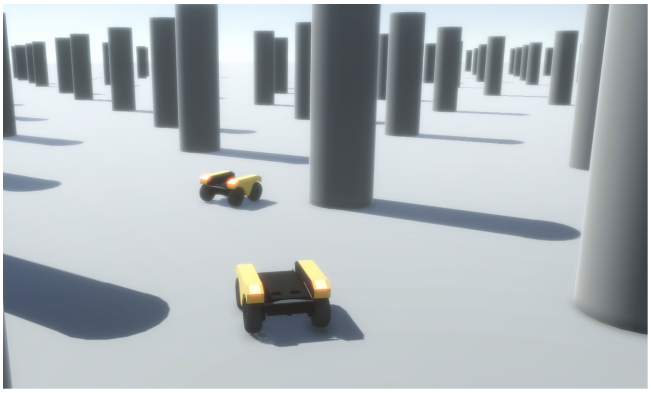


Fig. 2. The simulated environment with the clearpath warthog robots used to learn the catch-up game.

Similarly, the reward for the *chaser* at any time step t can be computed as:

$$R^{t,(chaser)}(\mathbf{o}_i^t, \mathbf{s}^t) = r_{dist}^{t,(chaser)} + r_{align}^{t,(chaser)} \tag{2}$$

where $r_{dist}^{t,(chaser)} = -(r_{dist}^{t,(target)})$ and $r_{align}^{t,(chaser)} = 0.5 - {}^{chaser}\phi_{target}$. In addition, both the *target* and *chaser* receive a very high negative reward ($\ll -1$) when they hit an obstacle when navigating.

In the catch-up game, the objective of both the agents is to maximize the expected return of rewards as:

$$\max \sum_i \mathbb{E}_{\mathbf{a}_i^{t,\pi_\theta^{agent}}} \left[\sum_{\tau=t}^{t+H} \gamma^{\tau-t} R_i^{\tau,agent} \right] \tag{3}$$

where γ is the discount factor that determines the importance placed on future rewards and a higher γ prioritizes long-term rewards, while a lower value emphasizes immediate rewards. Here, *agent* is either the *chaser* or the *target* and π_θ^{agent} represents the agent's policy with θ as its parameters. In Eq. 3, H represents the time horizon for which the robots should optimize their expected rewards.

Using our ROS2MLAgents framework, we first learn the policies for the agents in a simulated catch-up game scenario. We utilize Clearpath Warthog robots navigating a large open area with obstacles, as shown in Fig. 2. To evaluate the transferability of the learned policies, we also conduct experiments in a real-world setting using Clearpath Jackal robots. The catch-up game scenario is replicated in a controlled indoor environment, as illustrated in Fig. 3. Note that the Jackal robots have different dynamics compared to the Warthog robots used for training in simulation. Additionally, the real-world environment includes concave obstacles and higher levels of sensor noise, which result in a different data distribution compared to the simulated environment.

We train and evaluate agent policies using two distinct observation modalities:

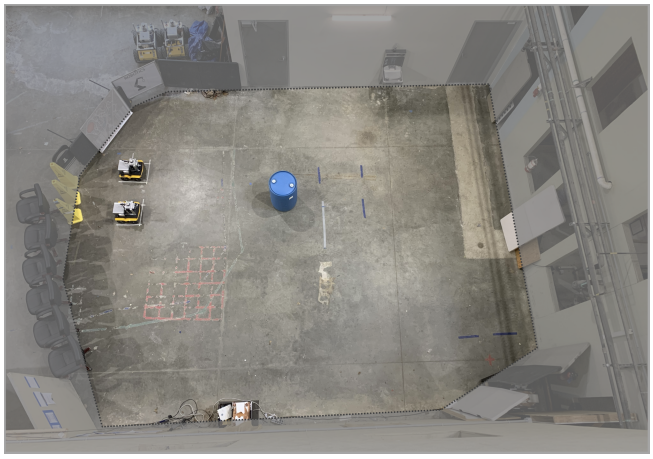
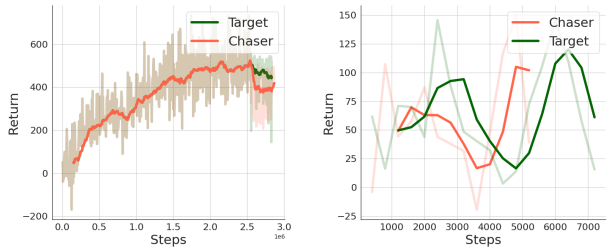


Fig. 3. The indoor arena used for evaluating the catch-up game.

- *Raw Laser Scan Observations:* In this setup, the agent receives the raw laser scan data from the robot's LiDAR sensor as input. The high-dimensional laser scan data was processed and fed directly into the RL algorithm.
- *Costmap Observations:* Instead of raw sensor data, we leverage the ROS2 perception abstractions to preprocess the laser scan data and generate costmaps, which represent the local environment as an occupancy grid map. These costmap observations serve as input to the RL algorithm, providing a more abstract and compact representation of the environment.

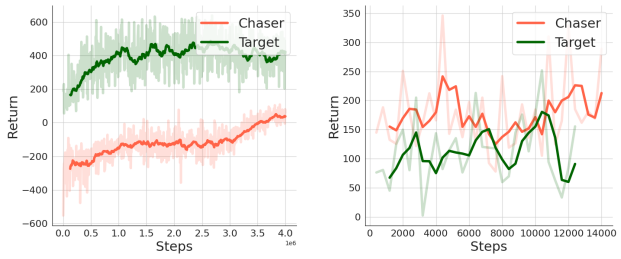
B. Results on Game with Raw Laser Scan Observation

In this scenario, both the chaser and target agents received high-dimensional laser scan data directly from the robot's LiDAR sensor as observations without any intermediate processing or abstraction. To train the agents, we employed a curriculum learning approach, where we first trained a common maneuver policy using a navigation reward function. This policy aimed to teach the agents basic navigation skills, such as avoiding obstacles and reaching target locations. Once the common maneuver policy converged, as shown in Fig. 4(a) (convergence of expected reward/episode length graph), we fine-tuned separate policies for the *chaser* and *target* agents using a catch-up game reward function. At this point, in Fig. 4(a), we see the divergence of the expected rewards for the *chaser* and *target* agents as they specialized for their respective roles in the catch-up game scenario. The *chaser*



(a) Training in Simulation (b) Deployment in Indoor Arena

Fig. 4. Expected Rewards when Training with Raw Laser Scan Observations.



(a) Training in Simulation (b) Deployment in Indoor Arena

Fig. 5. Expected Rewards when Training with Costmap Based Observations.

agent learned to efficiently pursue and catch up to the *target* agent, while the *target* agent learned evasive maneuvers to avoid being caught.

To evaluate the transferability of the learned policies from simulation to the real world, we deployed the trained policies on the Clearpath Jackal robot platform in the indoor arena. Fig. 4(b) illustrates the rewards of the *target* and *chaser* in the hardware platform. For the 7000 steps ($\sim$ 700 seconds) the agents were executed, we observe that their rewards is significantly less as compared the the simulation environment. Mainly because, the indoor arena used for the real-world experiments was significantly more constrained than the open spaces in the simulation environment. The presence of walls and other obstacles limited the agents' ability to maneuver freely, reducing their overall performance. In addition, the real-world arena contained several concave obstacles, such as corners and edges, which were more difficult for the agents to navigate around compared to the convex obstacles in the simulation.

Also we observe that there is little difference between the rewards of *chaser* and *target* as the *target* agent was started closer to the *chaser* agent, reducing the initial distance between them. This made it more challenging for the *chaser* agent to catch up to the *target*, as the *target* had less distance to cover to reach the goal. The *chaser* agent in the real-world experiments was also started with a better initial alignment towards the *target*, compared to the simulation. This reduced the need for the *chaser* to perform complex maneuvering to orient itself towards the *target*, making the task slightly easier for the chaser.

C. Results on Game with Costmap Observations

In this set of experiments, we explored the use of costmap observations instead of raw laser scan data. The agents received a 2D costmap representation of the environment as their observations, which was processed through a Convolutional Neural Network (CNN) encoder to extract relevant features.

The cumulative rewards obtained from the agent in illustrated in 5. In this scenario, we do not use a curriculum based learning and observe the learning of the agents. We observe that the *target* learns fast to navigate away from the *chaser*, while the *chaser* slowly improves its rewards.

When deploying the trained policies on the Clearpath Jackal robot in the indoor arena, we observed that the agents' performance in the real-world setting was much better

compared to the experiments with raw laser scan observations as seen in 5(b). Interestingly, the rewards achieved by the agents in the simulation environment were on par with the results from the raw laser scan observation experiments. This suggests that the costmap representation was able to capture the necessary information for the catch-up game task, without significant loss of performance compared to the higher-dimensional laser scan data. Furthermore, the costmap representation is less sensitive to sensor noise present in the raw laser scan data. This likely contributed to the better transferability of the learned policies from simulation to the real-world hardware platform, as the agents were able to generalize more effectively to the physical environment.

Overall, with the costmaps based perception abstraction, ros2mlagents is able to demonstrate the several benefits including better real-world performance as compared to the raw laser scan approach.

V. Conclusion

In this paper, we introduced the ROS2MLAgents framework, which offers seamless integration of Robot Operating System 2 (ROS2) with Unity's ML-Agents. Our framework enables integration of abstract perception derived from raw sensory data using ROS2, allowing agents to learn from environment-agnostic features. This allows our framework to run various reinforcement learning and imitation learning methods with any environment of choice supporting ROS2. In addition, our approach also enables learning against asynchronous data from different robot sensors and also provides seamless integration with physical robots, enabling the deployment of learned policies on real-world systems. Through our experiments, we have demonstrated the effectiveness of the ROS2MLAgents framework in enabling safe and efficient transfer of learned multi-agent policies from simulation to real-world robotic systems.

References

[1] X. Xiao, J. Dufek, T. Woodbury, and R. Murphy, "UAV Assisted USV Visual Navigation for Marine Mass Casualty Incident Response," in *IROS*, 2017.

[2] R. Murphy, J. Dufek, T. Sarmiento, G. Wilde, X. Xiao, J. Braun, L. Mullen, R. Smith, S. Allred, J. Adams *et al.*, "Two Case Studies and Gaps Analysis of Flood Assessment for Emergency Management with Small Unmanned Aerial Systems," in *SSRR*, 2016.

[3] A. Dutta, S. Roy, O. P. Kreidl, and L. Bölöni, "Multi-Robot Information Gathering for Precision Agriculture: Current State, Scope, and Challenges," *IEEE Access*, 2021.

[4] M. J. Schuster, M. G. Müller, S. G. Brunner, H. Lehner, P. Lehner, R. Sakagami, A. Dömel *et al.*, "The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration," *RAL*, vol. 5, no. 4, pp. 5315–5322, 2020.

[5] P. Gao, S. Siva, A. Micciche, and H. Zhang, "Collaborative scheduling with adaptation to failure for heterogeneous robot teams," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1414–1420.

[6] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[8] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Applied Intelligence*, vol. 53, no. 11, pp. 13 677–13 722, 2023.

[9] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[10] M. Tan, "Multi-agent reinforcement learning: independent versus cooperative agents," in *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, ser. ICML'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, p. 330–337.

[11] M. L. Littman, "Friend-or-foe q-learning in general-sum games," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 322–328.

[12] J. Serrino, M. Kleiman-Weiner, D. C. Parkes, and J. Tenenbaum, "Finding friend and foe in multi-agent games," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[13] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[14] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine, "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model," *Advances in Neural Information Processing Systems*, vol. 33, pp. 741–752, 2020.

[15] Y. Liu, Y. Hu, Y. Gao, Y. Chen, and C. Fan, "Value function transfer for deep multi-agent reinforcement learning based on n-step returns." in *IJCAI*. Macao, 2019, pp. 457–463.

[16] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg, "Daydreamer: World models for physical robot learning," in *Conference on Robot Learning*. PMLR, 2023, pp. 2226–2240.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[18] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 032–15 043, 2021.

[19] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[20] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[21] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[22] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell, "Adversarial policies: Attacking deep reinforcement learning," *arXiv preprint arXiv:1905.10615*, 2019.

[23] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *ArXiv*, vol. abs/1804.10332, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:13750177

[24] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[25] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[26] H. Niu, Y. Qiu, M. Li, G. Zhou, J. Hu, X. Zhan *et al.*, "When to trust your simulator: Dynamics-aware hybrid offline-and-online reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 36 599–36 612, 2022.

[27] C. Currier, "Protocol buffers," in *Mobile Forensics–The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices*. Springer, 2022, pp. 223–260.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[29] A. Cohen, E. Teng, V.-P. Berges, R.-P. Dong, H. Henry, M. Mattar, A. Zook, and S. Ganguly, "On the use and misuse of absorbing states in multi-agent reinforcement learning," *arXiv:2111.05992*, 2021.