Virtual Testing and Policy Deployment Framework for Autonomous Navigation of an Unmanned Ground Vehicle Using Reinforcement Learning

Tyrell Lewis, Patrick Benavidez and Mo Jamshidi
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249
Email: lewis.clay.ty@gmail.com, patrick.benavidez@utsa.edu, mojamshidi4@gmail.com

***Abstract*—The use of deep reinforcement learning (DRL) as a framework for training a mobile robot to perform optimal navigation in an unfamiliar environment is a suitable choice for implementing AI with real-time robotic systems. In this study, the environment and surrounding obstacles of an Ackermann-steered UGV are reconstructed into a virtual setting for training the UGV to centrally learn the optimal route (guidance actions to be taken at any given state) towards a desired goal position using Multi-Agent Virtual Exploration in Deep Q-Learning (MVEDQL) for various model configurations. The trained model policies are to be transferred to a physical vehicle and compared based on their individual effectiveness for performing autonomous waypoint navigation. Prior to incorporating the learned model with the physical UGV for testing, this paper outlines the development of a GUI application to provide an interface for remotely deploying the vehicle and a virtual reality framework reconstruction of the training environment to assist safely testing the system using the reinforcement learning model.**

***Index Terms*—Reinforcement Learning, Virtual Reality, Autonomous Car, Simulation, Multi-Agent Systems, Exploration**

I. Introduction

A. Reinforcement Learning for Robotic Systems

The use of deep reinforcement learning (DRL) as a framework for training a UGV to perform optimal navigation in an unfamiliar environment is a suitable choice for implementing AI with real-time robotic systems. Deep Q-Network (DQN) and similar variations are capable of approximating functions to provide generalized solutions for model-free Markov Decision Processes (MDP) and optimizing complex control problems by producing a learned optimal policy function to govern the behavior of an agent in an unfamiliar environment.

The state-action space of the robotic system is a representation of all possible "actions" (motor actuation commands, task execution) that a robot can take given any "state" which often describes the robot's position, orientation, and any other information considered pertaining to its surrounding environment. The classical approach to reinforcement learning, as it pertains to navigation, aims to discretize the space into a *grid-world* representation to allow for a fixed number of cell-based movement actions that can be feasibly implemented by a low-level controller. This technique can be extended to reduce the planning horizon of modern applications by decomposing complex tasks into *options*, or sub-tasks composed of elementary components. These options can be executed to help ensure safe exploration of the environment, a key necessity for reliable performance in the learning process [1], [2].

*This work was supported by Grant number FA8750-15-2-0116 from Air Force Research Laboratory and OSD through a contract with North Carolina Agricultural and Technical State University.

B. Reinforcement Learning Model Implementation

In this implementation, the environment and surrounding obstacles of an Ackermann-steered UGV are reconstructed into a virtual setting for training the UGV (represented as a bicycle model) to centrally learn the optimal route (guidance actions to be taken at any given state) towards a desired goal position using Multi-Agent Virtual Exploration in Deep Q-Learning (MVEDQL) [3], which has been shown to converge faster than standard Q-Learning techniques. Combining this method of learning with a physical UGV can be accomplished by making inferences of newly-seen obstacles by the UGV's distance / ranging sensors and incorporating them into the simulated environment for iterative learning (hence re-learning an optimal route given unfamiliar conditions). By designing a navigation controller with this "hands-off" approach and allowing the DRL model to perform the work, it is possible to modify the configuration of the physical system (a non-holonomic dynamical system) and have it autonomously navigate through unfamiliar environments without having to strictly modify the design of a controller and instead re-learn the optimal path based on state configuration changes.

The state of the virtual UGV $s(t)$, or "agent", at any given time t is represented by the following equations:

$$\phi = [\|\vec{d}\|_{scaled}, sin(\rho), cos(\rho), s_{-30^\circ}, s_{0^\circ}, s_{30^\circ}] \quad (1)$$

$$\vec{d} = (x_a - x_d)\hat{i} + (y_a - y_d)\hat{j} \quad (2)$$

Table I: Cases 1-3 aim to compare the effect of randomizing the agent initial position and destination. Cases 4-5 will test the performance of using inter-agent collisions during training.

Configuration Cases	**1**	**2**	**3**	**4**	**5**
Random Agent Position	T	F	T	T	T
Random Destination Position	T	F	F	F	T
Box Collision Model	T	T	T	T	T
Inter-agent Collision	F	F	F	T	T

$$\rho = \measuredangle \vec{d} - \omega \tag{3}$$

where $(x_a, y_a), (x_d, y_d)$ represents the position of the agent and destination respectively, ρ is defined as the angle offset between the vector directed towards the destination and the current position, ω is the agent's orientation relative to the world frame, and $s_{-30^\circ}, s_{0^\circ}, s_{30^\circ}$ are inverse scaled sensor values that indicate the presence of an obstacle as the agent approaches it. The actions a that can be taken consist of velocity and steering commands allowing movement towards a desired state position, and are notated as $a = \{(\upsilon, \psi); \upsilon \in \{0.2, 1.5\}, \psi \in \{0.0, 0.6, -0.6\}\}$.

II. Design Goals

A. Problem Overview

This project aims to develop a framework for efficient deployment of a trained deep reinforcement-learning (DRL) model that will optimally navigate an autonomous wheeled mobile robot to a user-provided waypoint goal location while avoiding collisions with various obstacles in the surrounding environment. The DRL model has been trained using reinforcement learning (Multi-Agent Virtual Exploration in Deep Q-learning, MVEDQL) in a custom Python simulation environment [3], [4] that can spawn multiple virtual agents representing the vehicle which perform reward-based learned-actions to optimally navigate to the goal location after spawning from a random position within the virtual environment during each iteration.

The results obtained from simulation demonstrated the fastest success using MVEDQL with 6 agents and a box collision model. The algorithm parameters affecting the *robustness* of the robot's performance are those that randomize the environment to elicit greater exploration and will likely have a significant affect on policy transfer-ability, or the effectiveness to which the physical robot takes actions to perform a task in real-time given a policy that was learned offline. These training parameters will be unique for several simulation model configurations (shown in Table I) to be compared during test evaluation.

The primary focus of this research is to compare the performance of implementing these pre-trained model configurations based on the shortest time taken to reach a destination position (if at all) for a fixed number of trials. The total trajectory length needed to reach the goal point for this time will also recorded, in addition to the percentage of successful trials that reach the destination without collision. The model

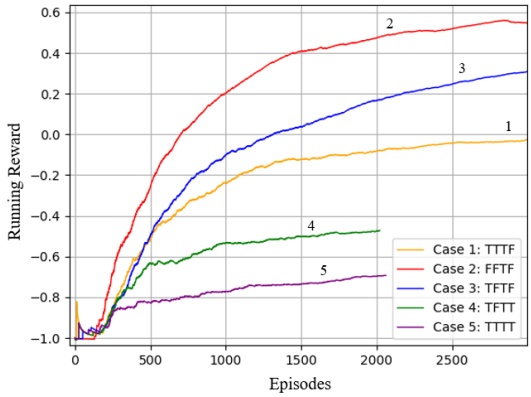

Figure 1: Reinforcement Learning Model Simulation Case Training Results

configuration parameters are all boolean-valued, and the list of configurations being tested is not exhaustive. The individual training performances are presented in Figure 1, where a fixed agent position, fixed destination, and no inter-agent collision learns the fastest for the early stages of training. While this model may perform best in an arena constructed to be identical to the one it used for training, it will likely suffer in practice when the environment has changed, as it is not adapted for exploration. However, this can be used to offer insight into the effectiveness of the framework developed in the following sections.

III. Background

A. Reinforcement Learning for Robotic Navigation Systems

Expanding upon the claim of using reinforcement learning as an ideal machine learning framework for training real-time robotic systems should be prefaced with a review of the challenging complications involved with these often highly-complex physical systems [5]. Firstly, training a DRL model requires repeated sampling from the robot's internal and external sensors so that an adequate representation of it's state-action space can be properly constructed without needing an unreasonable amount of exploration that will ultimately converge to a globally optimal solution.

For mobile navigation concerns, time-discretization of the actuation is challenging since neither the motors nor the vehicle can move instantaneously (an inherent time delay) which can distort the distance measurements collected between states and therefore delay the observable actions by multiple time steps. While this problem can be handled by placing some of the most recent actions into a buffer, this strategy increases the dimensionality of the problem. Instead, the duration of the time steps can be increased with the penalty of reducing the precision of the controller.

The internal sensors are used to monitor the states of the system that pertain to the position and orientation of the robot's joints and linkages. As the number of joints and linkages increase (as with modern robotic systems) so does the total

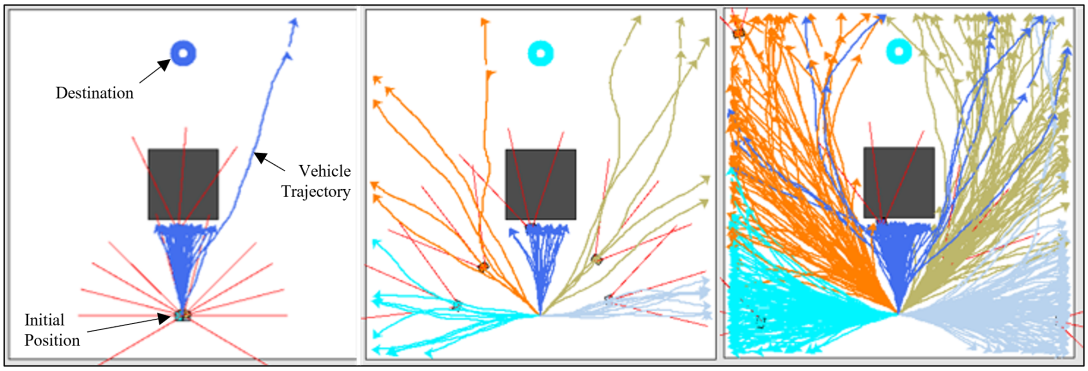

Figure 2: Training a Reinforcement Learning Model for Autonomous Waypoint Trajectory Generation Using Multi-Agent Virtual Exploration in Deep Q-Learning (MVEDQL) [3]

number of degrees of freedom (DoF) which must all be monitored. Over time, the dynamics of the system can change due to external effects such as wear and corrosion, affecting the robot's behavior and ultimately the learning process. This effect is more drastic for higher-velocity systems such as drones and UAVs.

External sensors are then used to monitor the environment surrounding the robot (e.g., nearby obstacles, landmarks, chemical concentrations, local wind velocity, ambient temperature, etc.) expanding the complexity of the overall system. Even with a limited quantity of sensors used to sample from the environment for a simple system, obtaining these samples can be very costly due to time-extensive task execution periods and any necessary interventions from human supervisors (maintenance, repairs, position resets, etc.). Furthermore, to efficiently implement an autonomous navigation algorithm in real-time, any delays in communication and actuation, filtering signal noise and task execution must be accounted for.

Increasing the number of sensors for systems of higher complexity not only demands attention to these challenges, but additionally introduces a new problem commonly referred to as the curse of dimensionality, where exponentially more data collection (and processing) is needed to cover the complete state-action space. Clearly, limitations on the amount of real-world interaction time along with sample-efficient algorithms that can learn from a small number of trials are essential, especially for non-episodic settings where movements cannot be paused while undergoing training and actions must be selected within a restricted time budget. A real-time architecture for model-based value-function reinforcement learning methods has been proposed by [6] where individual threads are created for planning, model-learning, and action selection. This particular example offers thorough insight into the difficulties of real-time policy deployment.

Training the reinforcement learning model on-line involves a major component of agent *exploration* to sample data from the environment in order to learn an optimal policy to accomplish a given objective (reach a destination without collision with obstacles). A consequence of this approach is that learning a policy while attempting to select new actions comes at a significant computational cost. Alternatively, training the model in simulation and transferring the learned policy to the physical robot is an idealistic approach that largely depends on the accuracy of the system's dynamic model. Small model errors can accumulate for under-modelled systems (e.g. by neglecting contact friction between mechanical interactions, neglecting air drag, etc.) resulting in divergence from the real system. The technique of transferring policies works best for self-stabilizing systems that do not require active control to maintain a safe state for the robot. For unstable tasks or approximate models, this approach still has utility for testing algorithms in simulation, verifying theoretically optimal solutions, and identifying alternative data-collection strategies. Addressed in [5], a general framework that can be used for issuing comparable experiments and consistent evaluation of reinforcement learning models on real robotic systems is highly desireable, but has yet to be developed. This paper seeks to identify a possible solution to this problem.

IV. Method

A. Testing Environment for the Trained Model

A demonstration of the learning process using MVEDQL for a simple box-shaped indoor environment with a single obstacle is provided in Figure 2, where multiple vehicles are spawned at the same location with different initial orientations with respect to the desired goal position. During testing, the agents employ the learned policies and attempt to reach the input destination position. The process of testing for the chosen environment is visualized in Figure 3.

Accomplishing the primary objective of implementing the trained DRL model for autonomous navigation of the robot can *seemingly* be satisfied by constructing a simple indoor arena with a physical obstacle placed between the vehicle and a goal point. Short walls restrict the domain or *search space* and the gray centered box acts as the obstructing object. The lighthouse sensors on the robot are visible at all positions within the domain as required to accurately track the pose of the robot. However, when using a testing arena with *real* obstacles, it is apparent that testing reinforcement learning models with the mobile robot will often result in a collision for

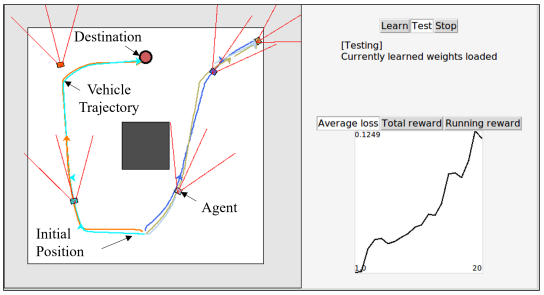

Figure 3: Testing a Reinforcement Learning Model for Autonomous Waypoint Trajectory Generation Using Multi-Agent Virtual Exploration in Deep Q-Learning (MVEDQL) [3]

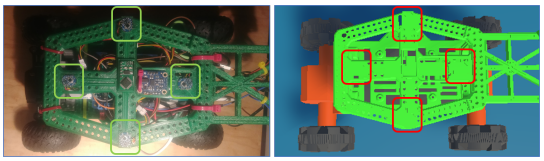

Figure 4: Light-to-digital Sensors Compatible With SteamVR Lighthouse Sensors

poorly performing models. Over time, this will introduce wear, potentially damage the robot and alter its dynamics, negatively influencing the outcome of the results.

Seeking a safer approach, consideration has been given to a potentially new method for testing pre-trained reinforcement learning models with physical robots without the possibility of damaging the system or altering its dynamics over time using virtual reality (VR). Instead of placing physical obstacles in an empty indoor environment, a virtual rendering of the object(s) can be constructed in VR so that during testing if the actual robot's position (tracked in the virtual environment using the light-to-digital sensors shown in Figure 4) coincides with the virtual object then the test result counts as a collision, ending the test without damaging the robot. This approach offers many benefits beyond safety as new environments can be constructed virtually, saving time and material costs. One caveat to this approach is that the readings from the robot's ToF distance sensors will need to be artificially reproduced to account for obstacle detection based on the heading of the vehicle at any time step. Although this artificial fabrication of the readings subtracts from the "real" hardware implementation aspect of testing, the vehicle dynamics are still unaffected, which is the primary necessity for valid testing.

An example application similar to this method can be found in [7], where a simulated environment of an underground mine roadway is used to test visual sensing algorithms. A framework for combining Unity and ROS has been proposed in [8]. These two studies are very recent, but highlight the possibility for using virtual reality as a common framework for reinforcement learning inference.

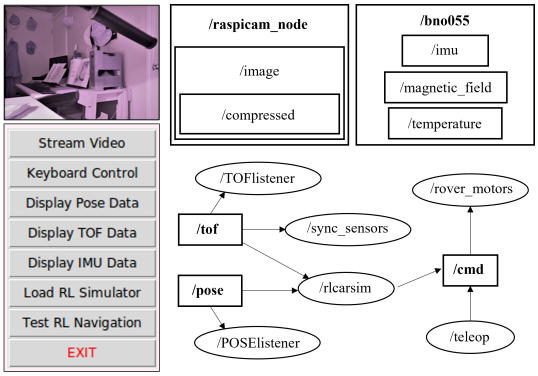

Figure 5: Host GUI and ROS Network Configuration for Testing and Deployment of the Unmanned Vehicle

V. Experimental Setup and Results

A. ROS Network Configuration and Remote Deployment GUI

Deployment, testing and control of the mobile robot (client) is facilitated by a custom GUI application running on a separate PC (host) by providing several utilities accessible via a user-friendly interface with labeled button inputs initiating several utilities, spawning threads for the individual processes. To initialize the sensors and configure the ROS network for communication with the host PC, a single script on the robotś microcomputer (Raspberry Pi 3B) is executed upon startup which enables the ROS nodes necessary for controlling the vehicle and publication of sensor data over associated topics that become available to the remote host. The GUI framework is displayed in Figure 5 where all thread processes for testing and deployment have been spawned, with a visualization of their connection to the ROS network:

The actions to be taken by the physical vehicle are produced by the RL model during testing (the "rlcarsim" node, which reads data from the ToF sensors and Pose topic) and published to the "/cmd" topic which acts as input to control the vehicle motors. Alternatively, the "teleop" node enables user-controlled remote navigation through this same route. This network is maintained during testing within the Unity Virtual Reality environment described in the following sections. The connection between the established ROS network and the Unity VR framework is assisted by the ROS# [9] Unity asset package which provides tools for publishing to the topics representing the robot pose and simulated ToF sensor readings as visualized in Figure 6.

B. Virtual Reality Testing Arena Construction

A virtual reality testing environment shown in Figure 8 has been constructed in Unity using the SteamVR SDK which replicates the simple arena used for training the robots using reinforcement learning. The testing arena developed for this implementation matches the simple arena used for initial training. It includes a simple box obstacle centered within a bounded arena, and a single fixed waypoint destination for the vehicle. The distance to nearby objects are continuously

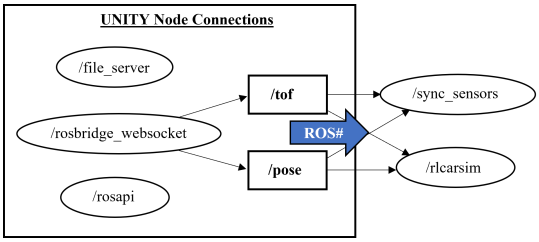

Figure 6: UNITY Nodes and Topic Messages Published Using ROS#

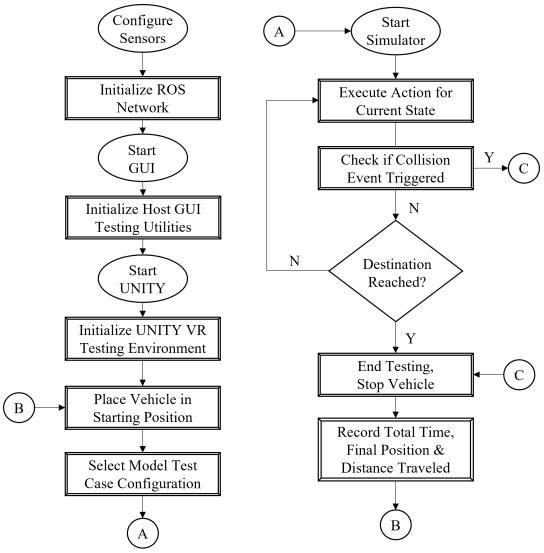

Figure 7: Program Execution Order and State Transition Diagram

updated and displayed above the testing area. In this environment, it is possible for the user to issue waypoint destinations for the vehicle, or configure static destination for consistent testing. New complex environments can be constructed quickly in Unity to provide sophisticated interactions between the physical robot and virtual obstacles. With the ability to safely visualize testing in real-time with hardware-in-the-loop, this method has the potential to offer greater insight into the behavior of the trained models, and even promote on-line learning beyond validation testing.

The execution of individual programs and condition checks are visualized in the state transition diagram of Figure 7. This flowchart outlines the general procedure to be followed while testing different model configurations within the virtual environment.

C. Robot Model Collisions in Unity

Tracked using Lighthouse sensors that continuously scan the testing area, the robot's estimated position can be monitored and linked to the 3D model produced in Figure 8. The kinematic motion of the vehicle model within Unity is to be controlled by script code, *not* by the physics engine, where the actions of the vehicle at any given state guide the motion of the robot. However, the model should still "collide" with any obstacles it encounters, which are detectable by *triggers*. Triggers are used to allow the scripting system to detect when collisions occur, then initiate actions using the "OnCollisionEnter" function. With the "Is Trigger" enabled, the model does not behave as a solid object, and other colliders will pass through, which calls the "OnTriggerEnter" function on the trigger object's scripts. If the position of the vehicle collides with an obstacle (the gray virtual box, or the arena boundary, as shown in Figure 9) in the testing arena, the triggered collision event will halt the testing and physical vehicle (through ROS), recording the performance and reporting a failed trial.

D. Artificial ToF Sensor Data

This virtual framework requires the addition of an artificial reconstruction of the vehicle's distance sensor readings so that the proper state of the robot can be represented in accordance with its position relative to any nearby obstacles. A script for the ToF array produces RayCast vectors from an initial transform position located at the center of the top platform on the vehicle and scales a line GameObject to visualize the ray between the vehicle and detected rigid body object, shown as three orange lines in Figure 9. These readings are used to update the state of the robot as if it were interacting with physical objects in its environment. The distances are published over ROS and used as input for the rlcarsim_ros.py script to generate new actions for the current state.

E. DRL Testing User Interfaces

An in-game menu for allowing selection of model weights and other various simulation configurations is constructed for testing. Similarly, an overlay of the test performance as it is occurring in real-time enhances the utility of the application. The Canvas used for the UI has a Render Mode setting which can be used to make it render in screen space or world space. The screen space is used in this project to be visible at all times during in-game testing, showing the current task performance of the robot model and sensor readings. A separate world space canvas is used to allow the selection of the current loaded model parameter configuration. The actions initiated upon selecting "Test RL Navigation" are ordered as follows:

1) Request user selection of environment and waypoint destination point.
2) Simulate the environment and vehicle, starting in the initial position matching the physical robot.
3) Load the pre-trained weights for the user's chosen environment.
4) Run "rlcarsim_ros.py" which generates actions to control the linear and angular velocities of the physical vehicle over the ROS network based on the state of the simulated vehicle (the vehicle pose relative to surrounding obstacles).

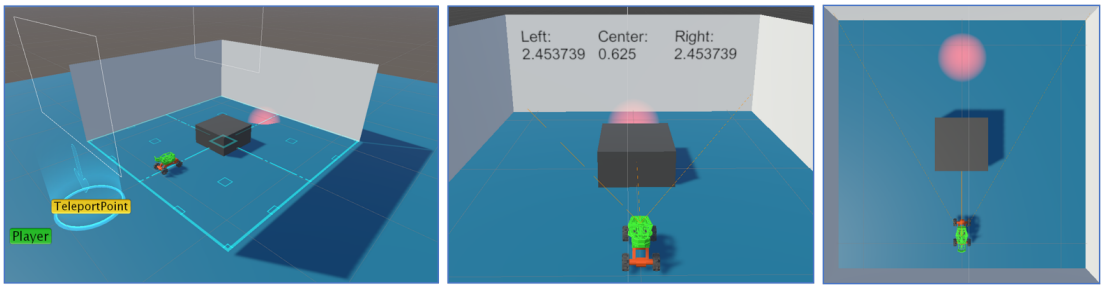

Figure 8: Reinforcement Learning Testing Arena in Unity Environment

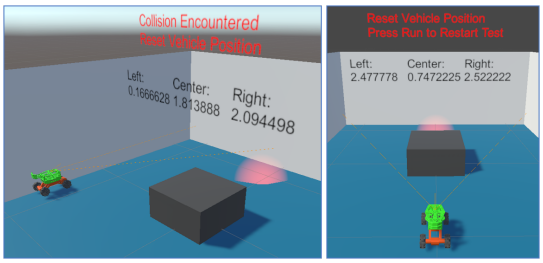

Figure 9: Collision Event Message Display

VI. Future Developments

A final step for the VR simulation framework is to evaluate its effectiveness for testing the model cases established in section II-A in real-time. After validating this approach, improvements to the virtual testing arena and reinforcement learning simulator can be considered. The vehicle in this research has been initially trained in a python simulation environment with various environment configurations, including tracks and arenas of more complex geometry than the simple box used in this demonstration. One objective could be to recreate these environments in Unity and test the model performances for further validation of the virtual framework approach. Additionally, the ML_agents Unity asset package [10] provides many utilities for training virtual agents using python scripts, and may benefit the development of the simulator by incorporating training into the new framework.

It is of primary interest to adapt the virtual testing arena framework to be applicable for more general cases of testing robotic systems. As has been shown, there are many useful packages currently available for integrating Unity with ROS, implementing AI algorithms, and establishing hardware-in-the-loop connections to robotic systems using SteamVR's lighthouse sensor tracking technology. These systems in combination can potentially provide a useful resource for robotics researchers by establishing a common framework for testing reinforcement learning algorithms on real-time robotic systems, opening up the possibility for collaborative efforts among robotics engineers.

VII. Conclusion

The creation of a virtual reality environment for testing reinforcement learning-based navigation has many useful utilities, the most important of which is to ensure that collisions between the vehicle and obstacles in its environment are prevented. In a more general sense, this method is a promising approach to safely testing and training reinforcement learning techniques for indoor robotics applications. While there is additional effort to be made before the testing environment is robust, the development of a user-friendly GUI framework for deployment of the vehicle presented in this work offers additional measures of safety during testing. Upon completing the development of the virtual reality testing arena, it is of interest to adapt the environment for general compatibility in testing reinforcement learning models for other robotic systems that have been trained using deep reinforcement learning techniques.

References

[1] J. G. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," *Advances in neural information processing systems*, pp. 1047–1053, 1997.

[2] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," 2012. [Online]. Available: https://arxiv.org/abs/1205.4810

[3] A. Majumdar, P. Benavidez, and M. Jamshidi, "Multi-agent exploration for faster and reliable deep q-learning convergence in reinforcement learning," in *2018 World Automation Congress (WAC)*. Stevenson, WA, 2018, pp. 1–6.

[4] ——, "Lightweight multi car dynamic simulator for reinforcement learning," in *2018 World Automation Congress (WAC)*. Stevenson, WA, 2018, pp. 1–6.

[5] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[6] T. Hester, M. Quinlan, and P. Stone, "Rtmba: A real-time model-based reinforcement learning architecture for robot control," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 85–90.

[7] Y. Wang, P. Tian, B. Zheng, Y. Zhou, Y. Li, and X. Wu, "Special robot vision algorithm test platform in virtual reality environment," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2019, pp. 416–421.

[8] E. Rosen, D. Whitney, E. Phillips, D. Ullman, and S. Tellex, "Testing robot teleoperation using a virtual reality interface with ros reality," in *Proceedings of the 1st International Workshop on Virtual, Augmented, and Mixed Reality for HRI (VAM-HRI)*, 2018, pp. 1–4.

[9] A. Hussein, F. García, and C. Olaverri-Monreal, "Ros and unity based framework for intelligent vehicles control and simulation," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2018, pp. 1–6.

[10] 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," 2018. [Online]. Available: https://arxiv.org/abs/1809.02627