MODIFLY : A Scalable End-to-end Multi-Agent Simulation for Unmanned Aerial Vehicles

First Author[1][0000−1111−2222−3333], Second Author[1][0000−1111−2222−3333], and Third Author[1][0000−1111−2222−3333]

Princeton University, Princeton NJ 08544, USA

Abstract. Multi-agent unmanned aerial vehicle (UAV) systems have emerged as a promising solution for complex applications such as industrial automation, surveillance, and disaster response. However, the application of multi-agent UAV coordination remains challenging due to the lack of consideration of real-world constraints such as communication link degradation, scalability issues, and the need for realistic training environments. Existing simulation platforms often lack the fidelity and flexibility required to bridge the gap between simulation and deployment. To address these limitations, we propose MODIFLY , a scalable, cross-platform, end-to-end simulation platform tailored for multi-agent UAV control. MODIFLY introduces dynamic communication modeling, including link degradation, to accurately simulate real-world UAV operations. It supports distributed execution across multiple UAVs, seamless coordination, real-time monitoring, and user input capture. MODIFLY uniquely integrates real drones with virtual environments, allowing UAVs to interact with simulated obstacles and peer ones for hybrid reality testing. Additionally, the platform is designed to facilitate reinforcement learning (RL) research by providing compatibility with popular libraries like OpenAI Gym and PettingZoo, supporting both single-agent and multi-agent RL environments. MODIFLY offers an intuitive interface for real-time parameter tuning and performance analysis, making it a versatile tool for researchers and practitioners to develop and validate UAV coordination strategies under realistic conditions.

Keywords: Multi-agent Systems · UAVs · Reinforcement Learning.

1 Introduction

Unmanned vehicles (UVs) [4] have gained a lot of attention over the past decade due to their wide range of applications from agriculture [7] to law enforcement [14], from military surveillance [10] to disaster management [5]. UVs can be categorized based on their modes of transportation, including aerial, land, and aquatic vehicles. Among aerial UVs, quadcopters are the most famous due to their ability to hover, take off, and land in confined spaces, making them particularly suitable for surveillance tasks such as industrial inspections, urban reconnaissance, and structural assessments of unsafe buildings.

With the increasing complexity of robotic systems, multi-agent coordination has become an important factor in solving large-scale decision-making systems. These systems play an important role in our daily lives where agents interact with each other at operational, tactical, and strategic levels. For example, the social interactions between humans are essential for many social activities that lead to sophisticated yet fundamental social structures in various aspects of our lives. Motivated by the ubiquitous multi-agent systems in social networks, the creation of similar structures for robotics systems, such as aerial UVs to perform tasks autonomously by following their humans' counterpart, has become an emergent research topic. One fundamental technical challenge is to uncover the key principles and ideas behind the multi-agent system's structure of social networks, such that multi-agent robotic systems demonstrate human-level intelligence and beyond. Addressing such a challenge can not only replace humans from tedious and dangerous tasks but also provide more economical and efficient solutions for numerous tasks, such as industrial automation, surveillance, and emergent response. However, before real-world deployment, these methods must be evaluated extensively in controlled simulation environments to ensure reliability, safety, and scalability.

Unmanned aerial vehicles (UAVs), also called drones, have been popular in various emerging applications, such as delivery, environmental monitoring, and agriculture. Simulating multi-agent UAV coordination presents significant challenges due to limited communication, computation, and sensing capabilities. Meanwhile, the scalability is much needed to ensure that multi-agent UAV coordination is robust to the addition of new UAVs in the team. Hence, these systems require robust distributed control to enable efficient collaboration among drones for tasks such as autonomous navigation [18], search, and rescue [17]. However, existing simulation platforms for multi-agent UAVs struggle with scalability, high degrees of freedom, realistic modeling of real-world communication, and non-stationarity, making it difficult to evaluate their real-world applicability.

There exist several simulation platforms [8, 1, 19, 3, 9]. Gazebo [8], a widely used open-source simulator, supports multi-agent physics-based simulations and integrates well with Robot Operating System (ROS). However, it has limited support for Windows, strong dependencies on ROS, and scalability constraints when simulating large drone fleets. Although some frameworks address multi-robot coordination such as Multi-robot Multi-target Potential Field (MMPF) strategies [19] or Byzantine threat modeling in ROS-based systems [3], they lack comprehensive end-to-end capabilities for realistic UAV simulations. Recently, ROS 2, an extension of ROS, has been introduced [9] which expedite the robotics research with freely available components and a modular framework. However, its support for UAV-specific simulation is limited. Several UAV-specific simulation efforts have been explored in the literature. Patel et al. [12] simulated quadcopters using the Euler Lagrange method to investigate its modeling, and altitude model validation. Kaya et al. [6] simulated a fuel battery hybrid powered UAV model in MATLAB/Simulink software for actual applications. Despite the success of these works, most existing simulation platforms, especially for UAVs

remain restrictive. They fail to account for communication link degradation, real-time multi-agent control, and mixed-reality integration, which are important factors for realistic UAV deployment.

To fill this gap, we propose MODIFLY , a scalable, cross-platform, end-to-end simulation platform for multi-agent UAVs. MODIFLY incorporates dynamic communication capabilities, including communication link degradation, to provide a realistic platform for UAV deployment in real-world conditions. It also supports distributed execution, which allows the coordination of multiple systems across multiple computers and the ability to monitor these devices easily in both a training and real-time setting. Each virtual device can capture user input through the monitoring software and simulate a camera based on the virtual environment. Users can import virtual obstacles and fly a simulated or real drone through these virtual obstacles, enabling highly dynamic, cost-effective, real-world testing. Additionally, to enhance the research in multi-agent quadcopter control, MODIFLY supports single-agent and multi-agent reinforcement learning (RL) environments based on OpenAI gym [2] and Pettingzoo [16][1].

Our contributions are summarized as:

- We propose MODIFLY , a realistic, scalable, cross-platform, end-to-end simulation platform for multi-agent quadcopter control.
- Our platform incorporates real-world communication constraints such as degraded links to enhance training fidelity.
- MODIFLY enables real drones to interact with virtual environments, including simulated obstacles and peer drones, for realistic training and testing.
- We provide an intuitive interface for real-time monitoring, parameter tuning, and dynamic interaction with UAVs.
- To facilitate RL research, MODIFLY includes wrappers for single and multi agent RL libraries such as stable-baselines3 [13], and EPyMARL [11] respectively.

2 Preliminaries

2.1 UAV Dynamics and Control

For simplicity of presentation, we here use quadcopters as a representation of the UAV platform. A quadcopter is an UAV equipped with four motors and can maneuver in six degrees of freedom (DoF). The maneuverability of quadcopters makes them particularly useful for control algorithm deployment and real-world applications. These applications include but are not limited to autonomous package delivery, aerial surveillance, search-and-rescue missions, etc. The Newton-Euler equations control the dynamics of a quadcopter, where the thrust, drag, and gravitational forces impact its trajectory. The translational motion along the x,y, and z axes and the rotational motion (roll, pitch, and yaw) form the six DoF of a quadcopter.

[1] All code and RL environment wrappers will be made publicly available upon paper acceptance

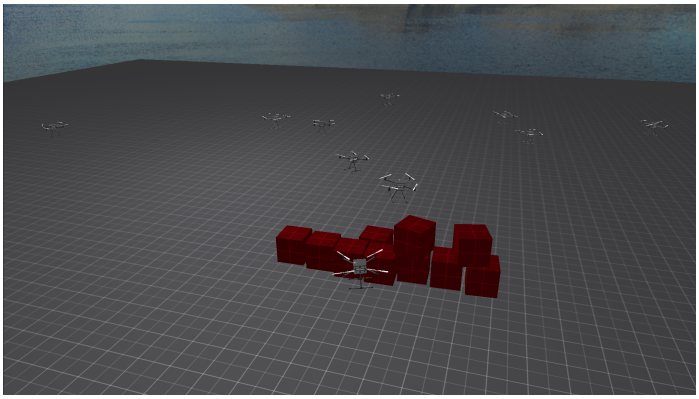


Fig. 1: 3D rendering of a multi-agent quadcopter environment with drones operating under a random policy.

A three-stage proportional-integral-derivative (PID) controller is used to control the simulated quadcopter. A velocity or position setpoint is the input to the first stage of the controller, which is then used in the subsequent attitude controller and angular velocity controller. All the communications of this system are built upon TCP and UDP sockets, where TCP sockets have reliable transmission and UDP sockets do not.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning where an artificial agent learns to perform decision-making under uncertainty. In RL, an agent sequentially interacts with an environment (see Figure 2), which is usually unknown, and learns about the dynamics of the environment in a trial-and-error manner [15]. The agent observes the current state, performs an action, receives a reward signal, and transits to the next state. Formally, an RL problem can be represented as a Markov Decision Process (MDP) which can be represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ represents the action space, $\mathcal{P}(s'|s, a)$ is the transition probability function that determines how an agent moves to the next state s' by taking an action a in the current state s, $\mathcal{R}(s, a)$ represents the reward function that provides feedback for every action and γ denotes the discount factor $[0, 1)$. In RL or MDP, the reward and state transition probability functions are usually unknown to the agent. The goal is to learn an optimal policy $\pi(a|s)$ that maximizes the expected cumulative reward. The policy is considered *Markov* if it solely relies on the current state for action selection. If the same procedure is applied at each time step, the policy is considered as *stationary*. A policy can be either deterministic or stochastic. A deterministic policy, denoted as $\pi(s) = a$, returns a single action for a given state, while a stochastic policy, denoted as $\pi(a|s) = \mathcal{P}_\pi[\mathcal{A} = a|\mathcal{S} = a]$, returns

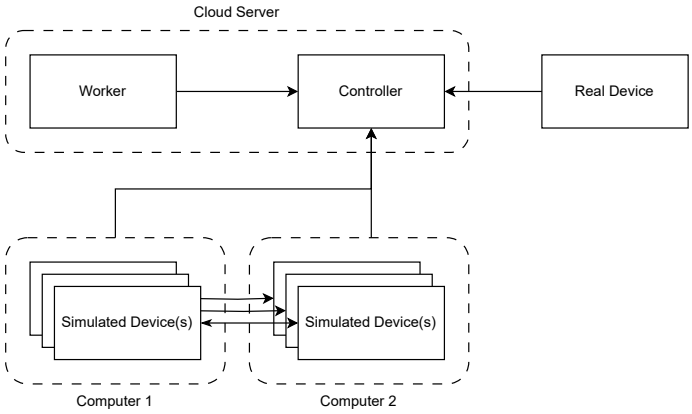


Fig. 2: System architecture: Multi-device simulation network with distributed computing nodes connected to a central controller, including integration with physical devices such as quadcopters.

a probability distribution of selecting all possible actions in a state s. In this paper, we consider policies to be stationary, Markov, and stochastic.

3 MODIFLY

MODIFLY is an end-to-end simulation platform designed to work for the development and testing of multi-agent UAV applications. It is highly scalable and compatible across various operating systems without performance degradation. MODIFLY is modular, which allows its software components to run on a single device or be distributed across multiple nodes. Moreover, it incorporates realistic communication constraints present in real-world drone operations. Figure 2 illustrates the overall system architecture. The platform supports multiple clients. In order to connect each client, there is a custom networking system designed to take advantage of the clustered architecture. Instead of using ROS, MODIFLY chooses to use allow users to write their own scripts in Lua instead as Lua is compatible with a clustered architecture due to its ease of deploying new packages or updates, along with advantages such as easy updates, configuration, and dependency management. Additionally, MODIFLY also comes with a monitoring tool that allows the user to load 3D scenes, visualize an entire system of devices, and record telemetry data. The full monitoring tool window, including the different views, cockpit, and visualizer, can be seen in Figure 3.

3.1 Architecture

In this section, we describe the system architecture for MODIFLY , which includes a main control connected to simulated devices, client devices, and workers. Next, we describe each component of the system.

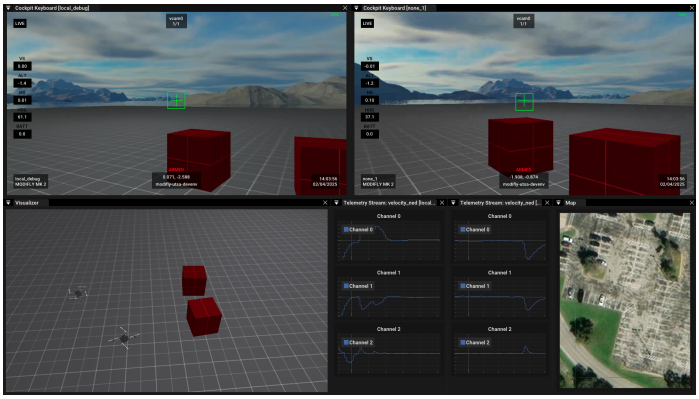


Fig. 3: Monitoring software viewing for MODIFLY .

Controller A controller is a central node that manages all client connections through TCP and UDP servers. Due to the safety and security in real-world operations, all the clients' connections to the controller are encrypted. It dynamically chooses whether a message should be sent through a TCP or UDP socket. Generally, messages that have tolerance for error will always be sent through UDP while messages that are dependent on order and have no tolerance for error will be sent through the TCP socket. This makes sure that all the transmissions are efficient and error-free.

A Controller also contains a master scene graph where simulated agents can update objects in this graph or create new objects by notifying the Controller. The Controller then accumulates and aggregates these update requests and periodically sends a notification to other agents so they can synchronize with this graph. Object updates are not instantly reflected on different agents, thereby simulating a realistic latent effect.

Device The device is the client that connects to the controller and responsible for providing control inputs to the simulated agents. Each simulated agent is self-contained, and can only communicate with other agents using the built-in inter-agent communication system. This allows for the ability to simulate communication errors between devices and with the controller.

Worker In MODIFLY , workers handle computationally intensive tasks offloaded from agents, such as inference of trained models. Workers can operate locally or in the cloud, hence making them highly useful for computationally extensive tasks. For instance, the inference of AI models especially the LLMs inference can be computationally challenging. As a consequence, they can be deployed on local networks or the cloud, addressing power and weight constraints on agents. All the workers have their own scene graphs that are connected to the controller.

3.2 Scalability

We now discuss the scalability of MODIFLY . It can be deployed across multiple end nodes simultaneously and can work with multiple clusters with minimal effort, equireing a single package file to be copied across nodes.

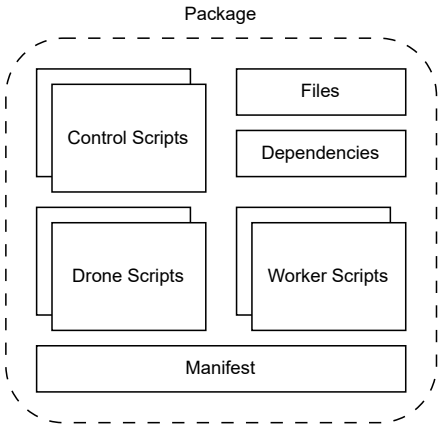
Networked Scalability MODIFLY is scalable as it allows networking of multiple computers together running multiple device instances. This is crucial for large-scale AI systems, such as reinforcement learning, where hardware limitations on a single resource—whether CPU or GPU—raise many challenges. MODIFLY facilitates horizontal scaling by adding new nodes rather than increasing the capabilities of a single system. This particular feature of MODIFLY creates an interconnected cluster of devices that efficiently share computational loads.

Packages A package is MODIFLY is a self-contained archive that includes all necessary scripts and resources to run a model across different environments. It also contains metadata, dependencies are roles for different devices and workers. Figure 4a shows the contents of a package. In MODIFLY , packages are particularly designed to operate without hard library dependencies which enables homogeneous deployment across the simulation environment. This makes it easy to push a single archive file to the controller, which then distributes the scripts to each client based on its role and ensures scaling from a single device to clusters of multiple computers that can be setup without additional configuration during updates or deployments. Furthermore, this guarantees that a single piece of code runs uniformly across all nodes where the runtime is installed and underscores the scalability, feasibility, and ease of use in complex multi-agent systems.

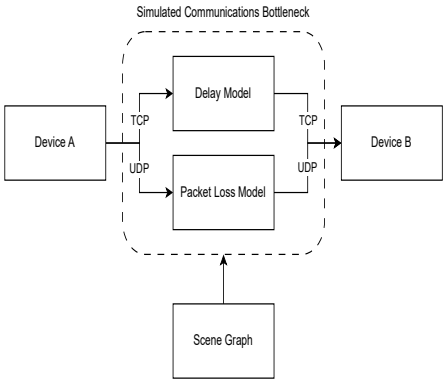
3.3 Deployment and Real-World Integration

In this section, we discuss how MODIFLY can be extended to real-world UAV applications through robust communication and mixed-reality simulation.

Communications Communication is an integral part of building real-world UAV systems. MODIFLY incorporates a simulated bottleneck that can add communications faults in both TCP and UDP connections. These faults have different effects based on the underlying systems. In this communication bottleneck, a simplified stochastic model manages UDP packet loss probabilities, while a delay model simulates TCP retransmissions in milliseconds. These delay models allow end users to test control latency in complex environments, video streaming, and RL under non-ideal conditions, making the development of complex multi-agent UAV systems more realistic and feasible.



(a) Package file structure showing core components, dependencies, and required execution resources.



(b) Network bottleneck simulation architecture with parallel UDP and TCP traffic handling through loss and delay models.

Fig. 4: System architecture showing package and network communication.

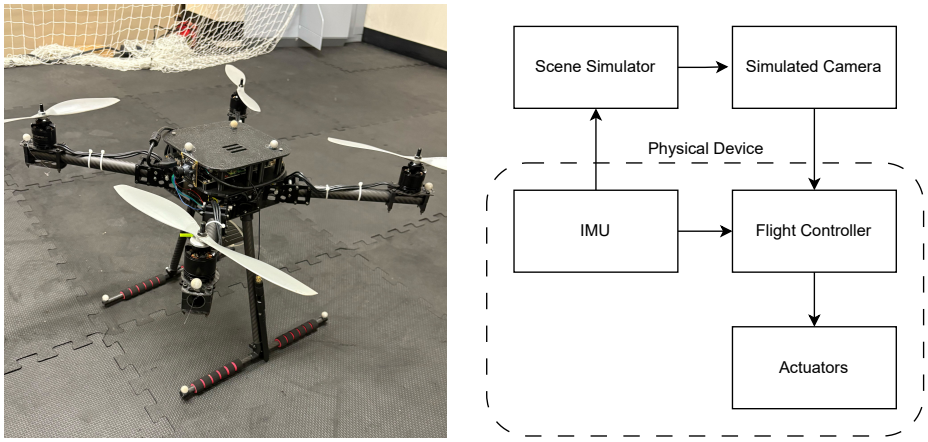
Mixed Reality Simulation MODIFLY bridges the gap between virtual simulations and physical UAV operations by integrating mixed reality simulation. In mixed reality simulation, a physical device like a drone operates in an empty room, while interacting with simulated 3D obstacles and scenarios. This setup allows the testing of many complex algorithms, such as collision avoidance, with real flight dynamics. Usually, these algorithms are tested in a simulation environment. These environments, however, abstract away important effects of the real world and provide a best-case-scenario for the demonstration of an algorithm. However, with MODIFLY , our goal is to provide a platform between fully simulated and physical demonstrations, which we call mixed reality simulations.

These simulations couple a real, physical device (see Figure 5a) operating in a blank space with a simulated environment. This means that developers can add simulated obstacles and goals while testing real flight dynamics. To achieve this, a scene graph is streamed from the controller to all client devices on the network. These devices then construct a 3D scene using a renderer with camera position and orientation data being received from the IMU (see Figure 5b).

4 Results

4.1 Scaling

To demonstrate the scalability of MODIFLY by just using CPUs, we performed analysis on two different computers using single-threaded execution. This analysis demonstrates the performance impact of increasing instances within a shared



(a) Physical drone running MODIFLY . (b) Mixed reality simulation.

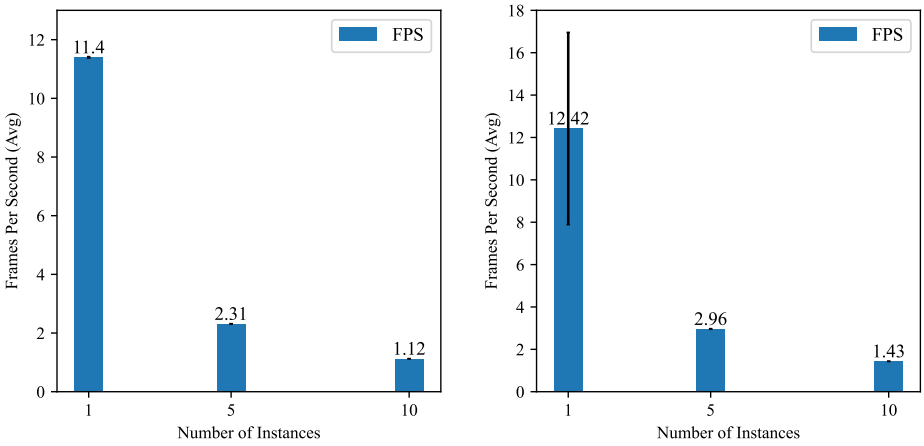
Fig. 5: System deployment demonstration: physical drone implementation and mixed reality simulation framework.

thread. To fairly evaluate the performance, we created an object detection task where each device ran all scripts, scheduled tasks, and physics simulation while also rendering a camera frame from a 3D scene. We used the Mobilenet SSD V3 model[2] trained on the COCO 2017 dataset [3]. The model utilized TensorflowLite with an XNNPack, integrated through control scripts that connected the camera feed to the inference engine. The inference pipeline rendered the 3D scene into an image, scaled this image down, and then pushed this image into the model.

As shown in Figure 6a, the single-threaded performance scales linearly with an increasing number of instances. While the performance for a single thread initially looks suboptimal, this is consistent with the computational overhead from parallel tasks and CPU-only inference. Future works could explore scaling with respect to a GPU or other accelerator, though our current focus was to assess the CPU performance on differing devices. Similarly, in the Windows-based system (see Figure 6b), the single instance performance shows higher variance. This variance is most likely caused by the Window's Operating System thread scheduler as MODIFLY yields the thread back to the operating system after each iteration. With a completely different operating system and CPU architecture, its performance is largely similar to the M3 Macbook Pro.

Based on these results, the most optimal configuration would require that each drone has its own dedicated thread. Due to the limited threading capabilities of processors, this demonstrates that clustering of multiple computers is necessary to achieve any large-scale end-to-end simulation of intelligent devices.

[2] https://github.com/chuanqi305/MobileNet-SSD
[3] https://cocodataset.org/



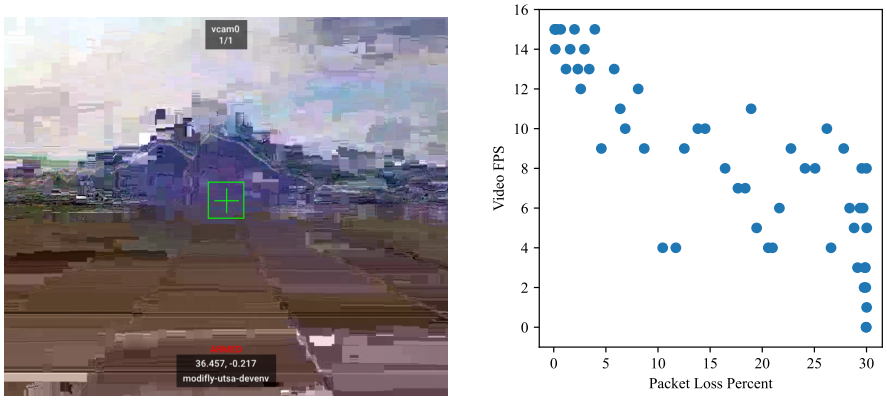
(a) Performance on M3 MacBook Pro. (b) Performance on AMD Ryzen 6900HX.

Fig. 6: Single-threaded simulator performance comparison.

4.2 Communications

To demonstrate communications degradation as part of the simulated bottleneck (see Figure 4b), we analyze the relationship between packet loss percentage and valid decoded video frame. In this context, a valid frame is defined as a frame that contains sufficient data for the H.264 decoder to generate an output frame. Our experimental setup simulates devices that render a 3D scene, encode that scene using H.264 compression, pushes the frame through the bottleneck to the controller. The controller then forwards this frame data to the monitoring software Figure 3 that measures the frame reception rate. The maximum frames per second that can be sent by the device has been limited to 15 frames per second.

While H.264 encoding handles packet loss, it generally expects these losses to be transient. Our results demonstrate that even the low sustained packet loss rates of 1% start to drop frames and impact the visual artifacts. As shown in Figure 7b at approximately 30% packet loss the received frame rate fluctuates between 0 and 8 frames per second. If this video is being streamed to another device to run any sort of inference or algorithm based on this feed, it would quickly become unusable after even a small sustained packet loss. As packet loss increases and fewer key frames are transmitted without error, an H.264 stream quickly becomes littered with artifacts (see Figure 7a). These degraded frames become particularly unstable for real-world applications. While utilizing TCP as a reliable transmission protocol can prevent such artifacts, it presents a fundamental trade-off between transmission reliability and performance overhead.



(a) Visual artifacts under 37% packet loss with H.264 encoding.

(b) Frame parse success rate vs. packet loss.

Fig. 7: Impact of packet loss on H.264 video and frame parsing performance.

5 Conclusion and Future Work

In this paper, we presented a scalable, cross-platform, end-to-end simulation platform MODIFLY for the control of multiple UAVs. The new platform differs from the existing ones by introducing realistic communication modeling, distributed execution, easy environment setup and deployment, operational scalability, mixed reality, and RL-compatible. As an illustration of the new platform, we presented some tests to show that our approach is scalable, supports different communication features, and is easily deployable.

While MODIFLY provides an abstraction of typical communication features, the development and optimization of multi-agent RL algorithms considering these features is significantly lacking. Hence, one interesting research direction is the development of new multi-agent RL algorithms and approaches to address the challenge that communication among agents can be noisy, unreliable, and range-dependent. Meanwhile, testing of these algorithms in simulated and real-world environments is a must for their deployment. Another interesting direction is to create benchmark scenarios that can be used to quantify the effectiveness of both existing and new algorithms.

References

1. Babu, V.S., Behl, M.: f1tenth. dev-an open-source ros based f1/10 autonomous racing simulator. In: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE). pp. 1614–1620. IEEE (2020)
2. Brockman, G.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)

3. Deng, G., Zhou, Y., Xu, Y., Zhang, T., Liu, Y.: An investigation of byzantine threats in multi-robot systems. In: Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses. pp. 17–32 (2021)
4. Finn, A., Scheding, S.: Developments and challenges for autonomous unmanned vehicles. Intelligent Systems Reference Library **3**, 128–154 (2010)
5. Griffin, G.F.: The use of unmanned aerial vehicles for disaster management. Geomatica **68**(4), 265–281 (2014)
6. Kaya, U., Bayrak, Z.U., Oksuztepe, E.: Fuel cell/battery hybrid powered unmanned aerial vehicle with permanent magnet synchronous motor. International Journal of Sustainable Aviation **3**(2), 130–150 (2017)
7. Kim, J., Kim, S., Ju, C., Son, H.I.: Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. Ieee Access **7**, 105100–105115 (2019)
8. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566). vol. 3, pp. 2149–2154. Ieee (2004)
9. Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W.: Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics **7**(66), eabm6074 (2022). https://doi.org/10.1126/scirobotics.abm6074, https://www.science.org/doi/abs/10.1126/scirobotics.abm6074
10. Ma'Sum, M.A., Arrofi, M.K., Jati, G., Arifin, F., Kurniawan, M.N., Mursanto, P., Jatmiko, W.: Simulation of intelligent unmanned aerial vehicle (uav) for military surveillance. In: 2013 international conference on advanced computer science and information systems (ICACSIS). pp. 161–166. IEEE (2013)
11. Papoudakis, G., Christianos, F., Schäfer, L., Albrecht, S.V.: Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. arXiv preprint arXiv:2006.07869 (2020)
12. Patel, K., Barve, J.: Modeling, simulation and control study for the quad-copter uav. In: 2014 9th International Conference on Industrial and Information Systems (ICIIS). pp. 1–6. IEEE (2014)
13. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)
14. Straub, J.: Unmanned aerial systems: Consideration of the use of force for law enforcement applications. Technology in Society **39**, 100–109 (2014)
15. Sutton, R.S.: Reinforcement learning: An introduction. A Bradford Book (2018)
16. Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L.S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., et al.: Pettingzoo: Gym for multi-agent reinforcement learning. Advances in Neural Information Processing Systems **34**, 15032–15043 (2021)
17. Waharte, S., Trigoni, N.: Supporting search and rescue operations with uavs. In: 2010 international conference on emerging security technologies. pp. 142–147. IEEE (2010)
18. Wang, C., Wang, J., Shen, Y., Zhang, X.: Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology **68**(3), 2124–2136 (2019)
19. Yu, J., Tong, J., Xu, Y., Xu, Z., Dong, H., Yang, T., Wang, Y.: Smmr-explore: Submap-based multi-robot exploration system with multi-robot multi-target potential field exploration method. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). pp. 8779–8785. IEEE (2021)