







OmniDrones: An Efficient and Flexible Platform for Reinforcement Learning in Drone Control

Botian Xu , Feng Gao , *Graduate Student Member, IEEE*, Chao Yu , Ruize Zhang ,
Yi Wu , *Member, IEEE*, and Yu Wang , *Fellow, IEEE*

Abstract—In this letter, we introduce *OmniDrones*, an efficient and flexible platform tailored for reinforcement learning in drone control, built on Nvidia’s Omniverse Isaac Sim. It employs a bottom-up design approach that allows users to easily design and experiment with various application scenarios on top of GPU-parallelized simulations. It also offers a range of benchmark tasks, presenting challenges ranging from single-drone hovering to over-actuated system tracking. In summary, we propose an open-sourced drone simulation platform, equipped with an extensive suite of tools for drone learning. It includes 4 drone models, 5 sensor modalities, 4 control modes, over 10 benchmark tasks, and a selection of widely used RL baselines. To showcase the capabilities of *OmniDrones* and to support future research, we also provide preliminary results on these benchmark tasks. We hope this platform will encourage further studies on applying RL to practical drone systems.

Index Terms—Reinforcement learning, simulation and animation, machine learning for robot control, software tools for benchmarking and reproducibility, cooperating robots.

I. INTRODUCTION

MULTI-ROTOR drones and multi-drone systems are receiving increasing attention from both industry and academia due to their remarkable agility and versatility. The ability to maneuver in complex environments and the flexibility in configuration empower these systems to efficiently and effectively perform a wide range of tasks across various industries, such as agriculture, construction, delivery, and surveillance [1].

Manuscript received 22 September 2023; accepted 7 January 2024. Date of publication 19 January 2024; date of current version 12 February 2024. This letter was recommended for publication by Associate Editor W. Pan and Editor J. Kober upon evaluation of the reviewers’ comments. This work was supported in part by the National Natural Science Foundation of China under Grants 62325405, U19B2019, and M-0248, in part by Tsinghua University Initiative Scientific Research Program, in part by the Tsinghua-Meituan Joint Institute for Digital Life, in part by the Beijing National Research Center for Information Science, Technology (BNRist), and in part by the Beijing Innovation Center for Future Chips and 2030 Innovation Megaprojects of China through Programme on New Generation Artificial Intelligence under Grant 2021AAA0150000. The work of Botian Xu was supported by internship in Tsinghua University. (Corresponding authors: Chao Yu; Yu Wang.)

Botian Xu, Chao Yu, Ruize Zhang, and Yu Wang are with Electronic Engineering Department, Tsinghua University, Beijing 10083, China (e-mail: btx0424@outlook.com; zoeuchao@gmail.com; jimmyzhangruize@gmail.com; yu-wang@tsinghua.edu.cn).

Feng Gao and Yi Wu are with the Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100083, China, and also with Shanghai Qi Zhi Institute, Shanghai 200232, China (e-mail: feng.gao220@gmail.com; jxwuyi@gmail.com).

For more resources, please visit: <https://omnidrones.readthedocs.io/>.

This letter has supplementary downloadable material available at <https://doi.org/10.1109/LRA.2024.3356168>, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3356168

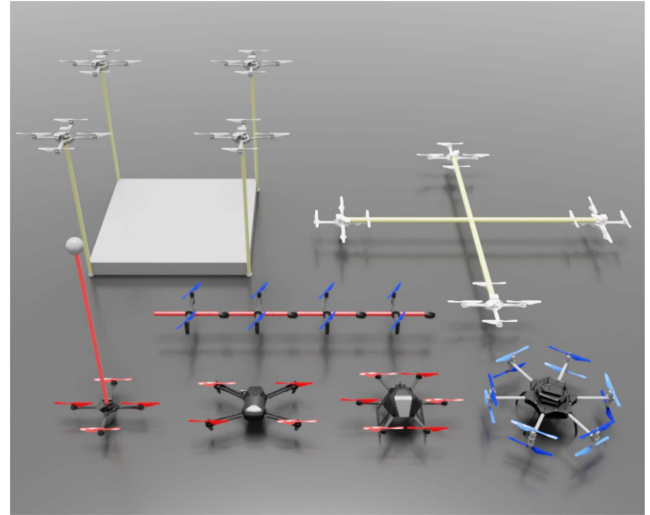


Fig. 1. Visualization of the various drone systems in *OmniDrones*, for which we offer highly efficient simulation, reinforcement learning environments, and benchmarking of baselines.

Recently, deep reinforcement learning (RL) has made impressive progress in robotics applications such as locomotion and manipulation. It has also been successfully applied to drone control and decision-making [2], [3], [4], [5], improving the computational efficiency, agility, and robustness of drone controllers. Compared to classic optimization-based methods, RL-based solutions circumvent the need for explicit dynamics modeling and planning and allow us to approach these challenging problems without accurately knowing the underlying dynamics. Moreover, for multi-drone systems, we can further leverage Multi-Agent RL (MARL), which is shown to be effective in addressing the complex coordination problems that arise in multi-agent tasks [6], [7], [8].

Efficient and flexible simulated environments play a central role in RL research. They should allow researchers to conveniently build up the problem of interest and effectively evaluate their algorithms. Extensive efforts have been made to develop simulators and benchmarks for commonly studied robot models like quadrupeds and dexterous arms [9], [10], [11], [12]. However, although a range of drone simulators already exists, they suffer from issues such as relatively low sampling efficiency and difficult customization.

To help better explore the potential of RL in building powerful and intelligent drone systems, we introduce *OmniDrones*, a platform featuring:

- **Efficiency:** Combining PyTorch-based multi-rotor dynamics and Nvidia Isaac Sim [13], [14], *OmniDrones* can notably achieve over 10^5 steps per second in terms of data collection, which is crucial for applying RL-based methods at scale.
- **Flexibility:** We provide drone models commonly used in related research while making it straightforward for users to extend the existing models, import new models, and add customized dynamics to meet diverse research needs.
- **RL-support:** *OmniDrones* includes a diverse suite of 10+ single- and multi-agent tasks, presenting different challenges and difficulty levels. The tasks can be easily extended and integrated with modern RL libraries.

To demonstrate the features and functionalities of *OmniDrones* and provide preliminary results, we implement and benchmark a spectrum of popular single- and multi-agent RL algorithms on the proposed tasks.

II. RELATED WORK

Simulated environments play a crucial role in the RL literature. We highlight the motivation of our work by reviewing the solutions developed out of various considerations, and related research in RL-based control of drones.

A. Simulated Environments for Drones

A common option in the control literature is to use Matlab to perform numerical simulations. This approach enjoys simplicity but has difficulty building complex and realistic tasks and is less friendly to reinforcement learning. Flightmare [15] and Airsim [16] leverage game engines such as Unity and Unreal Engine that enable visually realistic simulation. Flightmare’s efficient C++ implementation can notably achieve 10^6 FPS but at the cost of being inflexible to extend. Simulators based on the Robot Operating System (ROS) and Gazebo [17] have also been widely used [18], [19] as they provide the ecosystem closest to real-world deployment. For example, RotorS [18] provides very fine-grained simulation of sensors and actuators and built-in controllers for the included drone models, enabling sim-to-real transfer of control policies with less effort. However, Gazebo suffers from poor scalability and sample efficiency. Additionally, the working mechanism of ROS makes environment interaction asynchronous, which violates the common implementation practice in RL. To provide an RL-friendly environment, PyBulletDrones [20] introduced an OpenAI Gym-like environment for quadrotors based on PyBullet physics engine [21]. However, it relies on CPU multiprocessing for parallel simulation, which limits its scalability and leads to fewer steps per second. Aerial Gym [22] builds upon Isaac Gym to achieve efficient simulation of cluttered environments but is limited to one specific quadrotor model and navigation tasks.

Our platform aims firstly for efficiency and a friendly workflow for RL. While the highly parallelized GPU-based simulation ensures a high sampling performance, it is also convenient

to customize and extend the environment at Python level and seamlessly work with modern RL libraries such as TorchRL [23].

B. Reinforcement Learning of Drone Control

Reinforcement learning is seen as a potential approach for control and decision-making for multi-rotor drones. Prior works explored end-to-end training of visual-motor control policies [2], [24], [25], [26] to avoid the need for explicit dynamics modeling and hand-engineered control stack. Model-based reinforcement learning can combine learned forward dynamics models with planning methods, such as model predictive control (MPC), and has been investigated in [3], [27]. Applications to agile drone racing [4], [28] also demonstrated RL-based policies’ ability to cope with highly dynamic tasks, generating smooth and near-time-optimal trajectories in real-time. [29] benchmarked different choices of action spaces and control levels regarding learning performance and robustness. More recently, [5] trains a single adaptive policy that can control vastly different quadcopters, showing the potential of reinforcement learning in terms of generalization and adaptation capabilities.

To fully uncover the potential of RL on drones, a flexible and versatile platform that supports various research purposes is highly desirable. Therefore, *OmniDrones* is designed to be both efficient and flexible to meet the diverse needs.

III. OMNIDRONES PLATFORM

At a high level, *OmniDrones* consists of the following main components: (1) A simulation framework featuring GPU parallelism and flexible extension; (2) Utilities to manipulate and extend the drone models and simulation for various purposes; (3) A suite of benchmark task scenarios built from (1) and (2), serving as examples and starting points for customization.

An overview of *OmniDrones* is presented in Fig. 2. For comparison, Table I contrasts *OmniDrones* with existing drone simulators, highlighting the advantages of our platform. In the following subsections, we describe the details of these components and provide examples to demonstrate the overall capability of *OmniDrones*.

A. Simulation Framework

We achieve efficiency and flexibility by decoupling the simulation into a drone-specific part, namely actuator models (rotors) and aerodynamics effects such as downwash, and a general part including articulated and rigid-body dynamics and rendering, etc. We implement the former using PyTorch, which can be efficiently computed on GPUs as tensor operations, and let Isaac Sim handle the latter. This combination allows us to conveniently manipulate the physical configuration of the drones while enjoying GPU parallelism.

Regarding the multi-rotor dynamics, we follow the general model given by:

$$\dot{\mathbf{x}}_W = \mathbf{v}_W \quad \dot{\mathbf{v}}_W = \frac{1}{m} \mathbf{R}_{WB} \mathbf{f} + \mathbf{g} + \mathbf{F} \quad (1)$$

$$\dot{\mathbf{q}} = \frac{1}{2} \mathbf{q} \otimes \boldsymbol{\omega} \quad \dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(\boldsymbol{\eta} + \mathbf{T} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}) \quad (2)$$

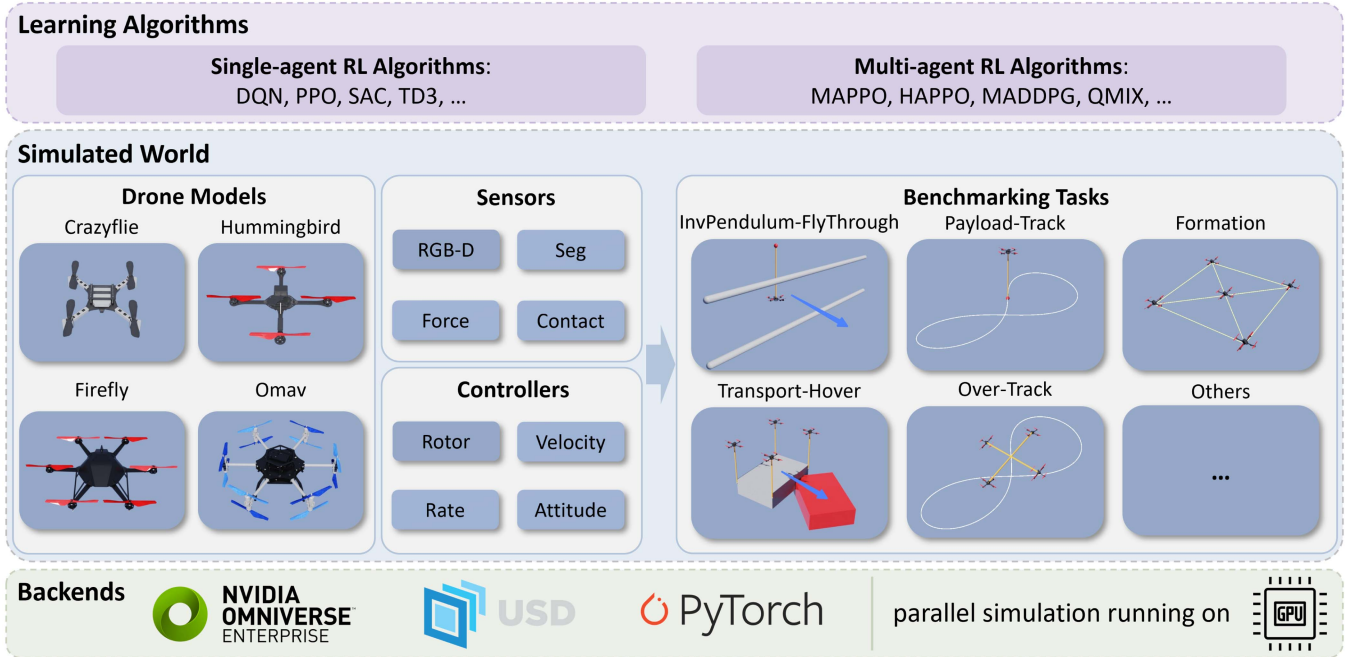


Fig. 2. Overview of *OmniDrones*. *OmniDrones* provides a foundational library of various sensors and drone models and offers multiple configurations to form diverse drone systems for multifaceted testing. In addition, *OmniDrones* incorporates several benchmark task logics, enabling the evaluation of the performance of different drone systems across various task objectives. Furthermore, we have implemented and assessed the capabilities of multiple learning algorithms on our benchmark tasks, serving as a baseline for subsequent work.

TABLE I
COMPARISON BETWEEN *OMNIDRONES* AND OTHER COMMONLY USED SIMULATED ENVIRONMENTS

| | Physics Engine | Renderer | Vectorization | | Drone Model | | | Runtime Operation | | Sync. / Steppable Physics & Rendering | Sensor | User Interface | |
|--------------------------|----------------|---------------|---------------|-----|-------------|-------|-------|-------------------|---------------|--|--------------------|----------------|---------------|
| | | | CPU | GPU | Quad. | Hexa. | Omni. | Configuration | Randomization | | | Task Spec | RL API |
| RotorS [18] | Gazebo-based | OpenGL | ✓ | X | ✓ | ✓ | ✓ | X | X | X | IMU, RGBD | - | - |
| Airsim [16] | PhysX | Unreal Engine | ✓ | X | ✓ | X | X | X | X | X | IMU, RGBD, S | C++&Python | Single&Multi. |
| Flightmare [15] | Flexible | Unity | ✓ | X | ✓ | X | X | X | ✓* | ✓ | IMU, RGBD, S | C++ | Single |
| PyBullet-Drones [20] | Bullet | OpenGL | ✓ | X | ✓ | X | X | X | X | ✓ | IMU, RGBD, S | Python | Single&Multi. |
| FlightGoggles [30] | Flexible | Unity | ✓ | X | ✓ | X | X | X | X | X | IMU, RGBD, S | C++ | - |
| CrazyS [31] | Gazebo-based | OpenGL | ✓ | X | ✓ | ✓ | ✓ | X | X | X | IMU, RGBD | - | - |
| OmniDrones (ours) | PhysX | Omniverse RTX | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | IMU, RGBD, S, F, C | Python | Single+Multi. |

In Drone Model columns, *Quad.*, *Hexa.*, *Omni.* stand for quadcopter, hexacopter, and omnidirectional, respectively. In Sensor column, *S* stands for segmentation, *F* stands for force sensors, and *C* stands for contact sensors.

where \mathbf{x}_W and \mathbf{v}_W indicate the position and velocity of the drone in the world frame. \mathbf{R}_{WB} is the rotation matrix from the body frame to the world frame. \mathbf{J} is the diagonal inertia matrix, m is the body mass, and \mathbf{g} denotes Earth's gravity. \mathbf{q} is the orientation (quaternion), and $\boldsymbol{\omega}$ is the angular velocity. \otimes denotes quaternion multiplication. \mathbf{F} and \mathbf{T} account for the external wrench (force and torque) introduced by, e.g., the drag and downwash effects. The collective thrust \mathbf{f} and body torque $\boldsymbol{\eta}$ are derived from single rotor thrusts \mathbf{f}_i as:

$$\mathbf{f} = \sum_i \mathbf{R}_B^{(i)} k_f \mathbf{c}_i \quad (3)$$

$$\boldsymbol{\eta} = \sum_i \mathbf{T}_B^{(i)} \times \mathbf{f}_i + k_m \mathbf{c}_i \quad (4)$$

where $\mathbf{T}_B^{(i)}$ and $\mathbf{R}_B^{(i)}$ are the local translation and orientation (tilt) of the i -th rotor in the body frame. The thrust and momentum are computed from rotor rotation \mathbf{c}_i and the force and momentum constants k_f and k_m . By default, \mathbf{c}_i changes according to $\mathbf{c}_i[t +$

$1] = \mathbf{c}_i[t] + \tau(\mathbf{c}_{\text{target}} - \mathbf{c}_i[t])$ at each time step where $\mathbf{c}_{\text{target}}$ is the commanded rotation.

The physical configuration of a drone model is specified by a Universal Scene Description (USD) file, which can be converted from the MJCF and URDF descriptions commonly used in existing workflows. Notably, with Isaac Sim, it is possible to programmatically edit the physical configuration, which we detail in the next section.

We offer a range of typical drone models with which users can build their applications. For example, *Crazyflie* is a small X-configuration quadrotor widely used for education; *Hummingbird* and *Firefly* are medium-sized quadrotor and hexacopter, respectively; *Omav* is an omnidirectional drone with tiltable rotors. They vary in size, design, and dynamic features. Moreover, our simulator provides an array of sensors such as IMUs, RGB-D cameras, segmentation sensors, force sensors, and contact sensors, addressing specific requirements for state estimation and perception. For most drones, we also implement the PD controllers introduced in [32]. They provide different control modes, including position/velocity, body rate,

and attitude, and can be used to construct the action spaces for RL.

B. Extending the Drone Models

Certain applications may require additional payloads to be attached. Also, it might be desirable to create multi-drone systems to cope with tasks beyond a single drone’s capability, e.g., to carry a bulky payload. With the flexible simulation framework, a key feature of *OmniDrones* is the support for building and extending a drone model’s physical/logical **configuration** in a highly parameterizable way.

Here, we introduce examples of interesting configurations provided in *OmniDrones*, some of which are shown in Fig. 2. The formed configurations considerably change the system dynamics and therefore present challenges for conventional controller design.

- **Payload & InvPendulum:** A single drone is connected to a weight through a rigid link. The attached weight will alter and destabilize the drone’s dynamics. The arrangement with the payload at the bottom is called **Payload**, while the arrangement with the payload on top is called **InvPendulum**.
- **Over-actuated Platform (Over):** An over-actuated platform consists of multiple drones connected through rigid connections and 2-DoF passive gimbal joints, similar to [33]. Each drone functions as a tiltable thrust generator. By coordinating the movements of the drones, it becomes possible to control their positions and attitudes independently, allowing for more complex platform maneuvers.
- **Transport:** A transportation system comprises multiple drones connected by rigid links. This setup allows them to transport loads that exceed the capacity of a single drone. Drones need to engage in coordinated control and collaboration for stable and efficient transportation.
- **Dragon:** A multi-link transformable drone as described in [34]. Each link has a dual-rotor gimbal module. The links are connected via 2-Dof joint units sequentially. The ability to transform enables highly agile maneuvers and poses a challenging control problem.

C. Domain Randomization

Due to the unavoidable gaps between the simulated dynamics and reality, domain randomization is an important technique for obtaining robust control policies that can be easily transferred and deployed to real-world robots. The parallel simulation capability allows us to efficiently collect diverse randomized trajectories. We list example factors that users can manipulate in Table II and train an adaptive control policy with them in the experiment section.

D. Benchmark Tasks

Based on the simulation framework and utilities introduced above, 15 tasks of varying complexity and characteristics are

TABLE II
RANDOMIZABLE SIMULATION ASPECTS

| Aspect | Examples | Startup | Runtime |
|------------------|--------------------------------|---------|---------|
| Physical config. | rigid connection, object scale | ✓ | X |
| Inertial prop. | mass, inertia, center of mass | ✓ | ✓ |
| Rotor param. | force constant, motor gain | ✓ | ✓ |
| External forces | wind, drag | ✓ | ✓ |

developed for benchmarking. They are formulated as decentralized partially observable Markov Decision Process (Dec-POMDP) [35], where partial observability comes from limited sensor capability and ignorance of other agents’ policy. A **task** specifies the POMDP on top of a certain **configuration**, similar to DMControl [36]. For example, **InvPendulum-Hover** is a task in which the agent (drone) is required to hover an inverted pendulum system introduced before at a desired state. For those that do not have a special configuration, we omit the first part.

According to their formulations and challenges, we divided the task specifications into categories that each might apply to a set of configurations. Here, we list and introduce several representative examples:

- **Hover:** The drone(s) need to drive the system to reach and maintain a target state. This basic task is simple for most configurations except the inherently unstable ones, e.g., **InvPendulum**.
- **Track:** The drone(s) are required to track a reference trajectory of states. The ability to (maybe not explicitly) predict how the trajectory would evolve and plan for a longer horizon is needed for accurate tracking.
- **FlyThrough:** The drone(s) must fly the system through certain obstacles in a skillful manner, avoiding any critical collision. The obstacles are placed such that a long sequence of coherent actions is needed. Such a task often challenges the RL algorithm in exploration.
- **Formation:** A group of drones needs to fly in a specific spatial pattern. This task examines the ability to deal with coordination and credit assignment issues.

Generally, each drone observes kinematic information such as relative position, orientation (in quaternions), and linear and angular velocities. They are concatenated with task-specific information such as the relative position of the target. The action space is such that the drones receive the target rotor throttles which go through a first-order system to obtain the thrusts and moments. For detailed specifications, please refer to the code.

Additionally, by integrating given with controllers, we can transform the action space to allow for the usage of higher-level control commands. We provide 4 control modes (rotor, velocity, rate, and attitude) for ordinary multi-rotor drones.

E. Reinforcement Learning With OmniDrones

It is common for robotics to have complex input and output structures due to, e.g., multi-modal sensory data and heterogeneous agents. Therefore, to have a flexible interface that conveniently handles tensors in batches, we follow TorchRL [23] in the environment specification and use TensorDict as the data

TABLE III
SIMULATION PERFORMANCE (FPS) OF *OMNIDRONES*

| #Envs | Track (1 agents) | Over-Hover (4 agents) |
|-----------|---------------------|--------------------------|
| 1024 Envs | 196074 ± 3754 | 115244 ± 1973 |
| 2048 Envs | 385027 ± 6688 | 204556 ± 7511 |
| 4096 Envs | 732109 ± 10362 | 310027 ± 12233 |

interface. We also provide utilities to transform the observation and action space for common purposes, such as discretizing action space, wrapping a controller, and recording state-action history.

With that, we implement and evaluate various algorithms to provide preliminary results and serve as baselines for subsequent research. They include PPO [37], SAC [38], DDPG [39], and DQN for single-agent tasks and MAPPO [8], HAPPO [40], MADDPG [41], and QMIX [42] for multi-agent ones. More algorithms can be easily added or adopted from other libraries.

IV. EXPERIMENTS

Leveraging the simulation framework and benchmark tasks, our platform can serve as a starting point for subsequent investigations. In this section, we showcase the features and functionalities of *OmniDrones* through experiments and evaluate a range of popular RL algorithms on the proposed tasks. In all the following experiments, we use a simulation time step $dt = 0.016$, i.e., the control policy operates at around 60 Hz.

A. Simulation Performance

We select a single-agent (Track) and a multi-agent (Over-Hover) task, respectively, to demonstrate the efficient simulation capabilities of our simulator under different numbers of environments.

As shown in Table III, the efficient PyTorch dynamics implementation and Isaac Sim’s parallel simulation capability allow *OmniDrones* to achieve near-linear scalability with over 10^5 frames per second (FPS) during rollout collection. The results were obtained on a desktop workstation with NVIDIA RTX4090, Isaac Sim 2022.2.0. The control policy is a 3-layer MLP with 256 hidden units per layer implemented with PyTorch. Note that there are additional computations for the observations/rewards and logging logic besides simulation.

B. Benchmarking RL Baselines

The algorithms are adapted following open-source implementations and modified to be compatible with large-scale training. All runs follow a default set of hyper-parameters without dedicated tuning. All the experiments in this part use direct rotor control.

For single-agent tasks, we evaluate PPO, SAC, DDPG, and DQN using *Hummingbird* and *Firefly*, which have 4 and 6 action dimensions, respectively, and differ in many inertial properties. For DQN, we discretize the action space by quantizing each dimension to its lower and upper bounds. We train each policy in 4096 parallel environments for 125 Million steps and repeat

using 3 different seeds. As suggested in Fig. 3(a), PPO, SAC, and DDPG are all good baselines for most tasks. However, various failures are observed in tasks that require substantial exploration to discover the optimal behavior, i.e., FlyThrough. DQN fails to make progress in all tasks.

Notably, PPO-based agents can be trained within 10–20 minutes. On the other hand, SAC and DDPG generally exhibit better sample efficiency. However, they require a longer wall time, since they need a significantly higher number of gradient steps with more data for each update.

For the more challenging multi-agent coordination tasks, we evaluate MAPPO, HAPPO, MADDPG, and QMIX using *Hummingbird*. We train all algorithms for 150 M steps. The results are shown in Fig. 3(b). The two PPO-based approaches are similar, and both achieve reasonable performance. The failure of MADDPG is potentially due to its exploration strategy being insufficient in multi-agent settings without careful tuning of the exploration noise. To apply the value-decomposition method, QMIX, we discretize the action space as we did for DQN. The results suggest that PPO-based algorithms may serve as robust baselines when the dynamics of the multi-drone system are hard to analyze directly.

C. Drone Models and Controllers

Different drone models can vary substantially in their dynamics and hence the task difficulties. We compare the 4 drone models on three tasks and the results are shown in Fig. 4. Interestingly, although being the most complex (with 12 rotors and 6 tilt units), *OmaV* can achieve comparable or even better performance on the same budget. This reveals the potential of RL in quickly obtaining a control policy for unusual drone models.

Moreover, the choice of action space can have a vital impact on the performance and robustness of learned policies [29]. Considering controllers as transforms of the action space, we verify this point using *Firefly* with the following control modes: (1) *rotor*, i.e., the policy directly commands the target throttle for individual rotors; (2) *velocity*, where the policy outputs the target velocity and yaw angle; (3) *rate*, where the policy outputs the target body rates and collective thrust; (4) *attitude*, where the policy outputs the target attitude and collective thrust. The actions are scaled and shifted to a proper range for each approach.

As shown in Fig. 5, direct rotor control and rate control consistently give the best performance, while velocity control appears to be insufficient for tasks that demand more fine-grained control. We remark that tuning the controller parameters can lead to better parameters. Nonetheless, the results suggest that a relatively low-level action space, despite being more subtle to transfer, is still necessary for agile and accurate maneuvers.

D. Domain Randomization and Adaptation

The ability to compensate for the mismatch between simulation and reality and adapt accordingly is vital for policy deployment. We showcase training of an adaptive control policy similar to that presented in [5], using *Hummingbird*. The simulation and drone parameters being randomized are shown in Table IV.

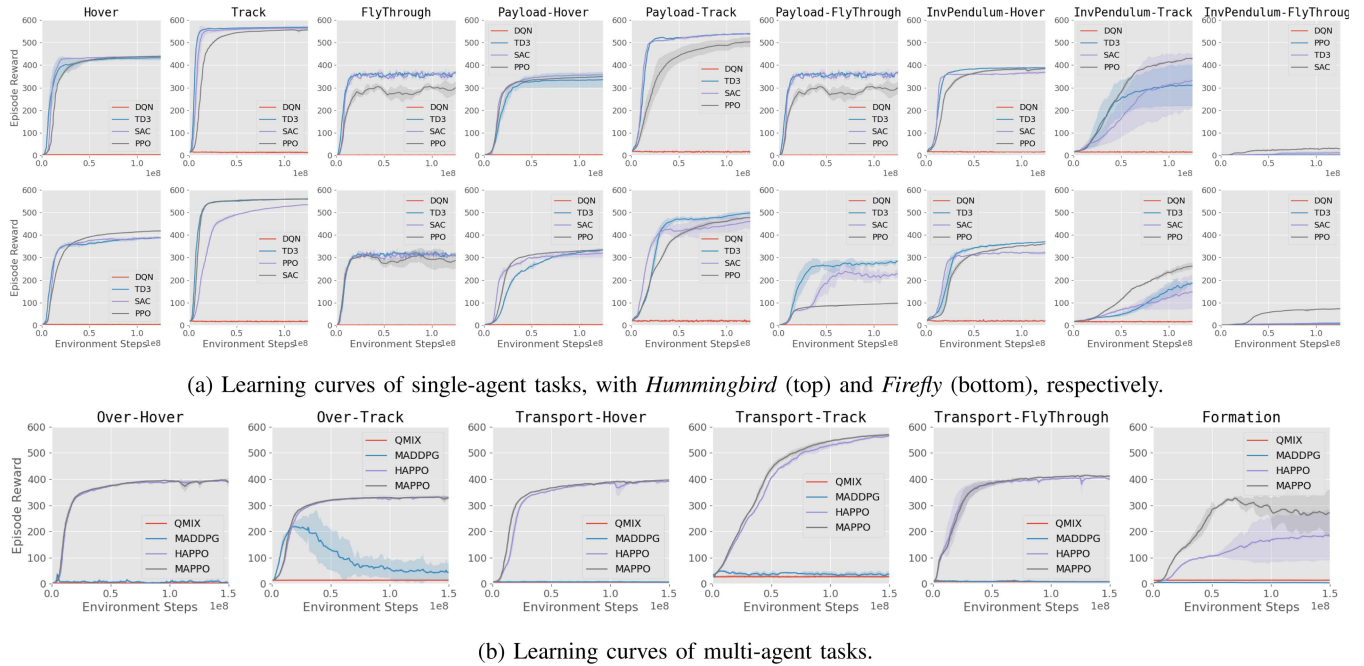


Fig. 3. Benchmarking results.

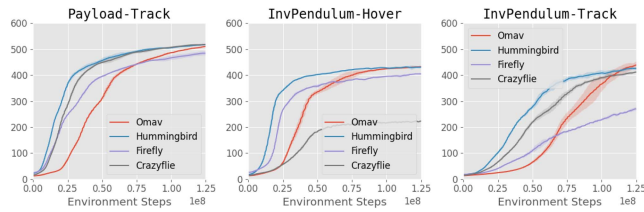


Fig. 4. Comparison of different drone models on three selected tasks.

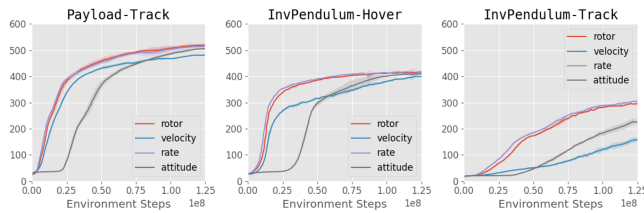


Fig. 5. Comparison of different choices of action space.

 TABLE IV
RANDOMIZED PARAMETERS

| Parameter | Range | Parameter | Range |
|-----------|----------------|---------------|----------------|
| *mass | [0.26, 1.74] | * k_f | [0.5556, 2.23] |
| *inertia | [0.026, 1.974] | * k_m | [0.625, 2.5] |
| com | [-0.05, 0.05] | *payload mass | [0.01, 1.0] |
| τ | [0.2, 1.0] | payload z | [-0.1, 0.1] |

A star (*) indicates the range is relative to the default value.

We additionally attach a payload with a degree of freedom along the z-axis. The values of these parameters, which we term “privileged observations”, are only available in the simulation. The adaptive policy is trained to infer these parameters from its observation-action history on the fly. To do this, we first train an expert policy with access to the privileged observations and

 TABLE V
COMPARISON OF PERFORMANCE EVALUATED IN RANDOMIZED ENVIRONMENTS, NORMALIZED BY THE EXPERT

| Policy | Expert | Baseline | GRU | Adaptive |
|--------|-----------------|------------------|------------------|------------------|
| Return | 1.0 ± 0.006 | 0.82 ± 0.017 | 0.89 ± 0.008 | 0.97 ± 0.008 |
| Error | 1.0 ± 0.011 | 1.24 ± 0.082 | 1.19 ± 0.011 | 0.96 ± 0.007 |

subsequently train an adaptation module to recover the features from past trajectories. We compare the episode reward (return) and tracking errors of the baseline, a recurrent policy using a GRU [43], the expert, and the adaptive policy. Refer to [5], [44] for more details. All policies are implemented based on PPO.

As shown in Table V, the gap between the expert policy and the baseline indicates the importance of the privileged observations. Adding a GRU helps by incorporating history observations to better infer the state information. The adaptive policy, which explicitly adapts to the environment by identifying the privileged information, achieves comparable performance to the expert. We believe the results could serve as a reference and starting point to facilitate sim-to-real transfer.

V. CONCLUSION AND FUTURE WORK

In this letter, we presented the *OmniDrones*: a platform for conducting RL research on multirotor drone control. Leveraging the parallel simulation capabilities of more GPUs, *OmniDrones* provides efficient and flexible simulation and a suite of RL tasks for multi-rotor drones. Through experiments, we demonstrate the features of the proposed platform and offer initial results on the tasks. We hope *OmniDrones* serves as a good starting point toward building more powerful drone systems regarding control and system design with reinforcement learning. In the future, we will provide long-term support and continue our development

to provide utilities for sim-to-real deployment. While this work focuses more on low-level control in an end-to-end setting, more complex and realistic scenarios, and higher-level tasks will be incorporated to complete the picture.

ACKNOWLEDGMENT

The abstractions and implementation of *OmniDrones* were inspired by ISAAC ORBIT [14]. Some of the drone models (assets) and controllers are adopted from or heavily based on the RotorS [18] simulator. Special thanks to Miranda Shi for picking the overview image and for her mental support during the refinement of this paper.

REFERENCES

- [1] J. Hilf and K. Umbach, "The commercial use of drones," *Comput. Law Rev. Int.*, vol. 16, no. 3, pp. 65–71, 2015, doi: [10.9785/crl-2015-0302](https://doi.org/10.9785/crl-2015-0302).
- [2] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 2, no. 4, pp. 2096–2103, Oct. 2017.
- [3] S. Belkhal, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-based meta-reinforcement learning for flight with suspended payloads," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 1471–1478, Apr. 2021.
- [4] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1205–1212.
- [5] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 1263–1269.
- [6] O. Vinyals et al., "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [7] C. Berner et al., "Dota 2 with large scale deep reinforcement learning," 2019, *arXiv:1912.06680*.
- [8] C. Yu et al., "The surprising effectiveness of PPO in cooperative multi-agent games," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 24611–24624.
- [9] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Proc. 5th Conf. Robot Learn.*, 2022, pp. 91–100.
- [10] Y. Chen et al., "Towards human-level bimanual dexterous manipulation with reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 5150–5163.
- [11] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 895–902, Apr. 2018.
- [12] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "RLBench: The robot learning benchmark & learning environment," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 3019–3026, Apr. 2020.
- [13] V. Makoviychuk et al., "Isaac gym: High performance GPU-based physics simulation for robot learning," in *Proc. Neural Inf. Process. Syst. Track Datasets Benchmarks*, J. Vanschoren and S. Yeung, Eds. 2021, vol. 1.
- [14] M. Mittal et al., "ORBIT: A unified simulation framework for interactive robot learning environments," *IEEE Robot. Automat. Lett.*, vol. 8, no. 6, pp. 3740–3747, Jun. 2023.
- [15] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Proc. Conf. Robot Learn.*, 2021, pp. 1147–1157.
- [16] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Proc. Field Serv. Robot.*, 2017, pp. 621–635.
- [17] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2004, pp. 2149–2154.
- [18] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—A modular Gazebo MAV simulator framework," in *Robot Operating System (ROS)*. Berlin, Germany: Springer, 2016, pp. 595–625.
- [19] G. Silano and L. Iannelli, "Crazys: A software-in-the-loop simulation platform for the crazyflie 2.0 nano-quadcopter," in *Robot Operating System (ROS)*. Berlin, Germany: Springer, 2020, pp. 81–115.
- [20] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—A gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 7512–7519.
- [21] E. Coumans and Y. Bai, "Pybullet, a Python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: <https://pybullet.org/wordpress>
- [22] M. Kulkarni, T. J. Forgaard, and K. Alexis, "Aerial Gym—Isaac Gym simulator for aerial robots," 2023, *arXiv:2305.16510*.
- [23] A. Bou et al., "Torchrl: A data-driven decision-making library for pytorch," 2023. [Online]. Available: <https://pytorch.org/rl>
- [24] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 528–535.
- [25] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2016.
- [26] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for UAV attitude control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 2, pp. 1–21, 2019.
- [27] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 4224–4230, Oct. 2019.
- [28] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Proc. Conf. Robot Learn.*, 2018, pp. 133–145.
- [29] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 10504–10510.
- [30] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 6941–6948.
- [31] G. Silano, E. Aucone, and L. Iannelli, "Crazys: A software-in-the-loop platform for the crazyflie 2.0 nano-quadcopter," in *Proc. IEEE 26th Mediterranean Conf. Control Automat.*, 2018, pp. 1–6.
- [32] T. Lee, M. Leok, and N. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proc. IEEE Conf. Decis. Control*, Jan. 2010, pp. 5420–5425.
- [33] Y. Su et al., "Downwash-aware control allocation for over-actuated UAV platforms," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 10478–10485.
- [34] M. Zhao, T. Anzai, F. Shi, X. Chen, K. Okada, and M. Inaba, "Design, modeling, and control of an aerial robot DRAGON: A dual-rotor-embedded multilink robot with the ability of multi-degree-of-freedom aerial transformation," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1176–1183, Apr. 2018.
- [35] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Berlin, Germany: Springer, 2016.
- [36] Y. Tunyasuvunakool et al., "Deepmind control suite," *Softw. Impacts*, vol. 6, 2018, Art. no. 100022.
- [37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
- [38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
- [39] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
- [40] J. G. Kuba et al., "Trust region policy optimisation in multi-agent reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–9. [Online]. Available: <https://openreview.net/forum?id=EcGGfKNTxdJ>
- [41] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 1–9.
- [42] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 7234–7284, 2020.
- [43] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.
- [44] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," 2021, *arXiv:2107.04034*.