Towards Task-Oriented Flying: Framework, Infrastructure, and Principles

Kangyao Huang[1,*], Hao Wang[2,*], Jingyu Chen[3], Jintao Chen[1], Yu Luo[1], Di Guo[4], Xiangkui Zhang[2], Xiangyang Ji[1,†], and Huaping Liu[1,†]

[1]Tsinghua University
[2]Dalian University of Technology
[3]Institue of Software, Chinese Academy of Sciences
[4]Beijing University of Posts and Telecommunications
[*]Contribute equally
[†]Corresponding author

arXiv:2504.15129v2 [cs.RO] 9 Dec 2025

ABSTRACT

Deploying robot learning methods to aerial robots in unstructured environments remains both challenging and promising. While recent advances in deep reinforcement learning (DRL) have enabled end-to-end flight control, the field still lacks systematic design guidelines and a unified infrastructure to support reproducible training and real-world deployment. We present a task-oriented framework for end-to-end DRL in quadrotors that integrates design principles for complex task specification and reveals the interdependencies among simulated task definition, training design principles, and physical deployment. Our framework involves software infrastructure, hardware platforms, and open-source firmware to support a full-stack learning infrastructure and workflow. Extensive empirical results demonstrate robust flight and sim-to-real generalization under real-world disturbances. By reducing the entry barrier for deploying learning-based controllers on aerial robots, our work lays a practical foundation for advancing autonomous flight in dynamic and unstructured environments.

1 Introduction

Aerial robots have emerged as revolutionary technologies that drive the development of the low-altitude economy[1, 2]. Their applications span critical domains including emergency rescue operations, precision agriculture, and urban air mobility, with the boosting of global drone market. This explosive growth is fueled by advancements in autonomous capabilities that enable complex operations in unstructured environments. Recent advances in lightweight sensing technologies and edge computing further accelerate their deployment across industries.

Deep reinforcement learning (DRL) has demonstrated superior performance in time-critical tasks including drone racing[3, 4, 5, 6, 7, 8], dynamic obstacle avoidance[9, 10], rapid trajectory tracking[11, 12, 13], agile flight[7, 14, 15, 16], as well as navigation and planning[7, 10, 17], achieving unprecedented perception and decision-making efficiency. Although many studies have provided methods and skills[6, 7, 11, 12, 14, 15, 16, 18, 19, 20] to improve algorithms and give experience on solving specific tasks, the principles of training and deploying a policy by DRL have not been summarized and refined, especially from a comprehensive and thorough perspective covering simulation, deployment, and their bridge. Most importantly, existing quadrotor works based on robot learning methods describe a lot about methodology but lack guidelines, principles, as well as underlying tools for implementation, and there are few open-source toolkits to facilitate the DRL training and transfer. These absences create significant barriers to practical application: researchers often struggle and feel confused about where to begin when designing and deploying a new task-oriented embodied flight task.

Our research aims to consolidate existing end-to-end DRL methods for aerial robots while incorporating our extensive experimentation and trials, distilling general task design techniques, and summarizing training and deployment guidelines for quadrotor DRL, thereby facilitating broader adoption by researchers. Specifically, we refine principles for designing successful quadrotor DRL tasks and consolidate the factors that affect final performance during the sim-to-real process. Additionally, to support our work and facilitate broader research efforts, we propose a fully open-sourced and robust infrastructure and workflow for quadrotor DRL, including simulation environments, software, firmware, and hardware platforms. We also introduce comprehensive experiments across several different types of aerial robot learning tasks. Our proposed platform supports training and sim-to-real transfer in a remarkably short period of time, significantly accelerating algorithmic iteration.

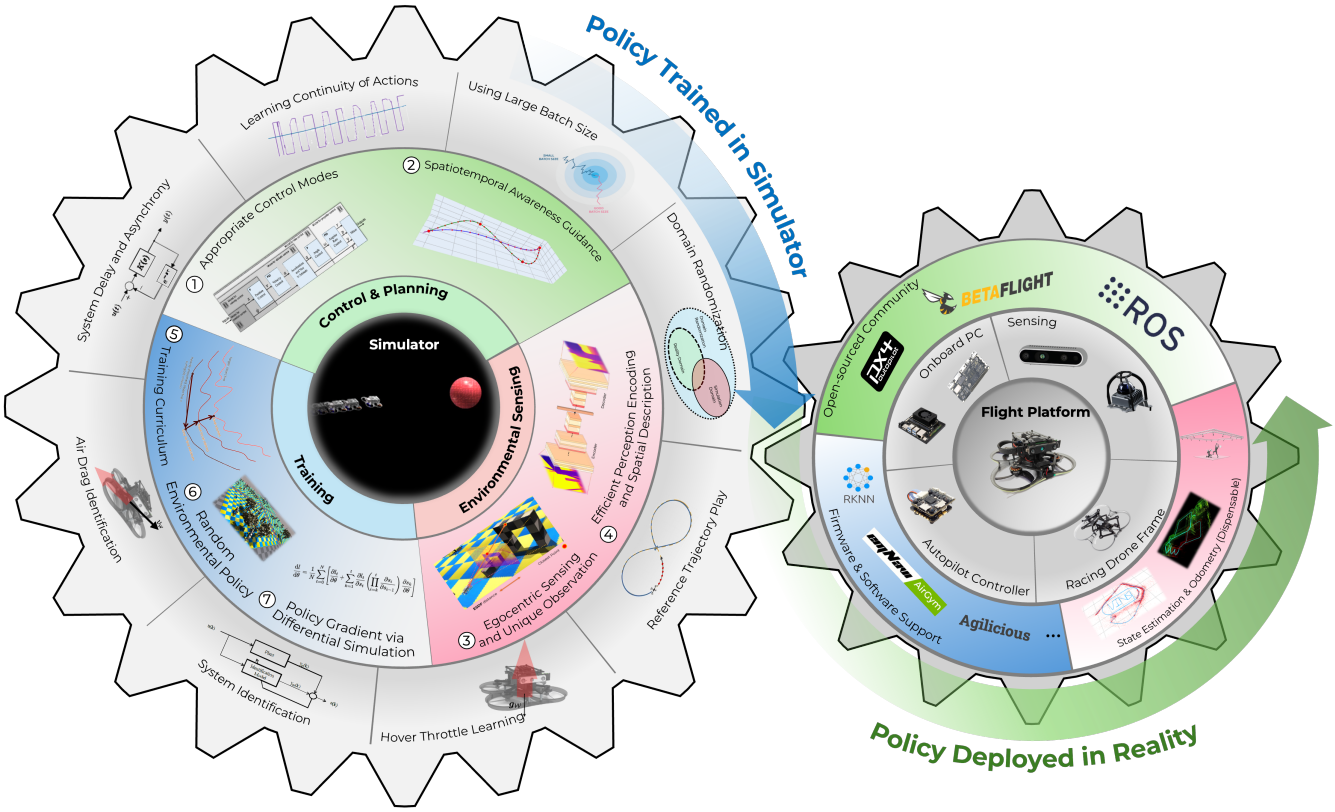


Figure 1. A DRL-based sim-to-real task-oriented framework. The left gear serves as the driving gear and represents simulation-related items, including simulator core, task principles middle layer, and sim-to-real techniques outermost layer. The right gear is the driven layer, involving hardware devices as inner layer, as well as firmware and software as outer layer.

2 Results

2.1 Framework

The task-oriented framework is illustrated in Fig.1. A pair of gears is employed as a metaphor to illustrate the interdependent relationship between simulation-based training and real-world deployment in aerial robot DRL. The left gear functions as the driving gear, symbolizing the training of policies within the simulator, while the smaller right gear serves as the driven gear, representing the deployment of these policies onto quadrotors in the real world.

Simulator is the core of the driving gear, for training a policy. The interactive simulation environment, built upon physics engines and rendering systems, serves as the training arena for DRL. The physics engine simulates and approximates Newton's physical laws, forming the dynamical foundation for all complex flight behaviors. Concurrently, environmental perception and pattern recognition inherently depend on rendering verisimilitude, where graphical frame-by-frame progression drives the interactive learning process. Built upon the simulator, task-driven policy training requires systematic design considerations. While end-to-end quadrotor missions can encompass diverse task typologies, their design framework must address three fundamental dimensions: Control & Planning, Environmental Sensing, and Training. This might involve multiple aspects, including command hierarchy selection, reward-punishment mechanisms, training environment design, and algorithm adaptation, each involving sophisticated techniques and diverse approaches. At this conceptual level, we systematically abstract and distill seven fundamental design principles addressing the aforementioned three dimensions, relying on both established literature and extensive empirical validation from our own practical works: ① appropriate control modes ② spatiotemporal awareness guidance ③ egocentric sensing and unique observation ④ efficient perception encoding and spatial description ⑤ training curriculum ⑥ random environmental policy ⑦ policy gradient via differential simulation. The detailed explanation of these principles will be comprehensively elaborated later. The outermost layer of driving gear is factors and tricks that directly influence the performance of sim-to-real deployment.

The driven gear can be conceptualized as the standard architecture of an autonomous unmanned system, embodying the following fundamental characteristics: platform, hardware device with firmware, and software application. As the validation

platform for this framework and workflow, the self-developed aerial platform `X152b` is centrally positioned within the gear diagram. Hardware components such as the onboard PC, flight controller, and sensors are integrated into this platform. Sensing by vision[7, 8, 21, 22, 23, 24, 25, 26, 27], LiDAR[26] and neuromorphic vision[3, 28, 29] as well as other perception devices can also be employed. At the outermost layer, the associated firmware & software, state estimation algorithms, and open-sourced communities serve as the medium for transferring and deploying a neural network onto the physical platform, and meshing with the outermost layer of driving gear.

2.2 Infrastructure & Workflow

To facilitate end-to-end policy training and deployment, we integrate insights from numerous existing RL studies along with our practical experience, and propose a general, task-oriented workflow for rapidly training and deploying DRL-based aerial robot flight policies. The pipeline of our platform is illustrated in Fig. 2, where we have implemented two separate closed-loops for training and deployment, tailored for simulation and real-world environments respectively. Our workflow pipeline includes:

- DRL simulation platform `AirGym`[1]: a large-scale parallel environment simulation based on IsaacGym[30], where we implement four DRL tasks: `tracking`, `avoidance`, `target hitting`, `planning`, as well as the most basic flight control task `hovering`.
- Parallel flight controller `rlPx4Controller`[2]: a parallel simulation flight geometric controller implemented in `C++` and encapsulated with `Python` interface, strictly aligned with the logic and parameters of the PX4 autopilot. It flexibly provides control at various levels, including position & yaw (*PY*), linear velocity & yaw (*LV*), collective thrust & attitude angle (*CTA*), and collective thrust & body rate (*CTBR*).
- Edge-side onboard inference runtime software `AirGym-Real`[3]: an onboard sim-to-real module compatible with `AirGym`, enabling direct loading of pretrained models, supporting onboard visual-inertial pose estimation, `ROS` topic publishing, and one-shot scripted deployment.
- Control bridge `control_for_gym` [4]: a middleware layer based on MAVROS for forwarding control commands at various levels to PX4 autopilot. It includes a finite state machine to facilitate switching between DRL models and traditional control algorithms.
- Quadrotor platform `X152b`: an open-sourced onboard sensing and inference flight platform, shown in Fig.2, where we integrate computation and perception devices and provide the mechanical model for simulation alignment.

Table 1. Time consumption for training a policy: training from scratch.

Task Name	hovering	tracking	avoidance	target hitting	planning
Comsumption (min)	2.4465 ± 0.1878	8.1619 ± 0.3811	5.3834 ± 0.4229	3.8277 ± 0.2125	7.1009 ± 0.3964

Our workflow demonstrates robust and rapid policy training-to-deployment capabilities, achieving mission-ready performance within minutes. As shown in Tab.1, the time consumptions of training a task in simulation among 10 trials are listed. In practice, the trained model is directly transferred to practical devices to realize zero-shot sim-to-real. Our training runs on a workstation with Intel Core i9-13900K processor and an NVIDIA RTX4090 graphic card, and the policy is deployed to an onboard computer with RK3588s processor. Since there is no collision during the training and flying, we can simply copy policies from the simulator to real-world robots without any fine-tuning which has been proved to be feasible in previous studies[5]. Additionally, our efficient workflow pipeline software offers one-click solutions for complex engineering implementations like robot state estimation and perception pre-processing, enabling immediate deployment and inference. Furthermore, the workflow is highly scalable; the modules and tools within the pipeline are not limited to the specific hardware platform we used. Features and usage of subsystems are described in supplementary materials.

[1] https://github.com/emNavi/AirGym
[2] https://github.com/emNavi/rlPx4Controller
[3] https://github.com/emNavi/AirGym-Real
[4] https://github.com/emNavi/control_for_gym

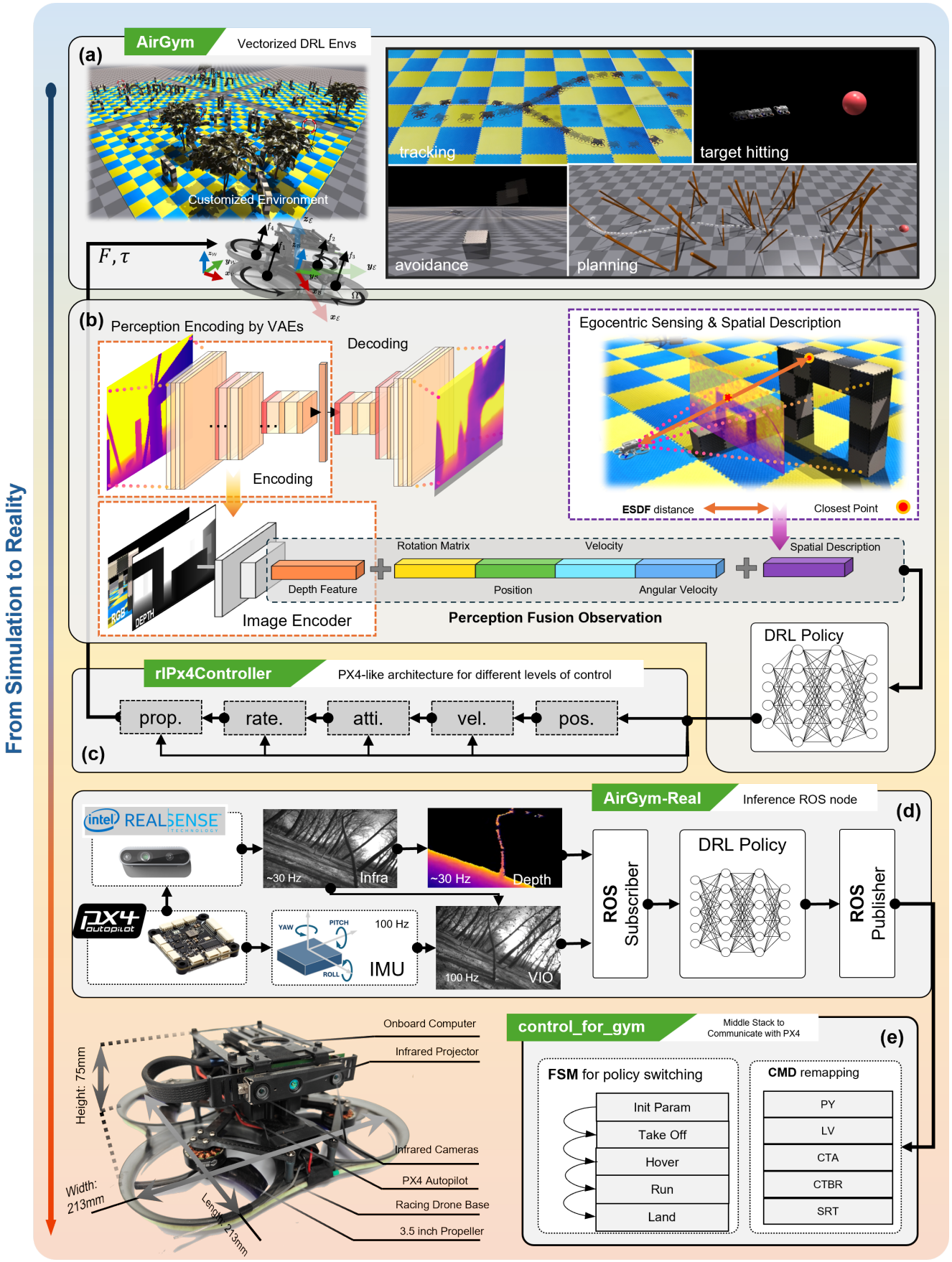

From Simulation to Reality
(a)
AirGym
Vectorized DRL Envs
Customized Environment
tracking
target hitting
avoidance
planning
F, τ
(b)
Perception Encoding by VAEs
Decoding
Encoding
Egocentric Sensing & Spatial Description
ESDF distance
Closest Point
Rotation Matrix
Velocity
Spatial Description
Depth Feature
Position
Angular Velocity
Image Encoder
Perception Fusion Observation
DRL Policy
rlPx4Controller
PX4-like architecture for different levels of control
prop.
rate.
atti.
vel.
pos.
(c)
AirGym-Real
Inference ROS node
(d)
~30 Hz
Infra
~30 Hz
Depth
100 Hz
IMU
100 Hz
VIO
ROS Subscriber
DRL Policy
ROS Publisher
Onboard Computer
Infrared Projector
Infrared Cameras
PX4 Autopilot
Racing Drone Base
3.5 inch Propeller
Height: 75mm
Width: 213mm
Length: 213mm
control_for_gym
Middle Stack to Communicate with PX4
(e)
FSM for policy switching
Init Param
Take Off
Hover
Run
Land
CMD remapping
PY
LV
CTA
CTBR
SRT

Figure 2. The proposed pipeline and the relationships between each part. In (a) `AirGym`, we provide four classical tasks: `tracking`, `avoidance`, `target hitting`, and `planning`. (b) illustrates the sensing processing and features fusion during the training phase. The left image in (b) shows visual input is compressed into a vector using VAEs. The depth image is reconstructed by using self-supervised learning, and the encoding part is used as the image encoder in the DRL loop. The right image in (b) illustrates ESDF distance which is used as an efficient spatial description, where we use the minimum value in depth array is used as the ESDF distance. (c) shows all different control modes that can be selected. (d) is the onboard inference module that runs on the flight platform. It is worth mentioning that the VIO part is not indispensable in our experiments because global states estimation is not necessary in our Ego-centric design. (e) builds the bridge between PX4 flight controller and the algorithm. Finally, bottom-left device is our flight platform `X152b`.

2.3 Principles in Training Design

Advances in Sim-to-Real for aerial robots increasingly enable tackling more complex tasks. However, designing effective task-oriented DRL training remains challenging. Poorly defined training objectives may misguide the learner into blind alleys, reducing the problem to unsolvable reward engineering; conversely, well-crafted training acts as a scaffold, steering learning toward generalizable and robust policies. This paradigm shift necessitates a principled approach to task design. In the following, we outline the fundamental principles to guide effective training design.

2.3.1 Control & Planning

The most fundamental question in learning-based control is deceptively simple: what exactly should the agent learn to control? The answer lies in selecting an appropriate level of abstraction for the control command—ranging from low-level motor thrusts to high-level position targets. Inspired by classical hierarchical control theory, which organizes control commands hierarchically, we adopt a similar strategy and define five distinct control modes: *PY*, *LV*, *CTA*, *CTBR*, and *SRT*. *CTBR* emerges as a robust mode, striking a balance between expressiveness and tractability. It performs consistently well across a range of tasks, indicating strong generalization capabilities. Meanwhile, *LV* mode demonstrates faster convergence in tasks with looser precision requirements, making it a practical alternative when high-frequency control is unnecessary. Notably, the choice of control mode directly influences shapes the agent's action space and physical embodiment, shaping both the learning space and the physical embodiment of policy execution. Thus, it is crucial to ① select appropriate control mode that matches the task at hand. For instance, while *SRT* offers the most sensitive control, it poses significant learning challenges due to the complexity of the underlying dynamics. In contrast, mid-level abstractions such as *CTBR* or *LV* strike a more favorable balance between controllability and learning efficiency.

Planning complex trajectories poses significant challenges, while ② spatiotemporal awareness guidance can facilitate it. Whether through incorporating desired poses as prior knowledge that embeds useful priors into the learning process[31] or providing environmental spatial descriptors in the state space as a temporary target[5, 32], both approaches can effectively enhance flexibility when handling complex trajectories. Whether these cues come from human commands or perception pipelines, their value lies in anchoring the agent's decision-making in both space and time, allowing the drone to plan ahead—without relying on brittle rule-based controllers. Therefore, when an agent gets lost in the "reward engineering", it is important to rethink whether sufficient guidance has been provided to the agent, and whether this guidance has been properly incorporated into the state space.

2.3.2 Environmental Sensing

In most traditional and classical quadrotor control and navigation scenarios, researchers typically use (i) the takeoff point as the origin of coordinate and (ii) adopt a North-East-Down (NED) coordinate frame. Because a locally constructed coordinate system like this allows the mathematical formulation of global/local path planning problems to be clearly defined from a third personal view, which is well-suited for control and planning approaches based on optimization but often hinders training in more embodied scenarios. The ③ egocentric sensing, where the drone perceives the world from its own point of view, enhancing learning stability and efficiency. We define an ego-centric coordinate system $\mathscr{E}$: the origin of this coordinate is fixed on the mass point of quadrotor, using the forward-facing direction of vehicle as the positive $\boldsymbol{x}_{\mathscr{E}}$. $\boldsymbol{z}_{\mathscr{E}}$ points vertically upward (toward the sky), and the $\boldsymbol{y}_{\mathscr{E}}$ is determined according to the right-hand rules. Ego-centric frame $(\boldsymbol{x}_{\mathscr{E}}, \boldsymbol{y}_{\mathscr{E}}, \boldsymbol{z}_{\mathscr{E}})$ is similar to the body frame $(\boldsymbol{x}_{\mathscr{B}}, \boldsymbol{y}_{\mathscr{B}}, \boldsymbol{z}_{\mathscr{B}})$ while the difference is $\boldsymbol{z}_{\mathscr{E}}$ in ego-centric frame always points to the sky. By anchoring perception to the drone's forward-facing frame, the system avoids the confusion introduced by arbitrary global positions or orientations. In practice, designing the observation space with respect to the agent's own perspective can greatly enhance learning stability and efficiency. This aligns not only with biological inspiration—humans and animals rely on self-centered perception—but also with practical benefits: observations become more compact, generalizable, and better suited for transfer to real-world deployments. Additionally, emphasize the importance of using unique state representations to ensure a one-to-one mapping from observations to actions. For example, while quaternions are commonly used for attitude representation. Since quaternions

are not unique representations of the same rotation, we recommend using rotation matrices to represent orientation. This helps the neural network learn an exclusive mapping from input to output, and it makes the training converge faster.

Moreover, what a drone perceives is only as valuable as its ability to interpret it, which highlights the importance of ④ efficient perception encoding and spatial description. While raw depth images or point clouds offer rich environmental information, they are often too high-dimensional and noisy for direct use in learning policies. Instead, feature extraction—either through hand-crafted heuristics or learned encoders—can distill relevant information into compact latent variables. Probabilistic models like variational autoencoders (VAEs) are especially promising [25]. They can produce low-dimensional features that still preserve critical environmental cues. These encodings also offer smooth variations and a structured latent space, which are beneficial for stable training. When used with additional spatial priors, such as Euclidean Signed Distance Fields (ESDFs), they enable reward functions that naturally reflect obstacle proximity—encouraging safe and agile navigation [26]. Therefore, both perception encoding and spatial description are indispensable: the former enhances the effective utilization of perceptual information, while the latter bridges perception and reward design.

2.3.3 Training

DRL training for drones must address the complexity of tasks. Many tasks involve intricate logical dependencies or require the integration of multiple skills that are hard to learn simultaneously, like dodging dynamic obstacles while hovering. A practical solution is to employ ⑤ training curriculum, such as deep hierarchical learning (DHL), to decompose the task into manageable sub-problems[33]. For example, a system can first learn a perception module that maps depth inputs to compact feature representations, and then train a separate policy to act based on those features to improve training stability and overall performance.

⑥ Random environmental policy is essential for improving the generalization capability of learned policies. For example, such as tracking arbitrary trajectories, drone racing across various tracks, or navigating in unknown scenes, the ability of a model to adapt to diverse conditions is a fundamental requirement. Randomization strategies have been shown to be effective in the previous works, like environment policy in study [34] and the unified tracking policy in SimpleFlight [11].

In more complex tasks, ⑦ differential simulation-based learning yields more efficient and intelligent results[7]. This approach involves designing the RL process—especially policy updates—as an end-to-end differentiable system, enabling the entire learning process to be efficiently optimized through gradient descent. Traditional RL often relies on sampling and the design of reward functions to obtain gradients, where the optimization objectives are often non-differentiable or suffer from high gradient estimation noise. However, differentiable reinforcement learning enables direct backpropagation of gradients, leading to lower estimation noise and more stable learning dynamics[15]. Therefore, rather than designing complex reward functions, it may be more effective to consider how to construct loss functions directly from differentiable components.

2.4 Task-oriented Flying

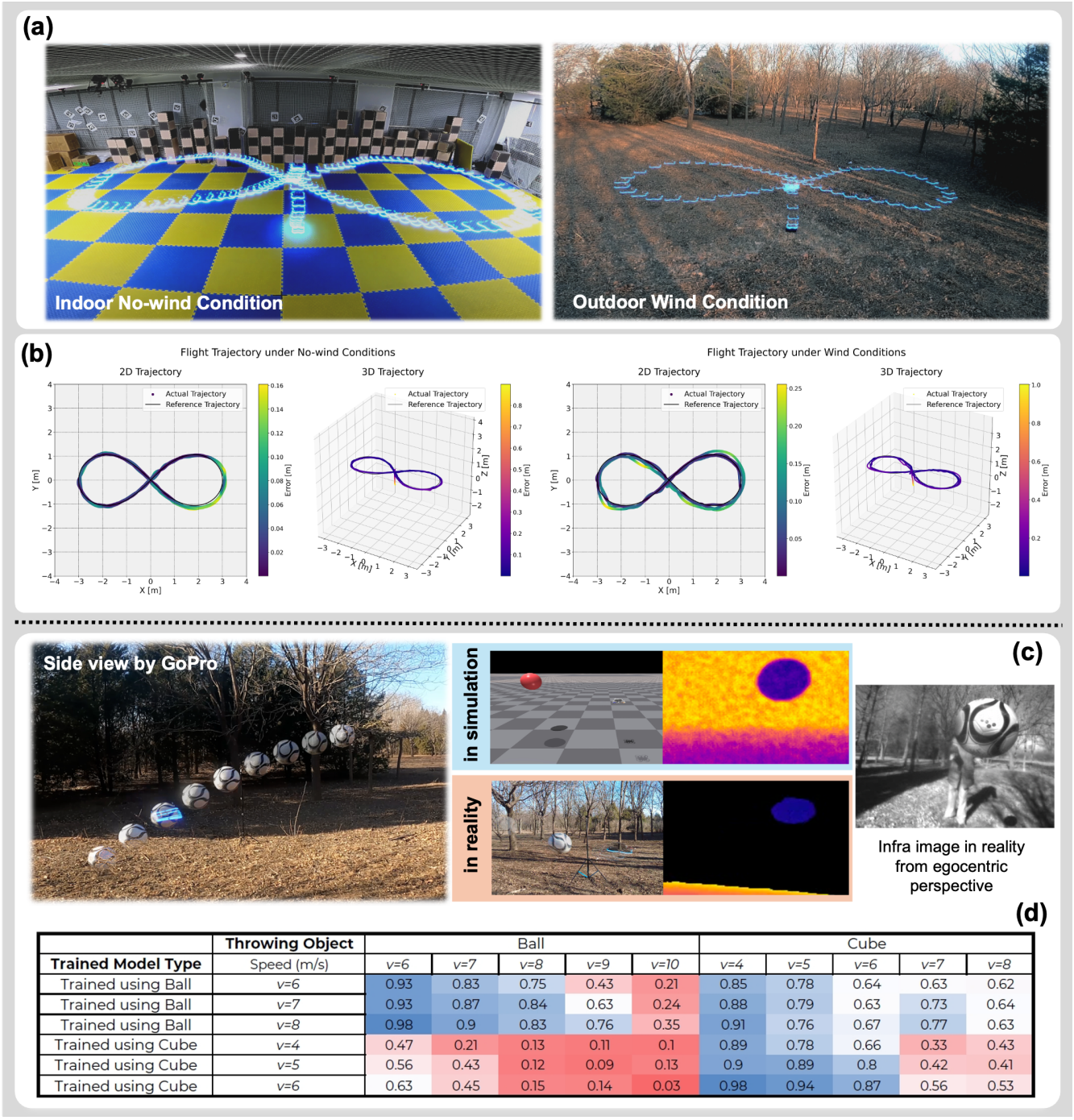
We implement four classic RL/DRL tasks and build a set of quadrotor experiments on top of `AirGym`: `tracking`, `avoidance`, `target hitting`, and `planning`, as shown in Fig.2(a). All tasks support end-to-end policy training across various control levels, allowing us to select and integrate specific levels where we wish to train and maintain classical control methods. `tracking` is a classical drone robot learning task: the quadrotor follows the given trajectory with a desired speed. Herein we primarily conduct tracking of a lemniscate trajectory flight. Task `avoidance` is a new scenario not addressed in previous RL tasks: the quadrotor starts at a fixed point, then an obstacle will be initialized at a random position with an appropriate speed to hit the quadrotor. The quadrotor needs to learn a policy to avoid the obstacle and keep hovering at the original position. `target hitting` is a waypoint navigation task: the quadrotor flies at a very high speed, rapidly dashing toward the target. Finally, we propose a `planning` task: a cluster of trunks is randomly generated at the beginning of an episode, and a quadrotor needs to fly through the forest and reach the target.

In our experiments, we evaluate the results of all different control types. The *CTBR* control mode is recommended as its effectiveness has been studied and shown to outperform others [18, 31]. Our experimental results also support this perspective: although *CTBR* may not offer the fastest training speed or the highest reward, its flight behavior is more stable and natural, based on real-world observations. Therefore, we uniformly use the *CTBR* mode for sim-to-real experiments.

2.4.1 Tracking with Unknown Disturbance

Tracking a lemniscate is a commonly used baseline for demonstrating control performance. Here, we conduct lemniscate tracking experiments under both ideal indoor and dim, windy outdoor conditions, as shown in Fig.3(a)(b) respectively. Meanwhile, we present the tracking performance at two different tracking speeds.

As illustrated in Fig. 3(b), we use the Mean Euclidean Distance (MED) to evaluate the tracking errors. The results report a maximum 0.09 ± 0.07 meters of MED under no-wind conditions. However, tracking lemniscate under moderate breeze conditions presents significant challenges. The tracking performance is shown in Fig. 3(b). Because the wind direction and strength in the outdoor environment are both uncertain, the disturbances can be considered approximately random. Thus, the



	Throwing Object	Ball					Cube				
Trained Model Type	Speed (m/s)	*v=6*	*v=7*	*v=8*	*v=9*	*v=10*	*v=4*	*v=5*	*v=6*	*v=7*	*v=8*
Trained using Ball	*v=6*	0.93	0.83	0.75	0.43	0.21	0.85	0.78	0.64	0.63	0.62
Trained using Ball	*v=7*	0.93	0.87	0.84	0.63	0.24	0.88	0.79	0.63	0.73	0.64
Trained using Ball	*v=8*	0.98	0.9	0.83	0.76	0.35	0.91	0.76	0.67	0.77	0.63
Trained using Cube	*v=4*	0.47	0.21	0.13	0.11	0.1	0.89	0.78	0.66	0.33	0.43
Trained using Cube	*v=5*	0.56	0.43	0.12	0.09	0.13	0.9	0.89	0.8	0.42	0.41
Trained using Cube	*v=6*	0.63	0.45	0.15	0.14	0.03	0.98	0.94	0.87	0.56	0.53

Figure 3. (a)(b) show the tracking performance under different conditions. We set a lemniscate trajectory as tracking reference and illustrate the cumulative flight path under indoor no-wind and outdoor windy environments. (c) is dynamic obstacle avoidance task by end-to-end DRL Sim-to-Real using depth sensing. The quadrotor with blue halo tries to dodge a fast flying football. Our experiment is conducted in a wild GNSS-deny environment and only relies on onboard sensing for perception and localization. The speed of flying ball reaches about $15\mathrm{m}\cdot\mathrm{s}^{-1}$. (d) shows the generalization experiments by using different throwing objects with various speed. One hundred episodes are evaluated for each experiment and success rate are recorded.

Table 2. Comparison between different conditions of lemniscate tracking.

Items	Speed	Relative Tracking Errors	Success Rate
SimpleFlight (Baseline)	1.6m/s	0.028 ± 0.0	–
	2.5m/s	0.051 ± 0.002	–
Indoor with no wind	1.6m/s	$0.022 \pm 0.013 \searrow$	1
	2.5m/s	$0.03 \pm 0.006 \searrow$	0.9
Outdoor with no wind	1.6m/s	$0.023 \pm 0.015 \searrow$	1
	2.5m/s	$0.032 \pm 0.024 \searrow$	0.8
Outdoor with wind	1.6m/s	0.073 ± 0.043	0.3
	2.5m/s	0.11 ± 0.041	0.2
Outdoor with wind & DR	1.6m/s	$0.043 \pm 0.028 \searrow$	$0.6 \nearrow$
	2.5m/s	$0.082 \pm 0.035 \searrow$	$0.5 \nearrow$

trajectory is irregular, and it deviates from the reference for most of the process. The wind disturbance directly affects the quadrotor's attitude, decreasing the control accuracy of the neural network. On the other hand, persistent wind disturbances may disrupt the quadrotor's control, making it difficult to follow up the desired playing trajectory and leading to a crash. When delay or ahead of trajectory playback occurs, the neural network encounters states that it has never seen during training, which can lead to unpredictable outcomes [35]. To conquer this, domain randomization can significantly improve the robustness, as shown in Tab.2, by adding the temporal margin [11] and randomization on observation. Furthermore, artificially adding Gaussian-distributed external force disturbances to simulate wind as a kind of DR method can also increase the success rate. The results show that the success rate nearly doubles after DR is applied.

To make the results more persuasive, we make a comparison with SimpleFlight[11]. Due to the different scales of flight platforms, we use the relative proportion of tracking error for a fairer comparison. Note that the maximum edge length of the trajectory in SimpleFlight is limited to $(-1,1)$, whereas in our experiment it reaches $(-3,3)$. In our experiment, the relative tracking error achieves a lower average compared to the baseline, as shown in Tab. 2, which demonstrates the effectiveness of our method in outdoor environments. Moreover, we apply DR to the timing of sending the desired tracking trajectory by introducing slight delays and random time offsets $T \sim \mathcal{N}(\mu, \sigma^2)$, where $(\mu, \sigma) = (0.3, 0.5)$. Compared with the experiments without DR under windy conditions, the incorporation of DR effectively improves the success rate of the tracking experiments and reduces the relative tracking errors. It is worth mentioning that the variance of our method is higher than the baseline because the observation in our experiments comes from VIO while an external optical positioning system in SimpleFlight. Although VIO drift and quadrotor vibrations may increase the variance in tracking performance, they do not affect the overall tracking trend, as our tracking error is also computed relative to the position data from VIO.

2.4.2 Dynamic Obstacle Avoidance

The experiments above already demonstrate the system's Sim-to-Real stability for routine tasks. In this experiment, we aim to highlight the end-to-end inference capability when faced with rapidly changing environments. As illustrated in Fig. 3(c), the quadrotor has to avoid a rapidly flying object, a ball or a cube assembled from EVA foam mats, thrown from 6 meters away. The speed of the flying ball is $10 \sim 15$ m/s. Fig.3(c) is the processed depth image from the camera, where depths beyond 4.5 meters are truncated. Considering the high speed of the obstacle, it places high demands on the real-time neural network inference. The success rate of dynamic obstacle avoidance reaches over 90% across more than 20 trials involving either a ball or a cube.

To evaluate the generalization capability of our end-to-end model, we conduct transfer experiments by training models with two different types of dynamic obstacles—ball and cube—each at distinct throwing speeds, and subsequently testing them under varying conditions. As illustrated in Fig.3(d), the training involved dynamic obstacles moving at speeds ranging from $4\ \mathrm{m}\cdot\mathrm{s}^{-1}$ to $8\ \mathrm{m}\cdot\mathrm{s}^{-1}$. We perform cross-testing using models trained under different configurations to assess performance when encountering previously unseen obstacle dynamics. Additionally, we increase the maximum velocity of incoming obstacles during testing to examine the robustness of the learned policies in more extreme scenarios. The results indicate that the trained model exhibits a certain level of robustness when dealing with dynamic objects of unknown types. The model trained with balls as dynamic obstacles demonstrates better generalization capability, likely because the balls move faster than cubes during training. As a result, the trained model can more easily adapt its motion planning strategy to handle high-speed objects.

2.4.3 High-speed Flight

In order to evaluate the performance of our workflow in dealing with extreme maneuvers, we conduct a series of high-speed impact and tracking experiments based on the `target hitting` and `tracking` tasks, as illustrated in Fig.4. The blue trajectory demonstrates the high-speed flight path. Here we test the sim-to-real performance on both the `target hitting` and `tracking` tasks. In the `target hitting` task, a virtual balloon is placed at a fixed position, and the quadrotor must accelerate from rest to strike the target. Furthermore, in the `tracking` task, the quadrotor follows a sigmoid-shaped trajectory

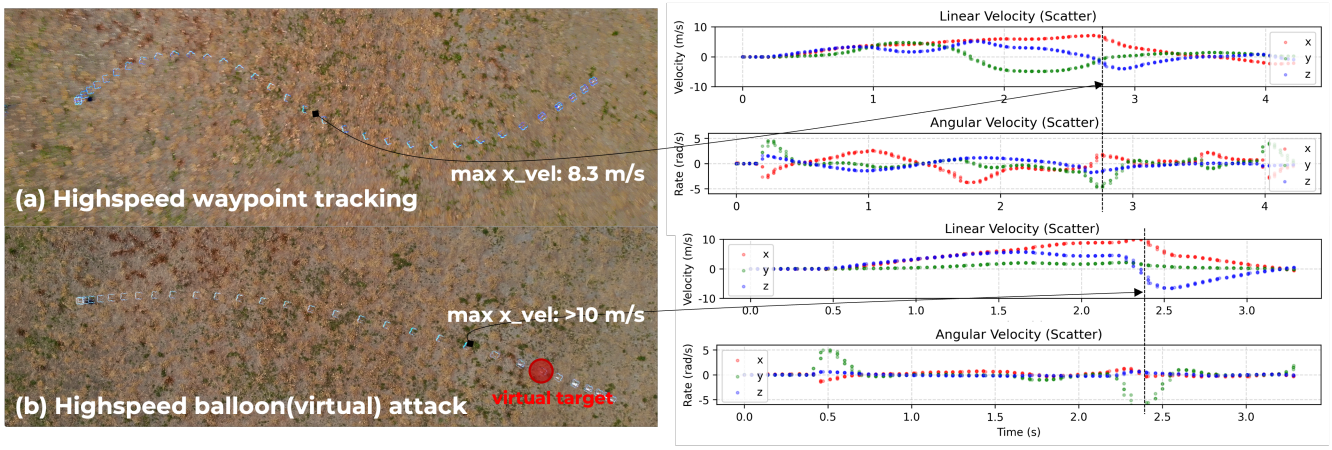


Figure 4. High-speed target hitting results. The blue halo is the quadrotor `X152b` with a strip light. (a) shows the high speed tracking with a sigmoid reference. (b) shows a `target hitting` task that accelerates from $0\,\mathrm{m}\cdot\mathrm{s}^{-1}$ and hits a virtual target.

starting from rest. In both experiments, no upper speed limit is imposed, allowing the quadrotor to reach the maximum velocity it can learn in `AirGym`. Linear and angular velocity records are shown on the right side of Fig.4. Remarkably, the platform achieves a top speed exceeding $10\,\mathrm{m}\cdot\mathrm{s}^{-1}$, demonstrating its strong maneuverability.

2.4.4 Navigation in Forest

We set up an outdoor wild flight scenario where the quadrotor navigates through and out of a small forest, illustrated in Fig.5. Herein, we evaluate the effectiveness of policies trained in the `planning` task. The quadrotor is initialized at the start point and then flies to the end with $1.5\,\mathrm{m}\cdot\mathrm{s}^{-1}$ maximum speed. The quadrotor navigates using depth images as input. Since the branches in the wild are dense and chaotic, differing somewhat from the simulation scene, we strictly constrain the actual flight altitude to around 0.5 meters to adjust the field of view (FOV) to prevent the camera from capturing images of canopies and other elements not present in the simulation. Moreover, we clip the depth image range to $0 \sim 4.5$ meters to further reduce the likelihood of capturing canopies. The quadrotor autonomously navigates a distance of over 20 meters and successfully reaches the target location as shown in the bird's-eye view Fig.5(b). In Fig.5(c), we use the recorded depth stream to generate the occupancy map of the area, highlighting the complexity of the forest environment.

3 Discussion

The direction of robot learning has emerged as a transformative force in aerial robotics, enabling autonomous systems to adapt, learn, and perform complex tasks in dynamic environments. This transformation goes beyond the conventional approach of merely grafting traditional pattern recognition methods onto aerial robot platforms to enhance onboard capabilities. Instead, it fundamentally redefines the control, planning, and perception architecture of quadrotor from the ground up, aiming to develop one task-driven end-to-end policy to realize all functions. This trend has sparked the emergence of increasingly challenging tasks in aerial robotics, as evidenced by the surge of related research in recent years[4, 5, 8, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21, 22, 25, 25, 27, 31, 32, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62]. In an increasing number of complex tasks, traditional methods are being replaced by end-to-end learning-based approaches, which serves as strong evidence of the generalization and effectiveness of learning-based methods. Among these, the majority of approaches are based on RL and its related variants.

Despite the numerous learning-based methods that have been successfully implemented, a significant challenge for many newcomers to the field lies in the complexity of building and implementing such system-level engineering solutions. Moreover, the reproduction process often depends heavily on extensive engineering experience, making it difficult to achieve rapid methodological iteration and innovation. Some previous highlight aerial robot learning works are listed in Tab.3. The last column outlines the task design principles used in previous approaches which support our task-oriented framework. Comparing with these, our framework and workflow differ significantly from existing works that primarily focus on methodological innovation or simulator. Instead, we emphasize the evaluation and integration of proven or widely accepted approaches, provide a comprehensive explanation of key principles for task design, and highlight the challenges of policy transfer along with practical solutions. In conjunction with the aforementioned design principles and techniques, the simulator, software, firmware, and hardware modules we develop collectively form a complete and cohesive workflow. Based on this, our task-oriented training leverages DRL to train policies and enable zero-shot sim-to-real transfer to real-world aerial robots. **We hope this**

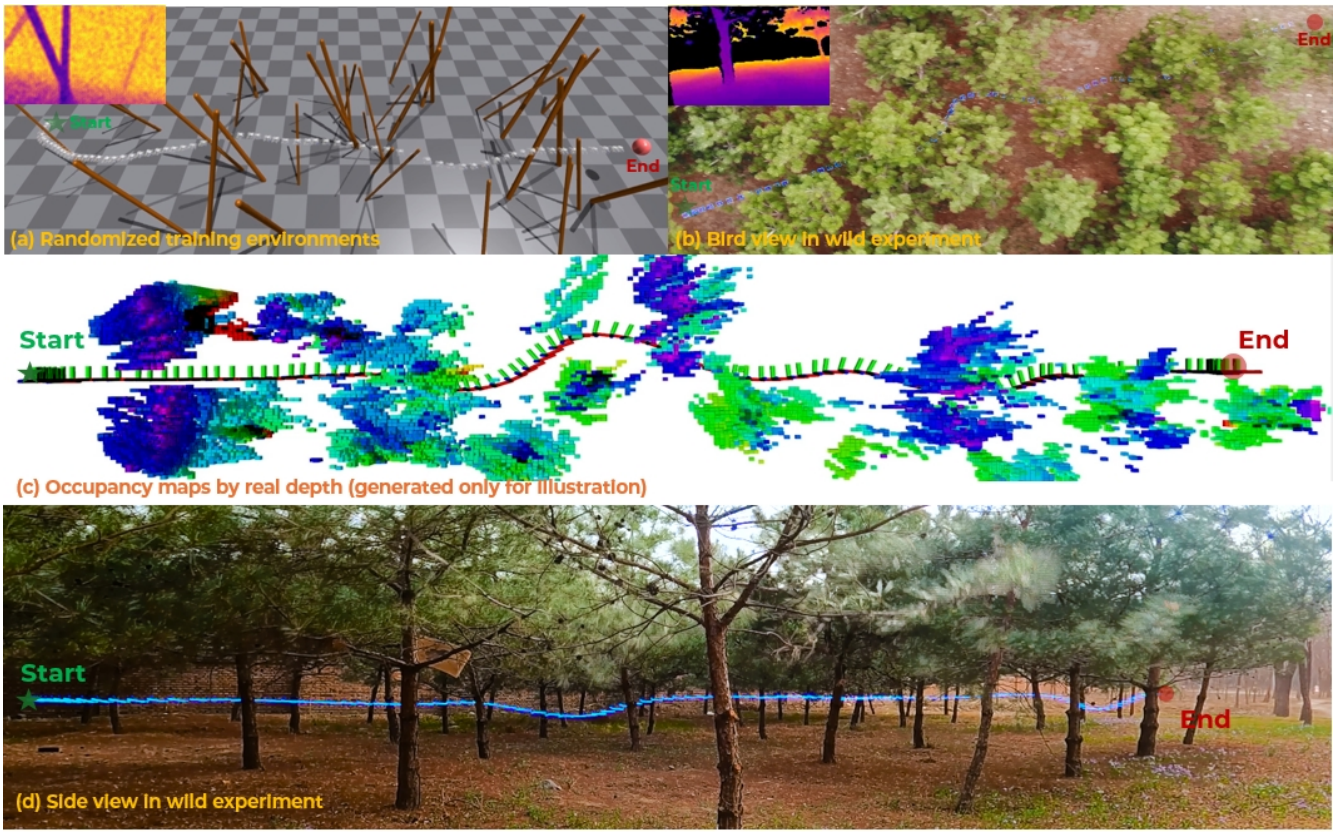


Figure 5. Navigating to fly out of the woods by task `Planning` sim-to-real. The sub-figure at the right bottom is generated using depth image after experiments and only for terrain illustration.

Table 3. Comparison of previous highlight works in learning-based aerial robotics field.

Work	Release	Type	Method	Tasks	SRT	CTBR	CTA	LV	PY/Traj	Controller	Principles
Control of a quadrotor with reinforcement learning[38]	2017	Methodology	RL	Control	✓	✗	✗	✗	✗	—	③
FlightGoggles[21]	2019	Simulator	—	Drone Racing	—	—	—	—	—	—	—
Deep drone racing[8]	2020	Methodology	DL	Drone Racing	✗	✗	✗	✗	✓	—	④
Flightmare[44]	2020	Simulator	RL	Control / Drone Racing	✓	✓	✗	✗	✗	—	①
NeuroBEM[63]	2021	Dataset	DL	Agile Flight / Control	—	—	—	—	—	Betaflight	—
gym-pybullet-drones[37]	2021	Simulator	RL	Control / Multi-agent Lead-follower	✓	✗	✗	✓	✗	—	①
Learning high-speed flight in the wild[39]	2021	Methodology	IL	Planning	✗	✗	✗	✗	✓	MPC	②③⑥
Autonomous drone racing with deep reinforcement learning[5]	2021	Methodology	RL	Drone Racing	✓	✗	✗	✗	✗	Betaflight	③④⑥
Agilicious[27]	2022	Software	—	Infrastructure for Any Tasks	✓	✓	✓	✓	✓	Betaflight	①
Aggressive quadrotor flight using curiosity-driven reinforcement learning[64]	2022	Methodology	RL	Planning	✗	✗	✓	✗	✗	AscTech Autopilot	④⑤
A deep multi-agent reinforcement learning framework for autonomous aerial navigation to grasping points on loads[42]	2023	Methodology	RL	Multi-agent Transport	✗	✗	✗	✓	✗	Crazyflie Controller	—
OmniDrones[31]	2023	Simulator	RL	Control / Tracking / Transport / Formation	✓	✓	✓	✓	✓	—	①
Champion-level drone racing using deep reinforcement learning[4]	2023	Methodology	RL	Drone Racing	✗	✓	✗	✗	✗	Betaflight	③④
Reaching the limit in autonomous racing[32]	2023	Methodology	RL	Drone Racing	✗	✓	✗	✗	✗	Betaflight	③④⑥
Aerial gym simulator[43]	2023	Simulator	RL	Control / Planning	✗	✗	✓	✓	✓	—	③
Deep-PANTHER[60]	2023	Methodology	RL	Planning	✗	✗	✗	✗	✓	—	②
Learning quadrotor control from visual features using differentiable simulation[15]	2024	Methodology	RL	Control	✗	✓	✗	✗	✗	Betaflight	⑦
Back to Newton's law[7]	2024	Methodology	RL	(Multi-agent) Planning	✓	✗	✗	✗	✗	Betaflight/PX4	③⑥⑦
What matters RL sim2real[11]	2024	Methodology	RL	Control / Tracking	✗	✓	✗	✗	✗	Crazyflie Controller	①②⑥
Dashing for the golden snitch[16]	2024	Methodology	RL	Multi-agent Tracking	✗	✓	✗	✗	✗	Betaflight	—
Reinforcement learning for collision-free flight exploiting deep collision encoding[25]	2024	Methodology	RL	Planning	✗	✗	✗	✓	✗	PixRacer	③④⑤
Learning-based navigation and collision avoidance through reinforcement for UAVs[19]	2024	Methodology	RL	Planning / Avoidance	✗	✗	✗	✓	✗	—	⑤
Flying on point clouds with reinforcement learning[26]	2025	Methodology	RL	Planning	✗	✓	✗	✗	✗	PX4	③④⑥
NavRL[65]	2025	Methodology	RL	Planning	✗	✗	✗	✓	✗	PX4	③④⑥
AirGym (Ours)	2025	Workflow	RL	Control / Tracking / Target Hitting / Avoidance / Planning / Customized	✓	✓	✓	✓	✓	PX4/rlPx4controller	①②③④⑤⑥

work serves as a stepping stone to inspire and enable more researchers to quickly engage in practical exploration within this field, rather than being hindered by limitations in infrastructure.

Our research has also identified key limitations, especially in the sampling efficiency and generalization of DRL. Model-free RL methods often rely on vast amounts of trial-and-error interactions, making them computationally expensive and

data-inefficient. Vectorized parallel simulation with simultaneous sampling in this work is one solution; however, it does not mask the inherent inefficiency of the sampling. Recently, differentiable simulations enable gradient-based optimization by backpropagating through environment dynamics, which can drastically reduce the number of required training samples[7, 15]. Although it offers a promising direction for improving sampling efficiency compared to traditional model-free RL, its advantages are not yet fully realized in complex, real-world scenarios. The applicability of differentiable simulation is currently constrained by its reliance on fully differentiable environments—a requirement that many physical or stochastic systems do not satisfy. Bridging this gap could unlock more efficient training paradigms but remains an open research challenge.

Generalization in task-oriented DRL policies remains a critical hurdle. While many DRL agents perform well on narrowly defined tasks, they often struggle when exposed to even slight variations in deployment scenarios. The most direct and effective approach to improving generalization is scaling up model parameters and training data. The success of large-scale models—such as large language models (LLMs) and vision-language models—demonstrates that scaling laws, where performance predictably improves with increased model size, data diversity, and computational resources, lead to remarkable generalization abilities. Analogously, aerial robot policies trained on massive, multi-domain datasets—combining simulation and real-world data across varying weather and obstacle conditions—are expected to exhibit stronger out-of-distribution (OOD) generalization. Embodied learning with large-scale models in aerial robotics thus offers a promising direction toward achieving robust generalization. Nonetheless, given the safety-critical and low-latency demands of aerial robot control, careful balance between model size and inference speed remains essential.

4 Methods

4.1 Deep Reinforcement Learning

We model the aerial robot learning problem into a Markov Decision Process (MDP) problem and solve it by DRL, which can be mathematically expressed as $\langle \mathscr{S}, \mathscr{A}, \mathscr{T}, \mathscr{S}', R \rangle$, where $\mathscr{S}$ is the state space and $\mathscr{A}$ is the action space. $\mathscr{T}$ is the state transition function, and $\mathscr{T}(s'|s,a)$ represents the transition probability distribution of the system when transiting into the next state $s' \in \mathscr{S}'$ from state $s \in \mathscr{S}$ after taking action $a \in \mathscr{A}$. Besides, R denotes the reward function, where $r = R(s,a)$ is the reward given to the agent after taking action a under state s. Our aim is to find the optimal policy π^*, to maximize the expected value of cumulative discount reward:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} J_\pi = \sum_{t=0}^{\infty} \gamma^t \cdot \mathbb{E}\left[R(s_t, a_t)|\pi\right] \tag{1}$$

where $s_t \in \mathscr{S}$ and $a_t \in \mathscr{A}$ are the state and action of the agent at time step t, and γ denotes the discount factor.

We use Proximal Policy Optimization (PPO) algorithm[66] to train the quadrotor policy, with a discount factor γ equal to 0.95. We use a shared actor-critic backbone with learning rate 3×10^4. Since our contribution focuses on the entire sim-to-real system rather than the algorithm itself, we use the default parameters from `rl_games`[67] without any specific tuning of the algorithm. The simulator's dynamics calculation time step $\boldsymbol{dt}$ is 0.01 s. The image render frequency is slower than dynamics calculation, with a render timestep of 0.04 s which is similar to the depth image in reality, because the real frequency of images varies from $25 \sim 30$ Hz. The experiments are completed on a 4090 GPU with 24G memory. During the sampling process, we encourage using a large batch size to improve network convergence, especially in DRL where the search space is extremely large. In this work, we do not set any learning curriculum but use hierarchical training in dynamics obstacle avoidance task.

4.2 Observation & Action Space

We commonly use typical quadrotor states as proprioceptive observation, illustrated in Fig.2, involving the position, velocity, attitude, and angular velocity of the quadrotor root, or the difference between its target. Rotation matrix is used to represent the attitude due to the quaternion representation of a rotation being not unique. Therefore, the basic dimension of proprioceptive observation is 18. Under the ego-centric observation design mode, we use states under $(\boldsymbol{x}_{\mathscr{E}}, \boldsymbol{y}_{\mathscr{E}}, \boldsymbol{z}_{\mathscr{E}})$ frame. Moreover, the exteroceptive observation is derived from a depth image captured from the forward-facing view. This depth image is processed through a simple convolutional neural network (CNN), a ResNet18, or a pretrained auto-encoder to extract features, where the feature dimensions can be adjusted depending on the complexity of the environment. Then, the policy network accepts the concatenation of both proprioceptive and exteroceptive observation and generates the action. Thus, the action space depends on the selected control levels. The action dimension is 4 in PY, LV, $CTBR$, and SRT mode, but 5 in the CTA mode.

4.3 Bridging Gaps in Sim-to-real

4.3.1 System Identification

The role of system identification in policy transfer is well-established and needs no further emphasis[11, 18]. The design of the controller relies on an accurate description of the system model. System identification enables precise acquisition

of the dynamic characteristics. This enhances the realism of simulation, ensuring that control policies designed are more effective when deployed in real-world scenarios. However, thanks to the differential flatness of quadrotor, it enables dynamic parameterization, meaning the RL policy in training only needs to address uncertainties within the flat output space instead of the full state space, greatly simplifying system identification[32]. Moreover, by leveraging large amount of sampling and domain randomization, DRL inherently addresses model uncertainty, diminishing the need for accurate system modeling. Here we emphasize three critical aspects that must be carefully addressed during the quadrotor system identification process: hovering throttle, air drag, and system delay. Hovering throttle refers to the precise throttle value required to maintain a stable hover which is necessary to counter the gravity, and will be discussed in details later. Air drag plays a significant role in the quadrotor's translational dynamics, especially at higher velocities[4, 68]. It introduces nonlinear effects that must be captured to ensure realistic modeling and robust control performance. Finally, the system delay[7], including sensor latency, actuator response time, and computation delay, affects the closed-loop behavior and must be quantified to build a reliable and predictive model of the entire system.

4.3.2 Hover Throttle Learning

In low-speed and low-maneuverability flights, system failures due to air drag and system delay are uncommon. While the hover throttle parameter estimation is typically the primary consideration during system identification. In the implementation of PX4 autopilot, the hover throttle is also a crucial setting. It is usually expressed as a normalized value between 0 and 1. It determines how the controller allocates thrust to counteract gravity and maintain altitude stability during hover. This setting enhances compatibility with different aircraft, making it more general. Moreover, this setting could decouple the altitude and attitude control, making the quadrotor more flexible. To ensure that the policy trained in simulation can maintain similarly stable outputs in real-world scenarios while preserving accuracy in altitude control, we recommend designing the reward during training to gradually guide the policy's total throttle output to converge to the actual hover throttle value. Otherwise, an inaccurate hover throttle can result in steady-state errors along the $\boldsymbol{z}$ axis and even lead to control divergence in the altitude loop.

4.3.3 Learning Continuity of Actions

The continuity of policy output is a key factor in sim-to-real control[11]. In simulation, the dynamics computation time step $\boldsymbol{dt}$ is strictly consistent. However, the control frequency may fluctuate slightly depending on operating conditions in practice. The latency, which can impact the completeness of actuator execution, should be considered. For instance, in learning to maintain hovering, the agent might acquire behaviours involving rapid oscillations within the $(0, 1)$ range, as shown in Fig.1. While the altitude may remain well performed in the simulator, when transferred to a physical quadrotor, the motors cannot execute instantaneous and high-frequency accelerations and decelerations, leading to a crash. Therefore, a continuity reward for actions is essential and should be incorporated throughout the entire learning process.

4.3.4 Reference Trajectory Play

As aforementioned, in tracking tasks, spatiotemporal awareness guidance is used for giving a reference in the near future. So that the policy could follow up the complicated trajectory. The previous work[11] has shown that RL methods with appropriate guidance can achieve high-precision trajectory tracking when a given velocity is specified. These experiments lay on the ideal environments such that the velocity can maintain steady. However, the situation when quadrotor is affected by disturbances is not taken into consideration. In reality, quadrotors might be affected by external factors like wind and inaccurate pose estimation, which can prevent them from flying at the expected speed. The result can be found in Fig.3. If the quadrotor cannot keep up with the playback of the trajectory guidance, the tracking policy may fail or even collapse. Therefore, in tasks involving external spatiotemporal guidance, it is important to consider both appropriate reference trajectory replay and external disturbances.

4.3.5 Domain Randomization (DR)

Although various domain randomization methods are already common in sim-to-real applications, we want to emphasize the importance of their effectiveness here. The role of domain randomization is not only to narrow the gap between simulation and reality but also to encourage exploration through parameter and state randomness introduced by random initialization. This is particularly critical in DRL with image-based inputs; otherwise, the policy may easily get trapped in local optimal. In this work, we used VIO as the data for state estimation of the quadrotor, without any external ground truth observation inputs. Therefore, domain randomization for the quadrotor's fundamental states is particularly crucial. Without this, the agent may encounter unseen states, which could lead to a failure or collapse. To ensure the generalization of images during sim-to-real transfer, DR should also be applied to the visual information during simulation training[39]. We process the depth maps generated in simulation by adding Gaussian noise and multiplicative noise to mimic sensor noise and errors. Additionally, we apply blurring to the depth to mimic lens distortion, motion blur, or inaccurate focus. Furthermore, we randomly scale and offset the depth to simulate sensor calibration discrepancies, changes in ambient lighting, or inconsistencies in depth measurement.

Data availability

All data necessary to evaluate the conclusions in the paper are available in the paper or the Supplementary Information.

Code availability

Simulation codes, software and firmware for fight platform are publicly accessible in the GitHub repository: https://github.com/emNavi/AirGym, https://github.com/emNavi/rlPx4Controller, https://github.com/emNavi/AirGym-Real, https://github.com/emNavi/control_for_gym.

References

1. Chu, H., Zhao, H., Li, P. & Guo, Y. X. Urban skies safeguarded: innovative drone detection with programmable metasurface periscope. *Nat. Commun.* **15**, DOI: 10.1038/s41467-024-54672-6 (2024).

2. Wüest, V. *et al.* Agile perching maneuvers in birds and morphing-wing drones. *Nat. communications* **15**, 8330, DOI: 10.1038/s41467-024-52369-4 (2024).

3. Hanover, D. *et al.* Autonomous Drone Racing: A Survey. *IEEE Transactions on Robotics* **40**, 3044–3067, DOI: 10.1109/TRO.2024.3400838 (2024). 2301.01755.

4. Kaufmann, E. *et al.* Champion-level drone racing using deep reinforcement learning. *Nature* **620**, 982–987, DOI: 10.1038/s41586-023-06419-4 (2023).

5. Song, Y., Steinweg, M., Kaufmann, E. & Scaramuzza, D. Autonomous Drone Racing with Deep Reinforcement Learning. *IEEE Int. Conf. on Intell. Robots Syst.* 1205–1212, DOI: 10.1109/IROS51168.2021.9636053 (2021). 2103.08624.

6. Nagami, K. & Schwager, M. HJB-RL: Initializing Reinforcement Learning with Optimal Control Policies Applied to Autonomous Drone Racing. *Robotics: Sci. Syst.* DOI: 10.15607/RSS.2021.XVII.062 (2021).

7. Zhang, Y., Hu, Y., Song, Y., Zou, D. & Lin, W. Back to Newton's Laws: Learning Vision-based Agile Flight via Differentiable Physics. (2024). 2407.10648.

8. Loquercio, A. *et al.* Deep Drone Racing: From Simulation to Reality with Domain Randomization. *IEEE Transactions on Robotics* **36**, 1–14, DOI: 10.1109/TRO.2019.2942989 (2020). 1905.09727.

9. Singla, A., Padakandla, S. & Bhatnagar, S. Memory-Based Deep Reinforcement Learning for Obstacle Avoidance in UAV with Limited Environment Knowledge. *IEEE Transactions on Intell. Transp. Syst.* **22**, 107–118, DOI: 10.1109/TITS.2019.2954952 (2021). 1811.03307.

10. Xiao, C., Lu, P. & He, Q. Flying Through a Narrow Gap Using End-to-End Deep Reinforcement Learning Augmented With Curriculum Learning and Sim2Real. *IEEE Transactions on Neural Networks Learn. Syst.* **34**, 2701–2708, DOI: 10.1109/TNNLS.2021.3107742 (2023). 2108.12869.

11. Chen, J. *et al.* What Matters in Learning A Zero-Shot Sim-to-Real RL Policy for Quadrotor Control? A Comprehensive Study. (2024). 2412.11764.

12. Ma, B. *et al.* Target Tracking Control of UAV Through Deep Reinforcement Learning. *IEEE Transactions on Intell. Transp. Syst.* **24**, 5983–6000, DOI: 10.1109/TITS.2023.3249900 (2023).

13. Ma, B. *et al.* Deep Reinforcement Learning of UAV Tracking Control under Wind Disturbances Environments. *IEEE Transactions on Instrumentation Meas.* **72**, DOI: 10.1109/TIM.2023.3265741 (2023).

14. Wu, T., Chen, Y., Chen, T., Zhao, G. & Gao, F. Whole-Body Control Through Narrow Gaps From Pixels To Action. (2024). 2409.00895.

15. Heeg, J., Song, Y. & Scaramuzza, D. Learning Quadrotor Control From Visual Features Using Differentiable Simulation. (2024). 2410.15979.

16. Wang, X. *et al.* Dashing for the Golden Snitch: Multi-Drone Time-Optimal Motion Planning with Multi-Agent Reinforcement Learning. (2024). 2409.16720.

17. Lee, J., Rathod, A., Goel, K., Stecklein, J. & Tabib, W. Rapid Quadrotor Navigation in Diverse Environments using an Onboard Depth Camera. DOI: 10.1109/SSRR62954.2024.10770033 (2024). 2411.04326.

18. Kaufmann, E., Bauersfeld, L. & Scaramuzza, D. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight. *Proc. - IEEE Int. Conf. on Robotics Autom.* 10504–10510, DOI: 10.1109/ICRA46639.2022.9811564 (2022). 2202.10796.

19. Azzam, R. *et al.* Learning-Based Navigation and Collision Avoidance Through Reinforcement for UAVs. *IEEE Transactions on Aerosp. Electron. Syst.* **60**, 2614–2628, DOI: 10.1109/TAES.2023.3294889 (2024).

20. Kalidas, A. P. *et al.* Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles. *Drones* **7**, DOI: 10.3390/drones7040245 (2023).

21. Guerra, W., Tal, E., Murali, V., Ryou, G. & Karaman, S. FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality. *IEEE Int. Conf. on Intell. Robots Syst.* 6941–6948, DOI: 10.1109/IROS40897.2019.8968116 (2019).

22. Amer, K., Samy, M., Shaker, M. & Elhelw, M. Deep convolutional neural network based autonomous drone navigation. 46, DOI: 10.1117/12.2587105 (2021). 1905.01657.

23. Ren, Y. *et al.* Bubble Planner: Planning High-speed Smooth Quadrotor Trajectories using Receding Corridors. *IEEE Int. Conf. on Intell. Robots Syst.* **2022-Octob**, 6332–6339, DOI: 10.1109/IROS47612.2022.9981518 (2022). 2202.12177.

24. Wang, X. *et al.* Path Planning for Air-Ground Robot Considering Modal Switching Point Optimization. *2023 Int. Conf. on Unmanned Aircr. Syst. ICUAS 2023* 87–94, DOI: 10.1109/ICUAS57906.2023.10156162 (2023).

25. Kulkarni, M. & Alexis, K. Reinforcement Learning for Collision-free Flight Exploiting Deep Collision Encoding. *Proc. - IEEE Int. Conf. on Robotics Autom.* 15781–15788, DOI: 10.1109/ICRA57147.2024.10610287 (2024).

26. Xu, G., Wu, T., Wang, Z., Wang, Q. & Gao, F. Flying on Point Clouds with Reinforcement Learning. (2025). 2503.00496.

27. Foehn, P. *et al.* Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Sci. Robotics* **7**, DOI: 10.1126/scirobotics.abl6259 (2022).

28. Delmerico, J., Cieslewski, T., Rebecq, H., Faessler, M. & Scaramuzza, D. Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset. *Proc. - IEEE Int. Conf. on Robotics Autom.* **2019-May**, 6713–6719, DOI: 10.1109/ICRA.2019.8793887 (2019).

29. Andersen, K. F., Pham, H. X., Ugurlu, H. I. & Kayacan, E. Event-based Navigation for Autonomous Drone Racing with Sparse Gated Recurrent Network. *2022 Eur. Control. Conf. ECC 2022* 1342–1348, DOI: 10.23919/ECC55457.2022.9838538 (2022). 2204.02120.

30. Makoviychuk, V. *et al.* Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. (2021). 2108.10470.

31. Xu, B. *et al.* OmniDrones: An Efficient and Flexible Platform for Reinforcement Learning in Drone Control. *IEEE Robotics Autom. Lett.* DOI: 10.1109/LRA.2024.3356168 (2024).

32. Song, Y., Romero, A., Müller, M., Koltun, V. & Scaramuzza, D. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Sci. Robotics* **8**, DOI: 10.1126/scirobotics.adg1462 (2023).

33. Hong, C., Huang, K. & Liu, H. Learning a Distributed Hierarchical Locomotion Controller for Embodied Cooperation. In *8th Annual Conference on Robot Learning, CoRL 2024* (2024).

34. Wang, H., Xing, J., Messikommer, N. & Scaramuzza, D. Environment as Policy: Learning to Race in Unseen Tracks. (2024). 2410.22308.

35. Gleave, A. *et al.* Adversarial Policies: Attacking Deep Reinforcement Learning. *8th Int. Conf. on Learn. Represent. ICLR 2020* (2020). 1905.10615.

36. Llanes, C., Kakish, Z., Williams, K. & Coogan, S. CrazySim: A Software-in-the-Loop Simulator for the Crazyflie Nano Quadrotor. *IEEE Int. Conf. on Robotics Autom.* (2024).

37. Panerati, J. *et al.* Learning to Fly-A Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control. *IEEE Int. Conf. on Intell. Robots Syst.* 7512–7519, DOI: 10.1109/IROS51168.2021.9635857 (2021). 2103.02142.

38. Hwangbo, J., Sa, I., Siegwart, R. & Hutter, M. Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics Autom. Lett.* **2**, 2096–2103, DOI: 10.1109/LRA.2017.2720851 (2017). 1707.05110.

39. Loquercio, A. *et al.* Learning high-speed flight in the wild. *Sci. Robotics* **6**, DOI: 10.1126/scirobotics.abg5810 (2021).

40. Lin, Y. *et al.* Autonomous aerial navigation using monocular visual-inertial fusion. *J. Field Robotics* **35**, 23–51, DOI: 10.1002/rob.21732 (2018).

41. Zhou, X., Wang, Z., Ye, H., Xu, C. & Gao, F. EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors. *IEEE Robotics Autom. Lett.* **6**, 478–485, DOI: 10.1109/LRA.2020.3047728 (2021). 2008.08835.

42. Chen, J., Ma, R. & Oyekan, J. A deep multi-agent reinforcement learning framework for autonomous aerial navigation to grasping points on loads. *Robotics Auton. Syst.* **167**, DOI: 10.1016/j.robot.2023.104489 (2023).

43. Kulkarni, M., Forgaard, T. J. L. & Alexis, K. Aerial Gym – Isaac Gym Simulator for Aerial Robots. (2023). 2305.16510.

44. Song, Y., Naji, S., Kaufmann, E., Loquercio, A. & Scaramuzza, D. Flightmare: A Flexible Quadrotor Simulator. *Proc. Mach. Learn. Res.* **155**, 1147–1157 (2020). 2009.00563.

45. Miera, P., Szolc, H. & Kryjak, T. LiDAR-based drone navigation with reinforcement learning. (2023). 2307.14313.

46. Moon, H. *et al.* Challenges and implemented technologies used in autonomous drone racing. *Intell. Serv. Robotics* **12**, 137–148, DOI: 10.1007/s11370-018-00271-6 (2019).

47. Mohta, K. *et al.* Fast, autonomous flight in GPS-denied and cluttered environments. *J. Field Robotics* **35**, 101–120, DOI: 10.1002/rob.21774 (2018). 1712.02052.

48. Foehn, P. *et al.* AlphaPilot: Autonomous Drone Racing. *Robotics: Sci. Syst.* DOI: 10.15607/RSS.2020.XVI.081 (2020).

49. Guerra, W., Tal, E., Murali, V., Ryou, G. & Karaman, S. FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality. *IEEE Int. Conf. on Intell. Robots Syst.* 6941–6948, DOI: 10.1109/IROS40897.2019.8968116 (2019).

50. Sun, S., Cioffi, G., De Visser, C. & Scaramuzza, D. Autonomous Quadrotor Flight despite Rotor Failure with Onboard Vision Sensors: Frames vs. Events. *IEEE Robotics Autom. Lett.* **6**, 580–587, DOI: 10.1109/LRA.2020.3048875 (2021). 2102.13406.

51. Romero, A., Penicka, R. & Scaramuzza, D. Time-Optimal Online Replanning for Agile Quadrotor Flight. *IEEE Robotics Autom. Lett.* **7**, 7730–7737, DOI: 10.1109/LRA.2022.3185772 (2022). 2203.09839.

52. Penicka, R., Song, Y., Kaufmann, E. & Scaramuzza, D. Learning Minimum-Time Flight in Cluttered Environments. *IEEE Robotics Autom. Lett.* **7**, 7209–7216, DOI: 10.1109/LRA.2022.3181755 (2022). 2203.15052.

53. Zhou, B., Pan, J., Gao, F. & Shen, S. RAPTOR: Robust and Perception-Aware Trajectory Replanning for Quadrotor Fast Flight. *IEEE Transactions on Robotics* **37**, 1992–2009, DOI: 10.1109/TRO.2021.3071527 (2021). 2007.03465.

54. Muller, M. *et al.* Learning a controller fusion network by online trajectory filtering for vision-based UAV racing. *IEEE Comput. Soc. Conf. on Comput. Vis. Pattern Recognit. Work.* **2019-June**, 573–581, DOI: 10.1109/CVPRW.2019.00083 (2019). 1904.08801.

55. Fu, J., Song, Y., Wu, Y., Yu, F. & Scaramuzza, D. Learning Deep Sensorimotor Policies for Vision-Based Autonomous Drone Racing. *IEEE Int. Conf. on Intell. Robots Syst.* 5243–5250, DOI: 10.1109/IROS55552.2023.10341805 (2023). 2210.14985.

56. Zhang, D. *et al.* Learning a Single Near-hover Position Controller for Vastly Different Quadcopters. *Proc. - IEEE Int. Conf. on Robotics Autom.* **2023-May**, 1263–1269, DOI: 10.1109/ICRA48891.2023.10160836 (2023). 2209.09232.

57. Yu, H., De Wagter, C. & de Croon, G. C. H. E. MAVRL: Learn to Fly in Cluttered Environments with Varying Speed. DOI: 10.1109/LRA.2024.3522778 (2024). 2402.08381.

58. Lamberti, L. *et al.* A Sim-to-Real Deep Learning-Based Framework for Autonomous Nano-Drone Racing. *IEEE Robotics Autom. Lett.* **9**, 1899–1906, DOI: 10.1109/LRA.2024.3349814 (2024).

59. Pham, H. X., Sarabakha, A., Odnoshyvkin, M. & Kayacan, E. PencilNet: Zero-Shot Sim-to-Real Transfer Learning for Robust Gate Perception in Autonomous Drone Racing. *IEEE Robotics Autom. Lett.* **7**, 11847–11854, DOI: 10.1109/LRA.2022.3207545 (2022).

60. Tordesillas, J. & How, J. P. Deep-PANTHER: Learning-Based Perception-Aware Trajectory Planner in Dynamic Environments. *IEEE Robotics Autom. Lett.* **8**, 1399–1406, DOI: 10.1109/LRA.2023.3235678 (2023). 2209.01268.

61. Panerati, J. *et al.* Learning to Fly-A Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control. *IEEE Int. Conf. on Intell. Robots Syst.* 7512–7519, DOI: 10.1109/IROS51168.2021.9635857 (2021). 2103.02142.

62. Bajcsy, A., Loquercio, A., Kumar, A. & Malik, J. Learning Vision-based Pursuit-Evasion Robot Policies. *Proc. - IEEE Int. Conf. on Robotics Autom.* 9197–9204, DOI: 10.1109/ICRA57147.2024.10610498 (2024). 2308.16185.

63. Bauersfeld, L., Kaufmann, E., Foehn, P., Sun, S. & Scaramuzza, D. NeuroBEM: Hybrid Aerodynamic Quadrotor Model. *Robotics: Sci. Syst.* DOI: 10.15607/RSS.2021.XVII.042 (2021). 2106.08015.

64. Sun, Q., Fang, J., Zheng, W. X. & Tang, Y. Aggressive Quadrotor Flight Using Curiosity-Driven Reinforcement Learning. *IEEE Transactions on Ind. Electron.* **69**, 13838–13848, DOI: 10.1109/TIE.2022.3144586 (2022).

65. Xu, Z., Han, X., Shen, H., Jin, H. & Shimada, K. NavRL: Learning Safe Flight in Dynamic Environments. DOI: 10.1109/LRA.2025.3546069 (2024). 2409.15634.

66. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal Policy Optimization Algorithms. (2017). 1707.06347.

67. Makoviichuk, D. & Makoviychuk, V. rl-games: A High-performance Framework for Reinforcement Learning. (2021).

68. Sun, S., Romero, A., Foehn, P., Kaufmann, E. & Scaramuzza, D. A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight. *IEEE Transactions on Robotics* **38**, 3357–3373, DOI: 10.1109/TRO.2022.3177279 (2022). 2109.01365.

Acknowledgements

We thank Wantong Qin for her assistance in the design and development of the flight platform, and Kai Tang for his support in VIO flight testing.

Author contributions

K.H and H.W designed the full flight system, including the simulator and physical platform. K.H built the framework and provided the visualization. K.H and J.Y.C wrote the paper. K.H and Y.L provided reward design in RL. J.T.C and D.G provided testing resources and validation. H.L, X.J, and X.Z were responsible for the project supervision and conceptualization.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to Huaping Liu.

A Dynamics Model

The world $\mathscr{W}$ and body $\mathscr{B}$ frames with the orthonormal basis in this work are defined: $\{\boldsymbol{x}_{\mathscr{W}}, \boldsymbol{y}_{\mathscr{W}}, \boldsymbol{z}_{\mathscr{W}}\}$, $\{\boldsymbol{x}_{\mathscr{B}}, \boldsymbol{y}_{\mathscr{B}}, \boldsymbol{z}_{\mathscr{B}}\}$. The frame $\mathscr{B}$ is at the root of the quadrotor. Symbols with subscript mean variables under the corresponding coordination frame. The system dynamics can be expressed by the following equations:

$$\begin{bmatrix} \dot{\boldsymbol{p}}_{\mathscr{W}} \\ \dot{\boldsymbol{q}}_{\mathscr{B}} \\ \dot{\boldsymbol{v}}_{\mathscr{W}} \\ \dot{\boldsymbol{\omega}}_{\mathscr{B}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\mathscr{W}} \\ \boldsymbol{q}_{\mathscr{B}} \cdot \begin{bmatrix} 0 & \boldsymbol{\omega}_{\mathscr{B}}/2 \end{bmatrix}^{\top} \\ \frac{1}{m}(\boldsymbol{q}_{\mathscr{B}} \odot \boldsymbol{f}) + \boldsymbol{g}_{\mathscr{W}} \\ \boldsymbol{J}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega}_{\mathscr{B}} \times \boldsymbol{J}\boldsymbol{\omega}_{\mathscr{B}}) \\ T_{\mathrm{m}}(\boldsymbol{\Omega}_{\mathrm{cmd}} - \boldsymbol{\Omega}) \end{bmatrix} \tag{2}$$

where $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\omega}, \boldsymbol{\Omega}$ represent position, quaterion, velocity, body rate (angular velocity), propeller rotation speed, respectively. $\boldsymbol{\Omega}_{\mathrm{cmd}}$ is the command of rotational speed, and T_{m} is the motor time constant, which can be estimated by rotational speed testing. $\boldsymbol{J} = \mathrm{diag}(I_x, I_y, I_z)$ denotes diagonal moment of inertia matrix. m is the mass of quadrotor. Note the quaternion-vector product is denoted by $\odot$ representing a rotation of the vector by the quaternion as in $\boldsymbol{q} \odot \boldsymbol{v} = \boldsymbol{q}\boldsymbol{v}\boldsymbol{q}^{-}$, where $\boldsymbol{q}^{-}$ is the quaternion's conjugate. $\boldsymbol{g}_{\mathscr{W}} = [0, 0, -9.81]^{\top}$ denotes gravity. $\boldsymbol{f}$ and $\boldsymbol{\tau}$ are the collective force and the torque produced by all propellers. The quantities are calculated as follows:

$$\boldsymbol{f} = \sum_i f_i, \boldsymbol{\tau} = \sum_i \tau_i + \boldsymbol{r}_{P,i} \times f_i \tag{3}$$

where $\boldsymbol{r}_{P,i}$ denotes the location of propeller i expressed in the body frame, f_i and τ_i are forces and torques generated by the ith propeller, where we have:

$$f_i(\Omega) = \begin{bmatrix} 0 & 0 & c_l \cdot \Omega^2 \end{bmatrix}^{\top}, \tau_i(\Omega) = \begin{bmatrix} 0 & 0 & c_d \cdot \Omega^2 \end{bmatrix}^{\top} \tag{4}$$

where c_l and c_d are corresponding thrust and drag coefficients that can be obtained by system identification.

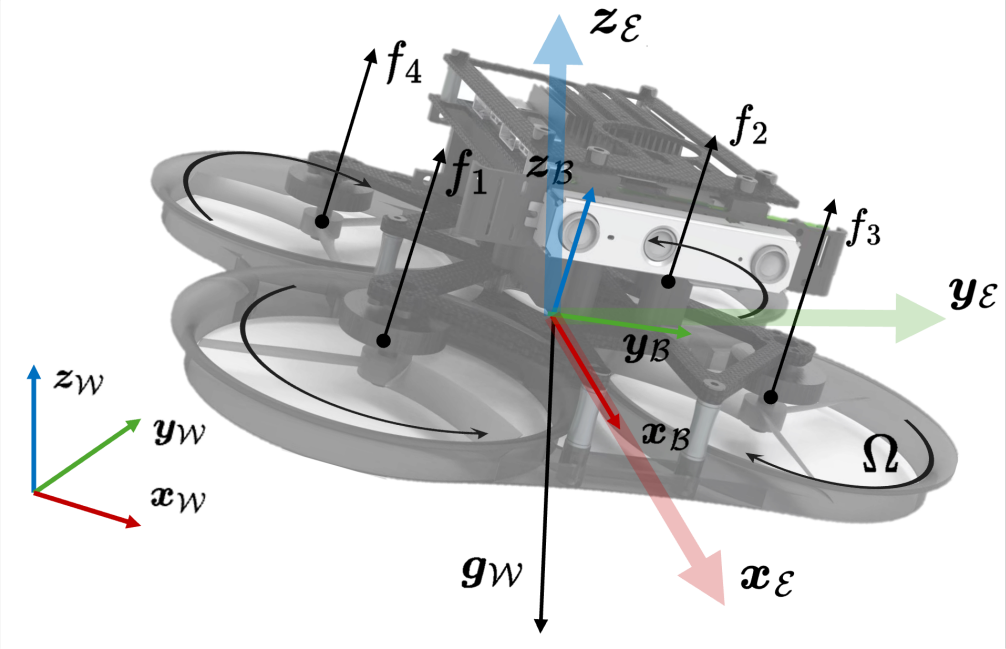


Figure 6. Diagram of the `X152b` depicting the world and body frames and propeller numbering. The background image showcases the construction and visualization of the `X152b` quadrotor in IsaacGym.

B Task Design & Reward Settings

B.1 hovering

Hovering is the most fundamental task of RL-trained quadrotor. We defined an 18 dimensional quadrotor state:

$$(\mathrm{R}^{\rightarrow}, p_x, p_y, p_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$$

where $\mathrm{R}^{\rightarrow}$ denotes the flattening of rotation matrix R. Then the differential between the current state and expected state is used as the observation. It brings advantages that all states will converge to zero, leading to a more uniform gradient in the data probability distribution. Additionally, by modifying the desired state during flight, commands can be sent to the quadrotor, enabling waypoint control similar to that in traditional flight controllers. The reward function is set as below:

$$\begin{aligned}
\boldsymbol{R} &= R_{smooth} + R_{effort} + R_{pos} + R_{throttle} && (5)\\
&\quad + R_{pos} \times (R_{ups} + R_{spin} + R_{heading} + R_{vel_dir}) && (6)\\
R_{smooth} &= K_1 \times e^{-\|\Delta a\|} && (7)\\
R_{effort} &= K_2 \times \sum_{i}^{n} (1 - \Omega_i), n = 4 && (8)\\
R_{pos} &= K_3 / (1 + K_4 \times \|\Delta \boldsymbol{p}\|^2) && (9)\\
R_{throttle} &= K_5 \times (1 - |T_{hover} - T|) && (10)\\
R_{ups} &= K_6 \times (\boldsymbol{q}\boldsymbol{v}_z\boldsymbol{q}^{-} + 1)^2 && (11)\\
R_{spin} &= K_7 / (1 + \omega_z^2) && (12)\\
R_{heading} &= K_8 / (1 + (\Delta\psi)^2) && (13)\\
R_{vel_dir} &= K_9 \times e^{-\boldsymbol{v}\cdot\boldsymbol{d}/\pi} && (14)
\end{aligned}$$

where R_{smooth} denotes the reward for continuous actions; R_{effort} is to encourage quadrotor to consume as little energy as possible; R_{pos} is position reward related to absolute position error; $R_{throttle}$ helps quadrotor to learn the expected hover throttle T_{hover}; R_{ups} maintains a stable horizontal attitude and avoids excessively large roll and pitch angles; $R_{heading}$ guides the target yaw direction ψ and R_{spin} tries to fix the spin around $\boldsymbol{z}^{\mathscr{B}}$-axis; finally, R_{vel_dir} leads to a straight flying direction from the current point to the target, using the dot product between the current flight direction vector $\boldsymbol{v}$ and the vector pointing to the target $\boldsymbol{d}$. Note that the $R_{throttle}$ equals zero when using *PY* and *LV* control modes that do not learn a collective thrust.

B.2 tracking

The fundamental observation in this task is insufficient. A common approach is to use trajectory playing, adding a sequence of the coming trajectory. Herein, we set an observation sequence of length 30 to store the coordinates of the upcoming 10 points, guiding quadrotor to follow the trajectory. Therefore, we do not define the observation by the differential, while we extend the observation dimension to 48: $(\mathrm{R}^{\rightarrow}, p_x, p_y, p_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, \mathrm{ref}^{\rightarrow})$, where $\mathrm{ref}^{\rightarrow}$ is a concatenation of the positions of ten reference points in the future. The reference trajectory is calculated by the parametric equations of a lemniscate (figure-eight curve), which has the longest side to range from $(-6, 6)$ meters:

$$\boldsymbol{x}(t) = \frac{3\sin(kt)}{1+\cos^2(kt)},\ \boldsymbol{y}(t) = \frac{3\sin(kt)\cos(kt)}{1+\cos^2(kt)},\ \boldsymbol{z}(t) = 1 \tag{15}$$

where k denotes a scaling factor to control the speed of quadrotor, directly determining the duration to complete one lemniscate lap. We also add distance reward part in this reward setting. Reward functions:

$$\boldsymbol{R} = R_{smooth} + R_{effort} + R_{dist} + R_{throttle} \tag{16}$$
$$+ R_{dist} \times (R_{ups} + R_{spin} + R_{heading}) \tag{17}$$
$$R_{smooth} = K_1 \times e^{-\|\Delta a\|} \tag{18}$$
$$R_{effort} = K_2 \times \sum_{i}^{n} (1 - \Omega_i), n = 4 \tag{19}$$
$$R_{dist} = K_3/(1 + K_4 \times \|\Delta \boldsymbol{p}\|^2) \tag{20}$$
$$R_{throttle} = K_5 \times (1 - |T_{hover} - T|) \tag{21}$$
$$R_{ups} = K_6 \times (\boldsymbol{q}\boldsymbol{v}_z\boldsymbol{q}^- + 1)^2 \tag{22}$$
$$R_{spin} = K_7/(1 + \omega_z^2) \tag{23}$$
$$R_{heading} = K_8/(1 + (\Delta\psi)^2) \tag{24}$$

We have similar reward items compared with task `hovering`. R_{dist} is used to replace the position error item R_{pos} in `hovering`. In R_{dist}, $\boldsymbol{p}$ is the distance between the current position and the expected position in the trajectory sequence.

B.3 target hitting

This task has the same state description with `hovering`. The state of balloon is randomly generated at every begin of the episode. Two kinds of rewards are designed. Dense reward encourages quadrotor to learn how to fly and move to the balloon, then a large sparse reward is given if the balloon is hit. Reward functions:

$$\boldsymbol{R} = R_{smooth} + R_{effort} + R_{guidance} + R_{ups} \tag{25}$$
$$+ R_{heading} + R_{hit} \tag{26}$$
$$R_{smooth} = K_1 \times e^{-\|\Delta a\|} \tag{27}$$
$$R_{effort} = K_2 \times e^{-\Sigma_i^4 \|a\|^2} \tag{28}$$
$$R_{guidance} = K_3 \times (\|\boldsymbol{p}^b - \boldsymbol{p}_{t-1}\| - \|\boldsymbol{p}^b - \boldsymbol{p}_t\|) \tag{29}$$
$$R_{ups} = K_4 \times (\boldsymbol{q}\boldsymbol{v}_z\boldsymbol{q}^- + 1)^2 \tag{30}$$
$$R_{heading} = K_5/(1 + (\Delta\psi)^2) \tag{31}$$
$$R_{hit} = K_6 \text{ if balloon is hit, else } 0. \tag{32}$$

where $R_{guidance}$ is a guidance reward that encourage quadrotor move from the start point $\boldsymbol{p}_s$ to the target balloon $\boldsymbol{p}^b$. A big positive sparse reward R_{hit} is added once the balloon is hit. Also, a great hitting brings the termination of an episode.

B.4 avoidance

Avoiding a dynamic obstacle by vision is always a exciting task for autonomous robots. We define the quadrotor state by ego-centric observation design: $(d_x, d_y, d_z, e_x, e_y, e_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, a)^{\mathscr{E}}$, where d_* denotes the goal direction, e_* denotes the Euler angle. All states are under the ego coordinate frame $\mathscr{E}$. We use *CTBR* control mode which means the action a has 4 dimensions. Besides, a simple CNN($[1,16,5,2,2] > [16,32,3,2,1] > [32,64,3,2,1] > ['AdaptiveAvgPool', 1]$) is set as a feature encoder to extract depth feature from binocular vision. At the beginning of an episode, we generate a cube with a suitable throwing speed and throwing angle. The onboard camera outputs the depth arrays with $(1,212,120)$ as the input of feature extractor. The feature extractor infers a 30-dimensional feature vector and makes a concatenation with other states for

final policy MLP. Reward functions:

$$\boldsymbol{R} = R_{smooth} + R_{effort} + R_{throttle} + R_{pose} \tag{33}$$
$$+ R_{alive} + R_{pose} \times (R_{ups} + R_{spin}) \tag{34}$$
$$R_{smooth} = K_1 \times e^{-\|\Delta a\|} \tag{35}$$
$$R_{effort} = K_2 \times e^{-\Sigma_i^4 \|a\|^2} \tag{36}$$
$$R_{throttle} = K_3 \times (1 - |T_{hover} - T|) \tag{37}$$
$$R_{pose} = K_4/(1 + \|\Delta[\boldsymbol{p}, \theta, \phi, \psi]\|^2) \tag{38}$$
$$R_{ups} = K_5 \times (\boldsymbol{q}\boldsymbol{v}_z\boldsymbol{q}^- + 1)^2 \tag{39}$$
$$R_{spin} = K_6/(1 + \omega_z^2) \tag{40}$$
$$R_{alive} = K_7 \text{ if alive, else } K_8 \tag{41}$$

Here we use item R_{pose} to indicate both position and orientation, θ, ϕ, ψ represent pitch, roll, yaw angles respectively. Besides, alive reward is also used in this task, while it is a dense reward: if the quadrotor stay alive for every step, it obtains a small positive reward; if the quadrotor is hit by the throwing object, it receives a big negative penalty.

B.5 planning

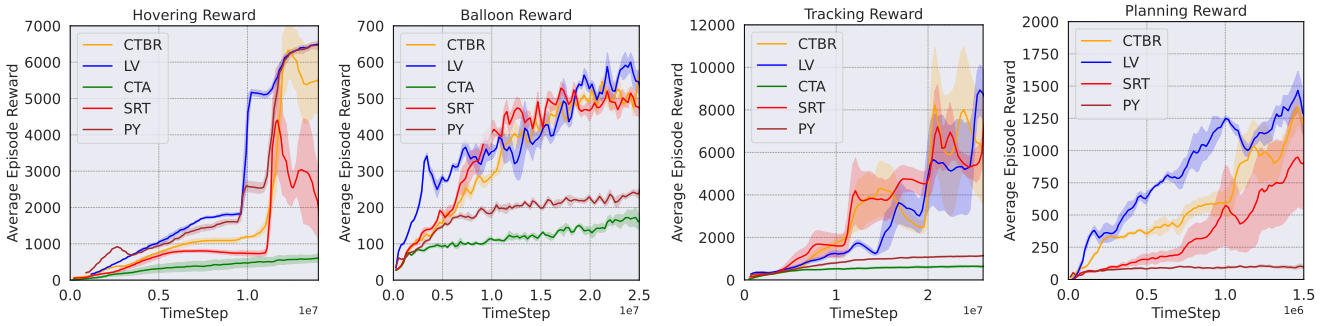
this work utilize the same observation definition in `avoidance`. However, navigation in unknown environments is much more complex than any other tasks. As the most important trick in `planning`, we introduce Euclidean signed distance field (*ESDF*) to describe the distance from quadrotor to the closest obstacle, and propose an *ESDF* reward to guide the quadrotor to the target, shown in the sub-figure (b) of the pipeline illustration. There is a problem: how to establish the connection between depth arrays $D \in \mathbb{R}^{H \times W}$ and *ESDF* distance x_{esdf}? In `planning`, we directly use the value of the point with the smallest depth in the depth array to approximate the *ESDF* distance:

$$x_{esdf} = D_{\min} = \min_{i,j} D_{i,j}$$

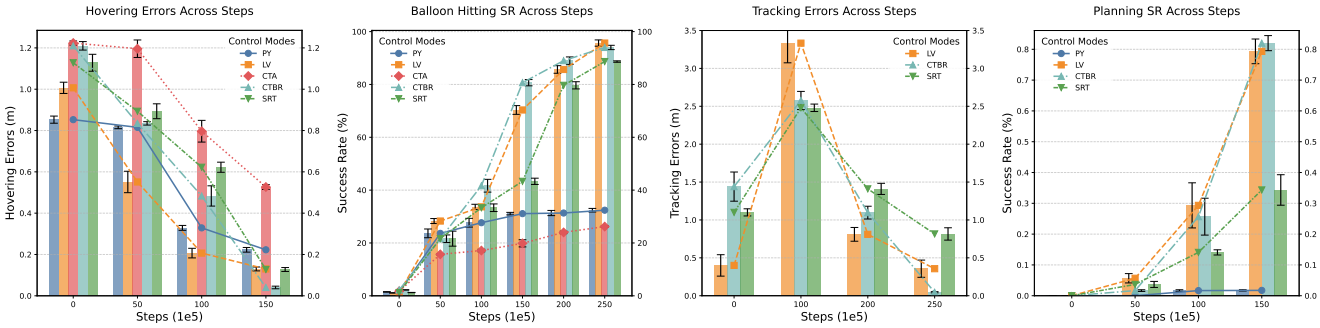
Rely on this, we define the *ESDF* reward as $R = 1 - e^{-x^2}$, to encourage quadrotor to pull itself away from the nearest *ESDF* point. Reward functions:

$$\boldsymbol{R} = R_{smooth} + R_{effort} + R_{throttle} + R_{guidance} \tag{42}$$
$$+ R_{speed} + R_{height} + R_{heading} + R_{ups} \tag{43}$$
$$+ R_{guidance} \times (R_{esdf} + R_{alive}) + R_{goal} \tag{44}$$
$$R_{smooth} = K_1 \times (\|\Delta a\| + \|\boldsymbol{\omega}_{\mathscr{E}}\|) \tag{45}$$
$$R_{effort} = K_2 \times e^{-\Sigma_i^4 \|a\|^2} \tag{46}$$
$$R_{throttle} = K_3 \times (1 - |T_{hover} - T|) \tag{47}$$
$$R_{guidance} = K_4 \times (\|\boldsymbol{p}^{target} - \boldsymbol{p}_{t-1}\| - \|\boldsymbol{p}^{target} - \boldsymbol{p}_t\|) \tag{48}$$
$$R_{speed} = -K_5 \times (1 - e^{-K_6 \times (\|\boldsymbol{v}_x^{\mathscr{E}}\| - K_7)^2}) \tag{49}$$
$$R_{height} = \min(\min(p_z - K_8, 0), K_9 - p_z) \tag{50}$$
$$R_{esdf} = K_1 0 \times (1 - e^{-K_1 1 \times x_{esdf}^2}) \tag{51}$$
$$R_{ups} = K_{12} \times (\boldsymbol{q}\boldsymbol{v}_z\boldsymbol{q}^- + 1)^2 \tag{52}$$
$$R_{alive} = K_{13} \text{ if } x_{esdf} > K_{14} \text{ else } 0 \tag{53}$$
$$R_{goal} = K_{15} \text{ if goal reached, else } 0 \tag{54}$$

In this task, most of observations are under $\mathscr{E}$ frame. K_7 denotes the maximum expected speed during the flight. K_8, K_9 are the minimum and maximum expected flight height. We use the minimum value in the depth array to replace *ESDF* distance x_{esdf}. Furthermore, we also use *ESDF* distance to judge whether the quadrotor is crashed or not by compute if x_{esdf} less than range K_{14}.



(a) `Hovering` training reward (b) `Target Hitting` training reward (c) `Tracking` training reward (d) `Planning` training reward



(e) `Hovering` errors convergency (f) `Target Hitting` SR convergency (g) `Tracking` errors convergency (h) `Planning` SR convergency

Figure 7. Training indexes changing in four tasks: `hovering`, `target hitting`, `tracking`, and `planning`. Task `avoidance` is not illustrated because it need training curriculum and the reward makes no sense to refer. (a-d) present the reward changes during the training process. (e) and (g) show the errors in task `hovering` and `tracking`; (f) and (h) are success rate of task `target hitting` and `planning` respectively.

C Control Results under Different Modes

First, we validate the performance of tasks in the `AirGym` using `rlPx4Controller`'s hierarchical controllers combined with NN-based policy control, illustrated in Fig.8. Our experiments involve five control levels: *PY*, *LV*, *CTA*, *CTBR*, and *SRT*. The RL-trained policy in each level of control mode generates the corresponding expected command, then `rlPx4Controller` executes the remaining loops of control. The training rewards are shown in Fig.7(a). From the point of training speed, the more controller layers replaced by the RL policy, the slower the learning process becomes. This trend aligns well with intuition, as the neural network requires more time to learn the attitude stabilization of the quadrotor's inner loop. The slow training under *CTA* mode is a special case. This may mean the attitude loop commands are not as direct as *CTBR* and *SRT* when policies are applied to the quadrotor. These two control styles are closer to the actuators, allowing the network to directly learn their relationships within the dynamics system without dealing with intermediate complex control logic. In contrast, in *PY* and *LV*, RL focuses on generating high-level commands for the outer loop, while the quadrotor itself can inherently maintain attitude stability, making it easier and faster to learn. Comparing the converging and hovering accuracy across these five modes, *CTBR* demonstrates an impressive advantage. We conduct eight trials for each type of experiment and calculate the average of hovering errors, as shown in Fig.7(e). In the scenario where the quadrotor is randomly initialized within a cubic space with a side length of 2 m and ultimately converges to the point $(0,0,1)$, the performance of *CTBR* can maintain hover accuracy within 0.1 m. Other methods can also maintain stability, but they fail to achieve the same level of control accuracy and robustness as *CTBR*. This result verifies the conclusion in again that *CTBR* shows a good performance and a strong resilience against dynamics mismatch, on different types of hardware.

In other tasks, we showcase the training performances of `target hitting` and `tracking`, which are shown in Fig.7(b)(c). In `target hitting`, we find *LV*, *CTBR*, and *SRT* control modes reach convergence much faster. Although rewards under *LV* and *SRT* obtain similar scores, the actual effects are not consistent, shown in Fig.7(f). Comparing the success rate (SR) of these five modes, control under *CTBR* and *LV* obtain higher SR. In *LV* control mode, the reward increases rapidly because the attitude loop in this mode is inherently stable, allowing the agent to focus only on learning how to move toward the target. However, the quadrotor under *SRT* mode does not first learn to maintain inner-loop stability but instead directly learns how to crash into the target without considering its own attitude, which simplifies the task. In brief, the quadrotor does not need to learn how to fly while only needing to learn how to hit the balloon. Task `tracking` achieves satisfactory results in three control modes: *CTBR*, *LV*, *SRT*, illustrated in Fig.7(c)(g) within limited timesteps. Here we record the tracking errors of these three control modes during the training process (tracking errors under *PY* and *CTA* are not described because its low SR and the value is meaningless). At the beginning, it is obvious that all control modes obtain errors less than 1.5 meters since the quadrotor is always reset at the first expected point, especially under *LV* modes because these two modes provide a robust inner-loop stability. After the training, *CTBR* and *LV* report a better tracking precision. Task `planning` is the most challenging one among all tasks since it faces much more uncertain circumstances. From the SR record in Fig.7(h), *PY* mode demonstrates obvious limitations for navigation purpose. *LV* and *CTBR* based learning shows an impressive advantage, with approaching 80% SR of navigation. Considering form the standpoint of visual effect, we believe the *CTBR* mode to deliver superior performance, as this mode enables direct attitude-loop control of the quadrotor, resulting in more agile flight characteristics.

In summary, control under *CTA* mode shows limited performance under all tasks, with longer duration and lower reward. The reasons can be diverse. The most important one is we employ quaternions as the network output, but the property of conjugate equivalence in quaternions may result in multiple valid output representations for the same orientation. This can lead to an imbalanced distribution in the training data.

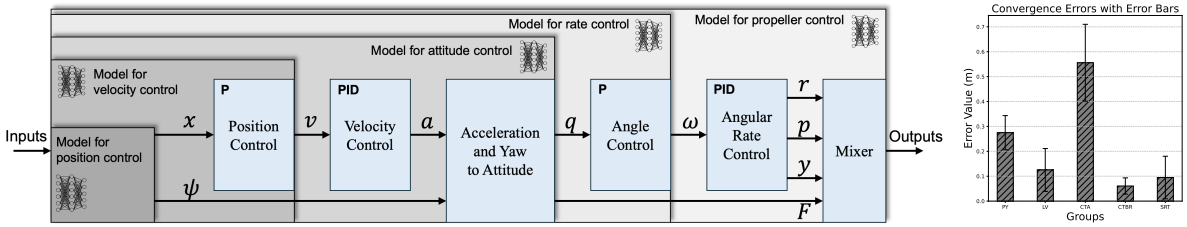
D Infrastructures

D.1 Quadrotor Hardware Device: X152b

We design a quadrotor `X152b` based on drone racing frame by OddityRC, illustrated in Fig.6. The diagonal distance of this quadrotor is 152mm. It is equipped with the Rockchip RK3588s as onboard processor and a mini open-sourced PX4 autopilot NxtPX4. The onboard computer delivers over 6 TOPs of AI performance, enabling us to perform lightweight neural network inference almost in real time directly onboard. Additionally, it integrates an Intel RealSense D430 camera for perception acquisition. To enhance the agility and flight endurance, the weight is strictly kept under 400g (excluding the battery). We report a more than $35\mathrm{m}\cdot\mathrm{s}^{-1}$ maximum flying speed under first personal view (FPV) mode and over $11\mathrm{m}\cdot\mathrm{s}^{-1}$ speed under sim-to-real mode.

D.2 Vectorized Simulation Platform: AirGym

We develop `AirGym` platform based on IsaacGym and inspired by previous work Aerial Gym. Compared with it, we rebuild the flight dynamics and make massive changes to align reality and simulation. (i) We use the physical dynamics parameters and



(a) Control Pipeline for Different Modes (b) Hovering Error

Figure 8. Control pipeline under different control modes. Five modes are validated: position (*PY*), velocity (*LV*), angle (*CTA*), angular rate (*CTBR*), propeller (*SRT*). Errors after converging are shown in (b).

real robot mesh file to replace general simplified quadrotor in Aerial Gym. (ii) We provide a new asset registor and manager, making code more concise and allowing it to load various classes and quantities of objects. Also, we add more assets to make the task more realistic. (iii) We provide multiple levels of RL policies to control a quadrotor, including *PY*, *LV*, *CTA*, *CTBR*, and single-rotor thrust (*SRT*), by replacing the corresponding control loop in a cascade PID.

We realize dynamics by directly applying force and torque to a rigid body to compute the collective thrust $\boldsymbol{f}$ and torque $\boldsymbol{\tau}$ applied to the root of the quadrotor, as shown in Eq.3. The relationship between propeller rotation speed Ω and both thrust f and torque τ are nonlinear in reality, typically exhibiting a quadratic correlation in Eq.4. Determining this function requires system identification. However, in controllers based on differential flatness (include PX4), there is a normalized hover throttle to counteract gravity, and we linearize the quadratic curve at this point because the variation in its second derivative is negligible. Therefore, in this work, we approximate the relationship between propeller speed and thrust as linear and use actual flight data to estimate the true hover throttle.

D.3 Parallelized Geometric Controller: rlPx4Controller

Implementing quadrotor flight in large-scale vectorized environments requires parallel flight control. To provide parallelized computing, one solution is to inherit the neural network (NN) forward propagation module in `PyTorch` to tensorfy vectors, such as Aerial Gym and OmniDrones. Nevertheless, current works have not fully aligned with the PX4 logic and parameters, which is critical in the engineering implementation of PX4. To address this issue, we implemented a parallel geometric controller strictly similar to PX4. This allows us to guarantee that the data trained in simulation aligns perfectly with the real-world system characteristics during the sim-to-real process, especially in holding the altitude and handling the hover throttle.

We rewrite the PX4 flight control in `C++` according to nonlinear quadrocopter attitude control, and enable parallel computation. While achieving the same mathematical calculation, a stringent correspondence between the simulation controller and PX4 is established.

Specifically, we extract and encapsulate the control part of the PX4 autopilot by pybind11, providing `Python` interfaces that can be directly invoked at various control levels. We give a simple example of the usage of `rlPx4Controller`. The code is accessible online, offering interfaces that are straightforward to install and easy to use:

```
git clone \
  git@github.com:emNavi/rlPx4Controller.git
cd rlPx4Controller
pip install -e .
```

Using the `rlPx4Controller` is straightforward. All we need to do is import the `rlPx4Controller`, then instantiate a controller of the expected type. It can be a position (*PY*), velocity (*LV*), attitude (*CTA*), or rate (*CTBR*) controller, and then call its `update` function:

```
from rlPx4Controller import ParallelRateControl, ParallelVelControl, ParallelAttiControl,
    ParallelPosControl
# According to which control layer is selected
# Use Rate control as an example
controller = ParallelRateControl(num_envs)
controller.set_status(status, dt)
cmd_thrusts = controller.update(actions)
```

`num_envs` is the number of parallel computing processes. The function `set_status` takes as input the status of the drone, including position, attitude in quaternion, linear velocity, and angular velocity, and settings for the controller's simulation

interval `dt`. The function `update` takes actions as input and outputs the final command of thrusts. More detailed features and usages of `rlPx4Controller` can be found in the online documentation[5].

D.4 Onboard ROS Inference Node: AirGym-Real

As aforementioned, `AirGym` provides a platform for fast training in simulator. During Sim-to-Real, we need software similar to AirGym that can directly load the trained neural network on the onboard system, interact with sensor and flight control data, and perform real-time inference from states to actions. Particularly in the wild, where there is no direct pose information available, such as from motion capture systems, we can only rely on other tricky methods to obtain the required observation for the inference.

We implement the above functionality as a ROS node to obtain state from PX4 autopilot and camera, as well as to conduct the inference. To obtain full states of quadrotor in the wild, we run a visual-inertial odometry (VIO) onboard. VINS-fusion is selected for this task, which is achieved by extracting feature points from the environment captured by the camera and combining them with the IMU information from PX4 Autopilot. It is worth mentioning that PX4 firmware will estimate IMU bias using input position data by default, which could contaminate the raw IMU data and lead to a nonconvergent solution during the VIO fusion. To address this issue, we modify the PX4 firmware by removing the IMU bias estimation component, which is commonly used in engineering but is less crucial for research. Meanwhile, we ensure the position estimation in the EKF to ultimately approach the VIO output position, to avoid re-fusion of IMU data. Furthermore, it brings another convenience: the coordinate alignment between the VIO system and the PX4 controller is not a concern in our system. The effectiveness and robustness of VINS classes have been extensively validated in previous studies. Therefore, this paper will not elaborate on their performance in detail. The raw depth data undergoes preprocessing, including resizing and depth truncation, before being used to construct the observation input for neural network inference. Finally, the computed action is published to the ROS topic.

D.5 PX4 Control Bridge: control_for_gym

`control_for_gym` is a PX4 middleware stack to publish NN-inferenced commands to expected levels of PX4 control loop. Two functions are primarily implemented: constructing a finite state machine (FSM) to enable switching between trained policy control and classical PID control; and forwarding control commands to the PX4 Autopilot controller based on the selected control hierarchy.

During DRL training, quadrotors are often initialized from a specific state, such as hovering stably at a certain position. This requires the quadrotor to stabilize to the desired state before inference in reality, ensuring continuous control and improving the success rate of sim-to-real transfer. To this end, a simple FSM is constructed for providing steady switching, which is shown in the pipeline in the manuscript. While transferring policy to the wild, we first conduct an automatic taking-off behaviour and then hover in place to ensure the quadrotor starts to infer with an appropriate state. Subsequently, the model computes the right commands.

After the inference is done to compute an action, it should be transmitted to the flight controller for execution. In our work, we remap the action topic to the corresponding execution topic, to send specific message type of control commands at different levels to the PX4 controller via MAVROS.

[5] https://rlpx4controller.readthedocs.io/en/latest/index.html