Hovering Flight of Soft-Actuated Insect-Scale Micro Aerial Vehicles using Deep Reinforcement Learning

Anonymous Authors

***Abstract*— Soft-actuated insect-scale micro aerial vehicles (IMAVs) pose unique challenges for designing robust and computationally efficient controllers. At the millimeter scale, fast robot dynamics (∼ms), together with system delay, model uncertainty, and external disturbances significantly affect flight performances. Here, we design a deep reinforcement learning (RL) controller that addresses system delay and uncertainties. To initialize this neural network (NN) controller, we propose a modified behavior cloning (BC) approach with state-action re-matching to account for delay and domain-randomized expert demonstration to tackle uncertainty. Then we apply proximal policy optimization (PPO) to fine-tune the policy during RL, enhancing performance and smoothing commands. In simulations, our modified BC substantially increases the mean reward compared to baseline BC; and RL with PPO improves flight quality and reduces command fluctuations. We deploy this controller on two different insect-scale aerial robots that weigh 720 mg and 850 mg, respectively. The robots demonstrate multiple successful zero-shot hovering flights, with the longest lasting 50 seconds and root-mean-square errors of 1.34 cm in lateral direction and 0.05 cm in altitude, marking the first end-to-end deep RL-based flight on soft-driven IMAVs.**

I. Introduction

Inspired by the exquisite maneuverability of natural insects, the robotics community has developed insect-scale micro aerial vehicles (IMAVs) that weigh less than a gram and are capable of stable hovering [1]–[4]. Among these platforms, a class of soft-actuated IMAVs [4] has gained particular attention due to their resilience to collisions [5]. Driven by muscle-like dielectric elastomer actuators (DEAs), these soft-actuated sub-gram robots can absorb external impacts, highlighting the potential of IMAV applications such as assisted pollination in unstructured environments.

Despite demonstrating unique capabilities, these DEA-driven IMAVs face distinct challenges in flight controller design due to their soft actuation. First, while the soft robots exhibit excellent impact resistance, they respond more slowly and require real-time communication between the off-board sensing, power, and control subsystems. Prior work [4] reported a 15- to 20-ms system delay that is contributed by the soft actuation and the communication between the robot and external apparatus. Such delay is critical to IMAVs, which have fast body dynamics in the millisecond range. Traditional model-based flight controllers [4] [5] mitigate this issue by setting non-aggressive control gains; however, this method greatly reduces the closed-loop control performance, making it difficult to track aggressive and long trajectories, like the ones shown on larger scale flying robots [6].

The second control challenge comes from model uncertainty and large external disturbances. The fabrication of soft IMAVs requires manual assembly, which leads to a 10-20% error in the estimation of the robot's moment of inertia (MoI). In addition, the soft actuators require 1.4- to 2-kV voltage for flights, so these DEA-driven IMAVs are tethered to a bundle of wires for offboard power during their aerial maneuvers. These wires contribute to position and attitude-dependent disturbances that are difficult to model.

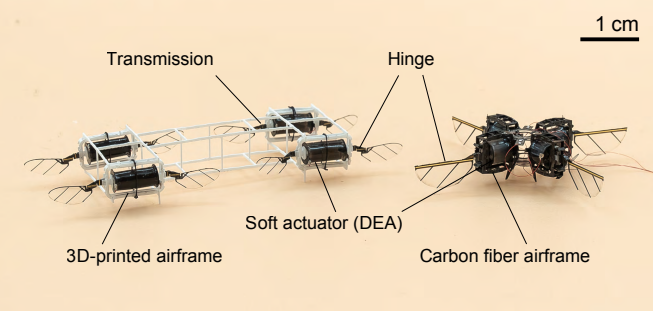

Fig. 1. An image of a 720-mg eight-wing micro-aerial-robot (left) and an 850-mg four-wing micro-aerial-robot (right) both driven by DEAs. The robot consists of either a 3D-printed or a carbon fiber airframe that connects four modules. Each module has a DEA, transmissions, wing hinges, and wings. The robot requires external systems for sensing, control, and power.

To address these challenges, learning-based methods were applied in several prior studies [7] [8]. In a previous work [7], researchers designed controllers with fixed structures and then used learning methods to identify the control parameters based on flight experiments. Another work [8] combined model-based and model-free methods to design a hovering flight controller. The simulations show substantial improvement, but the 1.5-second real-world flight demonstrations suffer a large position error of over 10 cm. This result shows that the unaccounted-for *Sim2Real* gap has a substantial influence on the flight performance of IMAVs and highlights the difficulties of bringing the controller from simulation to real-world IMAVs, emphasizing the importance of incorporating model uncertainty and system delay into the simulator.

Another work [9] presented a cascaded control architecture, which connects a positional neural network (NN) controller to a model-based attitude controller. The researchers used supervised learning to train a NN with expert demonstrations from a hand-tuned model predictive controller (MPC). While the flight results show hovering capability, this supervised learning method relies heavily on the performance of the expert controller and may lead to sub-optimal behavior as the time horizon increases [6].

Here, we propose a deep reinforcement learning (RL) approach for controlling IMAVs and demonstrate stable real-world flights on these soft-actuated platforms. In contrast to previous work [8], we successfully bridge the *Sim2Real*

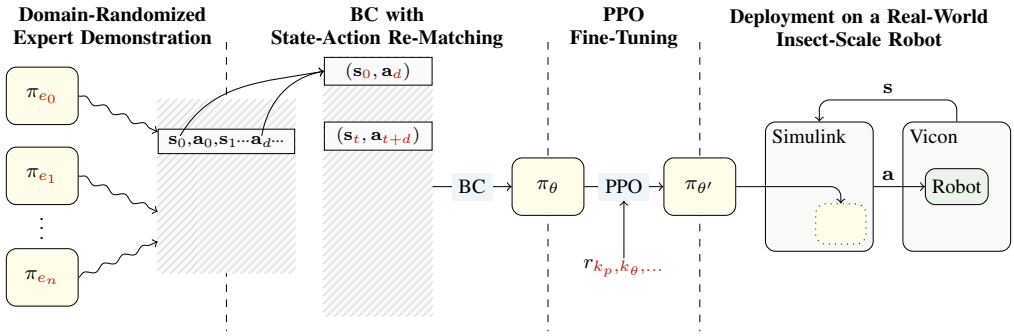

Fig. 2. Overview of our proposed controller design. First, from a model-based controller, π_{e_i}, a set of expert demonstrations is generated with randomized domain parameters. Then, we re-match the delayed state with the action to account for system delay. We implement behavior cloning to initialize a neural network controller. Next, in the RL phase, the control policy is fine-tuned with PPO to improve performance and reduce driving command fluctuations. Finally, the controller is integrated into the Matlab Simulink Real-Time environment for demonstrating robot hovering flight.

gap and showcase translational flight performance from the simulator to multiple real-world platforms at the insect scale. Compared to [9], our method replaces the entire control scheme with a single NN controller and is trained through unsupervised learning (RL) to seek optimal performance without relying on hand-tuned expert demonstrations.

Our new approach has two main features: 1) we explicitly account for the system delay of soft IMAVs during the initialization of NN by using state-action re-matching in behavior cloning (BC), and also incorporate this delay in the simulator for RL with proximal policy optimization (PPO). 2) we randomize domain parameters, including mass, MoI, and external disturbances, in both BC and RL phases to improve controller robustness against system uncertainty.

These design choices result in a new flight controller that is resilient to system delay and model uncertainty on soft-actuated IMAV. We deploy this new type of controller on two distinct DEA-driven IMAVs (Fig. 1) and evaluate their performance. Our results demonstrate multiple zero-shot hovering flights for both robots, marking the first successful deep RL flights at the insect scale. The longest flight we conducted lasts 50 seconds and achieves lateral and altitude root mean square errors (RMSEs) of 1.34 cm and 0.05 cm that outperform the state-of-the-art robots of similar scale [3] [10]. By explicitly accounting for system delay and model uncertainty, we achieved substantial flight performance improvement with deep reinforcement learning, representing a significant step toward unlocking the full potential of fast dynamics in soft-driven IMAVs.

II. Controller Design

In this section, we describe our flight controller design under the deep RL framework. First, we define the robot states and controller actions, together with the dynamics and flight simulator. Next, we develop a modified BC method to initialize the NN, which accounts for the system delay and uncertainties by using state-action re-matching and domain-randomized expert demonstrations. Finally, we design a reward function and use RL with PPO to further optimize the policy and improve driving commands' smoothness. The high-level design of our learning-based controller is illustrated in Fig. 2.

A. States & Actions

The states of our robot include positions $\mathbf{p} = [x, y, z]^T$, velocities $\mathbf{v} = [\dot{x}, \dot{y}, \dot{z}]^T$, rotational angles represented by quaternions $\mathbf{q} = [q_x, q_y, q_z, q_w]^T$, and angular velocities $\boldsymbol{\omega} = [p, q, r]^T$. The state vector is expressed as:

$$\mathbf{s} = [\ x,\ y,\ z,\ q_x,\ q_y,\ q_z,\ q_w,\ \dot{x},\ \dot{y},\ \dot{z},\ p,\ q,\ r\ ]^T,$$

where $\mathbf{p}$, $\mathbf{v}$, and $\mathbf{q}$ are in the world-fixed frame and $\boldsymbol{\omega}$ is on the body frame. The action of the robot is defined as

$$\mathbf{a} = [\ F,\ \tau_x,\ \tau_y\ ]^T,$$

where F represents the total thrust force along the body z-axis; τ_x and τ_y are the torques with respect to the body x-axis and body y-axis. While our robot cannot generate yaw torque (τ_z), other works [4] [5] have shown that hovering flight does not require yaw control authority.

B. Robot Dynamics & Simulator

The simulator for RL is constructed based on the 6-DOF rigid body dynamics. Compared to existing UAV simulators, our model accounts for yaw motion damping, external disturbances, and actuation delay.

We aim to develop the NN controller in a near-zero yawing condition to simplify training and use the method in [9] to re-map actions to the correct body frame. The robot yawing dynamics is thus intentionally constrained by a large damping term $-k_y\dot{r}$. In addition, the power tethers create external force disturbance ($\mathbf{F}_{dist}$) and torque disturbances ($\tau_{dist,x}$, $\tau_{dist,y}$) on the robot. The robot dynamics is described by:

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{1}$$

$$\dot{\mathbf{v}} = (\mathbf{R}\left[\ 0,\ 0,\ F\right]^T + \left[\ 0,\ 0,\ \text{-}mg\right]^T + \mathbf{F}_{dist})/m, \tag{2}$$

$$\dot{\mathbf{q}} = (\mathbf{q} \otimes [\ 0,\ \boldsymbol{\omega}^T\ ]^T)/2, \tag{3}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(\text{-}\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \left[\tau_x + \tau_{dist,x},\ \tau_y + \tau_{dist,y},\ \text{-}k_y\dot{r}\right]^T), \tag{4}$$

where $\mathbf{R}$ is the rotational matrix, $\mathbf{J}$ is the diagonalized moment of inertia tensor, and $\otimes$ is an operator representing quaternion multiplication.

To model system delay [11], we specify a delay time d. The action $\mathbf{a}_t$, which is computed at a time t, would be executed on the robot at the time $t + d$; the compact form of robot dynamics can be expressed as

$$\dot{\mathbf{s}}_{t+d} = f(\mathbf{s}_{t+d},\ \mathbf{a}_t), \tag{5}$$

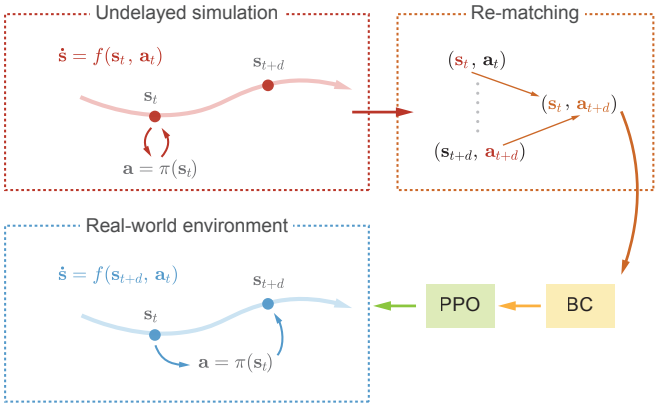

Fig. 3. Workflow of State-Action Re-matching. The expert demonstration is first rolled out in the undelayed simulator; then, we offset the state-action pairs by d and have $(\mathbf{s}_t, \mathbf{a}_{t+d})$ as a pair for supervised learning to clone the delay-compensated controller. The policy then goes through PPO fine-tuning and is deployed to the real-world environment.

where the function f represents the nonlinear robot dynamics described in Eq. (1)-(4). To solve Eq. (5) in discrete time, we use the forward Euler method with a step size of 1 ms.

C. Modified Behavior Cloning for Controller Initialization

To initialize the NN controller, we design a modified BC approach that accounts for model uncertainty and system delay. We first generate expert demonstrations using a model-based controller [12] with randomized robot parameters and disturbances to improve robustness. Then, we re-match the state-action pairs to account for system delay. Ultimately, we utilize supervised learning to train an NN controller that imitates the expert demonstrations.

1) Domain-Randomized Expert Demonstration: To bridge the discrepancy between simulation and real-world environments (*Sim2Real* gap), we randomize three types of domain parameters, including robot parameters $\mathcal{R} = \{m, I_{xx}, I_{yy}\}$, environmental disturbance parameters $\mathcal{E} = \{\mathbf{F}_{dist}, \tau_{dist,x}, \tau_{dist,y}\}$, and delay d, where I_{xx} and I_{yy} are MoI with respect to body x-axis and y-axis.

We specify a value range for each parameter that corresponds to fabrication variation and model uncertainty. For instance, we choose I_{xx} from the interval $[0.75I_{xx}, 1.25I_{xx}]$. Then we pick several ϕ_i where each of them is a mapping function from the set $\mathcal{P}$ ($\mathcal{P} = \mathcal{R} \cup \mathcal{E} \cup \{d\}$) to their corresponding values. The closed-loop dynamics under the expert policy π_e in the environment parameterized by ϕ_i becomes

$$\dot{\mathbf{s}} = f_{\phi_i}(\mathbf{s}, \pi_e(\mathbf{s})). \tag{6}$$

Based on this domain-randomized closed-loop dynamics, we roll out trajectories with various initial states, $\mathbf{s}_0$, to create expert demonstrations for the training data set.

In addition, by incorporating the disturbances, which push the robot slightly away from the nominal trajectory, we can efficiently sample more states and generate corresponding expert demonstrations during the rollout (Eq. 6) without using iterative dataset aggregation (DAgger) [13].

2) State-Action Re-Matching: To account for the system delay, we first turn off delay in the simulator to obtain an undelayed ideal demonstration $\mathcal{T}_{\text{ud}} = \{\mathbf{s}_0, \mathbf{a}_0, ..., \mathbf{s}_t, \mathbf{a}_t, ..., \mathbf{s}_{t+d}, \mathbf{a}_{t+d}, ..., \mathbf{s}_T\}$ with a model-based controller (Fig. 3, upper left).

Next, we "re-match" the state-action pairs. We recognize that in the real world, an action $\mathbf{a}_t$ would be executed at $\mathbf{s}_{t+d}$ due to system delay (Fig. 3, lower left). To have the optimal action $\mathbf{a}_{t+d}$ be executed at $\mathbf{s}_{t+d}$, the controller needs to generate $\mathbf{a}_{t+d}$ at $\mathbf{s}_t$. Hence, we re-match the state-action pairs with respect to time and choose $(\mathbf{s}_t, \mathbf{a}_{t+d})$ as a pair in the training data set $\mathcal{D}$, where $\mathcal{D}$ is defined as

$$\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_{t+d}) | \mathbf{s}_t, \mathbf{a}_{t+d} \in \mathcal{T}_{\text{ud}} \text{ where } 0 \leq t < T - d\}.$$

3) Behavior Cloning (Supervised Learning): With the re-matched dataset, $\mathcal{D}$, the NN controller π_θ can be initialized through BC by solving the following optimization equation

$$\max_{\theta} \sum_{(\mathbf{s},\mathbf{a}) \in \mathcal{D}} \log \pi_\theta(\mathbf{a}|\mathbf{s}).$$

This cloned policy, π_θ, has already accounted for the delay from the soft actuators.

D. Reward Function for Reinforcement Learning

To further optimize the BC policy, π_θ, with RL, we design a reward function that considers both states and actions. The state-dependent objective, $r_s(\mathbf{s})$, aims to minimize the distance between the current state and the setpoint (origin). To intuitively assign rewards, we use Euler angles (ϕ, θ, ψ) retrieved from $\mathbf{q}$. The reward function involves bringing positions ($\mathbf{p}$), velocities ($\mathbf{v}$), two Euler angles (ϕ and θ), and two angular velocities (p and q) to zero. Since we cannot control the body yaw rate, we do not assign rewards on the states ψ and r. The reward function on states is defined as:

$$\begin{aligned} r_s(\mathbf{s}) = -(\, & k_p ||\mathbf{p}||^2 + k_e(||\phi||^2 + ||\theta||^2) + k_v ||\mathbf{v}||^2 \\ & + k_\omega(||p||^2 + ||q||^2) \,), \end{aligned}$$

where k_p, k_e, k_v, and k_ω are hyperparameters that determine the relative weight of each state reward. To specify the action rewards, we penalize aggressive (fluctuating) control outputs and their deviations from the nominal action. The reward functions are defined as

$$\begin{aligned} r_f(\mathbf{a}) = -(\, & k_{ff} ||F_t - F_{t-1}||^2 + k_{\tau_x f} ||\tau_{x,t} - \tau_{x,t-1}||^2 \\ & + k_{\tau_y f} ||\tau_{y,t} - \tau_{y,t-1}||^2 \,), \; \forall t \in (0, T], \end{aligned}$$

$$r_n(\mathbf{a}) = -(k_f ||F - F_n||^2 + k_{\tau_x} ||\tau_x||^2 + k_{\tau_y} ||\tau_y||^2),$$

where k_{ff}, $k_{\tau_x f}$, $k_{\tau_y f}$, k_f, k_{τ_x}, and k_{τ_y} are hyperparameters that determine the relative weight of each action reward, and F_n is the nominal thrust at the hovering state. The action reward is given by $r_a(\mathbf{a}) = r_f(\mathbf{a}) + r_n(\mathbf{a})$. The total reward function sums contributions from states and actions:

$$r(\mathbf{s}, \mathbf{a}) = r_s(\mathbf{s}) + r_a(\mathbf{a}).$$

E. Reinforcement Learning through PPO

We utilize reinforcement learning to further optimize the policy π_θ that is initialized by modified BC. Specifically, we choose to train our policy in the delayed simulator described in Sec. II-B (Eq. 5), whose domain is parameterized by

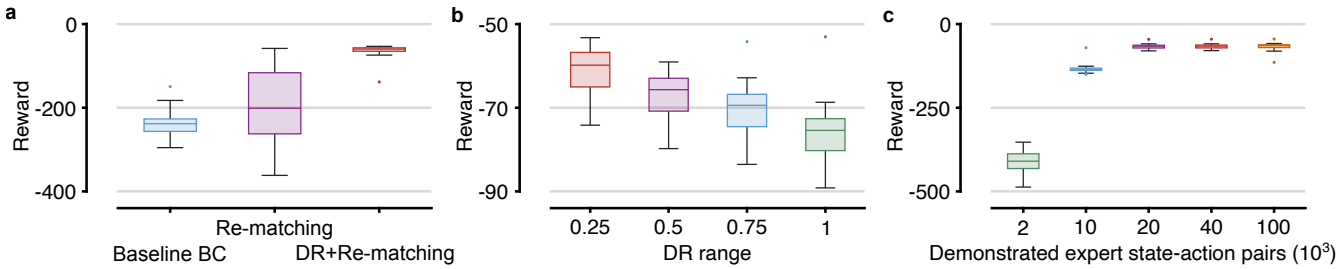

Fig. 4. Simulation results of behavior cloning. (a) Comparison of the baseline method, the method with state-action re-matching, and the method with both state-action re-matching and domain randomization. Colored boxes show 25%, 50%, and 75% percentiles and the black bars show non-outlier minimum and maximum. Dots are outliers that are 1.5 interquartile range (IQR) away from the top or bottom of the box. (b) Comparison of controller performance as the randomization range increases. (c) Controller performance as a function of training data set size.

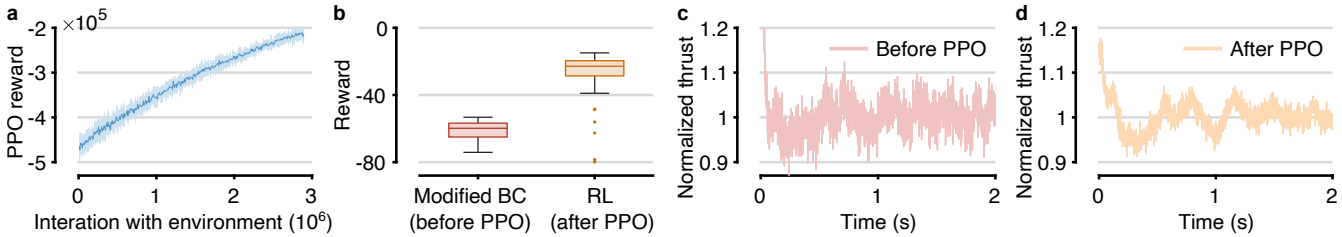

Fig. 5. Simulation results of before and after PPO fine-tuning. (a) shows the training curve of the PPO with respect to the chosen reward function. The dark blue line shows the median rewards and the light blue shaded region represents two standard deviations away from the median. (b) displays the performance improvement in simulation after PPO fine-tuning. (c-d) compare the aggressiveness of command before and after PPO, the fluctuation in command is greatly reduced after PPO fine-tuning.

set $\mathcal{P}$ in Sec. II-C.1 with various initial conditions, $\mathbf{s}_0$. The proximal policy optimization (PPO) is implemented to update the policy with the reward function defined in Sec. II-D.

The main challenge in implementing PPO involves setting appropriate hyperparameter values in the reward function. Quadrotor-like aerial robots are 4th-order systems, where the effects of commanded torques would appear in positional states after being integrated four times. This property makes positional rewards extremely sparse in the policy optimization formulation. As a result, although achieving position control is our primary objective, we still assign rewards to intermediate states such as velocity, Euler angles, and body angular velocity.

In addition to setting the state rewards, it is also crucial to set appropriate action reward hyperparameters. The actions generated through the BC policy are non-smooth [14] because BC learns discrete state-action pairs without considering the continuity in time. While fluctuating actions could generate reasonable results in simulation, they are harmful to the lifetime of robotic hardware [15]. Hence, we assign hyperparameters to the action reward function while minimizing the negative impact on control effectiveness.

F. Deployment on Soft-Actuated Robot

After performing PPO fine-tuning, we deploy the RL-trained policy, $\pi_{\theta'}$, on our customized experimental setup that runs Matlab Simulink Real-Time at 1 kHz. The flight arena consists of a commercial motion tracking system (Vicon), a specialized controller (Speedgoat), and high-voltage amplifiers (Trek). We built a 720-mg eight-wing IMAV and an 850-mg four-wing IMAV; both of them have four soft actuators (DEAs).

III. Results

We conduct simulations and flight experiments to evaluate controller effectiveness. In simulations, the modified BC improves the reward (median) by 75% compared to the baseline BC. The PPO further enhances the reward (median) by 62% and reduces the thrust fluctuation by 51%, which is crucial for performing experimental validation on real-world hardware. In flight experiments, we achieve multiple zero-shot stable hovering where our position errors outperform the state-of-the-art long endurance flights on IMAVs.

A. Simulation Results

To train the policy, we use a NN with 2 hidden layers and 32 neurons per layer. To compare their performance in simulation, we run each type of controller in the delayed simulator for 5 seconds and repeat 100 times with different robot parameters $\mathcal{R}$, environmental disturbances $\mathcal{E}$, delays d, and initial conditions $\mathbf{s}_0$.

First, we evaluate the effectiveness of the two techniques: state-action re-matching and domain-randomized expert demo. We compare the performance of three behavior cloned controllers: 1) baseline BC (non-randomized expert demo and no re-matching); 2) BC with state-action re-matching (non-randomized expert demo); and 3) BC with both state-action re-matching and domain-randomized expert demonstrations. In Fig. 4a, the baseline BC policy returns the lowest median reward of -238.2, while the policy trained on state-action re-matching (without randomized domains) scores -201.2, indicating that the proposed re-matching method improves controller performance in a delayed environment. The policy trained with the randomized domains (with re-matching), achieves the best median reward of -59.8. This result showcases that randomizing parameters at the behavior

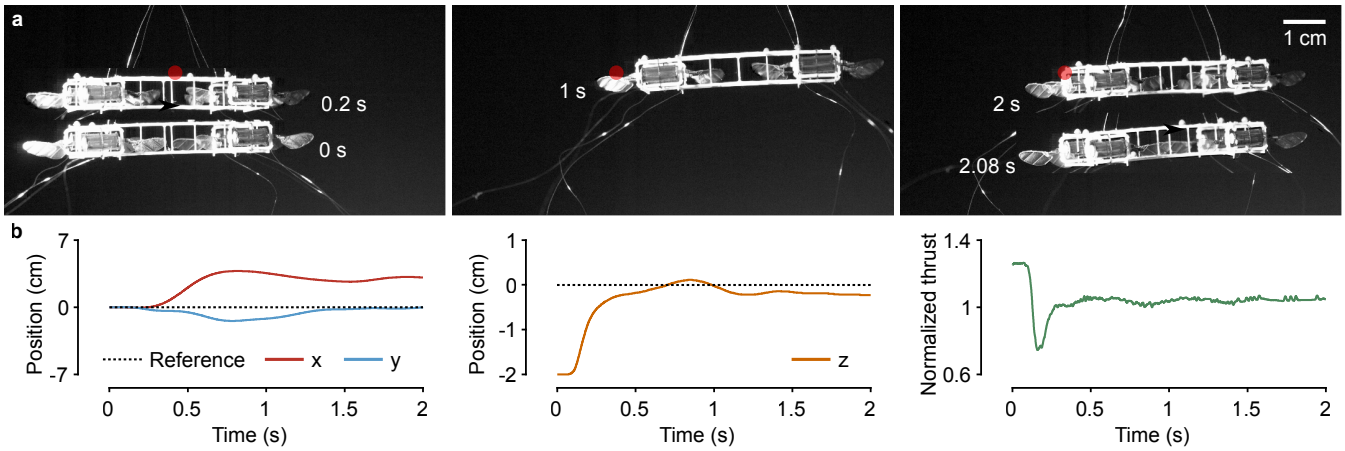

Fig. 6. A successful hovering flight performed by the deep reinforcement learning controller on a 720-mg soft-actuated IMAV. (a) A sequence of composite images illustrating a 2-second hovering flight. (b) Tracked robot lateral position, altitude, and the commanded thrust force.

cloning stage enhances the robustness of the controller to accommodate more model uncertainty.

We also vary the parameter range in our domain randomization (DR) implementation and evaluate its influence on controller performance. Fig. 4b shows that the mean reward remains similar (within 20% change) despite having large changes in parameter range. This result implies that larger parameter variation can lead to higher tolerance to model uncertainty without substantially sacrificing performance. Furthermore, we investigate the learning convergence rate. Fig. 4c shows the controller performance improves as the number of training state-action pairs increases. This result shows the trained policy only requires approximately 20000 state-action pairs to converge, equivalent to 20 seconds of expert flight demonstrations in simulation.

In addition, we investigate the performance of PPO fine-tuning and its influence on smoothing the control policy at the RL stage. Fig. 5a shows the mean reward increases as the number of agent-environment interactions increases. Fig. 5b demonstrates that the fine-tuned RL policy, $\pi_{\theta'}$, is improved through PPO by 62% (median reward). Fig. 5c and 5d compare the commanded thrust before and after PPO fine-tuning. The command fluctuation reduces by 51% without reducing trajectory tracking accuracy, making it more preferable to be deployed on real-world hardware.

B. Experimental Flight Results

To evaluate the effectiveness of state-action re-matching and DR in bridging the *Sim2Real* gap, we deploy the trained policy on two distinct soft-actuated robots. The results demonstrate successful and stable hovering flights on both real-world platforms.

We first conduct a flight test on a 720-mg eight-wing IMAV (Fig. 1, left) to evaluate the reinforcement learning policy after PPO fine-tuning (Fig. 6). The soft-actuated robot achieves a zero-shot stable hovering with small lateral drift, marking the first successful deployment of deep RL control on an insect-scale flapping-wing soft robot (Supplementary Video Part 1). Fig. 6 illustrates the tracked position and commanded thrust, with position errors comparable to other work [4] [5]. The commanded thrust is also reasonably smooth, highlighting the effectiveness of PPO fine-tuning.

We then attempt to fly a four-wing robot (Fig. 1 right), which has an MoI approximately six times smaller than the eight-wing version on the y-axis, making it even more challenging to control. To adapt to this design, we adjust only the robot parameters $\mathcal{R}$ for both BC and PPO, keeping other parameters unchanged for NN controller training. Using the trained policy, we achieve three consecutive 10-second flights (Fig. 7b and Supplementary Video Part 2) with lateral position and altitude RMSEs of 0.97–1.58 cm and 0.10–0.12 cm, respectively (error calculated after a 1-second delay to allow altitude to converge). To further evaluate the policy's reliability, we conduct an extended 50-second flight—longer than any other reported flight at the insect scale [15]. During this flight, the lateral position and altitude RMSEs are 1.34 cm and 0.05 cm, respectively (Fig. 7a,c and Supplementary Video Part 3, error calculated after a 1-second delay).

Compared with other long hovering flights ($>$ 10 s), the position errors of these successful flight attempts on the four-wing soft-actuated robot are smaller than those reported in state-of-the-art IMAV studies [3] [5] [10].

IV. Discussion & Conclusion

In this work, we develop a deep reinforcement learning-based controller for an insect-scale aerial robot. We address the challenges of system delay and model uncertainty by initializing the policy through the state-action re-matching method with domain-randomized demonstrations. Then, we apply PPO in the reinforcement learning stage to improve flight performance and reduce driving command fluctuation. The simulation results show that the proposed techniques for BC can effectively improve the mean reward, and PPO fine-tuning reduces variations of thrust. Most importantly, we deploy this controller on a 720-mg and an 850-mg soft-actuated IMAV and demonstrate a 50-second hovering flight with lateral position and altitude error of 1.34 cm and 0.05 cm, respectively.

Unlike most BC methods that rely on DAgger [13] to enhance neural network robustness after cloning, the proposed modified BC method incorporates disturbances, $\mathcal{E}$, directly

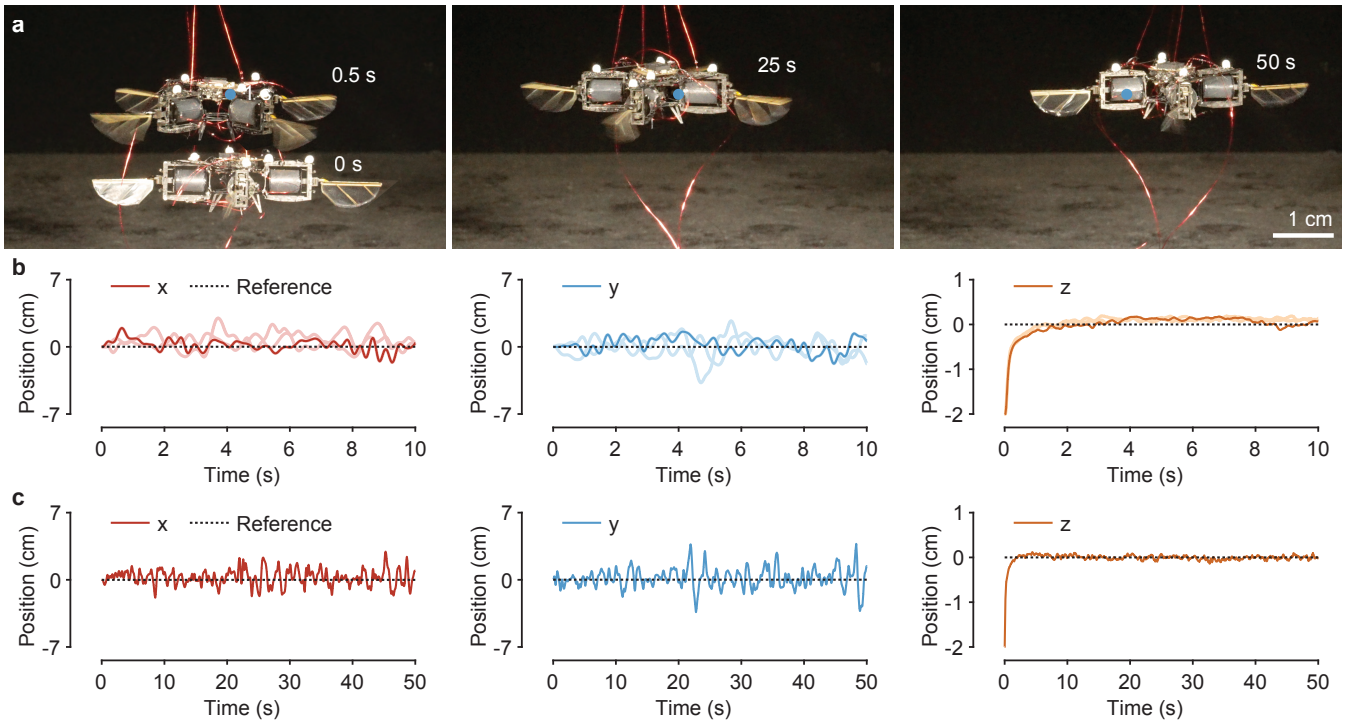

Fig. 7. Successful hovering flights performed by the deep reinforcement learning controller on an 850-mg soft-actuated four-wing IMAV. (a) A sequence of composite images illustrating a 50-second hovering flight. The blue dots in the images indicate the setpoint (origin) of the robot. (b)-(c) Tracked robot lateral position and altitude. (b) Three 10-second hovering flights. The light colors represent repeating flights. (c) The 50-second hovering flight.

during the expert demonstration stage. This early introduction allows us to sample more diverse state-action pairs without DAgger, making trajectory generation computationally efficient and reducing dataset creation time to under a minute. Including supervised learning, the modified BC approach initializes a delay-compensated NN ready for PPO in less than 5 minutes (on an M1 MacBook Air).

Achieving insect-like locomotion on a soft-actuated IMAV requires complex planning and high-rate feedback control, yet we are limited to lightweight onboard microprocessors with constrained computational capacity. Deep reinforcement learning is an ideal solution, as a small (32x32) multi-layer perceptron can be efficiently run on hardware of this size, and the neural network can learn an optimal policy over long time horizons through deep RL [6].

An essential aspect of this work is bridging the *Sim2Real* gap. For the first time, an insect-scale robot achieves stable hovering flight using model-free deep RL, demonstrating that controllers proven effective in the simulator can reliably translate to real-world soft-actuated robots. This milestone not only validates the robustness of our approach but also paves the way for testing more control strategies safely and efficiently within the simulation environment.

While this work focuses on hovering flights, it represents an intermediate step toward achieving insect-like agile maneuverability. By harnessing the potential of unsupervised deep reinforcement learning, the neural network controller can be trained on more complex tasks in simulation—such as wall perching [16], inverted ceiling landings [17], and aggressive trajectory following [6]—and perform these challenging maneuvers on real-world IMAVs in the near future.

References

[1] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, "Controlled flight of a biologically inspired, insect-scale robot," *Science*, vol. 340, no. 6132, pp. 603–607, 2013.

[2] Y. M. Chukewad, J. James, A. Singh, and S. Fuller, "Robofly: An insect-sized robot with simplified fabrication that is capable of flight, ground, and water surface locomotion," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2025–2040, 2021.

[3] R. M. Bena, X. Yang, A. A. Calderón, and N. O. Pérez-Arancibia, "High-performance six-dof flight control of the bee++: An inclined-stroke-plane approach," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1668–1684, 2023.

[4] Y. Chen, H. Zhao, J. Mao, P. Chirarattananon, E. F. Helbling, N.-s. P. Hyun, D. R. Clarke, and R. J. Wood, "Controlled flight of a microrobot powered by soft artificial muscles," *Nature*, vol. 575, no. 7782, pp. 324–329, 2019.

[5] Y. Chen, S. Xu, Z. Ren, and P. Chirarattananon, "Collision resilient insect-scale soft-actuated aerial robots with high agility," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1752–1764, 2021.

[6] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," *Science Robotics*, vol. 8, no. 82, p. eadg1462, 2023.

[7] N. O. Perez-Arancibia, P.-E. J. Duhamel, K. Y. Ma, and R. J. Wood, "Model-free control of a hovering flapping-wing microrobot: The design process of a stabilizing multiple-input–multiple-output controller," *Journal of Intelligent & Robotic Systems*, vol. 77, pp. 95–111, 2015.

[8] A. De, R. McGill, and R. J. Wood, "An efficient, modular controller for flapping flight composing model-based and model-free components," *The International Journal of Robotics Research*, p. 02783649211063225, 2021.

[9] A. Tagliabue, Y.-H. Hsiao, U. Fasel, J. N. Kutz, S. L. Brunton, Y. Chen, and J. P. How, "Robust, high-rate trajectory tracking on insect-scale soft-actuated aerial robots with deep-learned tube mpc," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3383–3389.

[10] S. Kim, Y.-H. Hsiao, Y. Lee, W. Zhu, Z. Ren, F. Niroui, and Y. Chen, "Laser-assisted failure recovery for dielectric elastomer actuators in aerial robots," *Science robotics*, vol. 8, no. 76, p. eadf4278, 2023.

[11] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems*, 2018.

[12] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Single-loop control and trajectory following of a flapping-wing microrobot," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 37–44.

[13] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[14] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *Robotics: Science and Systems*, 2023.

[15] Y.-H. Hsiao, S. Kim, Z. Ren, and Y. Chen, "Heading control of a long-endurance insect-scale aerial robot powered by soft artificial muscles," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3376–3382.

[16] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Perching with a robotic insect using adaptive tracking control and iterative learning control," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1185–1206, 2016.

[17] B. Habas, J. W. Langelaan, and B. Cheng, "Inverted landing in a small aerial robot via deep reinforcement learning for triggering and control of rotational maneuvers," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3368–3375.