Aerial Gym Simulator RA-L - Appendix 1

Mihir Kulkarni, Welf Rehberg and Kostas Alexis

January 2025

1 Additional Details Regarding the Simulation Framework

Simulator Architecture

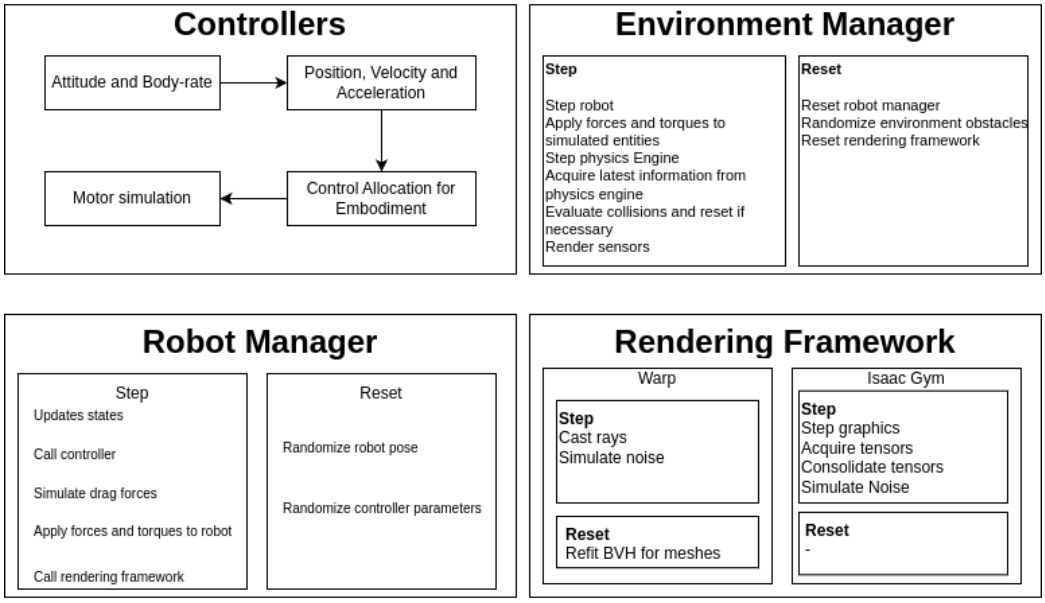

Figure 1: High-level representation of the components of the Aerial Gym Simulator Architecture

1.1 Environment Manager

The environment manager handles the coordination between sub-modules of the Aerial Gym Simulator. It maintains a copy of a robot manager, rending engine manager and environment asset manager, and a physics engine interface. A "Global Tensor Dictionary" is created, maintained and shared across each of these modules. Each module can read from, and write to the "Global Tensor Dictionary" (GTD) tensors related to their operation that are shared across the simulation framework. This data structure is also exposed to the user through the Aerial Gym Simulator API, allowing access to the underlying tensor representations present in the simulator.

The environment manager manages the high-level stepping of the parallel environments and selective resetting of each of them in the case of collisions or truncations of rollouts. The forces commanded by the robots are applied to the appropriate links in the Isaac Gym physics engine. Similarly, tensors from the physics engine after simulation of physics steps are obtained and updated in the GTD. These are then used by individual components.

1.1.1 Environment Randomization

Each environment when reset can be randomized (within the boundaries of the environment) as per the configuration in the config file. Users interested to know this process are encouraged to check the code and modify it to suit their needs.

1.2 Rendering Framework

The Aerial Gym Simulator supports both native Isaac Gym and custom-developed Warp rendering framework. The Warp rendering framework provides an order of magnitude speedup for rendering with lower resolution sensors, while Isaac Gym rendering engine generally maintains the throughput across resolutions for similar environments.

1.3 Robot Managers

The robot managers interface the robots with the controllers and sensors. First, the controllers are used to obtain reference force and thrust commands, subsequently these are converted to embodiment-specific motor commands based on robot geometry. Subsequently, a motor model is simulated to model the dynamic response of the actuators. The computed force and torque from the motor model is then applied to the corresponding links in the physics engine. After each physics simulation, the environment manager - via the robot manager, performs a rendering operation for the onboard sensors. The pose of the robot and the relative sensor placement on the robot is used to perform ray-casting for each sensor.

1.4 Controllers

Support for position, velocity, acceleration controllers is offered out-of the box. This is supported by implementation for attitude, angular rate controllers. The controllers provide reference thrust and torque commands for the robot.

1.4.1 Control Allocation

Control allocation is performed to track the above reference commands. Pseudo-inverse of the control-effectiveness matrix on the desired wrench provides the desired motor outputs. These are unconstrained and are subsequently clamped and commanded to the motors.

1.4.2 Motor Model

The response of the motor is modeled as a first-order system. The force and torque applied by the motor and propeller is proportional to the square of angular rate and is calculated based on the current angular rate of the motor.

(a) Ground-truth depth image (b) Simulated Gaussian noise and pixel dropouts.

Figure 2: Comparison of ground-truth depth image in simulation against a noisy depth image with Gaussian noise on depth (std. dev. linearly dependent on distance) and pixel dropouts.

1.5 Sensor Noise

The Aerial Gym Simulator offers a custom-developed NVIDIA Warp-based rendering framework. Users can simulate custom noise models that suit their sensor. We implement two noise models that can be applied simultaneously. First, we can add measurement noise using a Gaussian distribution with zero mean and the standard deviation linearly dependent on the measured (ground-truth) distance. This noise is added in place on the image tensors. Secondly, pixel dropouts can be performed by sampling from a Bernoulli distribution over the number of pixels. While the latter does not exactly match the real-world noise depth images, we provide this model to have robust policies that are exposed to (in training) and can deal with invalid pixels in the real-world depth images when deployed on real-world platforms. Figure 2 shows an image with ground truth measurement alongside a noisy image obtained from the simulator.

Noise models for LiDAR sensors are obtained using the corresponding sensor datasheets from the manufacturers. The quadratic dependence of distance to the standard deviation of measurement noise is obtained by fitting to the curves for noise as presented in the sensor datasheet. This relation is implemented for LiDAR sensors alongside an offset for the mean value of measurement noise.

1.6 Rendering Benchmarking

Table 1: Comparison of the time required to perform rendering operations.

Operation	*Time (s)*
Isaac Gym	
Randomizing Pose	0.2010
Stepping Graphics	0.0248
Rendering Image	0.5803
Acquiring Tensor	0.7712
Aerial Gym (Warp)	
Randomizing Pose	0.00029
Rendering Image	0.02027

Aerial Gym Simulator features a custom rendering framework that performs in-place operations on the tensors and provides a speedup as compared to the loop-based sensor API for native Isaac Gym rendering. Time required for various operations is measured and tabulated for a case of 2048 parallel environments with robots, each with a 64×64 resolution camera sensor. The Warp-based rendering framework performs in-place operations on tensors and therefore does not need to spend

time for commanding each sensor individually or acquiring the image data from each individual sensor in order to aggregate it into a consolidated tensor for training learning-based methods on them. Table 1 details the time in seconds required for each operation for the different frameworks used.

2 Feature Comparison Against Existing Simulation Frameworks

To provide an overview regarding of the simulation capabilities offered and their comparison against existing simulation frameworks, compare several openly available simulators discussed in [1]. We consider simulators that offer simulation of multirotor platforms and evaluate features provided alongside the simulation framework. Table 2 details the physics capabilities of the simulators and the features that support rapid development of solutions for multirotors. Features such as out-of-the box controllers, support for non-planar configurations, simulation of IMU sensors, and support for soft and reconfigurable joints are compared.

Table 3 compares rendering capabilities of these simulators, evaluating whether the software supports GPU-parallelization for rendering exteroceptive sensor data. Supported sensors and sensing data-types are compared to provide an overview of the offered capabilities of each of these simulators.

Table 2: Physics Simulation Feature Comparison

Physics Simulation							
Simulator	**Parallellized**	**Controllers**	**Non-planar**	**Motor Simulation**	**IMU**	**Soft-Joints**	**Reconfigurable**
RotorPY [2]	✗	✓	✗	✓	✓	✗	✗
PyFlyt [3]	✗	✓	✗	✓	✗	✗	✗
ARCAD [4]	✗	✓	✓	✓	✗	✗	✗
gym-pybullet-drones [5]	✗	✓	✗	✓	✗	✗	✗
QuadSwarm [6]	✓	✓	✗	✓	✗	✗	✗
L2F [7]	✓	✗	✓	✓	✗	✗	✗
FlightMare [8]	✓	✓	✗	✓	✗	✗	✗
FlightGoggles [9]	✗	✓	✗	✓	✓	✗	✗
Pegasus [10]	✗	✓	✗	✓	✓	✗	✗
OmniDrones [11]	✓	✓	✓	✓	✗	✗	✓
Aerial Gym Simulator	✓	✓	✓	✓	✓	✓	✓

Table 3: Rendering Feature Comparison

Rendering							
Simulator	**Parallellized**	**RGB-D**	**LiDAR**	**Stereo Camera**	**Segmentation**	**Face ID**	**Surface Normals**
RotorPY	✗	✗	✗	✗	✗	✗	✗
PyFlyt	✗	✓	✗	✗	✓	✗	✗
ARCAD	✗	✓/?	✗	✗	✗	✗	✗
gym-pybullet-drones	✗	✓	✗	✗	✓	✗	✗
QuadSwarm	✗	✗	✗	✗	✗	✗	✗
L2F	✗	✗	✗	✗	✗	✗	✗
FlightMare	✗	✓	✗	✗	✓	✗	✗
FlightGoggles	✗	✓	✓	✗	✓	✗	✓
Pegasus	✓	✓	✓	✗	✓	✗	✗
OmniDrones	✓	✓	✓	✗	✓	✗	✓
Aerial Gym Simulator	✓	✓	✓	✓	✓	✓	✓

3 Modeling and Training of Soft System in Simulation

Recent works investigate actively and passively morphing multirotors with flexible arms that change the robot morphology in flight [12, 13, 14]. In this section, the capabilities of The Aerial Gym Simulator to simulate compliant configurations are shown by modeling an existing soft system and training a position control policy for it. The Aerial Gym Simulator supports simulation of soft

airframes with both active and passive rotational joints. Each joint can be interfaced with a PD-controller provided by Isaac Gym, or with a custom implementation matching the actuator or the compliant-response properties of real-world robots.

3.1 Modeling of a Specific Soft System (Morphy)

Morphy [15], a quadrotor platform with compliant arm joints is modeled and simulated. Each arm on the real platform consists of a Hall-effect sensor embedded in silicon allowing sensing of the angular deflection of the arm. We simulate the robot as a rigid base frame with four arms attached to it with a 2-DoF passive joint. The joint dynamics are identified and modeled as a nonlinear spring alongside a linear damper system. The response is identified as $\ddot{\theta}_j = -K_p|(\theta_j - \epsilon_{\theta_j})|(\theta_j - \epsilon_{\theta_j}) - K_v\dot{\theta}_j$ and implemented in the simulator to apply moment $\tau_j = I_j\ddot{\theta}_j$ for arm j with moment of inertia I_j, the elevation deflection angle θ_j, and joint velocity $\dot{\theta}_j$ with a steady-state position $\epsilon_{\theta_j} = 0.125$ rad. We assume a similar response for the azimuth angles ϕ_j with a steady state position of $\epsilon_{\phi_j} = 0$ rad. The response of the arm on the real platform, the model and the response in simulation is shown in 3. Additionally, simulated joints can also be configured to be active and exert torque, or track velocity or position setpoints allowing simulation of reconfigurable systems and foldable drone platforms. To accelerate relevant research efforts, we also provide a model for a multi-linked flying platform inspired by the work in [16].

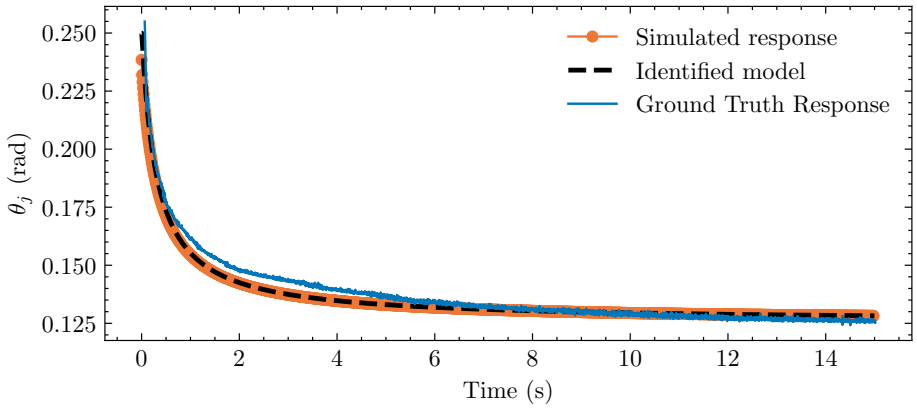

Figure 3: Comparison of the simulated response of *Morphy* against the identified model and the ground-truth response.

3.2 Simulation and Control for Soft Airframes

We investigate the performance of motor-control DRL policies for position-setpoint tracking trained for a rigid robot against one that accounts for joint dynamics to minimize oscillations and evaluate it on the simulated Morphy platform. The state vector is defined as $\mathbf{s}_t = [\mathbf{e}_t, \mathbf{R}_t, \mathbf{v}_t, \boldsymbol{\Omega}_t, \mathbf{a}_{t-1}, \boldsymbol{\theta}_t, \boldsymbol{\phi}_t, \dot{\boldsymbol{\theta}}_t, \dot{\boldsymbol{\phi}}_t]$, where $\boldsymbol{\theta}_t$ and $\boldsymbol{\phi}_t$ are the elevation and azimuth deflection angles of the robot arms respectively, while $\dot{\boldsymbol{\theta}}_t$ and $\dot{\boldsymbol{\phi}}_t$ correspond to the joint velocities. The state is fed into the network as observations. The commanded actions are motor force setpoints defined as $\mathbf{a}_t = \mathbf{U}_t^{\text{ref}}$. The physics timestep duration is set to $\Delta t = 0.002$ s for solver stability, and the controller setpoints are commanded at 100 Hz. RL Games [17] is used to train the policy in simulation with 4096 parallel environments.

3.2.1 Reward Design

The used reward is based on the following quantities:

$$\mathbf{h_e} = [e_x, e_z, e_z], \tag{1}$$
$$\mathbf{h_\Omega} = [\Omega_x, \Omega_y, \Omega_z], \tag{2}$$
$$\mathbf{h}_{joint} = [\dot{\theta}_1, ..., \dot{\theta}_4, \dot{\phi}_1, ..., \dot{\theta}_4], \tag{3}$$
$$\mathbf{h}_{\mathbf{a}_{diff}} = [a_{1,t} - a_{1,t-1}, ..., a_{4,t} - a_{4,t-1}], \tag{4}$$
$$\mathbf{h}_{\mathbf{a}_{abs}} = [a_1 - \hat{a}_1, ..., a_4 - \hat{a_4}], \tag{5}$$
$$h_{up} = 1 - (\mathbf{q}\mathbf{e}_3\mathbf{q}^{-1})_z. \tag{6}$$

The quantities $\mathbf{h_e}$, $\mathbf{h_\Omega}$ and $\mathbf{h}_{joint}$ are terms based on the position offset from the provided reference, angular rates and joint velocities. The term $\mathbf{h}_{\mathbf{a}_{abs}}$ is based on the difference of each commanded motor thrust to the thrust necessary for hovering ($\hat{\mathbf{a}}$). To incentivize smooth actions we add $\mathbf{h}_{\mathbf{a}_{diff}}$ rewarding small changes in the commanded thrust between two time steps. The quantity h_{up} is based on the alignment of the z-axis of the body-fixed frame and the z-axis of the world frame. This is calculated by rotating the corresponding unit vector $\mathbf{e}_3$ by the quaternion representing the orientation of the system $\mathbf{q}$ and evaluating its alignment where $h_{up} = 0$ for perfect alignment.

The in equation 1 - 6 described quantities $\mathbf{h}$ are used to calculate the individual parts r_h of the reward function using exponential kernels as follows:

$$r_h = \sum_i a e^{-b\mathbf{h}_i^2}, \tag{7}$$

where the coefficients a and b of the exponential kernel are adequately chosen weights defining its magnitude and width. The obtained individual reward terms are combined in the following reward:

$$r = r_\mathbf{e}(r_{up} + r_\mathbf{\Omega} + r_{\mathbf{a}_{diff}}) + r_\mathbf{e} + r_{\mathbf{a}_{diff}} + r_{\mathbf{a}_{abs}} + r_{joint} + p_{dist}. \tag{8}$$

Multiplying r_{up}, $r_{\mathbf{a}_{diff}}$ and $r_\mathbf{\Omega}$ with $r_\mathbf{e}$ leads to a weighted reward allowing for high angular rates, actions and orientation offsets far away from the target location. The term p_{dist} is added to incentivize straight convergence to the target location and is calculated using a potential-based shaping function calculated from the distance to the target [18].

3.2.2 Simulation Experiments with Flexible Airframe

The joint-aware policy is able to minimize the joint oscillations during flight and demonstrate a stable tracking performance compared to the policy trained for a similar platform with rigid joints as shown in 4. Only deflections and angular rates for the joint elevation are shown.

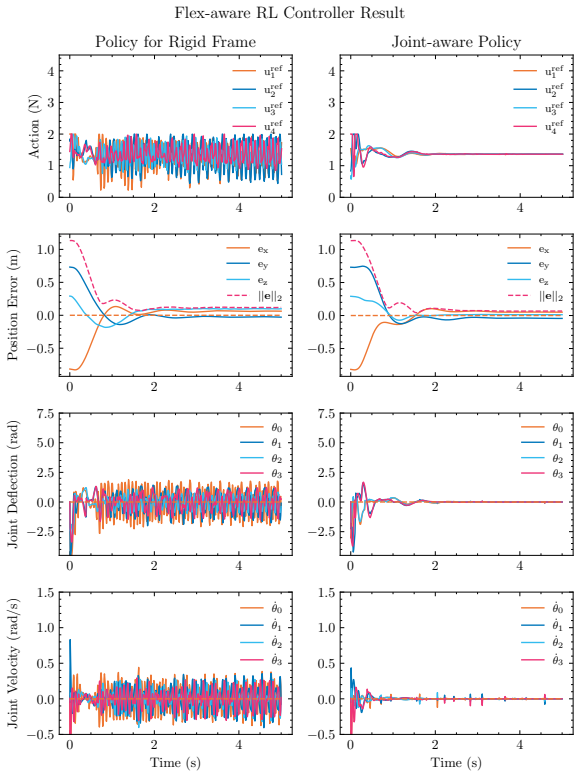

Figure 4: Simulation comparison between policies assuming rigid airframe (left) and a joint-aware policy (right) for *Morphy* [15]. The joint-aware policy tracks the target position and reduces oscillations in the flexible joint.

4 Further Details on RPM Experiment

4.1 System Description

This section describes the systems used for the motor-control experiment. For deploying motor level control policies, we use a planar quadrotor platform equipped with the ModalAI Voxl 2 Mini board and ModalAI Voxl ESC. Note that the ModalAI ESC comes with closed-loop RPM control. The electronics are mounted on a carbon sandwich frame. The used system is shown in figure 5. The motors used are iFlight XING2 1404 3000KV with HQProp Duct 75mm 6/5 Paddle 3-inch propellers. To estimate the robot's pose, a Qualisys Motion Capture system is used for all experiments.

4.2 System Identification

Estimating the physical parameters of the used system is a crucial part of deploying reinforcement learning policies in the real world. The system parameters with the highest impact on the simulation-to-reality fidelity are in our (with the literature agreeing [7, 19, 20]) opinion:

- system mass: m
- system inertia: $\mathbf{J}$
- thrust coefficient of the motor-propeller combination: c_t

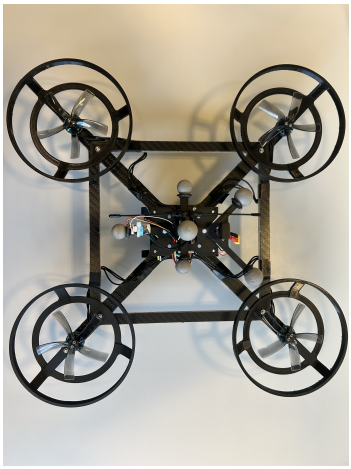

Figure 5: System used for RPM control policies.

- thrust-to-torque ratio of the motor-propeller combination: c_{t2t}
- time constant of the motor-propeller combination: τ_m

Identifying the system mass was done by simply weighing the system. The inertia of the system was identified using the system CAD models with the appropriate density settings for each component. The density of each component is assumed constant over the volume and was obtained by weighing each component and dividing the weight by the volume obtained from the CAD model. The time constants of the used motor-propeller combinations are estimated using data from RPM step responses, which were obtained by recording step responses for different RPM regions using the ESC calibration tool provided by ModalAI [1]. In the simulator the mean of the individual time constants over the full RPM range is used. The thrust coefficients for the motor-propeller combination were estimated from the hover state of the real system by relating RPM in the steady state ω_m with the theoretically necessary individual motor forces f_m to compensate for gravity. The following relation applies:

$$c_t = \frac{f_m}{\omega_m^2} \tag{9}$$

The estimation of the thrust-to-torque ratio is based on information from similar motor-propeller combinations and was not done specifically for this system. A summary of the identified parameters can be found in table 4.

We highlight that a better system identification will lead to improvements in the performance of the presented policies. Future users of the provided simulator are encouraged to use more advanced system identification tools [2] or measure the force and torque responses of their specific motor-propeller combination. For further convenience, we have provided scripts (`https://github.com/ntnu-arl/aerial_gym_simulator/tree/main/aerial_gym/sim2real`) to calculate the related parameters (c_t, c_{t2t}, τ_m) from RPM, force and torque curves. The provided scripts require data in a

[1] `https://gitlab.com/voxl-public/voxl-sdk/utilities/voxl-esc/-/tree/master/voxl-esc-tools?ref_type=heads`

[2] like `https://sysid.tools/`

Parameter	Value
m $[kg]$	0.373
$\mathbf{J}$ $[kgm^2]$	$\begin{bmatrix} 0.0015914 & -0.0000044 & 0.0000001 \\ -0.0000044 & 0.0015312 & 0.0000031 \\ 0.0000001 & 0.0000031 & 0.0025329 \end{bmatrix}$
c_t $[kgm]$	0.00001286412
c_{t2t} $[m]$	0.01
τ_m $[s]$	0.047

Table 4: Identified system parameters

CSV format from the user. Steady-state values provided by a force-torque bench, alongside the motor RPM can be used to calculate the force and torque constants of the motors. To calculate the time constant of the motors, the users need to provide timestamped RPM values for a step-response of a motor.

4.3 Policy Deployment with PX4

The ModalAI board runs a PX4 flight controller. Since PX4 does not offer native support for low-level neural control policies a custom module taking the system state as input and directly generating motor commands was implemented. The network in PX4 was rebuilt by loading the layer weights and biases from the trained Torch models in Eigen matrices and vectors. The neural controller runs at a frequency of 250 Hz. and outputs motor thrust commands from which RPM setpoints are calculated. Note that the network was trained on the simulator with a timestep duration of 0.01 s (effectively 100 Hz). However, the stable transfer of the policy at higher frequencies shows the robustness of the trained policies.

4.4 Learning Setup

The observations given to the network are defined as $\mathbf{o}_t = [\mathbf{e}_t^{\mathcal{W}}, \mathbf{R}_6, \mathbf{R}\mathbf{v}_t, \mathbf{\Omega}_t]$, where $\mathbf{e}_t^{\mathcal{W}}$ denotes $\mathbf{e}_t$ expressed in the world frame, while $\mathbf{R}_6$ denotes the 6-D rotation representation based on [21]. The network commands motor thrust setpoints $\mathbf{a}_t = [\mathbf{U}_t^{\text{ref}}]$. These are converted to RPM setpoints using identical thrust constants and commanded to the motors.

4.4.1 RL-Framework and PPO Parameter

We train the policies for the RPM control experiments using the Proximal Policy Optimization (PPO) implementation of Sample Factory [22]. The corresponding PPO parameter can be found in table 5. During training, we use no domain randomization but simulate sensor noise drawn from a Gaussian distribution. The chosen noise parameters are shown in figure 6. The training was done on a workstation with an NVIDIA GeForce RTX 3090 GPU using 4096 parallel environments.

4.4.2 Reward Function

As reward function, we use again a sum of exponential kernels (established in section 3.2.1) defined as follows:

$$r = r_{\mathbf{e}}(r_{forw} + r_{\mathbf{v}} + r_{\mathbf{\Omega}} + r_{\mathbf{a}_{diff}}) + r_{\mathbf{e}} + r_{\mathbf{v}} + r_{\mathbf{\Omega}} + r_{up} + r_{\mathbf{a}_{abs}} + p_{dist}, \tag{10}$$

Parameter	Value
Layer Size	[32, 24]
Activation	tanh
Gamma	0.99
Learning Rate	3e-4
Exploration Loss Coeff.	1e-2
KL Loss Coeff.	0.1
# Epochs	5
Batch Size	65536
# Batches per Epoch	4

Table 5: PPO parameters

State	Std
Position $[m]$	0.001
Orientation $[rad]$	0.003
Linear Velocity $[m/s]$	0.002
Angular Velocity $[rad/s]$	0.001

Table 6: Simulated sensor noise. The noise is drawn from a Gaussian distribution with zero mean and shown standard deviation.)

where the new reward terms $r_{\mathbf{v}}$ and r_{forw} are based on the linear velocity of the system and the alignment of the z-axis of the body-fixed frame and the z-axis of the world frame. These terms are calculated in a similar fashion as described in section 3.2.1.

4.5 Individual Experiment Results from Motor Control

In the following, the individual results of the experiments for RPM control with policies trained for multiple seeds are provided. Note that there is a significant offset in the commanded thrust and therefore the actual RPM for each motor in stable hover. We assume this is due to the center of mass not being at the geometric center of the system or the motor-propeller combinations having different thrust coefficients. Either way, the policy is able to overcome these unmodeled disturbances stabilizing the system with low tracking error. Modeling the system's thrust offset in hover correctly offers further potential to improve the shown performance.

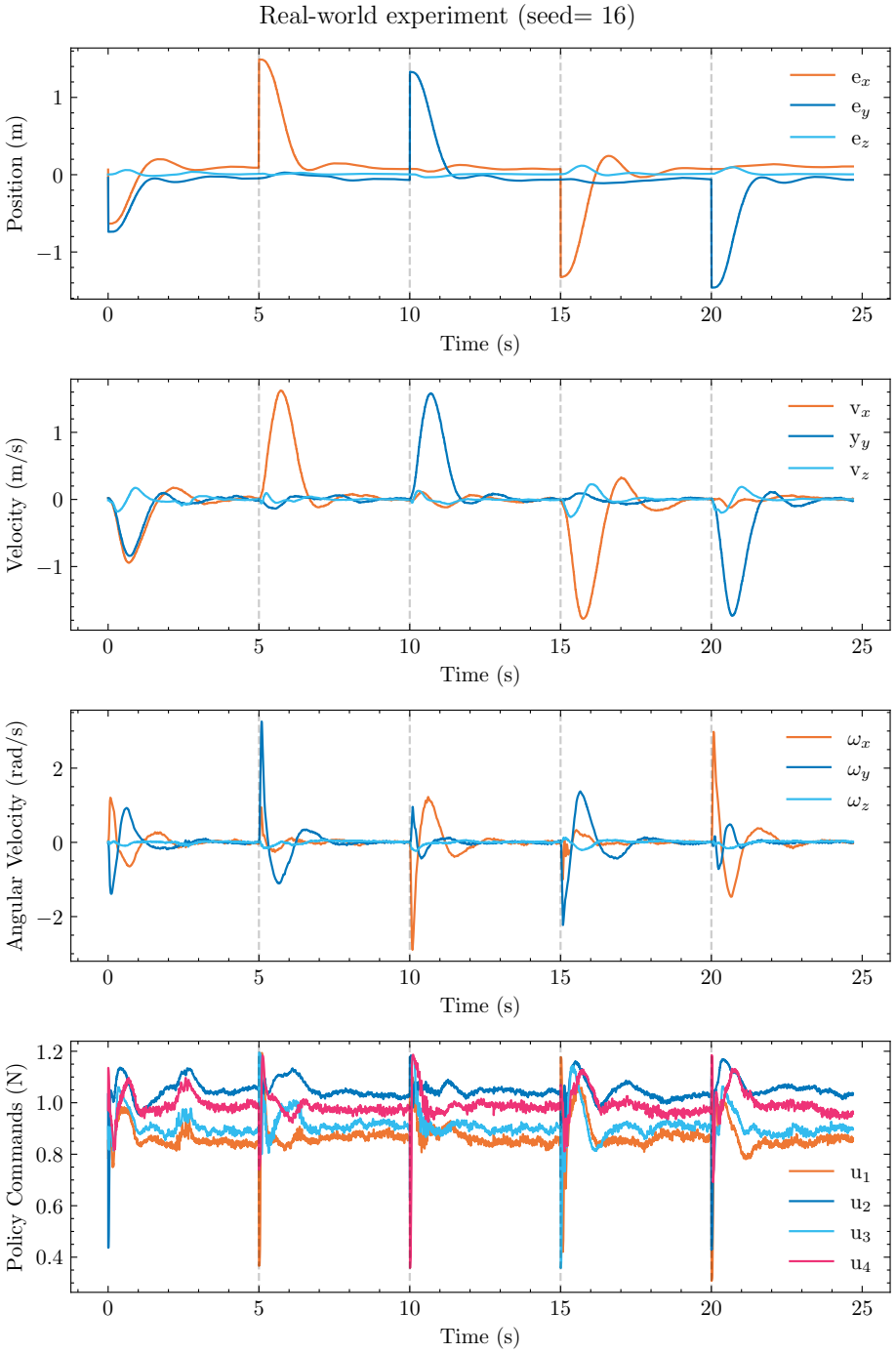

Figure 6: Experiment RPM control policy trained with seed 16.

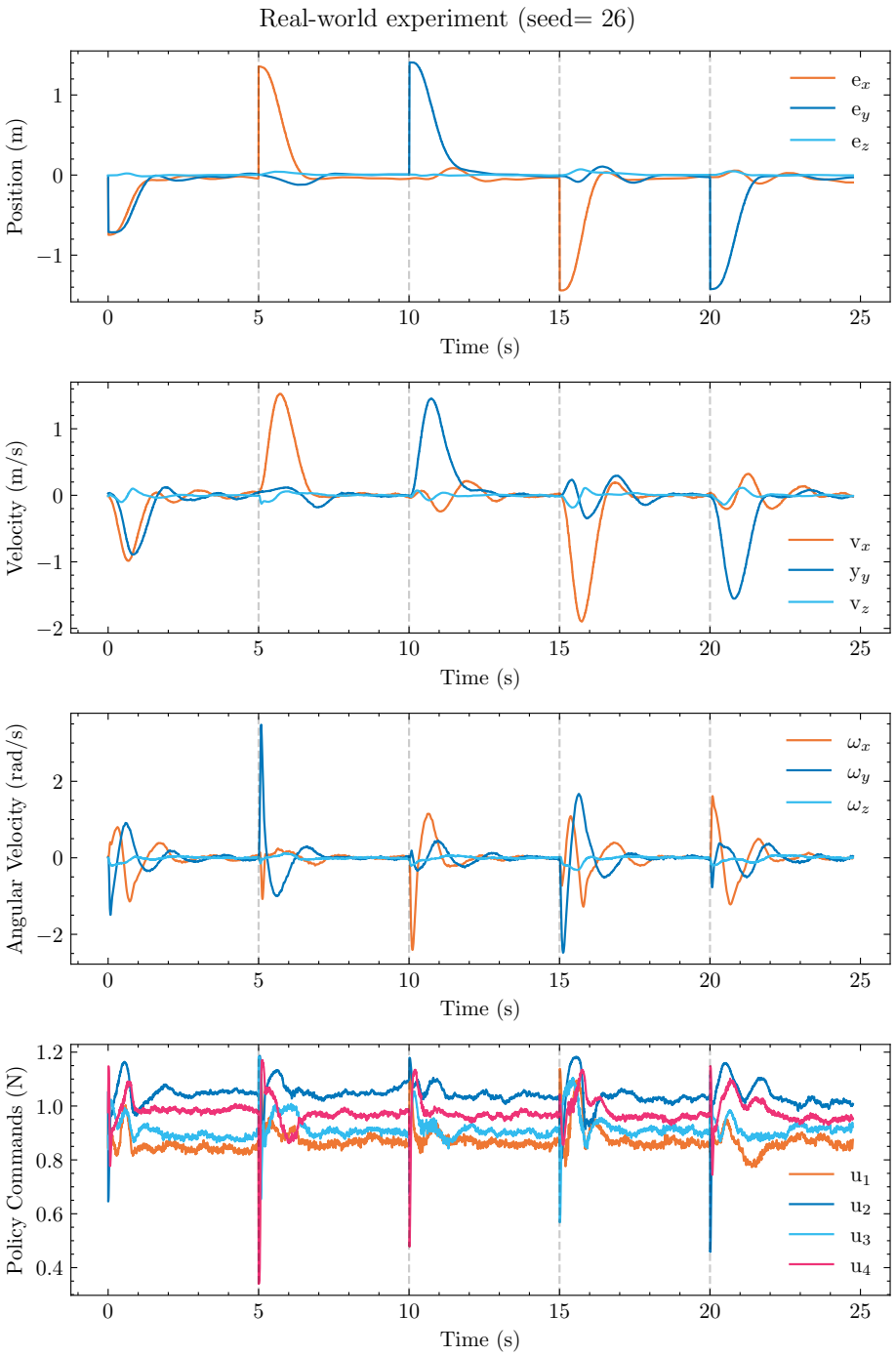

Figure 7: Experiment RPM control policy trained with seed 26.

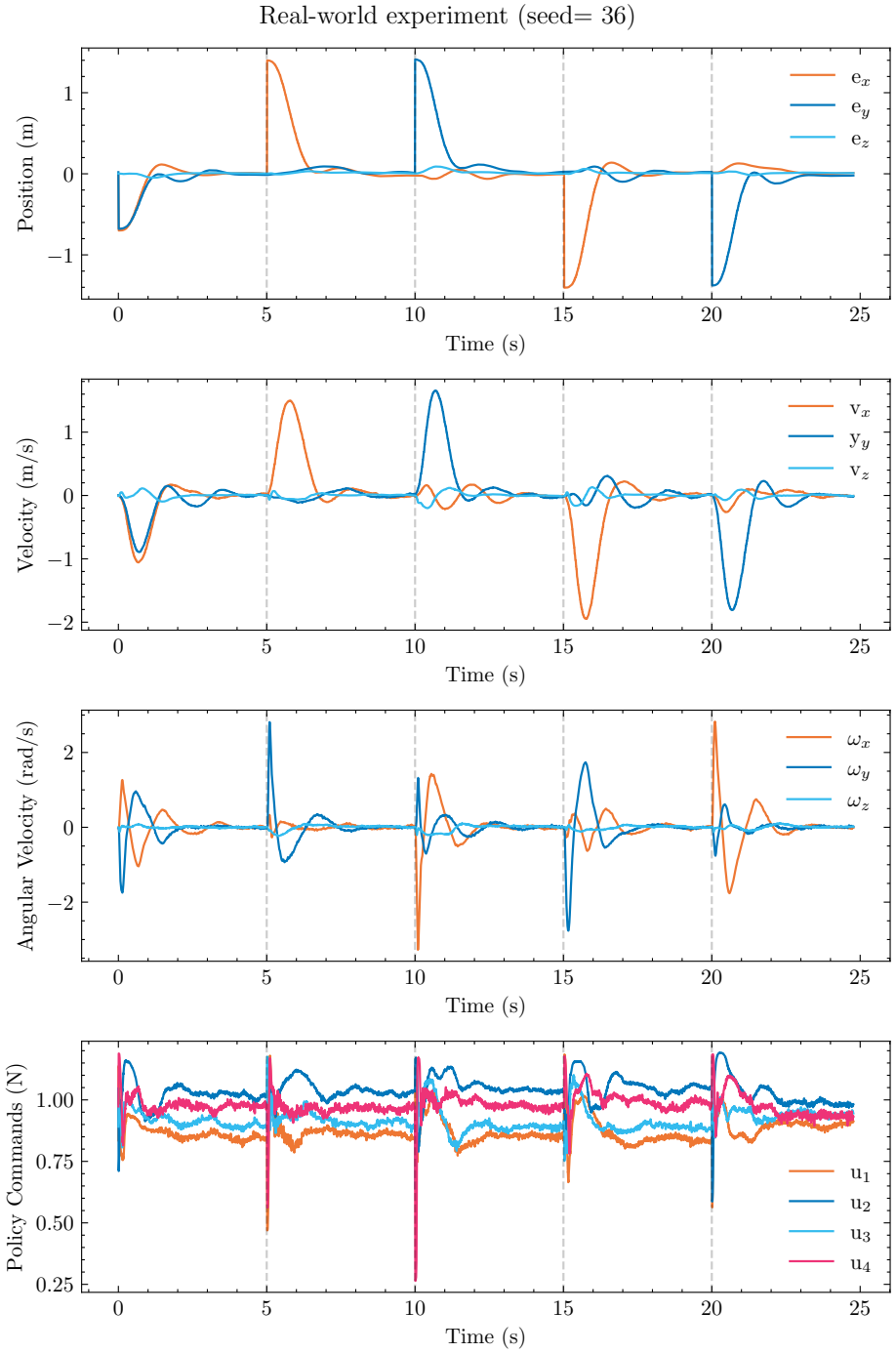

Figure 8: Experiment RPM control policy trained with seed 36.

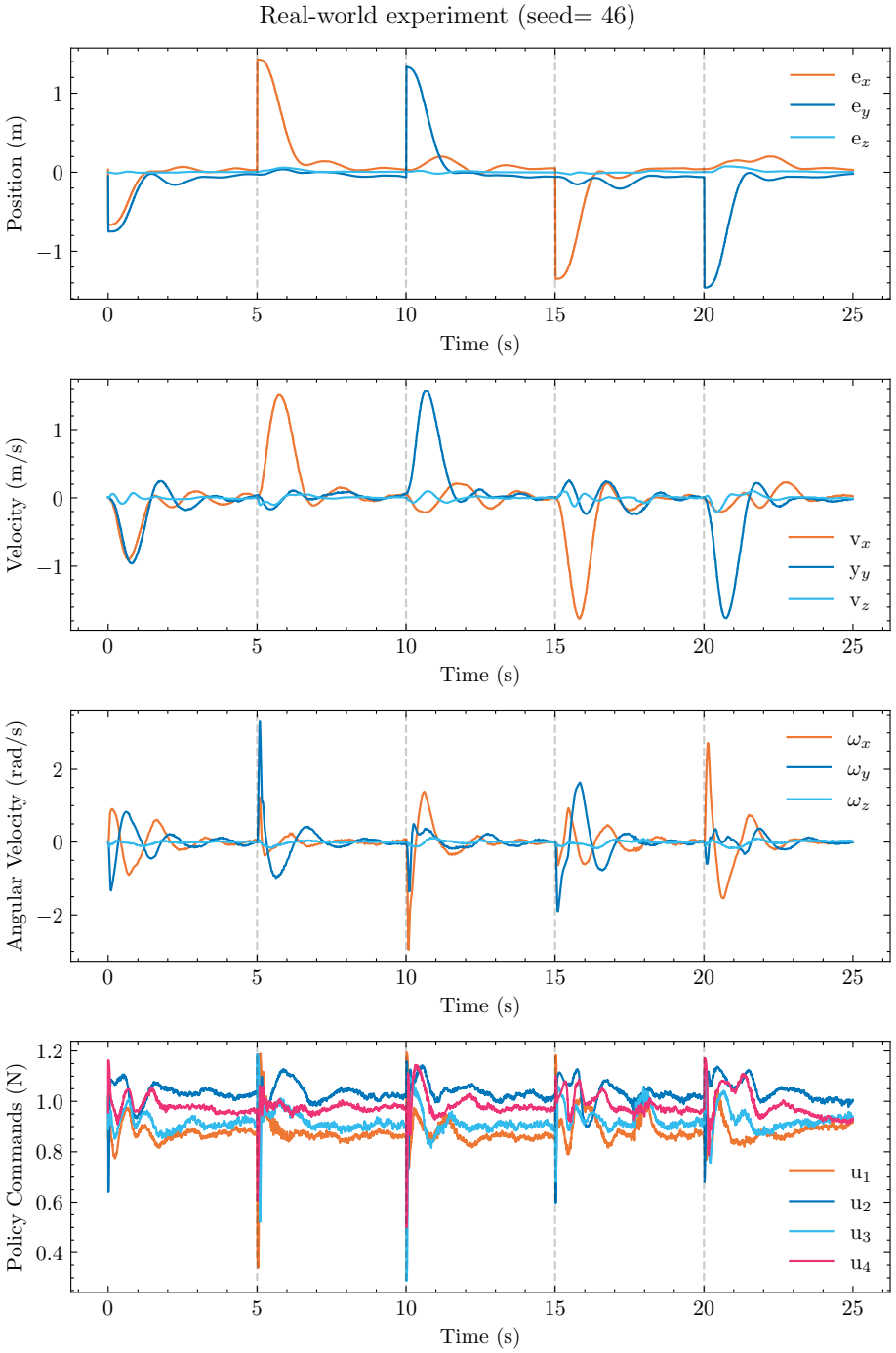

Figure 9: Experiment RPM control policy trained with seed 46.

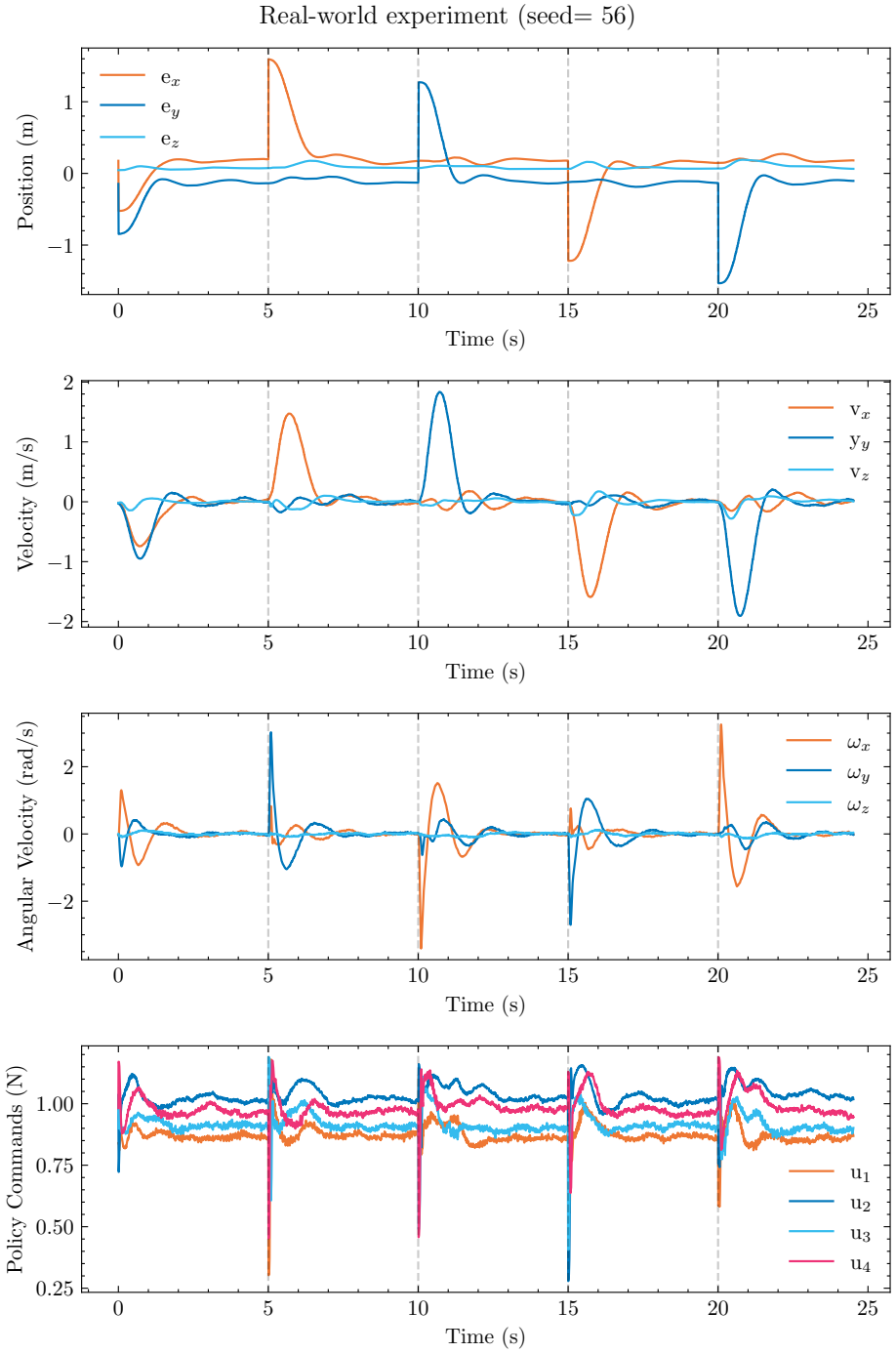

Figure 10: Experiment RPM control policy trained with seed 56.

References

[1] C. A. Dimmig, G. Silano, K. McGuire, C. Gabellieri, W. Hönig, J. Moore, and M. Kobilarov, "Survey of simulators for aerial robots: An overview and in-depth systematic comparisons," *IEEE Robotics & Automation Magazine*, 2024.

[2] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *4th Conference on Robot Learning (CoRL)*, 2020, pp. 1147–1157.

[3] J. J. Tai, J. Wong, M. Innocente, N. Horri, J. Brusey, and S. K. Phang, "Pyflyt – uav simulation environments for reinforcement learning research," 2023. [Online]. Available: https://arxiv.org/abs/2304.01305

[4] A. Keipour, M. Mousaei, J. Geng, D. Bai, and S. Scherer, "Uas simulator for modeling, analysis and control in free flight and physical interaction," in *AIAA SCITECH 2023 Forum*. American Institute of Aeronautics and Astronautics, Jan. 2023. [Online]. Available: http://dx.doi.org/10.2514/6.2023-1279

[5] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly – a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," 2021. [Online]. Available: https://arxiv.org/abs/2103.02142

[6] Z. Huang, S. Batra, T. Chen, R. Krupani, T. Kumar, A. Molchanov, A. Petrenko, J. A. Preiss, Z. Yang, and G. S. Sukhatme, "Quadswarm: A modular multi-quadrotor simulator for deep reinforcement learning with direct thrust control," 2023. [Online]. Available: https://arxiv.org/abs/2306.09537

[7] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," apr 2024. [Online]. Available: http://arxiv.org/abs/2311.13081

[8] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," 2021. [Online]. Available: https://arxiv.org/abs/2009.00563

[9] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," 2019.

[10] M. Jacinto, J. Pinto, J. Patrikar, J. Keller, R. Cunha, S. Scherer, and A. Pascoal, "Pegasus simulator: An isaac sim framework for multiple aerial vehicles simulation," in *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2024, pp. 917–922.

[11] B. Xu, F. Gao, C. Yu, R. Zhang, Y. Wu, and Y. Wang, "Omnidrones: An efficient and flexible platform for reinforcement learning in drone control," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2838–2844, 2024.

[12] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza, "The foldable drone: A morphing quadrotor that can squeeze and fly," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 209–216, 2019.

[13] M. Zhao *et al.*, "Design, modeling, and control of an aerial robot dragon: A dual-rotor-embedded multilink robot with the ability of multi-degree-of-freedom aerial transformation," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1176–1183, 2018.

[14] N. Bucki and M. W. Mueller, "Design and control of a passively morphing quadcopter," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9116–9122.

[15] P. De Petris, M. Nissov, and K. Alexis, "Morphy: a compliant and morphologically-aware flying robot," *Advanced Intelligent Systems*, 2024, to appear.

[16] M. Zhao, K. Kawasaki, X. Chen, S. Noda, K. Okada, and M. Inaba, "Whole-body aerial manipulation by transformable multirotor with two-dimensional multilinks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5175–5182.

[17] D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning," https://github.com/Denys88/rl_games, May 2021.

[18] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, p. 278–287.

[19] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, nov 2019, pp. 59–66. [Online]. Available: https://ieeexplore.ieee.org/document/8967695/?arnumber=8967695

[20] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, may 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10160836/?arnumber=10160836

[21] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5745–5753.

[22] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun, "Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning," 2020.