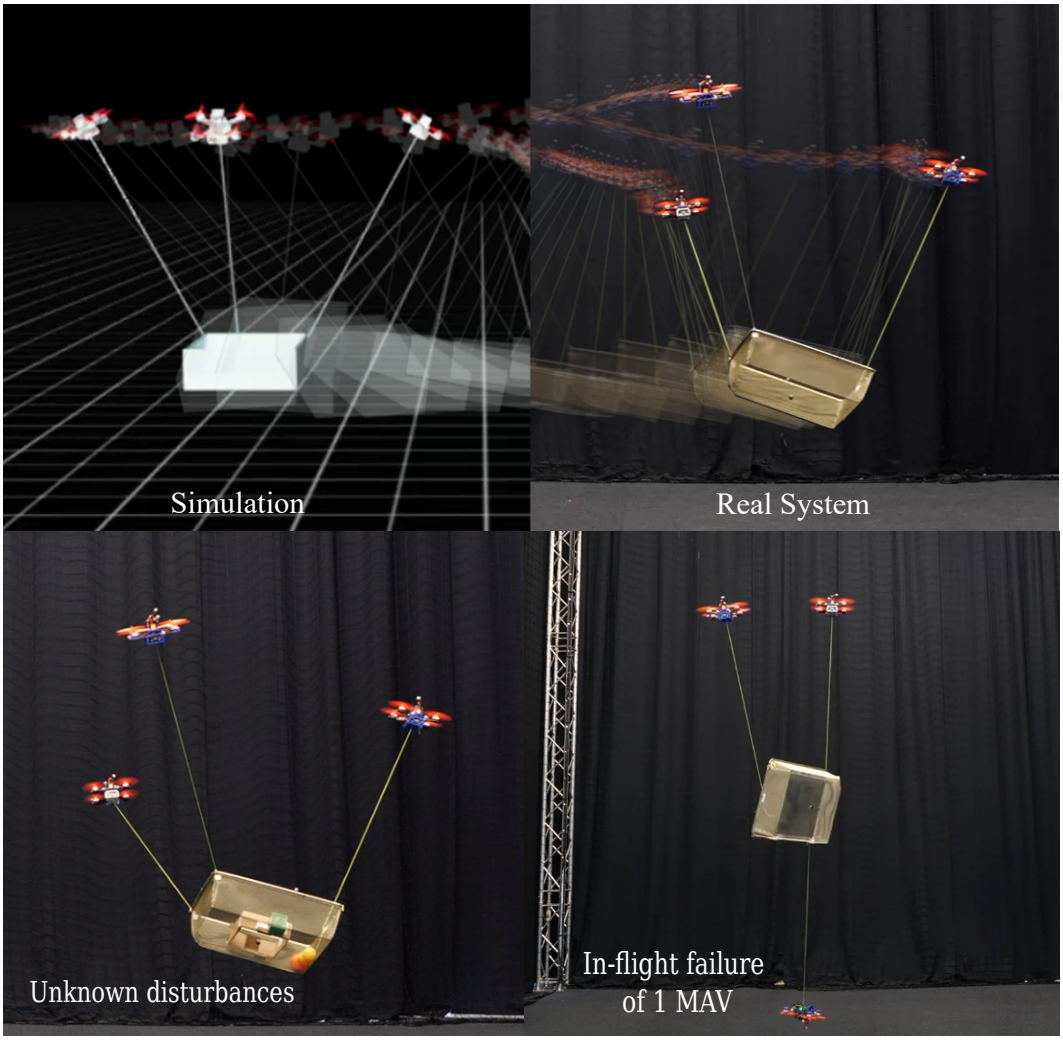
Decentralized Aerial Manipulation of a Cable-Suspended Load using Multi-Agent Reinforcement Learning

Jack Zeng
Master's Thesis Robotics



Decentralized Aerial Manipulation of a Cable-Suspended Load using Multi-Agent Reinforcement Learning

Jack Zeng
Master's Thesis Robotics

In partial fulfillment of the requirements
for the degree of Master of Science in Robotics
at Delft University of Technology.

To be defended on July 17, 2025, at 09:30

Supervisors: Dr. Javier Alonso-Mora
Dr. Sihao Sun
Andreu Matoses Gimenez
Dr. Eugene Vinitsky

Author: Jack Zeng
Student number: 5148987
Faculty: Mechanical Engineering
Research group: Learning and Autonomous Control, Autonomous multi-robots lab
Date: July 11, 2025

Decentralized Aerial Manipulation of a Cable-Suspended Load using Multi-Agent Reinforcement Learning

Jack Zeng
Department of Cognitive Robotics
Delft University of Technology, The Netherlands

Abstract:
This paper presents the first decentralized method to enable real-world 6-DoF manipulation of a cable-suspended load using a team of Micro-Aerial Vehicles (MAVs). Our method leverages multi-agent reinforcement learning (MARL) to train an outer-loop control policy for each MAV. Unlike state-of-the-art controllers that utilize a centralized scheme, our policy does not require global states, inter-MAV communications, nor neighboring MAV information. Instead, agents communicate implicitly through load pose observations alone, which enables high scalability and flexibility. It also significantly reduces computing costs during inference time, enabling onboard deployment of the policy. In addition, we introduce a new action space design for the MAVs using linear acceleration and body rates. This choice, combined with a robust low-level controller, enables reliable sim-to-real transfer despite significant uncertainties caused by cable tension during dynamic 3D motion. We validate our method in various real-world experiments, including full-pose control under load model uncertainties, showing setpoint tracking performance comparable to the state-of-the-art centralized method. We also demonstrate cooperation amongst agents with heterogeneous control policies, and robustness to the complete in-flight loss of one MAV. Videos of experiments: https://autonomousrobots.nl/paper_websites/aerial-manipulation-marl

Keywords: Aerial Manipulation, Multi-Agent Reinforcement Learning, Micro Aerial Vehicles

1 Introduction

Autonomous Micro Aerial Vehicles (MAVs) excel in agility, speed, and mobility, offering significant capabilities for transporting loads to hazardous or remote locations. This makes them well-suited to assist humans with tasks such as construction [1], delivery [2], and inspection [3]. Beyond transportation, UAVs are also increasingly explored for object manipulation, enabling tasks such as aerial grasping [4], and contact-based inspection [5]. Unlike traditional robotic arms, aerial manipulation must account for underactuation, external disturbances, and dynamic stability, making control and planning particularly challenging. These capabilities open up new possibilities in areas like disaster response, infrastructure maintenance, and industrial automation, where UAVs can interact with objects in environments that are otherwise inaccessible.

Different mechanisms can be used to achieve manipulation [6], such as actuated robotic arms [7], as well as passive mechanisms such as spherical joints [8, 9] and cables [10, 11]. Actuated mechanisms require more power and are also heavier, resulting in a more costly design of the UAV. Passive mechanisms, while lighter and less costly, add more degrees of underactuation to the system, making the system more difficult to control. In configurations where the load is directly attached to the UAVs

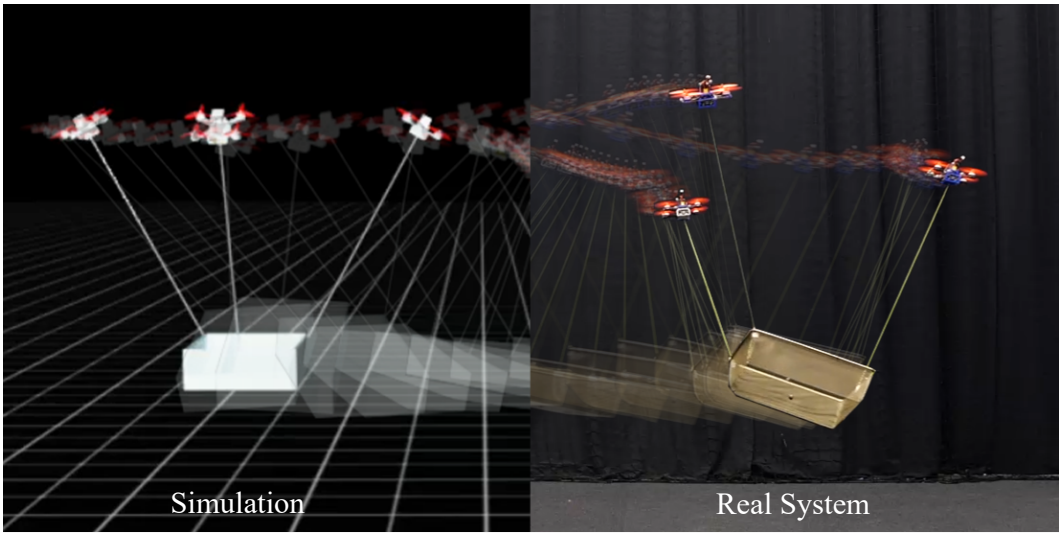


Figure 1: Multi-MAV lifting system performing full-pose control of a cable-suspended load. Left: simulation environment used to train the decentralized outer-loop control policy. Right: policy transferred to the real system.

(e.g. through spherical joints), the agility of the system is limited due to the extra inertia induced by the load. The cable-suspended load solution is a simple and effective strategy that allows agile transport of the load, but the non-linear coupling dynamics between the load and the UAV and the swing of the load affect the flight dynamics of the UAV.

While a single low-cost MAV has limited payload capacity, collaborative teams of MAVs can transport significantly heavier loads. In addition, by connecting at least 3 MAVs with the load at different points using tethers, the full pose of the load can be controlled by changing the position of the MAVs [12], yielding a cooperative cable-suspended manipulation solution, which shows great potential for aerial-based construction, inspection, and resecuring [13, 14, 15, 16, 17].

To coordinate and control MAV fleets, the state-of-the-art method [17] employs a centralized framework that accurately captures the strong dynamical coupling between the MAVs and the suspended load. This ensures safety and stability while addressing the significant underactuation inherent to cable-suspended systems, preventing actuator saturations and reciprocal collisions. However, using centralized control strategies for such systems suffers from critical drawbacks: computational complexity tends to scale exponentially with the number of agents for many approaches, rendering real-time control infeasible for larger teams with a centralized scheme [17, 18]. In addition, dependence on global state information and centralized communication is often impractical due to limits on sensors and communication bandwidth. A plausible solution, decentralization, remains an open challenge to effectively coordinate MAV fleets due to partial observability, limited communication bandwidth, and decision-making under strong dynamical coupling between agents while co-manipulating an object.

In this work, we present the first decentralized algorithm to achieve a real-world demonstrated full-pose manipulation of a cable-suspended payload using a team of MAVs. Our method leverages multi-agent reinforcement learning (MARL) and **does not require any inter-agent communication**. Instead, each agent only takes their own state and identity, the load pose, and the target load pose as observations. We train the policy through MARL in a centralized training with decentralized execution (CTDE) paradigm using multi-agent proximal policy optimization (MAPPO) [19]. Each MAV learns to communicate implicitly through the load pose information. To fill the sim-to-real gap in this highly dynamic cooperative task, we design the action space of the RL policy as reference linear accelerations and body rates of the MAV and combine the RL policy with a low-level controller based on incremental nonlinear dynamic inversion (INDI) [20, 21, 22]. The low-level

controller follows the linear acceleration command with the body rate reference as the feedforward commands, ensuring agile and smooth control maneuvers during the cooperative manipulation.

Our method enables zero-shot transfer of the policy from simulation to real-world deployment to achieve full-pose control accuracy comparable to the state-of-the-art centralized controller [17], and is deployed fully onboard. In addition, experiments with real MAVs demonstrate that our method remains robust under load model uncertainties, operates effectively in heterogeneous agent settings where one MAV uses a different controller, and remains functional even when one of the MAVs completely fails. We also show setpoint tracking with 4 MAVs, and evaluate its trajectory tracking capabilities.

Our core contributions are as follows:

- The first method to achieve fully decentralized and onboard-deployed cooperative aerial manipulation in experiments with real MAVs, without any inter-agent communication between agents.
- A novel action space design for MAVs manipulating a cable-suspended load, together with a robust low-level controller, enabling successful zero-shot sim-to-real transfer.
- First demonstration of robust full-pose control of the cable-suspended load under heterogeneous conditions and even under complete in-flight failure of an MAV.

2 Related works

Cooperative aerial manipulation of a cable-suspended load typically embraces a centralized paradigm to consider the cable-load-MAVs system as a whole and requires global state observations to ensure safety and performance. Early research on multi-MAV cable-suspended load problems often relied on model simplifications, such as assuming a quasi-static regime to ignore dynamic coupling effects [12, 23, 24, 25], which cannot address force-related constraints and perform dynamic motions. Another class of methods leverages system flatness [26] and dynamic equations to account for dynamic coupling effects. An example is the cascaded scheme, which employs an outer-loop geometric controller to generate the commanded wrench for the load, distributes it as desired cable tensions, and executes it through inner-loop controllers of MAVs [14, 27, 28, 29]. The outer-loop controller can be replaced by various approaches, such as inverse dynamics control [30], linear quadratic regulator [15], and nonlinear model predictive control (NMPC) [16]. Recent work [17] leverages whole-body dynamics and NMPC to generate reference trajectories followed by an adaptive low-level controller, showing high agility and accuracy.

However, these centralized methods require exponentially higher computational budgets and communication burdens with the number of agents involved. Therefore, decentralized controllers, such as distributed MPC [31, 32] have been proposed and tested in simulation to address the problem with the computational issues. But these methods still require reliable inter-agent data transfer to obtain real-time states from other agents, which does not fundamentally solve the problems with limited communication bandwidth.

Multi-agent reinforcement learning has been extensively studied for complex multi-agent systems, including cooperative scenarios [33, 34, 35]. Beyond achieving expert-level performance in video games [36, 37], MARL has been successfully applied to robotics, enabling decentralized control of multiple agents. For instance, researchers have leveraged MARL to develop cooperative strategies in robot football [38, 39], as well as multi-robot object manipulation with quadrupedal robots, including pushing [40] and cable-based towing [41]. Unlike our approach, these manipulation methods [40, 41] rely on neighboring agent information through communication or onboard perception. In many cases, MARL is employed to optimize high-level task objectives while relying on mid- and low-level controllers for motor and sub-task execution, capitalizing on RL's ability to optimize a long-horizon task-level objective [42].

Recent work by [43] demonstrates MARL's potential for cooperative object manipulation using simulated humanoids, relying solely on object bounding box information without explicit inter-agent communication. However, their approach depends on handcrafted reward functions that guide the humanoids toward predefined grasping points and walking behaviors. In MAV applications, MARL has been explored for tasks like swarming [44], but challenges remain due to the platform's agility, instability, and reliance on high-frequency, low-latency control [45]. Recently, MARL has shown potential for training multi-MAV lifting systems using global state observations [46]. However, a significant challenge remains to address the sim-to-real gap and partial observability, especially for the multi-MAV lifting system, where dynamic uncertainties are substantial due to complex aerodynamic disturbances and unknown cable tensions.

Our method effectively bridges this gap by leveraging multi-agent reinforcement learning (MARL) to achieve the first real-world demonstration of decentralized aerial manipulation, operating without global state observations or inter-agent communication. Furthermore, the method is deployed entirely onboard, enabled by its computational efficiency.

3 Preliminaries

3.1 Single-agent reinforcement learning

Reinforcement Learning (RL) is a framework for sequential decision-making in which an agent learns to act in an environment in order to maximize cumulative reward over time. The agent interacts with the environment by observing its state, taking actions, and receiving feedback in the form of rewards. Through repeated interactions, the agent learns a policy that maximizes the expected cumulative reward based on this feedback.

The RL problem is typically modeled as a Markov Decision Process (MDP), defined by the tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

where:

- $\mathcal{S}$: The set of possible **states** the agent can occupy.
- $\mathcal{A}$: The set of **actions** available to the agent.
- $P(s' \mid s, a)$: The **transition probability** function, which defines the probability of moving to state s' given the current state s and action a.
- $R(s, a)$: The **reward function**, giving the expected immediate reward for taking action a in state s.
- $\gamma \in [0, 1]$: The **discount factor**, which balances the importance of immediate and future rewards.

The agent's objective is to learn a **policy** $\pi(a \mid s)$, which defines a probability distribution over actions given a state, such that it maximizes the expected **return**, defined as the sum of discounted future rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

A central concept in reinforcement learning is the **value function**, which estimates the expected return associated with states or state-action pairs under a given policy. These functions help the agent evaluate how favorable it is to be in a given state, or to take a particular action in that state.

- The **state-value function** $V^\pi(s)$ is the expected return when starting in state s and following policy π thereafter:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right]$$

- The **action-value function** $Q^\pi(s, a)$ is the expected return when starting in state s, taking action a, and then following policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right]$$

These value functions form the basis of many RL algorithms. For instance, value-based methods such as Q-learning [47] aim to estimate the optimal action-value function $Q^*(s, a)$, from which an optimal policy can be derived. In contrast, policy-based methods [48] optimize the policy directly, often using gradients of the expected return with respect to the policy parameters.

3.2 Actor-Critic methods

Actor-critic methods [49] combine the strengths of both value-based and policy-based approaches. These methods maintain two separate models: an **actor**, which represents the policy $\pi_\theta(a \mid s)$ and is responsible for selecting actions, and a **critic**, which estimates the value function (typically $V^\pi(s)$ or $Q^\pi(s, a)$) and provides feedback to improve the actor's policy. The value function can be used as a baseline [48] and helps reduce the variance of the policy gradient estimates, leading to more stable and efficient learning.

A notable strength of actor-critic methods is their ability to operate in continuous action spaces, where the actor typically learns a parameterized distribution (e.g., Gaussian) over real-valued actions. This makes them particularly well-suited for robotics and control problems, where both actions and value estimates are continuous.

3.3 Multi-agent reinforcement learning

While traditional reinforcement learning considers a single agent interacting with a stationary environment, many real-world problems involve multiple agents that must learn and act simultaneously. MARL extends the RL framework to such settings, where each agent interacts with both the environment and other agents, either cooperatively, competitively, or in a mixed manner.

In the MARL setting, the environment is typically modeled as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [50], where each agent i observes local information o_t^i, selects an action a_t^i, and receives an individual or shared reward. The joint actions of all agents influence the global state transitions and reward structure, causing the optimal policies to adapt in response to the learning of other agents, which results in a non-stationary learning environment from each agent's perspective.

A key challenge in MARL is the increased complexity due to the joint action space and the non-stationarity introduced by concurrently learning agents. To address this, various approaches have been proposed, such as:

- **Centralized training with decentralized execution** (CTDE) [51], where agents are trained with access to global information but act using only local observations at test time.
- **Value decomposition** methods [52], which learn a centralized value function that factorizes across agents (e.g., VDN [52], QMIX [53]).
- **Communication and coordination** strategies that allow agents to share information or plan jointly.

Parameter Sharing is a common technique in MARL where multiple agents share and update the same policy network parameters during training. Instead of learning separate policies for each agent, a single shared policy is trained and used by all agents. This approach reduces the number of parameters to learn, improves sample efficiency, and promotes coordination by encouraging agents to learn common behaviors [54]. Parameter sharing is especially effective when agents are homogeneous and operate in similar environments or roles. However, it may be less suitable when agents have distinct capabilities or objectives.

In this work, we assume the agents are homogeneous and implement parameter sharing. The policies are trained using MAPPO [19], a multi-agent actor-critic algorithm, in CTDE fashion.

3.4 MAV dynamic model

To model the MAV in the simulation and make use of the robust low-level controller, we use a rigid-body model of a quadrotor [22] to model the MAV dynamics.

Using 6-DoF rigid body kinematic and dynamic equations, we can describe the translational dynamics by:

$$\ddot{\xi} = \frac{T\mathbf{z}_B + \mathbf{f}_a}{m} + \mathbf{g} \tag{1}$$

Where ξ is the position of the quadrotor center of mass (CoM), T is the collective thrust, m is the total mass of the quadrotor, $\mathbf{g} \in \mathbb{R}^3$ is the gravitational vector, and $\mathbf{f}_a$ are the external aerodynamic drag forces during high-speed flights.

The rotational kinematics and dynamics of the system are given by

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \mathbf{\Omega}^B \end{bmatrix} \tag{2}$$

$$\mathbf{I}_v \dot{\mathbf{\Omega}}^B = \mathbf{I}_v \boldsymbol{\alpha}^B = -\mathbf{\Omega}^B \times \mathbf{I}_v \mathbf{\Omega}^B + \boldsymbol{\tau} + \boldsymbol{d}_\tau \tag{3}$$

where $\otimes$ represents quaternion multiplication. The vector $\mathbf{\Omega}$ denotes the angular velocity of the body frame $\mathcal{F}_B$ relative to the inertial world frame $\mathcal{F}_I$, and its time derivative, $\boldsymbol{\alpha}$, corresponds to the angular acceleration. In our formulation, we work with $\mathbf{\Omega}^B = [\Omega_x, \Omega_y, \Omega_z]^T$, which is the representation of angular velocity in the body frame and can be directly obtained from the IMU. The matrix $\mathbf{I}_v$ defines the inertia matrix of the quadrotor. The term $\boldsymbol{\tau}$ captures the net torque produced by the propellers, while $\boldsymbol{d}_\tau$ models the uncertainties on the body torques caused by high-order aerodynamic effects, center of mass bias, or distinction among rotors.

The combined thrust and rotor-induced torques can be expressed as functions of the rotor speeds:

$$\begin{bmatrix} T \\ \boldsymbol{\tau} \end{bmatrix} = \mathbf{G}_1 \mathbf{u} + \mathbf{G}_2 \dot{\boldsymbol{\omega}} + \mathbf{G}_3(\mathbf{\Omega})\boldsymbol{\omega} \tag{4}$$

where

$$\boldsymbol{u} = c_t \boldsymbol{\omega}^{\circ 2} \tag{5}$$

represents the individual thrust produced by each rotor, with $\circ$ denoting the Hadamard power operation. Here, c_t is the thrust coefficient and $\boldsymbol{\omega}$ represents the vector of angular velocities for each propeller. The matrices $\boldsymbol{G}_1$ to $\boldsymbol{G}_3$ are defined as:

$$\boldsymbol{G}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ l\sin\beta & -l\sin\beta & -l\sin\beta & l\sin\beta \\ -l\cos\beta & -l\cos\beta & l\cos\beta & l\cos\beta \\ c_q/c_t & -c_q/c_t & c_q/c_t & -c_q/c_t \end{bmatrix} \tag{6}$$

$$\boldsymbol{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ I_p & -I_p & I_p & -I_p \end{bmatrix} \tag{7}$$

$$\boldsymbol{G}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ I_p\Omega_y & -I_p\Omega_y & I_p\Omega_y & -I_p\Omega_y \\ -I_p\Omega_x & I_p\Omega_x & -I_p\Omega_x & I_p\Omega_x \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

In these equations, c_q stands for the torque coefficient, while β is the angle between the forward direction and the propeller, and l is the arm length from the quadrotor base to the propeller. The term I_p denotes the rotor's moment of inertia about the $\boldsymbol{z}_B$ axis.

The terms $\boldsymbol{G}_2\dot{\boldsymbol{\omega}}$ and $\boldsymbol{G}_3(\boldsymbol{\Omega})\boldsymbol{\omega}$ correspond to torques arising from rotor angular acceleration and gyroscopic effects, respectively. These terms are often omitted in controller design. In our work, we incorporate the inertial torque term $\boldsymbol{G}_2\dot{\boldsymbol{\omega}}$ at test time using the INDI controller, but neglect it during training. $\boldsymbol{G}_3(\boldsymbol{\Omega})\boldsymbol{\omega}$ is omitted because of the symmetry of quadrotors, where the gyroscopic effects of the rotors (approximately) cancel each other out.

4 Methods

4.1 Problem formulation

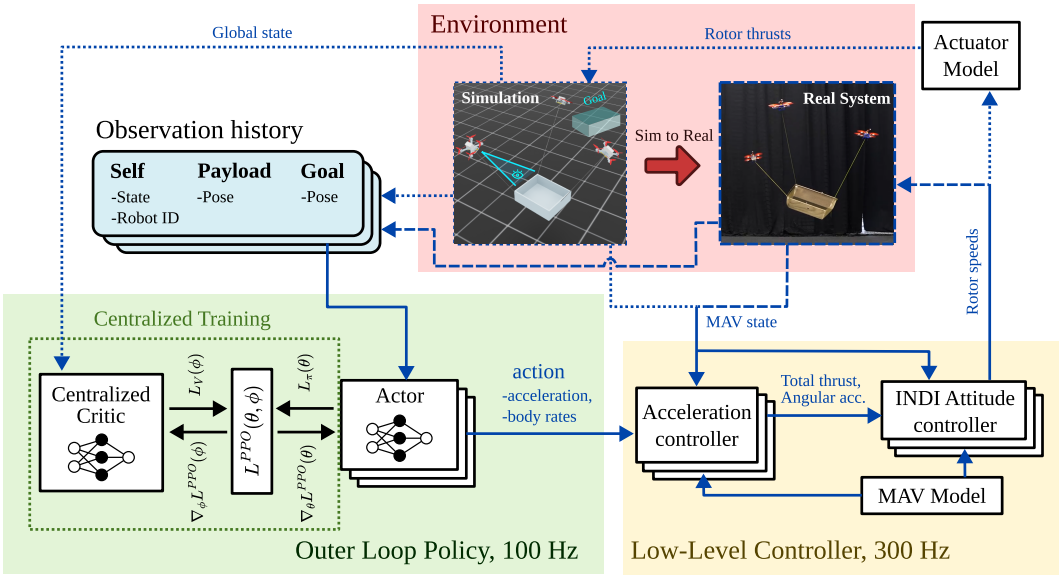


Figure 2: Overview of our method. Dotted lines indicate components only for training; dashed lines indicate those only for real-system deployment; solid lines for both. The training process involves the centralized critic (which observes the privileged global state), direct access to MAV states, and the actuator model that maps rotor speeds to thrust forces. Shared actors make decisions based on local observations, without access to other agents' states. The output actions, namely acceleration and body rates, are tracked by a robust model-based low-level controller based on INDI.

An overview of the full approach is shown in Figure 2. Our method utilizes MARL to train an outer-loop control policy, which generates reference accelerations and body rates for the low-level controller in real-time based on local observations of the ego-MAV state, payload- and goal pose. The low-level controller, including an INDI attitude controller, tracks these references based on the

MAV model and accelerometer measurements. The privileged full state is observed by the centralized critic during training, which is discarded at execution time. Collected experience is shared across actors to update the parameters of a shared policy. This enables training to be centralized while execution remains decentralized, allowing each agent to run the policy independently onboard after zero-shot transfer from simulation to the real world.

We model the cooperative aerial manipulation as a decentralized partially observable Markov decision process (Dec-POMDP) [50] with a shared reward function. A Dec-POMDP is defined by $\langle \mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{I}$ denotes the set of agents with the total number of agents being equal to N, $\mathcal{S}$ is the environment state, $\mathcal{A} = \{a_i\}_{i=1}^{N}$ is the joint action space of all agents, $\mathcal{O} = \{o_i\}_{i=1}^{N}$ represents each agent's partial observation of the environment, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition model, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the shared reward function and γ is the discount factor. At each timestep t, the current state $\mathbf{s}_t \in \mathcal{S}$ transitions to a new state $\mathbf{s}_{t+1}$ based on the joint action $\mathbf{a}_t \in \mathcal{A}$ and the transition function $\mathcal{P}$. Each agent i then receives the shared reward as feedback from the environment.

Our approach employs the CTDE paradigm [51], utilizing privileged global state information during training for the asymmetric centralized critic while relying solely on local observations for policy execution. Each agent i has a policy $\pi_i : \omega_i(o_i) \to a_i$ that maps its local observation, processed through its observation function ω_i, to an action a_i. We implement parameter sharing across agents (i.e., $\pi_i = \pi_j \; \forall i, j$), thus reducing π_i to a homogeneous policy π. The set of observation functions for all agents can be denoted as $\Omega = \{\omega_i\}_{i=1}^{N}$. The final decentralized partially observable problem is thus defined by the tuple $\langle \mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \Omega, \mathcal{P}, \mathcal{R}, \gamma \rangle$

4.2 Observations and rewards

The state of each MAV is given by $\boldsymbol{x}_i = \Big[\boldsymbol{p}_{M,i},\, \boldsymbol{R}_{M,i},\, \boldsymbol{v}_{M,i},\, \boldsymbol{\Omega}_{M,i} \Big]$, where $\boldsymbol{p}_{M,i} \in \mathbb{R}^3$ denotes the MAV's position, $\boldsymbol{R}_{M,i} \in \mathbb{R}^9$ is the vector composed of elements of its rotation matrix, $\boldsymbol{v}_{M,i} \in \mathbb{R}^3$ and $\boldsymbol{\Omega}_{M,i} \in \mathbb{R}^3$ denote its linear and angular velocities. We use the subscript i to denote the i-th MAV. The state of the load is given by $\boldsymbol{x}_L = \Big[\boldsymbol{p}_L,\, \boldsymbol{R}_L,\, \boldsymbol{v}_L,\, \boldsymbol{\Omega}_L \Big]$ where $\boldsymbol{p}_L \in \mathbb{R}^3$ denotes the load's position, $\boldsymbol{R}_L \in \mathbb{R}^9$ is the vector composed of elements of its rotation matrix, $\boldsymbol{v}_L \in \mathbb{R}^3$ and $\boldsymbol{\Omega}_L \in \mathbb{R}^3$ denote its linear and angular velocities. The state of the goal relative to the payload is denoted by $\boldsymbol{x}_G = \Big[\boldsymbol{d}_G,\, \boldsymbol{R}_G \Big]$ where $\boldsymbol{d}_G \in \mathbb{R}^3$ and $\boldsymbol{R}_G \in \mathbb{R}^9$ represent the goal position relative to the current load position and the vector composed of elements of its relative rotation matrix from the current load orientation to the goal orientation respectively. All quantities are described in the inertial world frame $\mathcal{F}_I$. The global state that is observable to the centralized critic during training is then denoted as:

$$\boldsymbol{s} = \Big[\boldsymbol{x}_L,\, \boldsymbol{x}_G,\, \boldsymbol{x}_{M,1},\, \boldsymbol{x}_{M,2},\, \cdots,\, \boldsymbol{x}_{M,N} \Big] \tag{9}$$

where N is the total number of MAVs. The local policies have an observation space that only includes the load pose, relative goal terms, their own respective MAV state, and a one-hot vector $\boldsymbol{e}_i$ indicating their identity to enable role differentiation among homogeneous agents, as the policy network parameters are shared across all MAVs. The observation space for the i-th MAV is described as:

$$\boldsymbol{o}_i = \Big[\boldsymbol{p}_L,\, \boldsymbol{R}_L,\, \boldsymbol{x}_G,\, \boldsymbol{x}_{M,i},\, \boldsymbol{e}_i \Big] \tag{10}$$

As the problem is partially observable, we use a history of observations by stacking the current and last 2 observations of the policy [55].

We train the policies using MAPPO, a model-free MARL algorithm that extends PPO [56] with CTDE. The reward at time t, denoted as r_t, is defined as:

$$r_t = r_t^{\text{pos}} + r_t^{\text{ori}} + r_t^{\text{down}} + r_t^{\text{act}} + r_t^{\text{br}} + r_t^{\text{thrust}}, \tag{11}$$

where r_t^{pos} and r_t^{ori} are rewards to track the goal position and orientation for the load, r_t^{down} encourages the MAVs to aim their (proxy) downwash away from the load for stability against aerodynamic

disturbances, r_t^{act} and r_t^{br} penalize action changes from the last time step and large body-rate outputs respectively for smoother flight, r_t^{thrust} penalizes outputting large thrusts which encourages energy efficiency.

The reward function components are formulated as:

$$\begin{aligned} r_t^{\text{pos}} &= \lambda_1 \exp\left(-\lambda_2 \left\|\boldsymbol{p}_G - \boldsymbol{p}_L\right\|\right), \\ r_t^{\text{ori}} &= \lambda_3 \exp\left(-\lambda_4 \theta(\boldsymbol{q}_{\text{G}}, \boldsymbol{q}_{\text{L}})\right), \\ r_t^{\text{down}} &= \lambda_5 \left(1 - \exp\left(-\lambda_6 \cdot \min_i \left\|f_{\text{int}}(\boldsymbol{p}_{M,i}, \boldsymbol{t}_i) - \boldsymbol{p}_L\right\|\right)\right), \\ r_t^{\text{act}} &= \lambda_7 \exp\left(-\left\|(\boldsymbol{a}_t - \boldsymbol{a}_{t-1})/N\right\|^2\right), \\ r_t^{\text{br}} &= \lambda_8 \exp\left(-\left\|\boldsymbol{\Omega}_t^B/N\right\|\right), \\ r_t^{\text{thrust}} &= \lambda_9 \exp\left(-\max(\boldsymbol{T}_t/T_{\max})\right), \end{aligned} \tag{12}$$

The amount of MAVs is denoted by N, and $\boldsymbol{a}$ represents the control command, and $\boldsymbol{\Omega}^B$ the body rate part of the control command. $\boldsymbol{T} \in \mathbb{R}^{4N}$ is the vector containing the rotor thrusts from each MAV, which is then normalized by the maximum thrust output $T_{\max}$. $\lambda_1, \lambda_2 \cdots \lambda_9$ are different positive hyperparameters. All components are normalized by the simulation frequency. The chosen hyperparameters can be found in Table 2.

Here $\boldsymbol{p}_G$ and $\boldsymbol{p}_L$ denote the goal and load positions respectively. $\theta(\boldsymbol{q}_{\text{G}}, \boldsymbol{q}_{\text{L}})$ denotes the quaternion error magnitude function which is calculated using the quaternion representation of the goal orientation $\boldsymbol{q}_{\text{G}}$, and the load orientation $\boldsymbol{q}_{\text{L}}$. The error is calculated by taking the norm of the axis-angle representation of the quaternion difference $\boldsymbol{q}_{\text{G}} \otimes \boldsymbol{q}_{\text{L}}^*$, where $\boldsymbol{q}_{\text{L}}^*$ is the conjugate of $\boldsymbol{q}_{\text{L}}$.

$$\theta(\mathbf{q}_{\text{G}}, \mathbf{q}_{\text{L}}) = \left\|\phi\left(\mathbf{q}_{e,L}\right)\right\| \tag{13}$$

$$\mathbf{q}_{e,L} = \mathbf{q}_{\text{G}} \otimes \mathbf{q}_{\text{L}}^* = \left[q_{e,L,w}, q_{e,L,x}, q_{e,L,y}, q_{e,L,z}\right]^T \tag{14}$$

$$\phi(\mathbf{q}_{e,L}) = \frac{\mathbf{v}}{\frac{\sin(\theta/2)}{\theta}} \quad \text{where } \mathbf{v} = \begin{bmatrix} q_{e,L,x} \\ q_{e,L,y} \\ q_{e,L,z} \end{bmatrix} \tag{15}$$

When $\theta = 0$, the bottom fraction in Equation 15 will be undefined. Therefore, as $\theta \to 0$, we use the Taylor approximation $\frac{\sin(\theta/2)}{\theta} \approx \frac{1}{2} - \frac{\theta^2}{48}$.

The function $\boldsymbol{f}_{\text{int}}(\boldsymbol{p}_{M,i}, \boldsymbol{t}_i)$ computes the intersection point between two elements: the line defined by the i-th MAV's position $\boldsymbol{p}_{M,i}$ and its thrust direction $\boldsymbol{t}_i$, and the plane containing the payload. This payload plane is characterized by its normal vector $\boldsymbol{n} = \boldsymbol{\ell}_x \times \boldsymbol{\ell}_y$, where $\boldsymbol{\ell}_x$ and $\boldsymbol{\ell}_y$ represent arbitrary vectors spanning the load's local x-y plane. From all such intersection points computed for each MAV, the operator $\min$ selects the closest one to the payload position, corresponding to the most significant downwash effect.

The intersection calculation expands to:

$$\boldsymbol{f}_{\text{int}}(\boldsymbol{p}_{M,i}, \boldsymbol{t}_i) = \boldsymbol{p}_{M,i} + \left(\frac{d - \boldsymbol{n} \cdot \boldsymbol{p}_{M,i}}{\boldsymbol{n} \cdot \boldsymbol{t}_i}\right) \boldsymbol{t}_i \tag{16}$$

where $d = \boldsymbol{n} \cdot \boldsymbol{p}_L$ defines the payload plane's offset from the origin through the payload position $\boldsymbol{p}_L$.

4.3 Action space and low-level controller

To balance reliable sim-to-real transfer with sufficient control authority, the choice of action space is critical. Prior work in single MAV control demonstrates that high-level outputs (e.g., position or

velocity) enhance robustness to disturbances and sim-to-real gaps but limit performance, whereas low-level outputs (e.g. snap) improve tracking precision at the cost of larger transfer discrepancies [57, 58]. To address this trade-off, we propose a mid-level action space in desired accelerations and body rates (ACCBR). This approach preserves adequate control capability while also being robust against uncertain disturbances and model mismatches from the cable-suspended load.

The low-level controller converts the acceleration reference $\boldsymbol{a}_{i,\text{ref}}$ from the outer-loop policy to the thrust direction command through the following acceleration controller:

$$\boldsymbol{z}_{i,\text{des}} = \|\boldsymbol{a}_{i,\text{ref}} - \boldsymbol{g} - \boldsymbol{f}_{\text{i,ext}}\|, \quad \boldsymbol{f}_{i,\text{ext}} = m_i \boldsymbol{a}_{i,\text{filtered}} - \boldsymbol{f}_{i,\text{filtered}} \tag{17}$$

where $\boldsymbol{g}$ is the gravity vector, external forces $\boldsymbol{f}_{\text{ext}}$, primarily due to the cable tensions, are estimated using the MAV mass m_i, and filtered unbiased accelerometer measurements $\boldsymbol{a}_{i,\text{filtered}}$ and collective thrust $\boldsymbol{f}_{i,\text{filtered}}$ are computed from a classical quadratic thrust model and filtered rotor speed feedbacks [22]. In the training stage, the acceleration and collective thrust measurements are directly taken from the true simulation states, without any noise or filtering. Given a reference yaw angle ψ_r, which is always zero in our case for all MAVs, we get an intermediate axis $\boldsymbol{x}_{C_i,d}$ from which the desired attitude command can be obtained through the following equations:

$$\mathbf{x}_{C_i,d} = [\cos\psi_r, \sin\psi_r, 0]^T \tag{18}$$

$$\mathbf{y}_{i,d} = \frac{\mathbf{z}_{i,des} \times \mathbf{x}_{C_i,d}}{\|\mathbf{z}_{i,des} \times \mathbf{x}_{C_i,d}\|} \tag{19}$$

$$\mathbf{x}_{i,d} = \mathbf{y}_{i,d} \times \mathbf{z}_{i,des} \tag{20}$$

$$\mathbf{R}(\mathbf{q}_{i,d}) = [\mathbf{x}_{i,d}, \mathbf{y}_{i,d}, \mathbf{z}_{i,des}] \tag{21}$$

where $\mathbf{q}_{i,d}$ is the desired attitude quaternion for the i-th MAV.

Since controlling the heading of a quadcopter is more likely to cause motor saturation, because its control effectiveness is less effective than pitch and roll control [22], we use tilt-prioritized attitude control [59] to handle yaw error $\tilde{\boldsymbol{q}}_{e,i,\text{yaw}}$ separately from the pitch and roll (reduced-attitude) errors $\tilde{\boldsymbol{q}}_{e,i,\text{red}}$ for the i-th MAV:

$$[q_{e,i,w}, q_{e,i,x}, q_{e,i,y}, q_{e,i,z}]^T = \boldsymbol{q}_{i,d} \otimes \boldsymbol{q}_i^{-1} \tag{22}$$

$$\tilde{\boldsymbol{q}}_{e,i,\text{red}} = \frac{1}{\sqrt{q_{e,i,w}^2 + q_{e,i,z}^2}} \begin{bmatrix} q_{e,i,w}q_{e,i,x} - q_{e,i,y}q_{e,i,z} \\ q_{e,i,w}q_{e,i,y} + q_{e,i,x}q_{e,i,z} \\ 0 \end{bmatrix} \tag{23}$$

$$\tilde{\boldsymbol{q}}_{e,i,\text{yaw}} = \frac{1}{\sqrt{q_{e,i,w}^2 + q_{e,i,z}^2}} \begin{bmatrix} 0 \\ 0 \\ q_{e,i,z} \end{bmatrix} \tag{24}$$

$$\boldsymbol{\alpha}_{i,d}^B = k_{q,\text{red}} \tilde{\boldsymbol{q}}_{e,i,\text{red}} + k_{q,\text{yaw}} \operatorname{sgn}(q_{e,i,w}) \tilde{\boldsymbol{q}}_{e,i,\text{yaw}} + K_\Omega (\boldsymbol{\Omega}_{i,d}^B - \boldsymbol{\Omega}_i^B) \tag{25}$$

where $k_{q,\text{red}}$ and $k_{q,\text{yaw}}$ are positive gains for the reduced-attitude and yaw control errors, respectively. K_Ω is a positive gain for the angular velocity error from the policy output body rates $\boldsymbol{\Omega}_{i,d}^B$. The gains $k_{q,\text{red}}, k_{q,\text{yaw}}$ and K_Ω are identical for all MAVs and are specified in Table 4.

The equations above do not capture all MAV dynamics, such as certain unmodeled rotational effects ($\boldsymbol{d}_\tau$ in Equation 3). These terms are often difficult to model accurately in real-world systems. To handle this, we use INDI, a robust controller that combines direct sensor measurements with a dynamic model to improve robustness against model uncertainties and external disturbances. Its effectiveness has been demonstrated in prior work [20, 21, 22].

Given the angular acceleration $\boldsymbol{\alpha}_{i,d}^{B}$ and linear acceleration commands $\boldsymbol{a}_{i,\text{ref}}$, the desired collective thrust $T_{i,d}$ is retrieved by:

$$T_{i,d} = \|\boldsymbol{z}_{i,\text{des}}\| \, m_i \tag{26}$$

Then, from INDI, the desired body torque is retrieved by:

$$\boldsymbol{\tau}_{i,d} = \boldsymbol{\tau}_{i,f} + \boldsymbol{I}_v \left(\boldsymbol{\alpha}_{i,d}^{B} - \dot{\boldsymbol{\Omega}}_{i,f}^{B} \right) \tag{27}$$

So, the unmodeled rotational effects are captured by $\dot{\boldsymbol{\Omega}}_{i,f}$ and filtered body torque $\boldsymbol{\tau}_{i,f}$, where:

$$\boldsymbol{\tau}_{i,f} = \bar{\boldsymbol{G}}_1 \boldsymbol{\omega}_{i,f}^2 + \Delta t^{-1} \bar{\boldsymbol{G}}_2 (\boldsymbol{\omega}_{i,f} - \boldsymbol{\omega}_{i,f,k-1}) \tag{28}$$

is calculated from rotor speed measurements. The body rates $\boldsymbol{\Omega}_{i,f}^{B}$ and rotor speeds $\boldsymbol{\omega}_{i,f}$ are processed through identical low-pass Butterworth filters (second-order, 12 Hz cutoff), which allows them to be synchronized. Here, the subscript $k-1$ denotes the previous time step value, while Δt is the sampling interval. The matrices $\bar{\boldsymbol{G}}_1$ and $\bar{\boldsymbol{G}}_2$ are constructed using the final three rows of their respective complete matrices.

From the following equation, the rotor speed command can be solved:

$$\begin{bmatrix} T_{i,d} \\ \boldsymbol{\tau}_{i,d} \end{bmatrix} = \boldsymbol{G}_1 \boldsymbol{\omega}_{i,c}^2 c_t + \Delta t^{-1} \boldsymbol{G}_2 (\boldsymbol{\omega}_{i,c} - \boldsymbol{\omega}_{i,c,k-1}) c_t, \tag{29}$$

where $\omega_{i,c}$ represents the sole unknown quantity that can be determined through numerical methods.

4.3.1 Training time simplifications

It should be noted that the low-pass filters are not required during training time, since there is no noise present in the simulation. In this case, all MAV states are retrieved directly from the simulation data buffers. Moreover, since angular rotor dynamics and complex aerodynamic torques (G2 and G3 terms in Equation 4) are not simulated in the training phase, the yaw torque command is directly retrieved by taking the last element of the desired body torques in simulation $\boldsymbol{\tau}_{i,ds}$ [60] from Equation 30, which neglects uncertainties on the body torque.

$$\boldsymbol{\tau}_{i,ds} = \boldsymbol{I}_v \boldsymbol{\alpha}_{i,d}^{B} + \boldsymbol{\Omega}_i^{B} \times \boldsymbol{I}_v \boldsymbol{\Omega}_i^{B} \tag{30}$$

With the simplified rotor dynamics, we can drop the $\boldsymbol{G}_2$ term from Equation 29 and retrieve the commanded rotor speed commands in the simulation during the training phase $\boldsymbol{\omega}_{i,s}$ through:

$$\boldsymbol{\omega}_{i,s} = \sqrt{\boldsymbol{G}_1^{-1} \begin{bmatrix} T_{i,d} \\ \boldsymbol{\tau}_{i,d} \end{bmatrix} / c_t} \tag{31}$$

The implementation of the low-level controller is modified from [61]. We refer readers to [20, 21, 22, 62] for derivations and further details on INDI.

To simulate the effect of motor inertia, we implement a first-order low-pass filter that acts as an actuator model. Given the commanded rotor speeds $\boldsymbol{\omega}_{i,s} \in \mathbb{R}^4$, the final rotor speed command $\boldsymbol{\omega}_{i,sa} \in \mathbb{R}^4$ is calculated as:

$$\boldsymbol{\omega}_{i,sa} = (1 - \beta) \boldsymbol{\omega}_{i,s} + \beta \boldsymbol{\omega}_{i,sa,k-1} \tag{32}$$

where $k-1$ indicates the previous timestep, and

$$\beta = \exp(\frac{-\Delta t}{c_\tau}) \tag{33}$$

where c_τ is the motor time constant. The rotor speeds are then converted into forces $\boldsymbol{F}_{i,s} \in \mathbb{R}^4$ and yaw torques $M_{i,s} \in \mathbb{R}$, and applied to the MAV rotor bodies and MAV base respectively in the simulation through:

$$\boldsymbol{F}_{i,s} = c_t \boldsymbol{\omega}_{i,sa}^2 \tag{34}$$

$$M_{i,s} = c_q \boldsymbol{\omega}_{i,sa}^2 \cdot \boldsymbol{d} \tag{35}$$

Where $\boldsymbol{d} \in \mathbb{R}^4$ is the vector indicating the direction of each rotor (clockwise or counter-clockwise).

4.4 Training setup

We train our method completely in simulation and achieve zero-shot transfer to real-world experiments. The simulation environment is built using NVIDIA's Isaac Lab [63], and the MARL algorithms are modified from [64]. Training was conducted on a consumer-grade RTX 3090 GPU and completed in 17 hours. The network architecture is a 4-layer MLP of size $[1024, 512, 256, 128]$ for both the shared policies and the centralized critic. The inputs to the network are normalized stacked observation histories with history size $H = 3$. We also implement a form of advantage filtering [65] where 50% of the samples with the lowest advantage magnitude are dropped. This approach prioritizes learning from the most informative state transitions—specifically the underexplored extremes of the data distribution where actions have a clearly better or worse outcome—thereby improving data efficiency during training. For a complete overview of the network and agent parameters, we refer the readers to Table 3.

The MAVs with the cable-suspended load spawn uniformly in a random location between -1 and 1 in the xy-plane, between 0.5 and 1.5 along the z-axis, with a random heading orientation. The goal position is uniformly sampled from the same set, but also allows for pitch and roll angles of ± 45 degrees. Despite sampling of the goal is limited to the predefined sets, the policy is still able to generalize and reach goal poses outside of it during execution. For setups with more than 3 MAVs, the mass of the load is sampled from a uniform distribution between 1.0 and 1.8 kg (the mass of the real payload is 1.4 kg). For the 3-MAV setup, the cables are modeled as rigid rods of 1 meter in length, connected to both the payload and the MAVs via ball joints. When using more than 3 MAVs, the system becomes overconstrained, which can lead to cable slack [12]. To address this, the cables are instead modeled as three rigid segments linked by ball joints.

The episodes have a duration of 20 seconds, where a single goal pose is given to encourage stable hovering of the payload. The episode times out after 20 seconds, in which case the return is bootstrapped using the value function estimate, or it terminates earlier if:

- any MAV or the payload is too close to the ground,
- the angle between the payload and the cable exceeds a certain threshold,
- the angle between the cable and the MAV exceeds a certain threshold,
- cables collide with each other,
- MAVs collide with each other,
- any rigid body is outside a specified bounding box,
- any of the cable tensions are below a specified threshold. (> 3 MAVs)

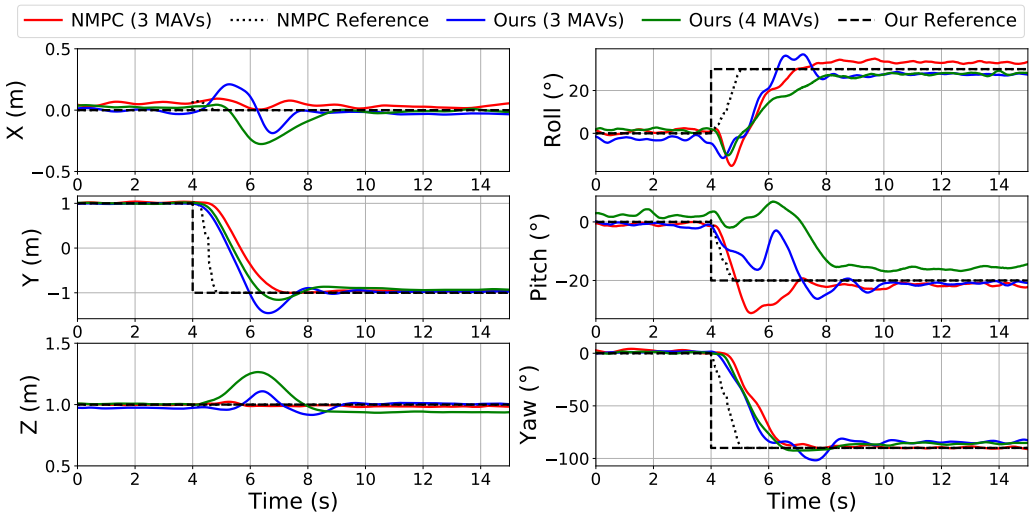


Figure 3: Time series of pose tracking results comparing our method and a centralized NMPC method [17]. Our method also includes a setup with 4 MAVs.

5 Experiments and Results

Real-world evaluation setup We evaluate our method in real-world experiments. All our experiments include 3 MAVs, unless stated otherwise, built based on the Agilicious [61] flight stack. Each MAV is connected to a basket-shaped payload with 1-meter cables at three distinct locations. The MAVs weigh 0.6 kg, and the payload weighs 1.4 kg. We conduct the experiment in an indoor flight space with motion capture systems. We attach motion capture markers to the MAVs and the payload to measure their positions and orientations and distribute them to each MAV through ROS at 100 Hz. The MAV state estimate is obtained by fusing the MAV pose and IMU measurements into an extended Kalman filter (EKF). The trained policy and low-level controllers are deployed onboard each MAV. The policy is inferred at 100 Hz to send acceleration and body-rate commands. The low-level controller is executed at 300 Hz to generate rotor speed commands.

Gazebo simulation environment For safety reasons, the ablation studies and comparisons of partially observable and fully observable policies in the failure and heterogeneous scenarios were done in simulation. We ran the Agilicious flight stack together with the Gazebo simulator [66] with quadrotor and sensor plugins provided by the RotorS [67] library, which introduces sensor noise, aerodynamic disturbances, and potential system latencies in a ROS environment. Note that **all experiments were evaluated in the real world unless explicitly stated otherwise.**

5.1 Real-world experiments

Setpoint tracking Our real-world experiments demonstrate agile pose control of three MAVs with a cable-suspended load, tracking a 2 m displacement with (30°, -20°, -90°) attitude commands. We compare our decentralized method with the state-of-the-art centralized NMPC approach [17] in Figure 3. Despite being fully decentralized, our method achieves comparable tracking performance with positional and attitude RMSEs of 0.52 m (vs 0.45 m) and 22.93 degrees (vs 16.24 degrees), respectively. Note that RMSE comparisons favor NMPC as it tracks a reference trajectory while we only track target poses, resulting in a larger RMSE in the transient area of the step command. We also show successful pose control with 4 MAVs (without cable slack), resulting in tracking RMSEs of 0.92 m and 42.67 degrees. The increased error, compared to the 3 MAV case, may be due to the system becoming overconstrained, which introduces more complex coordination and (cable) dynamics [28]. We emphasize that the main advantage of larger teams lies in increased payload capacity rather than improved tracking accuracy. In terms of computational efficiency, we run the

NMPC and our method onboard a Raspberry Pi 5 (2.4 GHz quad-core ARM Cortex-A76). Our method inferences in **6 ms** at 100 Hz, versus NMPC's **78 ms** at 10 Hz. Crucially, while NMPC's computation time grows exponentially with agent count, e.g. 174 ms and 267 ms for 5 and 6 agents respectively, our agent-independent approach maintains **constant computation time regardless of team size**.

Robustness against load model mismatch To evaluate robustness, we introduce five additional objects (0.216 kg, 15.4% of load mass), including four freely movable items that dynamically perturb both mass distribution and center of mass. Despite no inertia randomization during training, the system maintains strong tracking performance (0.63 m vs 0.60 m position RMSE; 26.93 degrees vs 26.49 degrees attitude error). The low-level feedback controller automatically compensates for these disturbances, demonstrating inherent robustness to model uncertainties. A snapshot and experimental results are shown in Figure 4.

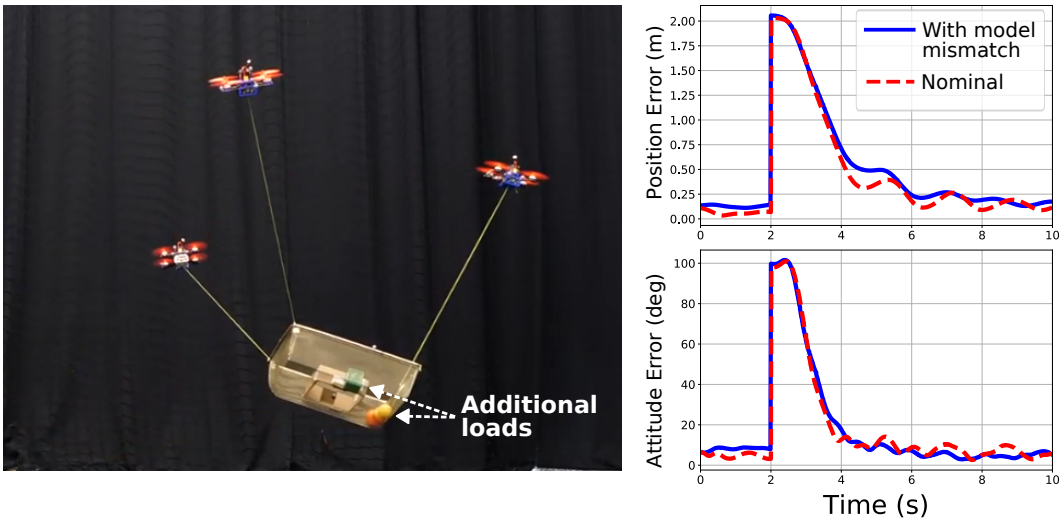


Figure 4: Snapshot of the test where additional load is added to the original load, and the pose error with and without such model mismatch.

Heterogeneous agents Although our policy is trained under the assumption of homogeneous agents, it remains effective when deployed with heterogeneous agents. In this experiment, we let the load hover at a fixed point. Then we hacked one of the MAVs by replacing its RL policy with a model-based controller [22], and provided it with different setpoints to observe the behavior of the other two MAVs controlled by the RL policy. Specifically, we commanded the hacked MAV to move outwards on the y-axis by 0.7 m to pull the load away from the reference; we then commanded the hacked MAV to move inwards by 0.3 m to push it closer to the other two MAVs. Figure 5 provides a snapshot of the experiments.

Since the policy is conditioned solely on the load pose and not on the states of the other agents, the two remaining MAVs utilizing the policy are able to compensate for load pose deviations from the reference. Figure 6 compares the performance of partially observable and fully observable policies (in simulation) in the heterogeneous agents scenario where the hacked MAV first pushes in, and then pulls out. The partially observable policy, being independent of other agents' states, allows the unaffected MAVs to compensate for the hacked agent, maintaining system stability. In contrast, the fully observable policy—which relies on neighboring agents' states—performs worse, exhibiting larger tracking errors (0.42 m vs. 0.28 m in position, 30.08 degrees vs. 8.88 degrees in attitude) and large oscillations during the inward push.

In-flight failure of one MAV The effectiveness of our method with a heterogeneous agent setup and robustness against load model uncertainties also offers strong fault tolerance in the case of agent failure. In this experiment, we deliberately turned off the hacked MAV (one of the two on

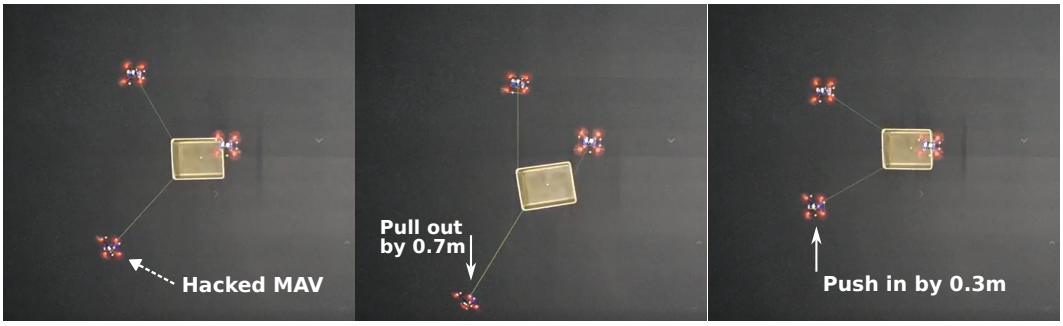


Figure 5: Snapshot of the test with heterogeneous agents in which one MAV is manually controlled (hacked) to pull out and push in, and the other two MAVs counteract the interference of the hacked MAV.

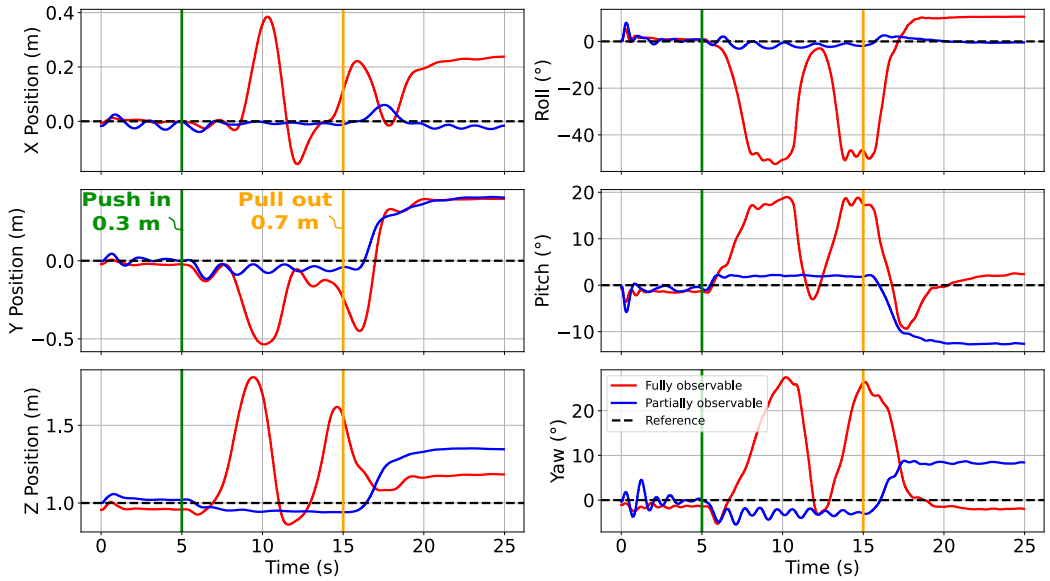


Figure 6: Time series of the load pose in the heterogeneous agents scenario, comparing the performance of the partially observable policy and the fully observable policy. The time points at which control commands are issued to push the load inward by 0.3 m relative to the desired policy position, or to pull it outward by 0.7 m, are indicated in green (push-in) and orange (pull-out), respectively.

the same side). As a result, the load was controlled by the remaining two MAVs. Note that with only two MAVs, the load orientation around the line joining the remaining two attachment points becomes unactuated. Even worse, the failed MAV hangs underneath the load, leading to additional disturbances to the post-failure system. Despite that, our method allows the other two MAVs to effectively control the remaining 5 DoFs of the load. We show that the system is still able to yaw by -180 degrees and is also able to maintain position control by flying 0.5 meters down along the z-axis and maneuvering along the y-axis by 1 meter. The snapshots of the setup and tracking results after the failure are seen in Figure 7 and Figure 8 respectively. As in the heterogeneous agent case, the remaining agents can compensate for the missing MAV since the policy operates independently of other agents' states, thereby avoiding unstable behavior in out-of-distribution scenarios. In contrast, the fully observable policy fails under these conditions due to its reliance on the states of all agents.

Figures 9 and 10 show the tracking performance of the partially observable and fully observable policies (in simulation) following an in-flight failure of one MAV. Figure 9 corresponds to the scenario in which no additional command inputs are issued, whereas Figure 10 corresponds to the

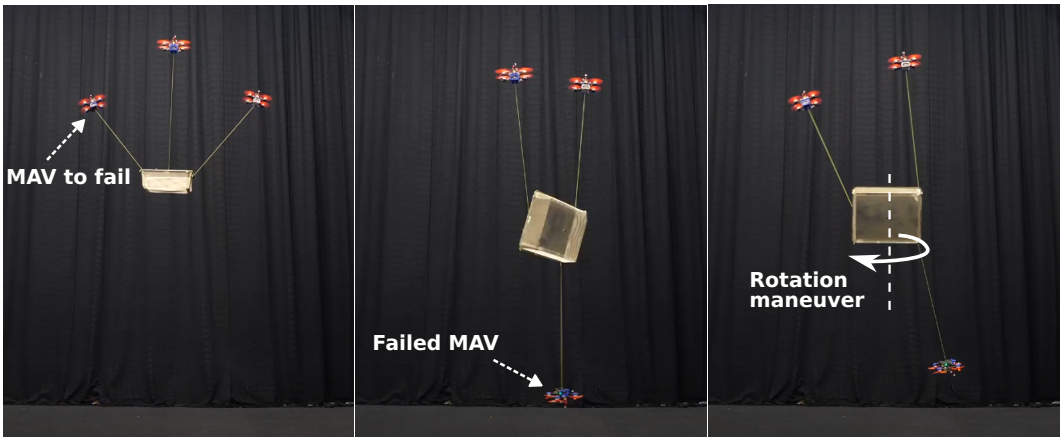


Figure 7: Snapshots of the case where one MAV fails in flight and the remaining two MAVs manage to control the load.

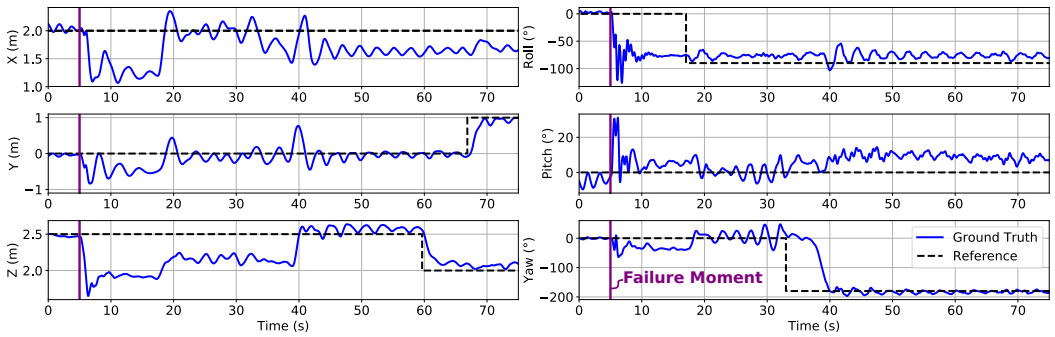


Figure 8: Time series of the case where one MAV fails in flight and the remaining two MAVs manage to control the load. The thick purple line indicates the moment the MAV fails.

scenario in which new attitude and position commands are introduced at $t = 15$ s and $t = 25$ s. In both scenarios, the partially observable policy successfully compensates for the MAV failure. In contrast, the fully observable policy exhibits strong oscillatory behavior, causing the suspended MAV to repeatedly crash to the ground. When new pose commands are sent, the fully observable policy fails to track them accurately, whereas the partially observable policy is still able to track 5 DoF. This results in larger tracking errors for the fully observable policy, which incurs position and attitude RMSEs of 1.50 m and 73.37 degrees, respectively, compared to 0.67 m and 50.31 degrees for the partially observable policy. The robustness of the partially observable policy is attributed to its independence from the states of neighboring agents, which helps prevent cascading failures.

Trajectory tracking Although our method is not trained for trajectory tracking, we evaluate its trajectory tracking capabilities against that of the centralized NMPC [17] in Figure 11. The reference trajectory is a figure-eight trajectory with a maximum velocity of 1 m/s and a maximum acceleration of 0.5 m/s^2. It is worth noting that our method only considers the reference pose information, while the NMPC also takes velocity information from the reference trajectory into account. For future specialized trajectory tasks, incorporating higher-order information such as velocity, as well as future reference points [68] into the observations would significantly improve tracking performance and make for a fairer comparison. Nonetheless, our method is able to successfully track the figure-eight trajectory, albeit with a high tracking error. Our method achieves positional and attitude RMSEs of 0.82 m (vs 0.10 m), and 18.22 degrees (vs 4.80 degrees).

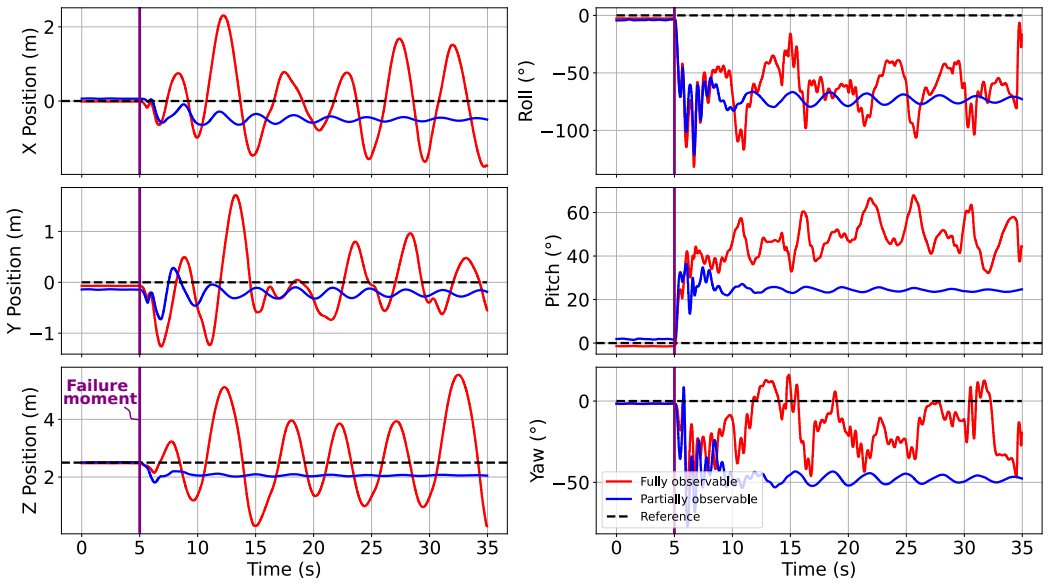


Figure 9: Time series of load pose in the in-flight failure of one MAV case without sending any commands, comparing a partially observable policy vs a fully observable policy. The thick purple line indicates the moment the MAV fails.

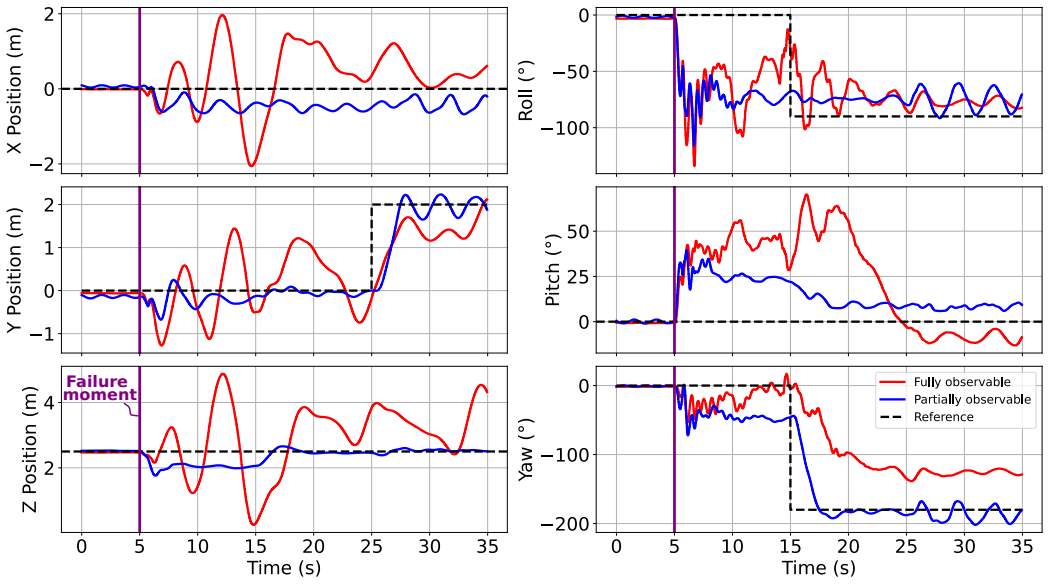


Figure 10: Time series of load pose in the in-flight failure of one MAV case, comparing a partially observable policy vs a fully observable policy. An attitude command is sent after 10 seconds and a positional command after 20 seconds. The thick purple line indicates the moment the MAV fails.

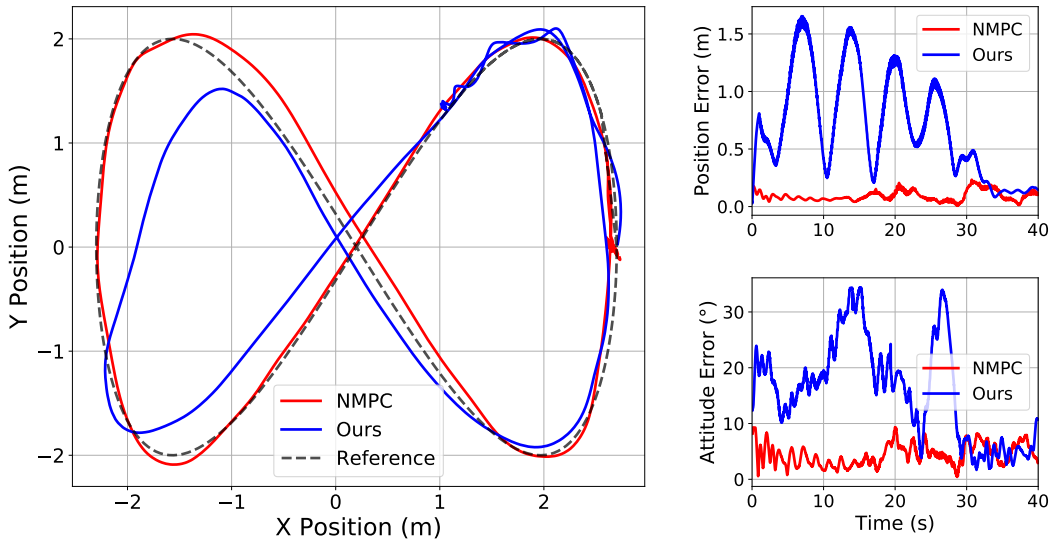


Figure 11: Comparison of our method, which is not trained for trajectory tracking, against the centralized NMPC in [17]. **Left**: top view of the flight path of the center of mass of the load while tracking a figure-eight trajectory with a maximum velocity of 1 m/s and maximum acceleration of 0.5 m/s^2. **Right**: position (top) and attitude (bottom) tracking errors time series.

5.2 Ablation studies

All comparisons are performed in the Gazebo simulation environment for safety reasons.

5.2.1 Comparison among different action and observation spaces

We compare our selected observation and action space with other alternatives. All policies are trained on a limited budget of 1 billion environment steps (10 hours of training), and are evaluated 10 times in the Gazebo environment.

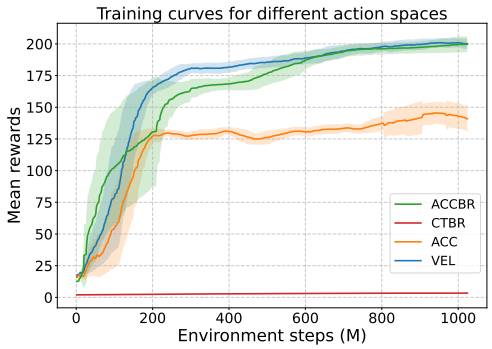


Figure 12: Training plots comparing different action spaces.

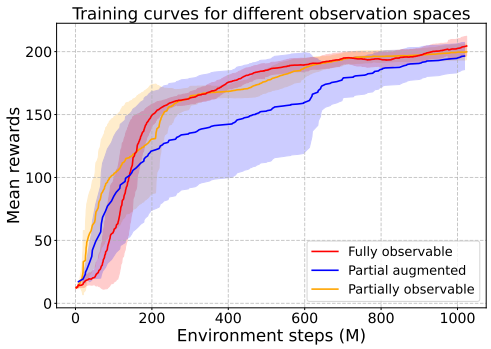


Figure 13: Training curves of fully observable, partial augmented, and partially observable observation spaces.

Action space We compare the ACCBR action space with three alternatives: velocity (VEL), linear acceleration (ACC), and collective thrust with body rates (CTBR). The ACCBR, VEL, and ACC outputs all utilize the same low-level controllers, which compensate for disturbances such as aerodynamic forces and cable tension. In contrast, CTBR outputs feed directly into the INDI attitude controller without additional disturbance compensation. The training curves are shown in Figure 12.

The RMSE results in Table 1 demonstrate that the VEL action space achieves the best performance, followed by ACCBR, while ACC fails to track the load orientation accurately. Notably, the widely used CTBR approach [69, 42] fails to learn effectively. Since CTBR directly commands collective thrust without leveraging the proposed low-level controller's disturbance compensation, we hypothesize that the unpredictable cable forces exerted on each MAV make the learning process prohibitively difficult, as there are no cable force sensors mounted for both training and evaluations.

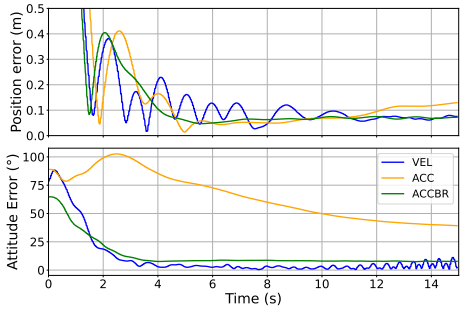


Figure 14: Positional and attitude errors comparing different action spaces at test time in the Gazebo environment.

Action space	Pos RMSE	Att RMSE
ACCBR	0.64 ± 0.00	33.87 ± 0.91
CTBR*	NaN	NaN
ACC	$\mathbf{0.54 \pm 0.00}$	87.89 ± 1.85
VEL	0.56 ± 0.06	$\mathbf{25.74 \pm 1.49}$

*Not able to take off

Table 1: Pose tracking RMSEs of different action spaces at test time in the Gazebo environment.

However, while VEL yields superior RMSE, Figure 14 reveals that it induces **hazardous oscillations** during execution. In contrast, ACCBR exhibits more stable hovering despite higher initial errors. For real-world tasks like inspection or delivery—where stability is critical—we argue that

ACCBR is the safer and preferred choice. Contrary to the aforementioned belief in Section 4 that higher-level action spaces reduce the sim-to-real gap for single MAV cases, the VEL action space has a larger sim-to-real gap than ACCBR. We speculate that in this multi-MAV lifting system, where uncertainties and disturbances are significantly larger than in the single MAV case, the VEL action space may not have enough control authority to reject uncertain high-frequency disturbances.

Observation space To benchmark the decentralized policy's performance, we compare three observation space cases: (1) the fully observable case with global state $\boldsymbol{s} = [\boldsymbol{x}_L, \boldsymbol{x}_G, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3]$, (2) an augmented partial observability case where each MAV i also receives the load twist and other MAVs' positions ("Partial augmented") $\boldsymbol{o}_i = [\boldsymbol{x}_L, \boldsymbol{x}_G, \boldsymbol{p}_{j_1}, \boldsymbol{p}_{j_2}, \boldsymbol{x}_i, \boldsymbol{e}_i]$ with $\boldsymbol{p}_{j_1}, \boldsymbol{p}_{j_2}$ representing the neighboring agents' positions, and (3) the partially observable case. For partially observable cases, we include observation histories ($H = 3$) to improve state estimation and decision-making under uncertainty [55]. Figure 13 reveals comparable convergence across all configurations, indicating that load pose alone serves as a sufficient statistic for implicit MAV coordination, while the full global state contains redundant elements.

5.2.2 Performance without centralized critic

To assess the impact of using a centralized critic with access to privileged global state information, we compare its performance against a policy trained with a shared local critic. The local critic has access only to local observations, which are the same as those available to the actor. The training curves in Figure 15 show that the setup with the local critic fails to converge to the same performance as with the centralized critic, and even collapses at the end. Specifically, the policy with the local critic fails to learn the position and orientation rewards effectively. We hypothesize that access to global state information allows the centralized critic to produce more accurate value estimates, which can indirectly support more effective credit assignment during learning [54], thereby improving task performance.

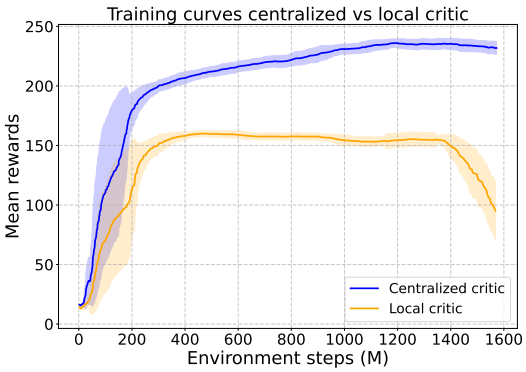


Figure 15: Training curves using a centralized critic vs using a local critic.

5.2.3 Performance with different history lengths

Despite that adding history of observations in partially observable settings is a common practice in literature [55], we compare the performance of the partially observable policy with different history lengths in the observation space. $H = 1$ means that the history only contains the observations of the current timestep (no previous observations). All policies are trained on a limited budget of 2 billion environment steps and are evaluated in the Gazebo environment.

Figures 16 and 17 show that including historical observations has little impact on performance. We hypothesize that the load's pose—even without historical data—contains enough information to estimate the other agents' states, enabling implicit communication among the MAVs. Further investigation into the role of history in more complex scenarios, such as those with higher noise or additional MAVs, is left for future work.

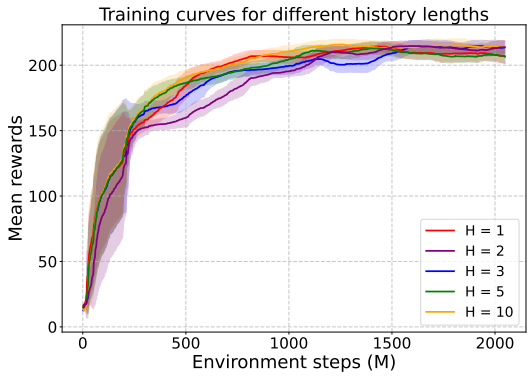


Figure 16: Training curves comparing different history lengths for the partially observable policy.

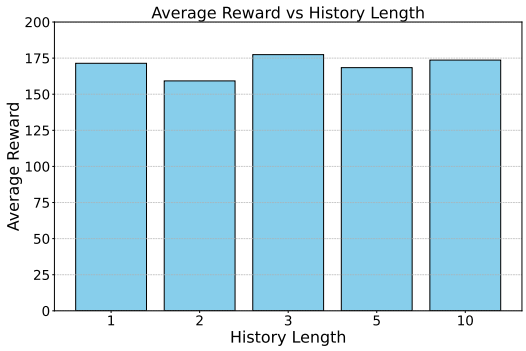


Figure 17: Mean reward of policies with different history lengths over 10 runs at test time in the Gazebo environment.

6 Conclusion

We introduced a decentralized method using MARL that allows for full-pose control of a cable-suspended load using three MAVs without any inter-MAV communication or neighboring MAV information. The policy is computationally tractable and executes entirely onboard. We proposed a novel action space of accelerations and body rates (ACCBR) along with a robust low-level controller and showcase zero-shot transfer from simulation to real-world deployment. Extensive testing with real MAVs shows that the setpoint tracking performance of our method is comparable to that of the state-of-the-art centralized NMPC [17], despite being fully decentralized and having significantly lower computation time. Our method demonstrates robustness against unknown disturbances, heterogeneous agents, and even the complete in-flight failure of one MAV. We attribute this resilience to two key factors: 1) closed-loop reference tracking by the low-level controller, which maintains stability despite perturbations, 2) decentralized policy independence, where local agents operate without dependence on neighboring states, preventing cascading failures. Our work shows promising results to enable scalable and robust cooperative aerial manipulation with minimal onboard sensing and no internal communications required.

7 Limitations

Our method requires pose measurement of the load, which is not often practical beyond lab environments. In our experiment, we require an external motion capture system to provide high-frequency load pose measurement. For future real-world outdoor deployment, onboard sensing (e.g., a downward-facing camera for load pose estimation and SLAM for MAV localization) would be necessary. This would introduce new challenges, such as different reference frames for the load and MAVs—requiring additional transformation and synchronization—, observation delays, imperfect state estimates, and sensor noise. Additionally, our current framework does not address obstacle avoidance, as we assume collision-free paths to the goal—an unrealistic assumption in unstructured environments. Future work will focus on integrating a robust perception stack and obstacle avoidance capabilities.

Acknowledgments

I would like to extend my gratitude to my supervisors who have guided me and helped me in the process of my literature review and thesis; Dr. Sihao Sun, Andreu Matoses Gimenez, Dr. Eugene Vinitsky and Dr. Javier Alonso-Mora. I would also like to thank Dr. Yunlong Song, Dennis Benders and Shlok Deshmukh for the insightful discussions, and the lab engineers Maurits Pfaff and Kseniia Khomenko for their help with the experiments. Furthermore, I am grateful to my family, friends, and girlfriend for supporting me throughout this period. Lastly, I would like to thank my friends from the MSc Robotics for making the past 2 years a very special, memorable and enjoyable time. I will miss you guys!

References

[1] Q. Lindsey, D. Mellinger, and V. Kumar. Construction with quadrotor teams. *Autonomous Robots*, 33(3):323–336, 2012.

[2] E. N. Barmpounakis, E. I. Vlahogianni, and J. C. Golias. Unmanned aerial aircraft systems for transportation engineering: Current practice and future challenges. *International Journal of Transportation Science and Technology*, 5(3):111–122, 2016.

[3] M. A. Trujillo, J. R. Martínez-de Dios, C. Martín, A. Viguria, and A. Ollero. Novel aerial manipulator for accurate and robust industrial ndt contact inspection: A new tool for the oil and gas inspection industry. *Sensors*, 19(6):1305, 2019.

[4] S. Kim, S. Choi, and H. J. Kim. Aerial manipulation using a quadrotor with a two dof robotic arm. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4990–4995. IEEE, 2013.

[5] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. *Robot Operating System (ROS) The Complete Reference (Volume 2)*, pages 3–39, 2017.

[6] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi. Past, present, and future of aerial robotic manipulators. *IEEE Transactions on Robotics*, 38(1):626–645, 2021.

[7] F. Ruggiero, V. Lippiello, and A. Ollero. Aerial manipulation: A literature review. *IEEE Robotics and Automation Letters*, 3(3):1957–1964, 2018.

[8] A. Tagliabue, M. Kamel, R. Siegwart, and J. Nieto. Robust collaborative object transportation using multiple mavs. *The International Journal of Robotics Research*, 38(9):1020–1044, 2019.

[9] G. Loianno, V. Spurny, J. Thomas, T. Baca, D. Thakur, D. Hert, R. Penicka, T. Krajnik, A. Zhou, A. Cho, et al. Localization, grasping, and transportation of magnetic objects by a team of mavs in challenging desert-like environments. *IEEE Robotics and Automation Letters*, 3(3):1576–1583, 2018.

[10] P. J. Cruz and R. Fierro. Cable-suspended load lifting by a quadrotor uav: hybrid model, trajectory generation, and control. *Autonomous Robots*, 41:1629–1643, 2017.

[11] M. Bisgaard, J. D. Bendtsen, and A. la Cour-Harbo. Modeling of generic slung load system. *Journal of guidance, control, and dynamics*, 32(2):573–585, 2009.

[12] J. Fink, N. Michael, S. Kim, and V. Kumar. Planning and control for cooperative manipulation and transportation with aerial robots. In *Robotics Research: The 14th International Symposium ISRR*, pages 643–659. Springer, 2011.

[13] K. Sreenath and V. Kumar. Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots. *rn*, 1(r2):r3, 2013.

[14] T. Lee. Geometric control of quadrotor uavs transporting a cable-suspended rigid body. *IEEE Transactions on Control Systems Technology*, 26(1):255–264, 2017.

[15] J. Geng, P. Singla, and J. W. Langelaan. Load-distribution-based trajectory planning and control for a multilift system. *Journal of Aerospace Information Systems*, 19(5):366–381, 2022.

[16] G. Li and G. Loianno. Nonlinear model predictive control for cooperative transportation and manipulation of cable suspended payloads with multiple quadrotors. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5034–5041. IEEE, 2023.

[17] S. Sun, X. Wang, D. Sanalitro, A. Franchi, M. Tognon, and J. Alonso-Mora. Agile and cooperative aerial manipulation of a cable-suspended load. *arXiv preprint arXiv:2501.18802*, 2025.

[18] L. Bakule and M. Papik. Decentralized control and communication. *Annual Reviews in Control*, 36(1):1–10, 2012.

[19] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

[20] E. J. Smeur, Q. Chu, and G. C. De Croon. Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3): 450–461, 2016.

[21] E. Tal and S. Karaman. Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. *IEEE Transactions on Control Systems Technology*, 29(3):1203–1218, 2020.

[22] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza. A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight. *IEEE Transactions on Robotics*, 38(6):3357–3373, 2022.

[23] N. Michael, J. Fink, and V. Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30:73–86, 2011.

[24] M. Manubens, D. Devaurs, L. Ros, and J. Cortés. Motion planning for 6-d manipulation with aerial towed-cable systems. In *Robotics: science and systems (RSS)*, page 8p, 2013.

[25] D. Sanalitro, H. J. Savino, M. Tognon, J. Cortés, and A. Franchi. Full-pose manipulation control of a cable-suspended load with multiple uavs under uncertainties. *IEEE Robotics and Automation Letters*, 5(2):2185–2191, 2020.

[26] K. Sreenath, T. Lee, and V. Kumar. Geometric control and differential flatness of a quadrotor uav with a cable-suspended load. In *52nd IEEE conference on decision and control*, pages 2269–2274. IEEE, 2013.

[27] G. Li, R. Ge, and G. Loianno. Cooperative transportation of cable suspended payloads with mavs using monocular vision and inertial sensing. *IEEE Robotics and Automation Letters*, 6 (3):5316–5323, 2021.

[28] G. Li, X. Liu, and G. Loianno. Rotortm: A flexible simulator for aerial transportation and manipulation. *IEEE Transactions on Robotics*, 40:831–850, 2023.

[29] K. Wahba and W. Hönig. Efficient optimization-based cable force allocation for geometric control of a multirotor team transporting a payload. *IEEE Robotics and Automation Letters*, 9 (4):3688–3695, 2024.

[30] C. Masone and P. Stegagno. Shared control of an aerial cooperative transportation system with a cable-suspended payload. *Journal of Intelligent & Robotic Systems*, 103(3):40, 2021.

[31] J. Wehbeh, S. Rahman, and I. Sharf. Distributed model predictive control for uavs collaborative payload transport. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11666–11672. IEEE, 2020.

[32] B. Wang, R. Huang, and L. Zhao. Auto-multilift: Distributed learning and control for cooperative load transportation with quadrotors. *arXiv preprint arXiv:2406.04858*, 2024.

[33] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[34] K. Zhang, Z. Yang, and T. Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[35] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pages 66–83. Springer, 2017.

[36] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

[37] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[38] S. Liu, G. Lever, Z. Wang, J. Merel, S. A. Eslami, D. Hennes, W. M. Czarnecki, Y. Tassa, S. Omidshafiei, A. Abdolmaleki, et al. From motor control to team play in simulated humanoid football. *Science Robotics*, 7(69):eabo0235, 2022.

[39] Z. Li, F. Bjelonic, V. Klemm, and M. Hutter. Marladona-towards cooperative team play using multi-agent reinforcement learning. *arXiv preprint arXiv:2409.20326*, 2024.

[40] Y. Feng, C. Hong, Y. Niu, S. Liu, Y. Yang, W. Yu, T. Zhang, J. Tan, and D. Zhao. Learning multi-agent loco-manipulation for long-horizon quadrupedal pushing. *arXiv preprint arXiv:2411.07104*, 2024.

[41] W.-T. Chen, M. Nguyen, Z. Li, G. N. Sue, and K. Sreenath. Decentralized navigation of a cable-towed load using quadrupedal robot team via marl. *arXiv preprint arXiv:2503.18221*, 2025.

[42] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82): eadg1462, 2023.

[43] J. Gao, Z. Wang, Z. Xiao, J. Wang, T. Wang, J. Cao, X. Hu, S. Liu, J. Dai, and J. Pang. Coohoi: Learning cooperative human-object interaction with manipulated object dynamics. *Advances in Neural Information Processing Systems*, 37:79741–79763, 2024.

[44] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme. Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning. In *Conference on robot learning*, pages 576–586. PMLR, 2022.

[45] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza. Bootstrapping reinforcement learning with imitation for vision-based agile flight. In *8th Annual Conference on Robot Learning*.

[46] 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*, 9(3):2838–2844, 2024.

[47] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[48] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[49] V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[50] F. A. Oliehoek and C. Amato. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[51] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[52] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[53] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

[54] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[55] M. J. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI fall symposia*, volume 45, page 141, 2015.

[56] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[57] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10504–10510. IEEE, 2022.

[58] J. Eschmann, D. Albani, and G. Loianno. Learning to fly in seconds. *IEEE Robotics and Automation Letters*, 2024.

[59] D. Brescianini and R. D'Andrea. Tilt-prioritized quadrocopter attitude control. *IEEE Transactions on Control Systems Technology*, 28(2):376–387, 2018.

[60] S. Sun, C. C. de Visser, and Q. Chu. Quadrotor gray-box model identification from high-speed flight data. *Journal of Aircraft*, 56(2):645–661, 2019.

[61] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Science robotics*, 7(67):eabl6259, 2022.

[62] X. Wang, E.-J. Van Kampen, Q. Chu, and P. Lu. Stability analysis for incremental nonlinear dynamic inversion control. *Journal of Guidance, Control, and Dynamics*, 42(5):1116–1129, 2019.

[63] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

[64] A. Serrano-Muñoz, D. Chrysostomou, S. Bøgh, and N. Arana-Arexolaleiba. skrl: Modular and flexible library for reinforcement learning. *Journal of Machine Learning Research*, 24(254): 1–9, 2023.

[65] M. Cusumano-Towner, D. Hafner, A. Hertzberg, B. Huval, A. Petrenko, E. Vinitsky, E. Wijmans, T. Killian, S. Bowers, O. Sener, et al. Robust autonomy emerges from self-play. *arXiv preprint arXiv:2502.03349*, 2025.

[66] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004.

[67] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. Rotors—a modular gazebo mav simulator framework. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 595–625, 2016.

[68] J. Mayer, J. Westermann, J. P. G. H. Muriedas, U. Mettin, and A. Lampe. Proximal policy optimization for tracking control exploiting future reference information. *arXiv preprint arXiv:2107.09647*, 2021.

[69] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

A Supplementary Materials

A.1 Training configuration

Reward function weights The reward function weights shown in Table 2 are based on iterative tuning in simulation and real-world experiments.

Reward weight	Value
λ_1	1.5
λ_2	1.5
λ_3	1.5
λ_4	1.5
λ_5	0.5
λ_6	3.0
λ_7	0.5
λ_8	0.5
λ_9	0.5

Table 2: Reward function weights

Hyperparameters of MAPPO The hyperparameters of MAPPO are shown in table 3. The names of the parameters are based on the SKRL [64] learning library.

Hyperparameter	Value
number of envs	4096
rollouts	128
learing epochs	5
mini batches	4
discount factor	0.99
gae lambda	0.95
learning rate actor	5e-4
learning rate critic	1e-4
state preprocessor	RunningStandardScaler
shared state preprocessor	RunningStandardScaler
value preprocessor	RunningStandardScaler
grad norm clip	1.0
ratio clip	0.1
value clip	0.1
entropy loss scale	0.001
value loss scale	1.0
kl threshold	0.0

Table 3: MAPPO hyperparameters based on SKRL [64] learning library

A.2 Low-level controller parameters

Parameter	Value
$\mathrm{k}_{q,\mathrm{red}}$	150.0
$k_{q,\mathrm{yaw}}$	5.0
K_Ω	diag(25.0, 25.0, 8.0)

Table 4: Low-level controller gains