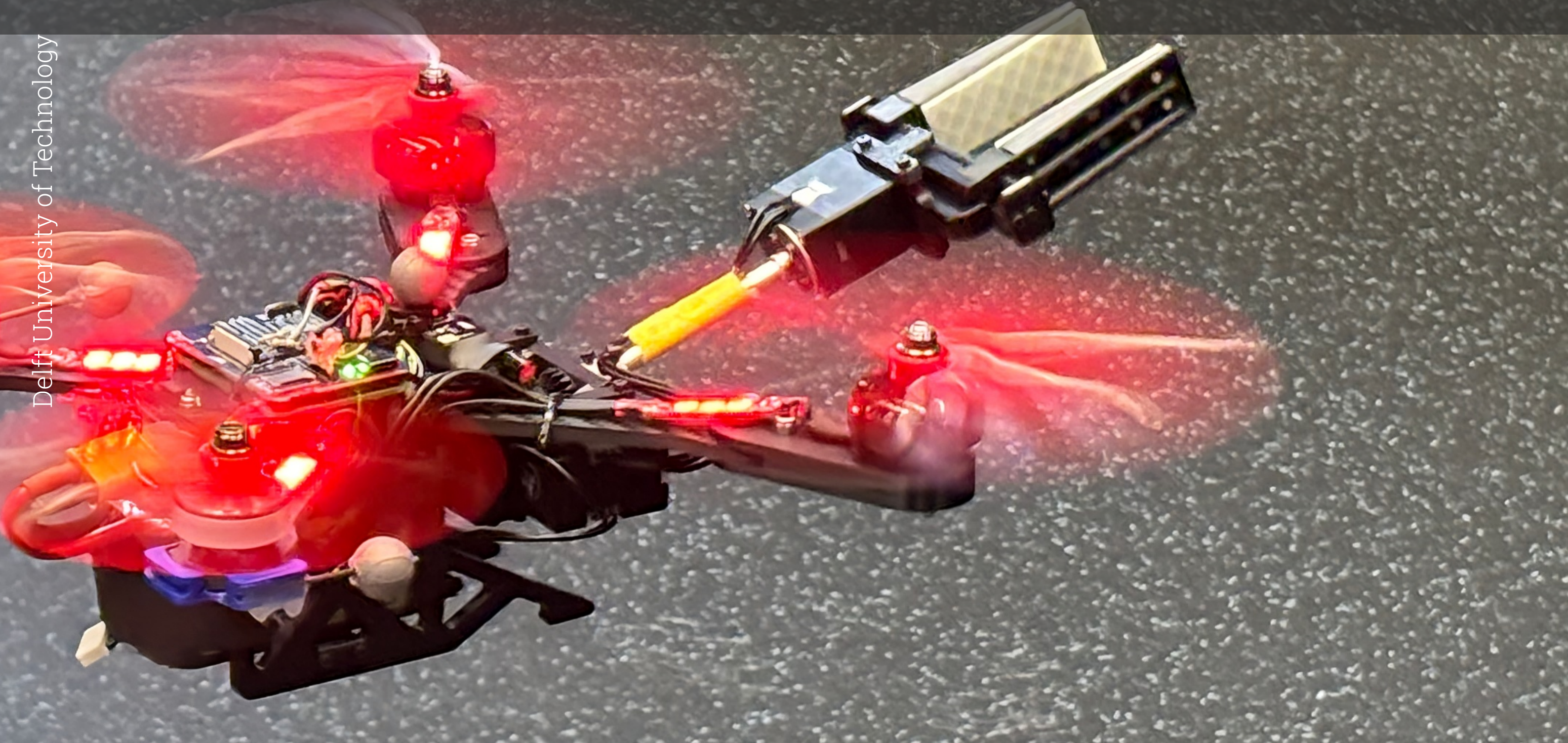
# Whole-body Control of an Aerial Manipulator with Reinforcement Learning

Master Thesis

Shlok Deshmukh

Delft University of Technology



TUDelft

# Whole-body Control of an Aerial Manipulator with Reinforcement Learning

Master Thesis

by

Shlok Deshmukh

To obtain the degree of Master of Science

at the Delft University of Technology,

to be defended on 27 August, 2025 at 3:00 p.m.

| | |
|---|---|
| Student Number: | 5928516 |
| Project Duration: | Nov, 2024 - August, 2025 |
| Main Supervisor: | Prof. Dr. Javier Alonso-Mora |
| Daily Supervisor: | Dr. Sihao Sun |
| Faculty: | Mechanical Engineering |

Cover: Captured at AMR Lab, Department of Cognitive Robotics, TU Delft

TU Delft

# Contents

# Nomenclature

## Terms

| Symbol/Term | Definition |
|---|---|
| Aerial Manipulator | A drone equipped with a robotic arm, capable of performing manipulation tasks while flying. |
| Aerial Vehicle / Platform | Any flying robot, typically multirotors, used as the base for an aerial manipulator. |
| Dynamics model | A model that describes the transition, given a state and action performed, returning the new state. Used as an umbrella term for analytical model and geometric model in this thesis. |
| Whole-body Control | Simultaneous control of locomotion and manipulation subsystems with a single unified algorithm. |
| End-Effector | The ”tool” at the tip of the manipulator (e.g., gripper) that physically interacts with the environment. |
| Trajectory | A time-parameterized path representing the drone’s desired motion in space. |
| Pose | The drone’s position and orientation in 3D space. |
| End-Effector Pose | The position and orientation of the manipulator’s tip, either in body or world frame. |
| Actuator Dynamics | The physical behavior and limitations of motors and servos, including delays and non-linearities. |
| Contact Dynamics | The forces and torques exchanged between the end-effector and the environment during manipulation tasks. |
| Proximal Policy Optimization (PPO) | A stochastic RL algorithm that balances exploration and exploitation with stable training. |
| Deep Deterministic Policy Gradient (DDPG) | A deterministic RL algorithm designed for continuous action spaces, combining value-based and policy-based methods. |
| Trust Region Policy Optimization (TRPO) | A stochastic RL algorithm that improves policy by adding a constraint using KL divergence making learning reliable without over updating. |
| Zero-Shot Transfer | The deployment of an RL policy (or any ML model) trained entirely in simulation to real-world without additional fine-tuning. |
| Partially Observable Markov Decision Process (POMDP) | A MDP where the agent has incomplete / partial knowledge of the environment. |
| Osprey | The aerial manipulator to be used in this study. It is a quadrotor base with a 2-DoF arm and a gripper as end-effector. |

## Symbol Definitions

| **Symbol** | **Space** | **Definition** |
|---|---|---|
| | **Section 5.1: Modeling and Problem Definition** | |
| $(x_W, y_W, z_W)$ | $\mathbb{R}^3$ | Basis vectors of world frame |
| $(x_B, y_B, z_B)$ | $\mathbb{R}^3$ | Basis vectors of quadrotor base |
| $(x_{EE}, y_{EE}, z_{EE})$ | $\mathbb{R}^3$ | Basis vectors of end-effector frame |
| ${}^W\mathbf{x}_b$ | $\mathbb{R}^3$ | Position of quadrotor base in world frame |
| ${}^W\mathbf{R}_b$ | $SO(3)$ | Rotation matrix of quadrotor base w.r.t. world frame |
| $\boldsymbol{\theta}$ | $\mathbb{T}^2$ | Vector of joint angles of 2-DoF manipulator arm |
| ${}^W\dot{\mathbf{x}}_b$ | $\mathbb{R}^3$ | Linear velocity of quadrotor base in world frame |
| ${}^B\boldsymbol{\Omega}_b$ | $\mathbb{R}^3$ | Angular velocity of quadrotor base in body frame |
| $\dot{\boldsymbol{\theta}}$ | $\mathbb{R}^2$ | Joint angular velocities of manipulator |
| $\mathbf{M}(q)$ | $\mathbb{R}^{8\times 8}$ | Inertia matrix of coupled quadrotor–manipulator system |
| $\mathbf{C}(q, \mathbf{v})$ | $\mathbb{R}^{8\times 8}$ | Coriolis and centrifugal matrix |
| $\mathbf{G}(q)$ | $\mathbb{R}^8$ | Gravity and external force vector |
| $\mathbf{B}(q)$ | $\mathbb{R}^8$ | Matrix that maps control inputs to generalized forces applied |
| $\mathbf{u}$ | $\mathbb{R}^n$ | Control input vector, $n$ depends on controller design |
| $\mathbf{G}_1$ | $\mathbb{R}^{4\times 4}$ | Control allocation matrix |
| $\mathbf{G}_2$ | $\mathbb{R}^{4\times 4}$ | Allocation matrix influencing torque generated due to rotor angular accelerations |
| $\mathbf{G}_3$ | $\mathbb{R}^{4\times 4}$ | Allocation matrix influencing torque generated due to gyroscopoic effects |
| $\mathcal{W}$ | $\mathbb{R}^4$ | Net force and torques generated by quadrotor base |
| $f$ | $\mathbb{R}$ | Net thrust generated by quadrotor in body frame |
| $\boldsymbol{\tau}$ | $\mathbb{R}^3$ | Net moment vector on the system |
| $\boldsymbol{\tau}_\theta$ | $\mathbb{R}^2$ | Applied joint torques |
| ${}^W\mathbf{x}_{ee}$ | $\mathbb{R}^3$ | Position of end-effector in world frame |
| ${}^W\mathbf{p}_{ee}$ | $SE(3)$ | Pose of end-effector in world frame |
| ${}^W\mathbf{p}_{ee}^{\text{des}}$ | $SE(3)$ | Desired pose of end-effector in world frame |
| $s_t$ | $\mathcal{S}$ | Observed system state at time $t$ |
| $\mathbf{u}_t$ | $\mathcal{A} \subseteq \mathbb{R}^n$ | Control action at time $t$ |
| $\pi_\phi$ | $-$ | Learned policy mapping states to actions, with parameters $\phi$ |
| | **Section 5.2: Effect of actuation on control** | |
| $F_1$ | $\mathbb{R}^{3\times n_u}$ | Force allocation matrix |
| $F_2$ | $\mathbb{R}^{3\times n_u}$ | Moment allocation matrix |
| $F$ | $\mathbb{R}^{6\times n_u}$ | Wrench allocation matrix |
| $\mathbf{f}$ | $\mathbb{R}^3$ | Force generated by fully actuated system |
| $\boldsymbol{\tau}$ | $\mathbb{R}^3$ | Torques generated by fully actuated system |
| $n_u$ | $\mathbb{N}$ | Number of control inputs |
| $\mathcal{W}$ | $\mathbb{R}^6$ | Quadrotor wrench space |
| $\text{rank}\{F_1\}$ | $1$ | Rank of force matrix |
| $\text{rank}\{F_2\}$ | $3$ | Rank of moment matrix |
| $\text{rank}\{F\}$ | $\geq 4$ | Rank of wrench matrix |
| $u$ | $\mathbb{R}^{n_u}$ | Control input vector |
| $m$ | $\mathbb{R}^3$ | Moment vector |
| $\forall\, u \in U$ | $U \subset \mathbb{R}^{n_u}$ | Valid control inputs |
| | **Section 5.3: Reinforcement Learning for Control** | |
| $\pi$ | $-$ | Control policy (agent) |

Continued on next page

| **Symbol** | **Space** | **Definition** |
|---|---|---|
| $a_t$ | $A$ | Action at time $t$ |
| $s_t$ | $S$ | State at time $t$ |
| $r_t$ | $\mathbb{R}$ | Reward at time $t$ |
| $R(s_t, a_t)$ | $\mathbb{R}$ | Reward function |
| $R(\tau)$ | $\mathbb{R}$ | Return over trajectory |
| $\gamma$ | $[0,1]$ | Discount factor |
| $\tau$ | $-$ | Trajectory of states and actions |
| $P(\tau \mid \pi)$ | $[0,1]$ | Trajectory probability under policy |
| $\rho_0(s_0)$ | $[0,1]$ | Initial state distribution |
| $J(\pi)$ | $\mathbb{R}$ | Expected return under policy |
| $\pi^*$ | $-$ | Optimal policy |
| $\mathbb{E}_{\tau\sim\pi}[\cdot]$ | $-$ | Expectation over trajectories |
| $r_t(\theta)$ | $\mathbb{R}$ | Probability ratio (new vs old policy) |
| $\theta, \theta_{\text{old}}$ | $-$ | Policy parameters |
| $\hat{A}_t$ | $\mathbb{R}$ | Advantage estimate |
| $L^{\text{CLIP}}(\theta)$ | $\mathbb{R}$ | PPO clipped surrogate loss |
| $V_\phi(s_t)$ | $\mathbb{R}$ | Value function (critic output) |
| $L^{\text{VF}}(\phi)$ | $\mathbb{R}$ | Critic loss |
| $R_t$ | $\mathbb{R}$ | Return estimate (e.g., from GAE) |
| $L(\theta, \phi)$ | $\mathbb{R}$ | PPO total loss |
| $\mathcal{H}(\pi_\theta(\cdot \mid s_t))$ | $\mathbb{R}$ | Policy entropy |
| $c_1, c_2$ | $\mathbb{R}$ | Loss balancing coefficients |
| | | **Section 6.1: Control Architecture** |
| $\boldsymbol{\theta}$ | $\mathbb{T}^2$ | Manipulator joint angles $(\theta_1, \theta_2)$ |
| $\boldsymbol{\theta}_r$ | $\mathbb{T}^2$ | Reference joint angles vector |
| ${}^W\mathbf{a}_b$ | $\mathbb{R}^3$ | Policy linear-acceleration command (world frame) |
| ${}^W\psi_r$ | $\mathbb{R}$ | Desired yaw reference (world frame) |
| ${}^B\boldsymbol{\Omega}_b$ | $\mathbb{R}^3$ | Policy body-rate reference (rad/s) |
| ${}^B\boldsymbol{\Omega}_b^{\text{cmd}}$ | $\mathbb{R}^3$ | Commanded body angular velocity (tilt-prioritized controller) |
| ${}^B\boldsymbol{\alpha}_d$ | $\mathbb{R}^3$ | Desired body angular acceleration |
| $q_d$ | $\mathbb{S}^3$ | Desired attitude quaternion of quadrotor |
| ${}^B\mathbf{z}_d$ | $\mathbb{R}^3$ | Desired body $z$-axis (unit vector) |
| ${}^C\mathbf{x}_d$ | $\mathbb{R}^3$ | Desired $x$-axis in intermediate $C$ frame (unit vector) |
| ${}^B\mathbf{y}_d$ | $\mathbb{R}^3$ | Desired body $y$-axis (unit vector) |
| ${}^B\mathbf{x}_d$ | $\mathbb{R}^3$ | Desired body $x$-axis (unit vector) |
| $R(q_d)$ | $SO(3)$ | Rotation matrix corresponding to $q_d$ |
| $\mathbf{K}_\Omega$ | $\mathbb{R}^{3\times3}$ | Angular-velocity feedback gain matrix |
| $\boldsymbol{\omega}_{cmd}$ | $\mathbb{R}^3$ | Commanded rotor angular velocities |
| $\boldsymbol{\omega}_f$ | $\mathbb{R}^3$ | Filtered rotor angular velocities |
| $f$ | $\mathbb{R}$ | Total thrust |
| $\boldsymbol{\tau}$ | $\mathbb{R}^3$ | Total body torque vector |
| $\boldsymbol{\tau}_{\text{f}}$ | $\mathbb{R}^3$ | Filtered torque vector |
| $\mathbf{J}$ | $\mathbb{R}^{3\times3}$ | Inertia matrix of quadrotor base |
| $\mathcal{W}$ | $\mathbb{R}^4$ | Wrench vector (total thrust and torques) |
| $\tau_x, \tau_y, \tau_z$ | $\mathbb{R}$ | Torques about body $x$, $y$, $z$ axes |
| $\tau_{\theta_1}, \tau_{\theta_2}$ | $\mathbb{R}$ | Torque generated for manipulator joint 1 and 2 |
| | | **Section 6.2: Training** |
| ${}^W\mathbf{a}_{\text{b}}$ | $\mathbb{R}^3$ | Linear acceleration of the body in world frame |

Continued on next page

| **Symbol** | **Space** | **Definition** |
|---|---|---|
| ${}^{B}\mathbf{v}_{\text{b}}$ | $\mathbb{R}^3$ | Linear velocity of the quadrotor base in body frame |
| ${}^{B}\mathbf{\Omega}_{\text{b}}$ | $\mathbb{R}^3$ | Angular velocity of the quadrotor base in body frame |
| ${}^{W}\psi_r$ | $\mathbb{R}$ | Reference heading in world frame |
| ${}^{W}\mathbf{R}_{\text{b}}$ | $SO(3)$ | Rotation matrix of the quadrotor base w.r.t. world frame |
| $\boldsymbol{\theta}_r$ | $\mathbb{T}^2$ | Reference vector of $2$ revolute joint angles of the manipulator |
| ${}^{B}\mathbf{x}_{\text{goal}}$ | $\mathbb{R}^3$ | Goal position expressed in quadrotor base frame |
| ${}^{EE}\mathbf{x}_{\text{goal}}$ | $\mathbb{R}^3$ | Goal position expressed in end-effector frame |
| ${}^{EE}\mathbf{R}_{\text{goal}}$ | $SO(3)$ | Goal orientation in end-effector frame |
| ${}^{EE}\mathbf{R}_{\text{goal}}[:, 0\!:\!2]$ | $\mathbb{R}^6$ | 6D continuous rotation representation from the first two columns of ${}^{EE}\mathbf{R}_{\text{goal}}$ |
| ${}^{W}\mathbf{x}_{\text{ee}}$ | $\mathbb{R}^3$ | End-effector position in world frame |
| ${}^{W}\mathbf{q}_{\text{ee}}$ | $S^3$ | Unit quaternion representing end-effector orientation in world frame |
| ${}^{W}\mathbf{q}_{\text{goal}}$ | $S^3$ | Unit quaternion representing goal orientation in world frame |
| $\mathcal{E}_{ori}({}^{W}\mathbf{q}_{\text{ee}}, {}^{W}\mathbf{q}_{\text{goal}})$ | $\mathbb{R}$ | Minimum geodesic angular distance between two unit quaternions in $SO(3)$ |
| ${}^{\text{drone}}\mathbf{a}_t$ | $\mathbb{R}^7$ | Drone action vector at time $t$: $\{{}^{W}\mathbf{a}_{\text{b}}, {}^{B}\boldsymbol{\omega}_{\text{b}}, {}^{W}\psi_r\}$ |
| ${}^{\text{joint}}\mathbf{a}_t$ | $\mathbb{R}^2$ | Joint position action vector at time $t$: $\{\boldsymbol{\theta}_r\}$ |

# 1 Acknowledgments

This thesis marks the culmination of my Masters at TU Delft, made possible with the support of several individuals, to whom I am grateful.

First and foremost, I would like to thank my supervisor Dr. Sihao Sun for his exceptional guidance and trust. Over the past nine months, I have learned a great deal about engineering, research and patience, and I sincerely appreciate the support and involvement. I would also like to convey sincere thanks to my main supervisor, Prof. Javier Alonso-Mora, for his guidance and honest feedback on the work.

Special thanks go to Maurits and Kseniia, for building the drones used in this thesis and for their prompt assistance after crashes.

On a personal note, I am grateful to my parents for their support, patience and inspiration, and Aditi for being a bundle of energy and the best sibling one could ask for.

I would like to express my deepest gratitude to friends Anurag, Hrishi, Tejas, Tarini and Srinivas for the company and encouragement throughout this journey. You have contributed in ways you may not realize. To my friends from home, Abhay, Jyoti, Manju, Shardul and many others, thank you for always cheering me on. Jack and Ivan, thank you for the company and for being a constant source of optimism. Dave and David, I'm grateful for the good times we shared.

Special thanks to Karthik for valuable feedback on my draft and Anmol for assisting in editing figures.

Finally, I'm grateful to all my peers from the MSc Robotics program who made the past two years rewarding, challenging and fun.

*Shlok Deshmukh*
*Delft, August 2025*

# 2 Abstract

Aerial manipulation control is challenging due to the inherently coupled and highly variable dynamics involved in simultaneously controlling both the drone platform and its manipulator. Traditional model-based methods have been widely used but are limited by their dependence on accurate dynamics models, high computational cost, and sensitivity to external disturbances. Recent studies have explored reinforcement learning (RL)-based methods to overcome these limitations. However, most of them focus on fully actuated drones or rigid-link manipulators, thereby avoiding the complexity of underactuated platforms and manipulator control. This work develops a robust RL-based controller framework capable of whole-body control of an underactuated aerial manipulator with a two-degree-of-freedom arm. The method is evaluated in both simulation and real-world experiments on three tasks: end-effector pose control, payload carrying, and path following. We achieve 6-DoF end-effector pose tracking with errors of **5.3 cm and 8.8°**, with inference times of **0.18 ms**, and demonstrate payload carrying capacity of up to **140 g** through real-world experiments. Through ablations we show that action smoothing and domain randomization are critical for reliable transfer. The results demonstrate the feasibility of using RL-based whole-body control for aerial manipulation and lay the foundation for more complex contact-based manipulation tasks. To the best of our knowledge, this is among the first demonstration of RL-based **whole-body** control of an **underactuated** aerial manipulator validated through real-world experiments.

# 3 Introduction

## 3.1. Background

Aerial manipulation is a growing field that combines the dexterity and precision of robotic arms with mobility of aerial vehicles, such as quadrotors. This integration enables capabilities to perform tasks while flying, such as applying forces to a wall, pushing, sliding and grasping objects [30], [1], [28]. With these capabilities, aerial manipulators can be used for a wide array of practical applications, such as transportation of material, remote operations, disaster response and industrial inspection, tasks that are often challenging and dangerous to humans.

At high altitude and in constrained workspaces aerial manipulators can be used to quickly and efficiently transport material, reducing operational costs and risk. In disaster-stricken areas, they can be used for transporting relief to regions inaccessible by traditional means of transport. In industrial settings, aerial manipulators can perform tasks such as installing sensor devices, inspection by direct contact with tanks, pipes, and other surfaces. Across these diverse applications, a common advantage emerges - aerial manipulators can significantly mitigate risk and cost associated with human workers operating in hazardous situations and time-critical scenarios.

While these capabilities seem attractive, deploying aerial manipulators in the real-world introduces significant challenges, primarily due to the complexity of control involved. This complexity arises from highly nonlinear and coupled dynamics between the flying base and arm. The robotic arm's motion generates forces and torques that can affect stability of the drone, if not compensated. For example, raising a robotic arm to a horizontal position shifts the center of gravity away from the drone's center, requiring rotor thrusts to be adjusted to maintain stability. Additionally, tasks involving interaction with the environment, such as pushing against a surface, maintaining contact and grasping create external disturbances, further complicating control.

## 3.2. Motivation

Thus, to design an effective controller for aerial manipulation, we consider the following requirements:

1. Ability to handle complex and coupled dynamics between arm and base.
2. Robustness and adaptation to external disturbances.
3. Real-time performance.

Existing studies have managed to implement tasks such as holding position and applying forces [13], opening doors [6] and pick-and-place [15] with model-based control methods. Typically, they are structured by separating trajectory planning and low-level control. However, these methods depend on a highly accurate model of system dynamics that might be unreliable and infeasible to obtain, especially in tasks involving interaction with environment [9] (further discussed in Chapter 4). Contact-based tasks introduce discontinuities, unmodeled disturbances and forces that are difficult to model and predict analytically. Moreover, some model-based control methods, such as MPPI (Model Predictive Path Integral)

[45], often require computationally expensive online optimizations, limiting use in real-time scenarios [7].

Using reinforcement learning for control shows promise as an alternative, eliminating the need for **accurate** dynamics modeling with added benefit of fast inference times enabling reactive decision making for complex tasks, as witnessed in drone control [21]. In aerial manipulation, limited studies have managed to show effectiveness of reinforcement learning to achieve contact-based tasks. Examples include, controlling a fully actuated vehicle with a rigid link to open and close doors [7] and an underactuated platform with 2-DoF planar manipulator [25] to perform pick and place, with a **decoupled** control approach.

However, RL for aerial manipulation shows positive results on simple tasks by focusing only on fully-actuated drones or rigid-link manipulators, thereby avoiding complexity of underactuated platforms and manipulator control, leaving **whole-body control of an underactuated aerial manipulator** largely unexplored. **Whole-body** control refers to using a unified algorithm to control a mobile manipulator with a locomotion and manipulation system. In aerial manipulators, this refers to control of both the quadrotor base and arm[1] using a single algorithm, necessary for tasks involving coordination between both systems and compensation for disturbances due to interaction within and outside the aerial manipulator. **Underactuated** refers to lack of independent control over all degrees of freedom on a quadrotor. We specifically look into underactuated drones due to lower hardware costs compared to fully actuated drones, making them attractive for practical deployment. However, they introduce additional control challenges (discussed in Section 5.2).

Before performing tasks such as grasping, applying forces or tool use, an aerial manipulator should be able to accurately control the pose of its end-effector [2] in three dimensional space. This is a fundamental prerequisite to contact-based manipulation, a task this thesis attempts to achieve with RL.

## 3.3. Research question

In particular, we aim to answer - *How can we develop **whole-body** control policies using reinforcement learning to achieve **end-effector pose control** of an **underactuated** aerial manipulator in the real-world?*

From a scientific perspective, this research is significant as it contributes to the growing field of learning-based robotic control, while addressing critical sim-to-real challenges necessary for real-world deployment. Successfully performing end-effector pose control using Reinforcement Learning serves as a foundational capability for enabling complex aerial manipulation tasks in the future, such as grasping, applying forces and other types of interaction with the environment.

## 3.4. Contributions

We develop a control policy capable of whole-body control of an underactuated aerial manipulator to perform end-effector pose control. We demonstrate the policy's effectiveness by extensively testing in simulation and validating with real-world experiments.

1. We demonstrate that RL-based control policies can be successfully trained using PPO (Proximal Policy Optimization), achieving **6-DoF end-effector pose control** with errors of **5.3 cm (position)** and **8.8° (orientation)** in real-world experiments (see A.0.6 for videos).
2. We **introduce an end-to-end pipeline for training and deploying** a reinforcement learning policy onboard a aerial manipulator with sub-millisecond inference times (**0.18 ms**).
3. Through ablations, we provide recommendations for action and observation space design, controller abstractions and show that domain randomization and action smoothing are critical for transfer.
4. Additionally, we demonstrate **robustness to external disturbances and generalization** by performing pose control while carrying payloads up to **140 g** in the real world and path following while maintaining constant orientation.

[1]We use the term "arm" and "manipulator" interchangeably.
[2]In our context, the end-effector is the gripper attached to the arm or manipulator.

# 4 Related work

In this chapter, we review literature regarding aerial manipulator designs, discussing platform and manipulator design separately. Then, we review the state-of-the-art methods to control aerial manipulators, including popular model-based and reinforcement learning-based methods. Finally, we present our motivation to use RL based control and highlight key aspects that differentiate our work from existing methods.

## 4.1. Aerial Manipulator Design

We define an aerial manipulator as an aerial robot, typically a drone equipped with a robotic manipulator, capable of performing manipulation tasks while flying. Such a system comes with several challenges, including complexity of controlling both the platform and manipulator simultaneously, need for precise torque and force application, and varying dynamic characteristics inherent to particular tasks.

Appropriate design of aerial manipulators is crucial because it directly impacts controllability and ability to perform a range of tasks. Different tasks such as pick and place, sliding or peg in hole need different designs. This section reviews aerial manipulator designs by separating platform (drone base) and manipulator design.

It helps to classify aerial manipulator design based on the drone platform's ability to generate forces and torques (admissible wrenches) directly affecting controllability.

### 4.1.1. Platform design based on admissible wrenches

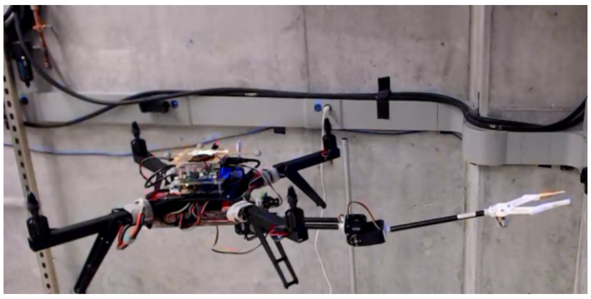


**Figure 4.1:** Underactuated drone platform (UDT) [15]

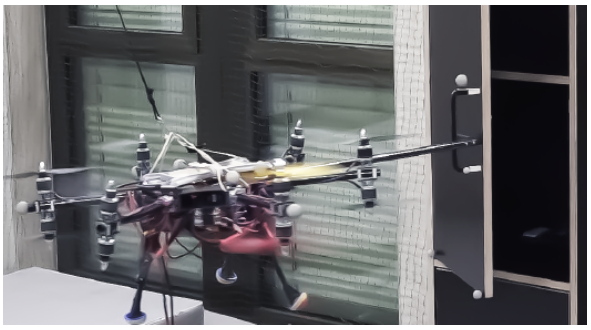


**Figure 4.2:** Fully actuated drone platform - with 6 independently tiltable rotors [7]

Platforms can be classified based on admissible wrenches [17], [30] which affects controllability - whether the platform can control its six degrees of freedom independently or in a coupled manner.

1. **Uni-Directional thrust (UDT) or Underactuated (UA)** — Platforms that can generate thrust along only one direction. This property makes control of thrust independent of total moment

impossible in more than one direction. A typical quadrotor with rigidly fixed rotors is a UDT system or underactuated system (see 4.1).

2. **Multi-Directional thrust (MDT)** — Platforms that can vary thrust in more than one direction independent of total moment.
3. **Fully Actuated (FA)** — A subclass of MDT platforms that can vary thrust in all directions independent of total moment. Examples of such platforms are a hexrotor with fully tiltable rotors (see 4.2).
4. **Overactuated (OA)** — Multirotor platforms that have more actuation inputs than degrees of freedom, meaning an FA has more than one input combination for every desired wrench.

### 4.1.2. Manipulator design

Aerial manipulators use various robotic arm designs to complete different tasks. They vary in morphology, kinematic configuration and mechanical construction [30].

Few common types of robotic arms / manipulators used include -

1. Rigid-link arms - Often used for point contact and sliding tasks [47], [46].
2. Grippers - Usually used for pick and place operations [28].
3. Passive links, cables in particular for transportation, point contact and manipulation [40], [3].
4. Articulated arms - can be used for a wide range of tasks such as pick and place [15], pushing/pulling, sliding and manipulation [34].

To reliably perform aerial manipulation tasks, effective control strategies need to be assessed. In the next section, we review existing control methods used for aerial manipulation tasks, starting with widely adopted model-based control and concluding with reinforcement learning-based methods.

## 4.2. Aerial Manipulator Control

Control of an aerial manipulator can be challenging due to the need to simultaneously manage the coupled dynamics of the drone and manipulator. Some studies use handcrafted trajectories based on heuristics to perform tasks like door opening in [43] or valve turning in [23]. Such methods become infeasible in contact rich tasks.

Beyond handcrafted methods, control of an aerial manipulator can be compared along two key aspects - **(1) whole-body vs decoupled control**, which differs in how platform-manipulator dynamics are handled, and **(2) model-based vs model-free control**, which differs in whether an explicit dynamics model is required.

In the first aspect, coupled (whole-body) control [15], [24], [33] which considers full system dynamics and decoupled control [49], [25], treats forces from the environment or manipulator as external disturbances to be compensated by the quadrotor. Coupled or whole-body control is suited for tasks requiring precise coordination between drone and manipulator, at a higher computation cost. While relatively simpler to implement, decoupled control might struggle with tasks involving simultaneous movement of both platform and manipulator.

In the second aspect, model-based control methods solve for optimal control inputs using a model of system dynamics, through predictive techniques such as MPC (Model Predictive Control). MPC uses a model to predict system's behavior over a finite horizon, solve an optimization problem at every time step and applies the first control input. Despite it's success in real world applications, requiring a differentiable model and cost function is a limitation in environments with uncertainty. Sampling based methods such as MPPI (Model Predictive Path Integral) shine here, by not requiring a differentiable model as it relies on sampling and cost evaluation of control inputs, performing better in uncertainty. These techniques have achieved high performance in aerial manipulation tasks such as grasping, knob turning and door opening, with well-modeled environments.

In contrast, model-free approaches such as reinforcement learning, work by directly mapping states to actions during deployment, without the need of an accurate dynamics model. Limited studies have achieved reliable performance on tasks such as object pushing, door opening and pose tracking (decoupled manner) by training RL-based control policies.

### 4.2.1. Model-based approach

Optimized model-based methods of generating trajectories primarily include MPC (Model Predictive Control) and MPPI (Model Preditive Path Integral) control.

An optimal control algorithm based on Nonlinear MPC to generate reference trajectories to pick an object with a 2-DOF manipulator with a quadrotor base was introduced in [15] (Figure 4.1). They use a coupled approach to model the system to execute the operation as fast as possible. Compared to manually generated reference trajectories, NMPC trajectories take less time (15 seconds and 5 seconds) to track due to simultaneous execution and coupled modeling. Due to it's computational complexity, NMPC operates at 10Hz, limiting it's use for real time control. A low-level controller is used to track reference trajectory at high frequencies.

Whole body torque level NMPC control is explored in [26]. They compare different methods of NMPC control with a hexacopter and 2, 3 and 5 DOF manipulator. Full body dynamics (or coupled dynamics) are considered to solve the control problem, equipping the robot to perform intricate maneuvers like Eagle's catch, monkey bar and push and slide. Again however, a key drawback of this method is computational cost of solving Optimal Control Problem (OCP) online, which can be limiting for longer horizons.

A MPPI [45] based method is used in [6], for door opening and knob turning tasks on a fully actuated drone, equipped with a rigid link to manipulate. MPPI works by sampling pose trajectories and sending references to an impedance controller for tracking. It repeatedly samples varying input trajectories and simulates resultant system dynamics in a physics engine, removing the need for solving an analytical model of interaction dynamics. They also show the framework's ability to handle intermittent contacts and disturbances by re planning trajectories. However, complexity of solving MPPI problem, depends on state, input size and the speed at which system dynamics can be computed, resulting in slow optimization at times [45].

Model-based methods reduce the amount of heuristics required but they are limited by two factors - computational cost and heavy reliance on accuracy of analytical models for system dynamics, making them susceptible to unaccounted external disturbances. Deriving this model to include all possible contact forces and kinematic constraints becomes intractable for a complex system like an aerial manipulator [6].

### 4.2.2. RL for Aerial Manipulator control

Despite RL's success in complex tasks, it has seen limited adoption in aerial manipulation tasks. Existing examples include aerial transportation of suspended loads [10], push and slide tasks [9] and door opening task [7] (Figure 4.2). This section reviews studies that use Reinforcement learning for aerial manipulation control.

| Study | Platform Design | Task | Arm Design | Sim/Hardware Implementation | RL Method | Whole body (coupled control) |
|---|---|---|---|---|---|---|
| [25] | Underactuated quadrotor | Pick and Place | 2 DOF + Shoulder & Elbow Joint | Both | Decoupled DDPG | No |
| [9] | Underactuated quadrotor | Push and Slide | Fixed Arm | Simulation | Actor-Critic | Yes |
| [7] | Fully Actuated OMAV | Door Opening Task | Fixed Arm | Both | PPO | Yes |
| **Our work** | **Underactuated quadrotor** | **Pose control** | **2 DOF (roll and pitch axis)** | **Both** | **PPO** | **Yes** |

**Table 4.1:** Comparison of RL-based Aerial Manipulation Studies

In [25], the authors take a decoupled approach for tracking control of an aerial manipulator utilizing a DDPG (Deep Deterministic Policy Gradient) algorithm for arm control with a robust controller for the quadrotor to reduce computational costs. Unlike the more common choice of using PPO, this study chooses DDPG, motivated by the fact that arm trajectory tracking is a deterministic task. In this case DDPG combines the advantages of policy gradient methods in continuous action spaces and its deterministic nature aligning with the task. The RL controller is trained to send joint angle commands to the arm while minimizing disturbances induced to the drone, while the adaptive controller compensates for disturbances created by picking and dropping a load.

Aerial manipulation requires interaction with environment, whose dynamic properties are usually unknown that effect stability of control. In [9], authors introduce a learning pipeline that allows the UAV to reliably position objects by pushing them using a non-prehensile aerial manipulator, without explicit modeling of their dynamics. The approach involves using DreamerV3 [16] to learn the world model of the environment and simultaneously learning a policy for interaction. The method is evaluated on pushing tasks in simulated environments with varying values of friction coefficient, showing the models ability to implicity understand contact dynamics while learning repeatable behaviors. However, the results are limited to a simulation.

RL to perform a door opening task is explored in [7]. The authors train a reactive motion control policy in simulation and successfully transfer it to an OMAV (Omni directional aerial vehicle) in real world. The policy learns in a physics simulator and outputs desired pose references to a low-level controller to be tracked. The study compares their RL based approach against the state-of-the-art MPPI controller showing considerably more robustness to errors in the environment observations, changes to initial conditions and even faster task completion. This highlights the potential of RL to train robust controllers without requiring accurate dynamics models or expensive online optimizations. However, this study uses a fully actuated drone with a rigid link, limiting manipulation degrees of freedom and trading control complexity with higher hardware costs of using a fully actuated platform.

## 4.3. A case for Reinforcement learning

| Aspect | MPC | MPPI | RL |
|---|---|---|---|
| Model Dependency | Requires accurate dynamics model | Requires model (allows non differentiable) | Model-free (for inference) |
| Computational Load | Moderate (real-time feasible) | High (GPU-dependent) | Low inference, high training |
| Flexibility | Fixed cost functions | Arbitrary costs/constraints | Multi-task generalization |

**Table 4.2:** Comparison of MPC, MPPI, and RL control approaches.

Optimal control and RL address the same problem, finding an optimal policy that optimizes an objective function, given the states, feasible controls and a model that describes the transition between states. However, optimal control requires a perfect transition model (that can predict next state, given current state and action) and provides strong guarantees, but breaks down in case of model and computational approximations, likely to happen when aerial manipulators start interacting with the environment. Whereas, reinforcement learning operates directly on the measured data and rewards generated while learning, eliminating the need for a perfect transition model [22].

Model-based methods discussed [6], [15] use their respective methods as trajectory planners that run at significantly lower frequencies than their low-level controllers used for tracking. Partly, this is due to computational cost associated with solving for next state in real time, especially for systems with complex dynamics and multiple degrees of freedom. On the other hand, model-free methods such as Reinforcement learning are well suited for contact rich tasks (see 4.2), eliminating the need of an accurate dynamics model and offering faster inference time, allowing real-time decision making.

Furthermore, RL directly optimizes for task level objectives, eliminating need for intermediate representations such as a reference trajectory and ability to perform broader range of adaptive responses [39], by leveraging domain randomization. On the other hand, model-based approaches decompose a problem into planning (generate reference trajectories) and control (trajectory tracking) steps, potentially limiting range of behaviors. This can be limiting when considering control for a task that involves both navigating and task execution, such as aerial manipulation [9].

Previously, RL-based controllers have been shown to perform well in robot control tasks such as in hand manipulation [2], legged robot control [19] and quadrotor control [18], [39]. Moreover, they are capable of behaviors that meet our previously defined requirements (see 3.2) for aerial manipulator controllers, making RL-based control for aerial manipulation worth exploring.

### 4.3.1. Our work

Prior work using RL for aerial manipulator control either adopted a decoupled control method or relied on a fully actuated platform, significantly reducing complexity of control. As summarized in Table 4.1, our work addresses a gap in literature by attempting to perform **whole-body** control for an **underactuated** aerial manipulator equipped with a two-degree-of-freedom manipulator, an approach that, to the best of our knowledge, has not been previously demonstrated.

# 5 Preliminaries

This chapter introduces theoretical foundations for the remainder of the thesis. We begin by mathematically modeling our system and defining the problem. Next, we examine and highlight the effect of underactuation on aerial manipulator control. Finally, we cover key concepts from reinforcement learning and introduce the algorithm we use in our method.

## 5.1. Modeling and Problem Definition

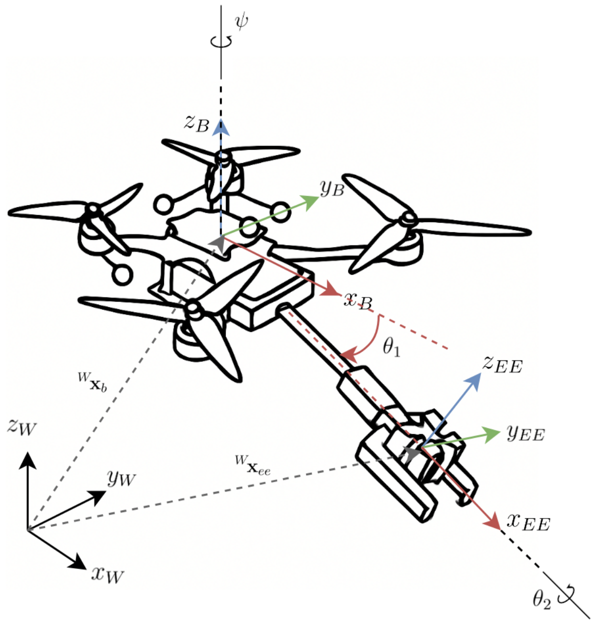


**Figure 5.1:** Schematic[32] of the underactuated aerial manipulator with 2-DoF joints used in this thesis

Our system is a quadrotor equipped with a two-degree-of-freedom manipulator arm, actuated by two revolute joints, each allowing rotation along the pitch and roll axes Figure 5.1.

Throughout this thesis we use three reference frames:

1. **World frame** $(x_W, y_W, z_W)$ - inertial frame of reference.
2. **Body frame** $(x_B, y_B, z_B)$ - located at the drone platform's center of mass.
3. **End-effector frame** $(x_{EE}, y_{EE}, z_{EE})$ - located at gripper's center of mass.

Modeling each link and arm as a rigid body (similar to [44]), we define the configuration and velocity of the system as

$$q = \begin{Bmatrix} {}^W\mathbf{x}_b \\ {}^W\mathbf{R}_b \\ \boldsymbol{\theta} \end{Bmatrix} \in \mathbb{R}^3 \times SO(3) \times \mathbb{T}^2, \quad \mathbf{v} = \begin{bmatrix} {}^W\dot{\mathbf{x}}_b \\ {}^B\boldsymbol{\Omega}_b \\ \dot{\boldsymbol{\theta}} \end{bmatrix} \in \mathbb{R}^8. \tag{5.1}$$

The system evolves according to coupled nonlinear dynamics given by generalized Lagrangian method presented in [12]

$$\mathbf{M}(q)\dot{\mathbf{v}} + \mathbf{C}(q, \mathbf{v})\mathbf{v} + \mathbf{G}(q) = \mathbf{B}(q)\mathbf{u}, \tag{5.2}$$

$$\mathbf{B}(q)\mathbf{u} = \begin{bmatrix} \mathcal{W} \\ \boldsymbol{\tau}_\theta \end{bmatrix}, \quad \mathcal{W} = \begin{bmatrix} f \\ \boldsymbol{\tau} \end{bmatrix}, \tag{5.3}$$

where $\mathbf{B}(q)\mathbf{u}$ represents the generalized forces due to control inputs. $\mathbf{B}(q)$ maps control inputs $\mathbf{u}_t = [\mathbf{u}_{\text{rotor}}, \boldsymbol{\tau}_\theta]$, to generalized forces applied to the system. $\mathcal{W} \in \mathbb{R}^4$ is the total wrench applied to the quadrotor base, $f$ is the net rotor thrust, $\boldsymbol{\tau}$ is the net moment on the quadrotor base and $\boldsymbol{\tau}_\theta$ are generated joint torques.

The generated wrench is given by the quadrotor dynamics model:

$$\mathcal{W} = G_1\mathbf{u}_{\text{rotor}} + G_2\dot{\boldsymbol{\omega}} + G_3(\boldsymbol{\Omega}_b)\,\boldsymbol{\omega}, \tag{5.4}$$

where $\mathbf{u}_{\text{rotor}}$ are rotor inputs (rotor speeds or thrust commands), $\boldsymbol{\omega} \in \mathbb{R}^4$ are rotor angular velocities and and $G_1, G_2, G_3$ are allocation matrices determined by the quadrotor's geometry.

Our objective is to control the end-effector pose of the system, defined by the forward kinematics:

$${}^W\mathbf{x}_{ee} = f_{\mathbb{R}^3}(q), \qquad {}^W\mathbf{x}_{ee} \in \mathbb{R}^3, \qquad \text{(position of the end-effector in world frame)}, \tag{5.5}$$

$${}^W\mathbf{p}_{ee} = f_{SE(3)}(q), \qquad {}^W\mathbf{p}_{ee} \in SE(3), \qquad \text{(pose of the end-effector in world frame)}. \tag{5.6}$$

We formulate the control task as a reinforcement learning problem, where a policy $\pi_\phi$ is learned that maps observed system states $s_t \in \mathcal{S}$ to generate control inputs $\mathbf{u}_t = [\mathbf{u}_{\text{rotor},t}, \boldsymbol{\tau}_{\theta,t}] \in \mathcal{A} \subseteq \mathbb{R}^6$:

$$\pi_\phi : \mathcal{S} \to \mathcal{A}, \quad \mathbf{u}_t = \pi_\phi(s_t). \tag{5.7}$$

The control inputs $\mathbf{u}_t$ directly influence the system dynamics by controlling the wrench ($\mathcal{W}$) and joint torques ($\boldsymbol{\tau}_\theta$). The learned policy aims to minimize the tracking error between the actual and desired end-effector poses ${}^W\mathbf{p}_{ee}(t)$ and ${}^W\mathbf{p}_{ee}^{\text{des}}(t)$, in the presence of unknown external wrench acting on the end effector.

## 5.2. Effect of actuation on control

For a system to be fully actuated in $SE(3)$, it must be able to control all directions of wrench

$$\mathcal{W} = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix} \in \mathbb{R}^6, \tag{5.8}$$

where wrench $\mathcal{W}$, refers to the forces and torques the system can generate.

In the case of a fully actuated drone, the platform can generate wrenches in all six dimensions independently, enabling direct and independent control of the drone's position and orientation. Subsequently, it simplifies the control problem as planners or policies have simpler dynamics to optimize for. In a fully actuated drone:

$$\text{rank}\{F\} = 6 \implies \dim(\mathcal{W}) = 6, \tag{5.9}$$

where $F$ is the control allocation matrix mapping actuator inputs to body wrench $\mathcal{W}$.

In the case of underactuated drones, the platform cannot generate wrenches in all six dimensions independently:

$$\text{rank}\{F\} = 4 \implies \dim(\mathcal{W}) = 4. \tag{5.10}$$

For instance, quadrotors cannot span the full wrench space $\mathbb{R}^3$, as thrust can be controlled only along $z_B$ and torques about $x_B, y_B, z_B$, coupling the force and torque generation. Specifically during lateral movement, requires the drone to tilt, i.e change its roll or pitch, making it a underactuated system.

Coupling between force and torque generation introduces non-linear dynamics and further complicates control when a manipulator arm is attached, introducing

1. Additional mass and inertia.
2. Internal forces and torque, challenging platform stability.
3. Arm movement dynamics requiring compensation by the quadrotor base.

These effects make precise control of an underactuated aerial manipulator significantly challenging. For instance, in a horizontal force application task, a fully actuated drone (tiltable hex rotor) can maintain orientation and apply a horizontal wrench by tilting few rotors. However, in an underactuated quadrotor, the drone will have to pitch or roll to move horizontally, requiring compensatory arm movement to maintain end-effector pose.

## 5.3. Reinforcement Learning for control

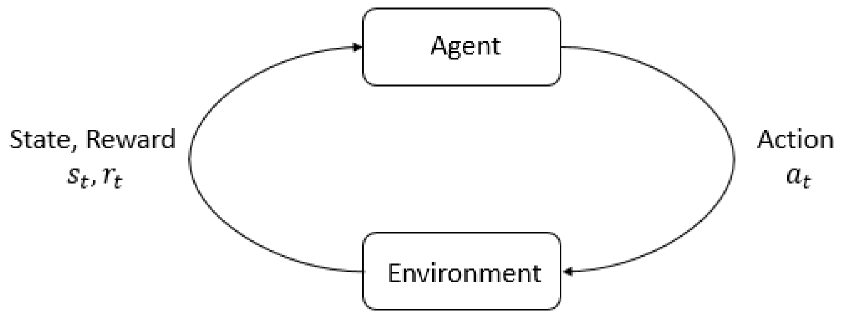


**Figure 5.2:** Agent-Environment interaction loop in Reinforcement learning [31]

In contrast to supervised learning where input features and appropriate labels are provided, in RL, both data collection and learning steps occur during training. The collected data includes tuples of states, actions and rewards, with the agent being tasked to learn relationships between them.

The objective in RL is to train an agent $\pi$ (used as a control policy) that provides appropriate actions $a \in A$ to achieve a certain goal, given the states $s \in S$, such that

$$a_t = \pi(s_t) \tag{5.11}$$

where $a_t$ is the action taken at time $t$.

For example, correct rotor thrust values (actions) to reach a goal, given the drone's position (state). This is done by training an agent or policy ($\pi$), typically in simulations.

The agent's learning is guided by rewards $r_t$, which indicate quality of actions and usually depend on current state or state-action pair, and are given by

$$r_t = R(s_t, a_t), \tag{5.12}$$

were $R$ is the reward function.

Reward functions ($R$) are designed to incentivize agents to achieve desired behaviors. For instance, negative distance to goal as reward can incentivize agent to generate actions to move towards the goal.

A common learning objective is to maximize an infinite horizon discounted return, which is the sum of all rewards obtained by the agent discounted by how far off they're obtained in the future [31], and is given by

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t, \quad \gamma \in [0,1], \tag{5.13}$$

where $\gamma$ is the discount factor.

The RL problem can be formulated as selecting a policy that maximizes expected return. Assuming stochastic environment transitions and policy, the probability of a $T$ step trajectory $\tau = (s_0, a_0, s_1, a_1, ...s_T, a_T)$ can be written as,

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \tag{5.14}$$

Then expected return can be written as,

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)]. \tag{5.15}$$

The Optimization problem RL solves can be given by,

$$\pi^* = \arg\max_{\pi} J(\pi). \tag{5.16}$$

| Where: | | |
|---|---|---|
| | $J(\pi)$ : | Expected return under policy $\pi$ |
| | $\pi$ : | Policy function |
| | $\pi^*$ : | Optimal policy that maximizes expected return |
| | $\tau$ : | Trajectory of states and actions |
| | $P(\tau\|\pi)$ : | Probability of trajectory under $\pi$ |
| | $\rho_0(s_0)$ : | Distribution over initial states |
| | $R(\tau)$ : | Total return of a trajectory |
| | $\mathbb{E}_{\tau\sim\pi}[\cdot]$ : | Expectation over trajectories drawn from $\pi$ |

Importantly, RL approaches rely on the Markov Decision Process (MDP) assumption. It states that future states of a system only rely on its current state and not any past states. In context of RL, the next state $s'$ and reward $r$ should only depend on the current state $s$ and action $a$ [42] [22].

### 5.3.1. Proximal Policy Optimization (PPO)

PPO (Proximal Policy Optimization) is state-of-the-art model-free, policy gradient based algorithm used for robot control. PPO does not need a model of the environment, i.e a function that predicts state transitions. It directly updates the policy (also called on-policy) based on data collected while acting using the current version of the policy.

PPO is built on a Actor-Critic network architecture. Actor is a neural network that takes states as input and generates mean and standard deviations of a gaussian distribution for actions to be sampled (continuous case). The Critic neural network, takes states as input and quantifies the value of that state, i.e the expected cumulative reward if actions are taken according to the current policy.

There are two main components to the objective used to update parameters of the policy.

1. **Clipped Surrogate Objective (Policy loss)**

   Let the probability ratio between the new and old policy be

   $$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{5.17}$$

   where $\theta_{\text{old}}$ and $\theta$ are the policy parameters before and after the update.

   Let $\hat{A}_t$ be the advantage estimate, which quantifies how much better an action is compared to the average action in that state. The PPO clipped objective is given by

   $$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{5.18}$$

   where the clip function controls how far the new policy can diverge from the old one, and $\epsilon$ determines the clipping range.

2. **Critic Loss (Value loss)**

   The critic loss minimizes the error in value estimation from the critic network:

   $$L^{\text{VF}}(\phi) = \mathbb{E}_t\left[(V_\phi(s_t) - R_t)^2\right], \tag{5.19}$$

   where $R_t$ is the target return, i.e., the expected cumulative rewards from time step $t$, usually estimated using GAE (Generalized Advantage Estimation).

The overall PPO objective combines the policy loss, value loss, and an entropy bonus (which adds randomness and encourages exploration):

$$L(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\phi) + c_2 \mathbb{E}_t\left[\mathcal{H}(\pi_\theta(\cdot|s_t))\right], \tag{5.20}$$

where $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the policy's entropy, and $c_1$ and $c_2$ are coefficients to balance the losses.

In a typical training loop, we collect rollouts using the current policy (actor network). These are a batch of tuples containing states, actions, returns and advantages from certain number (rollout) of environment interactions, across environments. The PPO loss is computed and parameters of actor and critic networks are updated using a gradient descent based optimizer. After training on a batch, the memory is cleared and agents start interacting with the updated actor network.

The simplicity of clipped objective differentiates PPO from other Actor-Critic network architectures, by limiting how far a new policy can travel. However, PPO trains a stochastic policy in an on-policy way, progressively making the policy less random, potentially causing the policy to be stuck in a local minima [31].

## 5.4. Summary

This chapter outlined the theoretical foundations for the remainder of the thesis, including the system model, problem definition, effect of underactuation for aerial manipulation, and key concepts of reinforcement learning with Proximal Policy Optimization. These formulations and definitions will be referenced in future chapters describing the control architecture, training and evaluation.

# 6 Methodology

In this chapter, we present the methodology developed to train and deploy an RL-based controller to perform end-effector pose control.

We first introduce the control architecture, which integrates the trained policy with low-level controllers. We then outline the training procedure used to develop our control policy, covering action and observation space, reward functions, command sampling strategy and domain randomization techniques.

## 6.1. Control architecture

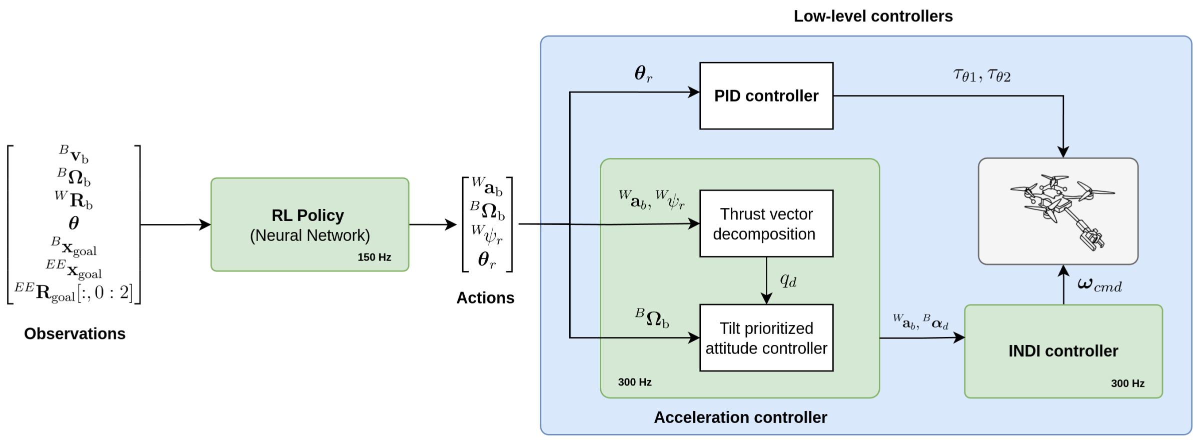

Low-level controllers
$\begin{bmatrix} {}^{B}\mathbf{v}_{\mathrm{b}} \\ {}^{B}\mathbf{\Omega}_{\mathrm{b}} \\ {}^{W}\mathbf{R}_{\mathrm{b}} \\ \boldsymbol{\theta} \\ {}^{B}\mathbf{x}_{\mathrm{goal}} \\ {}^{EE}\mathbf{x}_{\mathrm{goal}} \\ {}^{EE}\mathbf{R}_{\mathrm{goal}}[:, 0:2] \end{bmatrix}$
Observations
RL Policy
(Neural Network)
150 Hz
$\begin{bmatrix} {}^{W}\mathbf{a}_{\mathrm{b}} \\ {}^{B}\mathbf{\Omega}_{\mathrm{b}} \\ {}^{W}\psi_r \\ \boldsymbol{\theta}_r \end{bmatrix}$
Actions
$\boldsymbol{\theta}_r$
PID controller
$\tau_{\theta 1}, \tau_{\theta 2}$
${}^{W}\mathbf{a}_b, {}^{W}\psi_r$
Thrust vector decomposition
$q_d$
${}^{B}\mathbf{\Omega}_{\mathrm{b}}$
Tilt prioritized attitude controller
300 Hz
Acceleration controller
${}^{W}\mathbf{a}_b, {}^{B}\boldsymbol{\alpha}_d$
INDI controller
300 Hz
$\boldsymbol{\omega}_{cmd}$

**Figure 6.1:** Control architecture for simultaneous (whole-body) control of quadrotor base and manipulator arm using RL-based policy

In autonomous flight, a flight computer, typically referred as outer-loop controller generates higher level goals at low frequencies, such as position and attitude for the drone to achieve. A flight controller, also referred as inner-loop controller, follows high-level goals and maintains stability of the drone while operating at high frequencies.

Our policy behaves like an outer-loop controller and generates commands for inner-loop controllers to follow. However, the actions generated are not typical to a outer-loop controller used for drone control. The action space is motivated by the choice of low-level controllers that grant sufficient control authority while preserving sim-to-real transferability.

The proposed control architecture consists of two low-level controllers for the drone and one PID controller for the arm, all governed by our policy.

**RL Policy (Neural Network):**

The policy is a feedforward neural network with three hidden layers of sizes 512, 256, 128 neurons respectively. It receives observations from the environment and generates actions for subsequent controllers at 150 Hz. Mean and Variance statistics from training are used to normalize the observation vector, followed by inference using ONNX Runtime engine [8] onboard for shorter inference times. The drone base is controlled using policy generated linear accelerations, body rates and yaw reference $({}^{W}\mathbf{a}_b, {}^{B}\boldsymbol{\Omega}_{\boldsymbol{b}}, {}^{W}\psi_r)$. The manipulator arm is controlled using policy generated desired joint angle commands $(\theta_1, \theta_2)$.

Joint actions are scaled and sent to a PID controller in simulation that generates required torques, while in the phyisical system, these actions are sent to the Dynamixel actuators [35] via serial communication, which internally use a PID to generate required current (see A.2 for joint parameters used).

**Acceleration controller:**

The acceleration controller, running at 300 Hz, is a modified implementation of a DFBC controller [41, 48] to generate desired angular accelerations $({}^{B}\boldsymbol{\alpha}_d)$. Policy-generated linear accelerations $({}^{W}\mathbf{a}_b)$ and the desired yaw $({}^{W}\psi_r)$ are decomposed to generate the desired quadrotor attitude $(q_d)$. The desired body $z$-axis is given by

$$ {}^{B}\mathbf{z}_d = \frac{{}^{W}\mathbf{a}}{\|{}^{W}\mathbf{a}\|}. \tag{6.1}$$

The desired $x$-axis in the intermediate $C$-frame (new body frame after yaw) is

$$ {}^{C}\mathbf{x}_d = \begin{bmatrix} \cos {}^{W}\psi_r \\ \sin {}^{W}\psi_r \\ 0 \end{bmatrix}. \tag{6.2}$$

The desired body $y$-axis is computed as

$$ {}^{B}\mathbf{y}_d = \frac{{}^{B}\mathbf{z}_d \times {}^{C}\mathbf{x}_d}{\|{}^{B}\mathbf{z}_d \times {}^{C}\mathbf{x}_d\|}. \tag{6.3}$$

The desired body $x$-axis then follows from

$$ {}^{B}\mathbf{x}_d = {}^{B}\mathbf{y}_d \times {}^{B}\mathbf{z}_d. \tag{6.4}$$

The corresponding rotation matrix is

$$ R(q_d) = \begin{bmatrix} {}^{B}\mathbf{x}_d & {}^{B}\mathbf{y}_d & {}^{B}\mathbf{z}_d \end{bmatrix}. \tag{6.5}$$

Finally, the desired attitude quaternion is obtained from

$$ q_d = \Phi(R(q_d)), \tag{6.6}$$

where $\Phi : SO(3) \to S^3$ represents the mapping from a rotation matrix to its equivalent unit quaternion representation.

Next, the tilt-prioritized attitude controller proposed in [5] regulates the attitude error and generates the desired angular accelerations $({}^{B}\boldsymbol{\alpha}_d)$, using the policy-generated body rates $({}^{B}\boldsymbol{\Omega}_b)$ as reference:

$$ {}^{B}\boldsymbol{\Omega}_b^{\text{cmd}} = \text{tiltPrioritizedControl}(q, q_d), \tag{6.7}$$

$$ {}^{B}\boldsymbol{\alpha}_d = {}^{B}\boldsymbol{\Omega}_b^{\text{cmd}} + \mathbf{K}_\Omega \cdot \left({}^{B}\boldsymbol{\Omega}_b - {}^{B}\boldsymbol{\Omega}_b^{\text{curr}}\right). \tag{6.8}$$

Finally, the policy-generated linear accelerations $({}^{W}\mathbf{a}_b)$ and desired angular accelerations $({}^{B}\boldsymbol{\alpha}_d)$ are sent to the INDI controller.

**INDI controller:**

The Incremental Nonlinear Dynamic Inversion (INDI) controller, demonstrated in [41] runs at 300 Hz and accounts for unmodeled effects on rotational dynamics arising from aerodynamics, center-of-gravity bias, or differences between rotors, by using instantaneous sensor measurements. INDI calculates the final rotor speeds $(\boldsymbol{\omega}_{cmd})$ from the desired wrench $(\mathcal{W})$, which consists of combined thrust and torques in roll, pitch, and yaw directions.

Policy-generated linear accelerations (${}^{W}\mathbf{a}_b$) are normalized and multiplied by the system's mass to calculate the combined thrust ($f$), the first component of the wrench:

$$f = m \cdot \left\| {}^{W}\mathbf{a}_b \right\| . \tag{6.9}$$

The desired torques are computed as

$$\boldsymbol{\tau_d} = \boldsymbol{\tau}_f + \mathbf{J} \cdot \left( {}^{B}\boldsymbol{\alpha}_d - {}^{B}\dot{\boldsymbol{\Omega}}_f \right) , \tag{6.10}$$

where $\boldsymbol{\tau}_\mathrm{f}$ is the filtered body torque, $\mathbf{J}$ is the inertia matrix, ${}^{B}\boldsymbol{\alpha}_d$ is the desired angular acceleration from the acceleration controller, and ${}^{B}\dot{\boldsymbol{\Omega}}_f$ is the filtered angular acceleration measurement.

The unmodeled effects on rotational dynamics are captured by filtered angular acceleration measurement (${}^{B}\dot{\boldsymbol{\Omega}}_f$) and filtered body torque ($\boldsymbol{\tau}_\mathrm{f}$), where

$$\boldsymbol{\tau}_f = \bar{G}_1 \, \boldsymbol{\omega}_f^2 \; + \Delta t^{-1} \bar{G}_2 \left( \boldsymbol{\omega}_f - \boldsymbol{\omega}_{f,k-1} \right) \tag{6.11}$$

is computed from rotor speed measurements. ${}^{B}\dot{\boldsymbol{\Omega}}_f$ and $\boldsymbol{\omega}_f$ are measured body rates and rotor speeds obtained by applying a second order low-pass Butterworth filter with 12 Hz cut-off frequency to raw measurements, thereby synchronizing both signals. The subscript $k-1$ indicates the previously sampled value and $\Delta t$ is the sampling interval. $\bar{G}_1$ and $\bar{G}_2$, are the final three rows of the respective allocation matrices.

Then, the following equation can be used to solve for rotor speeds:

$$\begin{bmatrix} f \\ \boldsymbol{\tau}_d \end{bmatrix} = \bar{G}_1 \, \boldsymbol{\omega}_{cmd}^2 \; + \Delta t^{-1} \bar{G}_2 \left( \boldsymbol{\omega}_{cmd} - \boldsymbol{\omega}_{cmd,k-1} \right) , \tag{6.12}$$

with the only unknown $\boldsymbol{\omega}_{cmd}$ which can be determined numerically.

## 6.2. Training

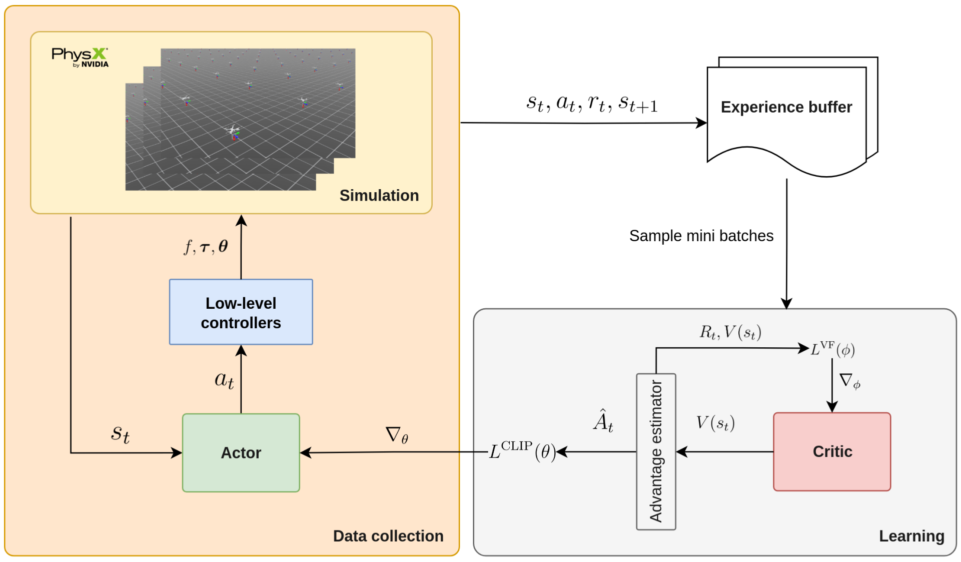


**Figure 6.2:** PPO training architecture for the aerial manipulator. **Left**: Actions sampled from the actor network are converted by low-level controllers into wrench and joint positions commands, which are applied in simulation via the physics engine. **Right**: Mini batches are sampled from memory, losses are calculated and networks are updated via backpropagation, to generate favorable actions and accurate state-value estimates.

We model the control architecture (6.1) in our training environment and use Proximal Policy Optimization to train our agent to perform end-effector pose control. The agent uses an actor-critic architecture where both networks are multi-layer perceptrons with hidden layers of size [512, 256, 128] and ELU activation functions. The actor is responsible for generating actions $(a_t)$ based on states $(s_t)$ obtained from the environment, while the critic generates value estimates [1] $(V(s_t))$.

A training iteration can be divided into data collection and learning steps. In data collection, data of batch size $B$ is collected by the actor taking $n_{steps}$ across environments, based on observed states. For learning, a single batch of these trajectories (actions, states, rewards) are split into mini batches and advantage estimates $(\hat{A}_t)$ are computed to measure how much better each action was compared to the expected value. Using this data, relevant losses $(L^{\text{CLIP}}(\theta),\ L^{\text{VF}}(\phi))$ are computed that PPO backpropogates to update the actor and critic networks. Iteratively, the actor network gets better at generating more advantageous actions to perform end-effector pose control and the critic gets better at estimating state values.

In the following sections we detail our training setup involving the training environment, observation space, action space, reward functions and techniques such as domain randomization for robust transfer.

### 6.2.1. Training environment

We use Isaac Lab for training our policy as it extensively utilizes GPUs for both simulation and learning, enabling high data throughput and faster learning. Apart from modeling the control architecture (Section 6.1) with low-level controllers, we also model filters, rotor models, physical properties of the system.

PPO offers control over multiple aspects of the algorithm, making hyperparameter selection challenging. We begin with the hyperparameters used in [36] as a starting point.

We train on 4096 environments in parallel, with a rollout $(n_{steps})$ size of 24 $(B = 4096 \times 24)$ and environment step frequency of 150 Hz. This corresponds to simulating 0.16 seconds of real time per environment, every training iteration. Our maximum episode length is 6s, after which the environment gets timed-out and reset, meaning a single episode will have multiple policy updates.

To achieve a working policy, we train for 2B environment steps $(4096 \times 500,000)$ that takes 5.5 hours on a consumer grade GPU (NVIDIA RTX 2080 Ti). For other training hyperparameters see Table A.1.

### 6.2.2. Action and Observation space

The system is controlled through an action space $a^t \in \mathbb{R}^9$ containing linear accelerations, body rates, a reference heading, and joint positions.

Policy generated linear acceleration and body rates serve as references for the low-level controllers - the acceleration controller and the INDI (Incremental Nonlinear Dynamic Inversion) controller [41], that send rotor speed commands to control the drone's movement.

Joint positions from the action vector are scaled and treated as references for a PID controller that generates torques required to actuate the arm. The action vector is given by

$$\mathbf{a}_t = \begin{bmatrix} {}^{W}\mathbf{a}_\text{b} \\ {}^{B}\boldsymbol{\Omega}_\text{b} \\ {}^{W}\psi_r \\ \boldsymbol{\theta}_r \end{bmatrix} \in \mathbb{R}^9, \tag{6.13}$$

where

$$ {}^{W}\mathbf{a}_\text{b} \in \mathbb{R}^3, \quad \text{linear acceleration of the body in the world frame,} \tag{6.14}$$

$$ {}^{B}\boldsymbol{\Omega}_\text{b} \in \mathbb{R}^3, \quad \text{angular velocity of the body in the body frame,} \tag{6.15}$$

$$ {}^{W}\psi_r \in \mathbb{R}, \quad \text{reference heading in the world frame,} \tag{6.16}$$

$$ \boldsymbol{\theta}_r \in \mathbb{T}^2, \quad \text{reference vector of two revolute joint angles.} \tag{6.17}$$

[1]Value estimates, also referred as state estimates in some literature, is the expected return when actions are taken according to the current policy.

The observation for the agent at any time step is a $29$-dimensional vector that includes linear velocity, angular velocity, orientation (as a $3 \times 3$ rotation matrix), normalized joint angles, goal position in the drone frame, and goal pose in the end-effector frame. We use goal pose information in the drone and end-effector frames instead of the world frame to make our policy invariant to global coordinates. The observation vector is given by

$$\mathbf{o}_t = \begin{bmatrix} {}^{B}\mathbf{v}_{\mathsf{b}} \\ {}^{B}\boldsymbol{\Omega}_{\mathsf{b}} \\ {}^{W}\mathbf{R}_{\mathsf{b}} \\ \boldsymbol{\theta} \\ {}^{B}\mathbf{x}_{\mathsf{goal}} \\ {}^{EE}\mathbf{x}_{\mathsf{goal}} \\ {}^{EE}\mathbf{R}_{\mathsf{goal}}[:, 0:2] \end{bmatrix} \in \mathbb{R}^{29}, \tag{6.18}$$

where

$$\begin{aligned}
{}^{B}\mathbf{v}_{\mathsf{b}} &\in \mathbb{R}^3, && \text{linear velocity of the quadrotor base in the body frame,} && (6.19)\\
{}^{B}\boldsymbol{\Omega}_{\mathsf{b}} &\in \mathbb{R}^3, && \text{angular velocity of the quadrotor base in the body frame,} && (6.20)\\
{}^{W}\mathbf{R}_{\mathsf{b}} &\in SO(3), && \text{rotation matrix of the quadrotor base w.r.t. the world frame,} && (6.21)\\
\boldsymbol{\theta} &\in \mathbb{T}^2, && \text{vector of two revolute joint angles of the manipulator,} && (6.22)\\
{}^{B}\mathbf{x}_{\mathsf{goal}} &\in \mathbb{R}^3, && \text{goal position expressed in the quadrotor base frame,} && (6.23)\\
{}^{EE}\mathbf{x}_{\mathsf{goal}} &\in \mathbb{R}^3, && \text{goal position expressed in the end-effector frame,} && (6.24)\\
{}^{EE}\mathbf{R}_{\mathsf{goal}} &\in SO(3), && \text{goal orientation in the end-effector frame,} && (6.25)\\
{}^{EE}\mathbf{R}_{\mathsf{goal}}[:, 0:2] &\in \mathbb{R}^6, && \text{6D continuous rotation representation from the first two columns of } {}^{EE}\mathbf{R}_{\mathsf{goal}}. && (6.26)
\end{aligned}$$

### 6.2.3. Rewards

We use reward functions to guide learning. For the pose control task, we define two primary rewards - one for position and one for orientation, along with additional penalty terms to smooth the action space and penalize large action magnitudes. These terms reduce oscillations and help ensure stable transfer. All rewards are scaled by a negative exponential transformation that maps raw reward values between $(0, 1]$, enabling comparable weighing of individual reward components.

**Position reward** ($\mathbf{R}_{\mathsf{pos}}$): We use the L2 norm of the difference between the end-effector and goal positions as the basis for position reward:

$$\mathbf{R}_{\mathsf{pos}} = w_1 \cdot e^{-\alpha_1 \cdot \| {}^{W}\mathbf{x}_{ee} - {}^{W}\mathbf{x}_{goal} \|_2}. \tag{6.27}$$

The $\alpha_1$ value tunes reward sensitivity for local convergence. Larger values increase the gradient magnitude near the target but can lead to vanishing gradients far from the target, as the exponential saturates near zero.

**Orientation reward** ($\mathbf{R}_{\mathsf{ori}}$): We use the smallest geodesic angular difference between the orientations of the end-effector and the goal, equivalent to our evaluation metric (see Equation (7.2)). We do not penalize the drone's orientation directly, as the end-effector orientation already constrains the possible drone orientations:

$$\mathbf{R}_{\mathsf{ori}} = w_2 \cdot e^{-\alpha_2 \cdot \mathcal{E}_{ori}({}^{W}\mathbf{q}_{ee}, {}^{W}\mathbf{q}_{goal})}. \tag{6.28}$$

**Drone action smoothing reward** ($\mathbf{R}_{\mathsf{ds}}$): The norm of the difference between the drone action values ${}^{\mathsf{drone}}\mathbf{a} = \left\{ {}^{W}\mathbf{a}_b, {}^{B}\boldsymbol{\Omega}_b, {}^{W}\psi_r \right\}$ between the current and previous time steps penalizes large deviations in consecutive actions:

$$\mathbf{R}_{\mathsf{ds}} = w_3 \cdot e^{-\alpha_3 \cdot \left\| {}^{\mathsf{drone}}\mathbf{a}_{t-1} - {}^{\mathsf{drone}}\mathbf{a}_t \right\|_2^2}. \tag{6.29}$$

We found this term beneficial for reliable deployment (discussed in Chapter 8).

**Joint action smoothing reward** ($\mathbf{R}_{\text{js}}$): The absolute difference between joint position action commands (${}^{\text{joint}}\mathbf{a} = \boldsymbol{\theta}_r$) is penalized to encourage smoother joint actions, producing near-constant torque with fewer jerks that might destabilize the platform:

$$\mathbf{R}_{\text{js}} = w_4 \cdot e^{-\alpha_4 \cdot \left\| {}^{\text{joint}}\mathbf{a}_{t-1} - {}^{\text{joint}}\mathbf{a}_t \right\|_1}. \tag{6.30}$$

**Drone action magnitude reward** ($\mathbf{R}_{\text{dmag}}$): The magnitude of drone actions ${}^{\text{drone}}\mathbf{a} = \left\{ {}^W\mathbf{a}_b,\ {}^B\boldsymbol{\Omega}_b,\ {}^W\psi_r \right\}$ is penalized to minimize oscillations and overshoot while reaching the goal pose:

$$\mathbf{R}_{\text{dmag}} = w_5 \cdot e^{-\alpha_5 \cdot \left\| {}^{\text{drone}}\mathbf{a}_t \right\|_2^2}. \tag{6.31}$$

The total reward is the weighted sum of all components:

$$\mathcal{R}_t = \mathbf{R}_{\text{pos}} + \mathbf{R}_{\text{ori}} + \mathbf{R}_{\text{ds}} + \mathbf{R}_{\text{js}} + \mathbf{R}_{\text{dmag}}. \tag{6.32}$$

| Index | Weight $w_i$ | Scale $\alpha_i$ |
|---|---|---|
| 1 | 4.0 | 1.2 |
| 2 | 1.0 | 1.0 |
| 3 | 0.5 | 1.0 |
| 4 | 1.0 | 1.0 |
| 5 | 0.1 | 1.0 |

**Table 6.1:** Reward weights $w_i$ and scaling factors $\alpha_i$ used in the total reward function.

### 6.2.4. References and Resets

We sample goal pose references[2] for the end-effector (gripper) during training. A new goal pose is sampled upon environment resets, which occur when an episode terminates either due to a timeout or when the drone falls below 0.3 m.

Positions are sampled from the ranges $(-1.0, 1.0)$ m in the $X$–$Y$ directions and $(3.0, 5.0)$ m in the $Z$ direction. Orientations are sampled from $(-120°, +120°)$ in yaw and $(-90°, +90°)$ in both pitch and roll, covering the reachable workspace.

After resets, the drone base spawns at the same initial position $(0.0, 0.0, 3.0)$ m. However, the manipulator configuration is randomly sampled from $(-90°, +90°)$ for each joint, which encourages the drone to learn to track poses from arbitrary arm configurations.

### 6.2.5. Domain Randomization

After every reset, we randomly sample a payload mass to be added to the end-effector from [-15g, 120g] and scale inertia accordingly. This results in a generalized policy that is able perform pose tracking while carrying loads up to 200 g in simulation and 140 g in the real world, shown in Results 8.1.2, 9.1.2.

| Simulator / Environment | Tunable Parameter | Friction Model |
|---|---|---|
| Isaac Lab (Training Sim) | Friction Coefficient | $F_{\text{thresh}} \leq \mu \cdot F_{\text{spatial}}$ |
| Gazebo (Deployment Sim) | Static Friction Value (N·m) | $F_{\text{thresh}} =$ Static Friction Value |
| Real-world | Not directly tunable | Unknown, typically approximated |

**Table 6.2:** Comparison of friction models in simulation and the real world. In real-world systems, the exact friction model is typically unknown, motivating domain randomization during training.

The joint friction models (see table 6.2) differ between training and deployment environments resulting in joint behavior that varies with same control input (see A.0.2 for step responses). This mismatch

[2]We use the terms *references* and *commands* interchangeably, following Isaac Lab terminology for references in goal-conditioned tasks

introduces both a sim-to-sim gap and sim-to-real gap, leading to unreliable transfer of arm dynamics. To address this, we first improve our simulation fidelity by finding friction and stiffness values that result in similar step response from joint actuators across training, deployment simulators and the hardware. Once this gap is reduced, we apply domain randomization by scaling joint stiffness and friction to [0.75, 1.25] and [0.0, 1.5] times their default values, respectively.

## 6.3. Summary

In this chapter, we presented a methodology for training a policy to perform end-effector pose control of an aerial manipulator. We describe the control architecture showing how the RL policy integrates with low-level controllers. We also detailed the training setup, covering action space, observation space, reward functions, training procedures such as command sampling and domain randomization. Following chapters describe the implementation details and experimental setup used to validate our methodology.

# 7
# Experimental setup

This chapter describes the experimental setup for deploying[1] the learned policy on the aerial manipulator.

First, we present deployment setups, controller integration, modeling and real-world hardware. We then describe the three tasks chosen to evaluate the trained policy, outline the rationale for their selection and define metrics used to assess performance.

## 7.1. Deployment environments

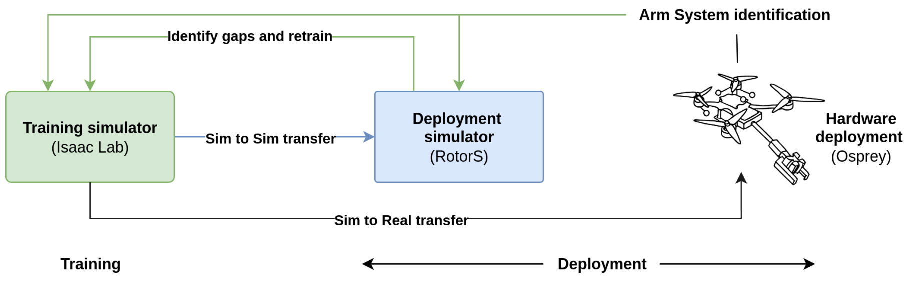

Arm System identification
Identify gaps and retrain
Training simulator (Isaac Lab)
Sim to Sim transfer
Deployment simulator (RotorS)
Hardware deployment (Osprey)
Sim to Real transfer
Training
Deployment

**Figure 7.1:** Iterative training and deployment using deployment simulator for validating policies

To deploy a functional real-world implementation, we progressively validate our solution in simulation and real-world. After we train a policy, we deploy it in a simulation that closely mimics the real drone's flight conditions, referred as the deployment simulator. Once satisfactory performance is achieved in the deployment simulator, we deploy our policy on hardware.

### 7.1.1. Deployment simulator

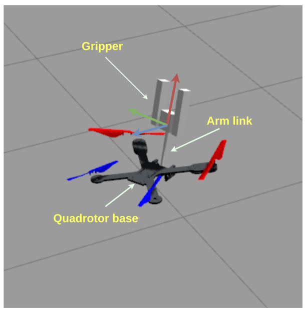


**Figure 7.2:** Transferred agent performing end-effector pose control in deployment simulator

[1] In this context, deploying or transferring refers to using the trained agent from the training simulation environment in another environment, such as deployment simulator or real-world.

We developed a deployment simulation that closely mimics the real-world flight conditions, with the same low-level controllers and filters being used in both environments. We use *rotor_simulator* (a Gazebo plugin) [14] to simulate the quadrotor flight and *ros_control* [27] for PID-based joint actuator control.

**System model:** We model the physical properties of the aerial manipulator in a URDF, using above mentioned plugins. The drone body, rotors and arm are modeled as separate links joined to the same base, seen in figure 7.2. The arm is attached to drone base with two revolute joints, controlling each degree of freedom. Lastly, a gripper consisting of three links is attached as a fixed body at the arm's end.

**Control software framework:** For control, we use the same software framework, Agilicious[11], in both real-world and deployment simulator, ensuring the simulation reliably reflects real-world deployment. The framework is modified to accommodate the reinforcement learning policy as an outer-loop controller performing inference onboard and generating commands for subsequent controllers, as shown in the control architecture (Section 6.1).

**Agent transfer:** Transferring a trained agent from Isaac Lab involves running a script that converts PyTorch model weights into ONNX format and provides mean and variance statistics for normalizing observations. The resulting model file and observation statistics are transferred to Agilicious code base, where their paths can be specified in a YAML configuration file for deployment.

### 7.1.2. Real-world setup

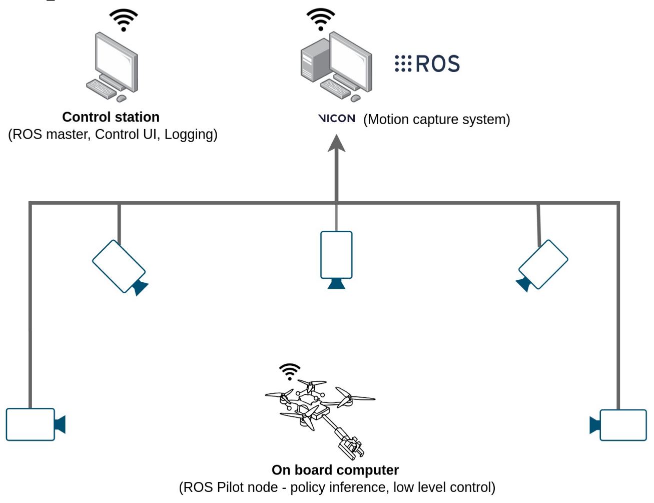

Control station
(ROS master, Control UI, Logging)
ROS
VICON (Motion capture system)
On board computer
(ROS Pilot node - policy inference, low level control)

**Figure 7.3:** Hardware communication setup with motion capture system

To validate the proposed method, the trained agent is deployed on *Osprey*, the aerial manipulator platform. Because the hardware and simulation share the same software framework, deployment requires minimal changes.

**Hardware:** The aerial manipulator weighs 860 grams, powered by a Raspberry Pi 5. The manipulator assembly weighs 200 grams, including the arm with a gripper attached to the base. Two Dynamixel [35] actuators use a differential gearbox to control both degrees of freedom of the arm.

**Policy execution:** For short inference times and near real-time performance, the entire policy deployment code is written in C++. The policy is executed onboard with ONNX Runtime Engine resulting in an average inference time of **0.18 ms**.

**Communication:** Experiments are performed in a facility (AMR Lab [4]) spanning 13 x 7 m, equipped with a VICON motion capture system that estimates states using reflective markers on the drone, with sub-millimeter accuracy. The state estimates are published on a ROS node, used by the drone as observations for the policy. A pilot ROS node is launched onboard that runs the entire control stack, including policy inference and low-level controllers. A ROS master, command center node and logger run on an external system, communicating with the drone via WiFi, which also provides a UI, allowing the user to control the drone's basic functionality and set desired end-effector poses.

**Flight procedure:** We take off with a safety controller to hover the drone at a certain height (0.5 m). Then, we switch off the safety controller, triggering the policy to take over and control the system to achieve commanded end-effector poses. For safety purposes, we define an area for the policy to operate, when crossed the safety controller is triggered, bringing the drone to hover state immediately.

Table 7.1 summarizes differences and similarities between deployment and training environments.

| Aspect | Training Sim (Isaac Lab) | Deployment Sim (Gazebo) | Deployment Hardware |
|---|---|---|---|
| Simulation Engine | Isaac Sim | Gazebo | – |
| Physics Model format | USD | URDF + Gazebo plugins | Real-world physics |
| Controllers | Acceleration, INDI, PID | Acceleration, INDI, PID | Acceleration, INDI, PID |
| Rotor Dynamics | Simulated using rotor model | rotor_simulator plugin | Actual rotors |
| Policy Inference | PyTorch (Python) | ONNX (C++) | ONNX (C++) |
| Sensing | Simulated state estimates | Simulated VICON | VICON motion capture |
| Actuation | Simulated rotors and joints | ros_control + plugins | Dynamixel |
| Control Framework | - | Agilicious (C++) | Agilicious (C++) |
| Parallelism | 4096 environments | Single drone sim | One physical platform |
| Evaluation Purpose | Training + ablations | Policy validation | Real-world validation |

**Table 7.1:** Comparison of training simulation, deployment simulation and deployment hardware

## 7.2. Evaluation tasks and metrics

### 7.2.1. Tasks

We evaluate policies on three tasks:

1. **End-effector pose control:** This task assesses our primary objective of precise 6-Dof positioning of end-effector in a bounded workspace. The policy is given a sequence of goal poses and the aerial manipulator must reach and stabilize the end-effector at every commanded pose. Evaluation is performed in both deployment simulator and real-world to validate our method with unmodeled disturbances.
2. **Load-carrying[2] pose control:** To evaluate robustness under physical disturbances, the aerial manipulator repeats the pose control task, carrying loads between 50 to 200 grams in simulation and up to 140 grams in real-world experiments. This tests the policy's ability to accommodate shift in mass distribution and inertia while maintaining position and orientation control.
3. **Path following:** This task evaluates the policy's ability to perform continuous path tracking while maintaining orientation, serving as a bridge to performing precise contact based tasks such as push and slide. The end-effector is commanded to follow a figure-8 and straight-line path with constant orientation. The figure-8 path challenges the system to coordinate joint movements in two axes, to maintain orientation, while the straight-line path serves as a simpler baseline. Successfully performing this task indicates readiness to perform contact based manipulation.

Simulation experiments are performed either in the training or deployment simulators. Ablation studies discussed in the next chapter are done in training environment and their performance on above mentioned tasks is evaluated by deploying ablated agents in deployment simulator.

### 7.2.2. Metrics

1. **Mean reward** per episode is used to compare and evaluate training runs in ablation studies. In some cases, we compare individual components of training reward, such as position or orientation for additional insight.
2. **Position error** measures the Euclidean distance between the end-effector and goal positions:

$$\mathcal{E}_{pos} = \left\| {}^{W}\mathbf{x}_{ee} - {}^{W}\mathbf{x}_{goal} \right\|_2 , \tag{7.1}$$

[2]Also referred as payload-carrying.

where ${}^{W}\mathbf{x}_{ee}, {}^{W}\mathbf{x}_{goal} \in \mathbb{R}^3$ are the measured end-effector and goal positions in the world frame, reported in meters.

3. **Orientation error** is computed from quaternions as the smallest geodesic rotation angle on $SO(3)$ between the measured and goal orientations:

$$\mathcal{E}_{ori} = 2 \arccos\big( |q_{goal} \cdot q_{ee}| \big), \tag{7.2}$$

   where $q_{goal}$ and $q_{ee}$ are unit quaternions, reported in degrees.

4. **Success rate** is defined as the ratio of poses successfully reached to the total commanded poses. If the aerial manipulator crashes or touches the ground before reaching a pose, it is marked as not reached.
5. **Rise time** is the time taken for the end-effector position to progress from $10\%$ to $90\%$ of the total distance to the commanded goal position.
6. **Settling time** is the time taken for the end-effector position to first enter and remain within $10\%$ of the total distance to the commanded goal position.
7. **RMSE (Root Mean Square Error)** for path tracking is computed over the entire commanded path:

$$\mathcal{E}_{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left\| {}^{W}\mathbf{p}^{i}_{ee} - {}^{W}\mathbf{p}^{i}_{goal} \right\|_2^2}, \tag{7.3}$$

   where $\mathbf{p}^i$ contains position and/or orientation components depending on the metric being evaluated, and $N$ is the number of points considered for evaluation.

We evaluate pose-tracking performance by commanding a pose for 10 s to allow stabilization. Position and orientation errors are calculated over the final 4 s to evaluate steady state behavior. To interpret transition behavior we calculate rise time, settling time and RMSE is calculated over the entire path. This protocol is followed in all experiments unless mentioned otherwise.

## 7.3. Summary

In this chapter, we described the deployment environments used to validate the proposed method to arrive at a functional control policy. Flight conditions were replicated by implementing appropriate controllers, filters, modeling physical properties of the system and simulating its dynamics with high fidelity for reliable transfer. All training and deployment environments share the same controller framework and setup, ensuring deployment simulator closely reflected real-world behavior.

We define primary tasks to evaluate our method - end-effector pose control, load-carrying pose control and path following. Then, we introduced metrics used to evaluate performance on these tasks - position error, orientation error, rise time, settling time and RMSE.

# 8 Simulation Results

In this chapter, we present the results of experiments conducted to validate the proposed method in both training and deployment simulators.

We first report the performance of a baseline policy deployed in our deployment simulation for pose-tracking, payload-carrying and path following tasks. We then present ablation studies on our training setup and compare resulting policies with the baseline to assess and validate our design choices.

## 8.1. Task results

### 8.1.1. End-effector Pose control

We evaluate pose control performance by deploying our policy in the deployment simulator and tracking a set of $N = 250$ randomly generated goal poses $(x, y, z, r, p, y)$. The sampled goal poses span a bounded workspace - $X \in [-1, 1]$, $Y \in [-1, 1]$, $Z \in [1, 2]$ with Roll $\in [-90°, 90°]$, Pitch $\in [-90°, 90°]$ and Yaw $\in [-120°, 120°]$. We sample a new pose every 10s, allowing sufficient time for the system to reach and maintain a pose to evaluate steady state behavior (seen in Figure 8.2).

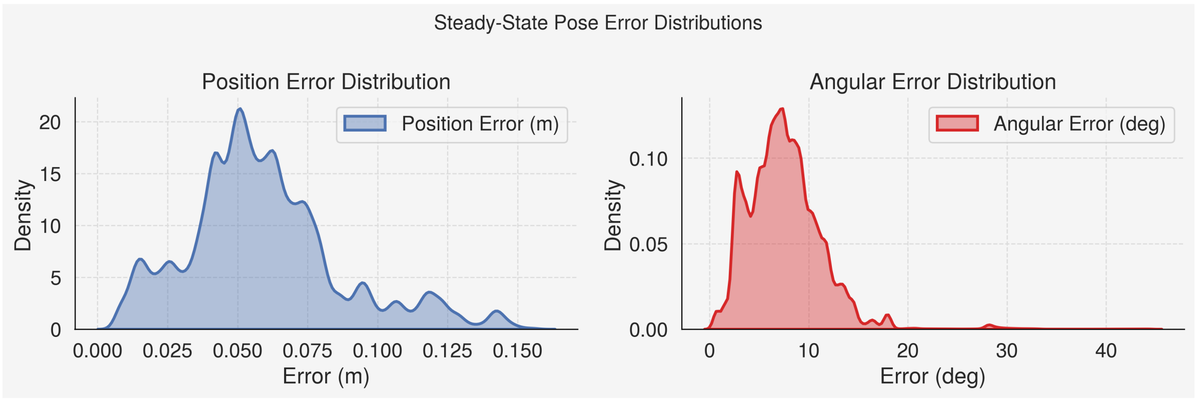


**Figure 8.1:** Pose Error distribution of end-effector tracking 250 randomly generated goals

| **Position Error (m)** | | **Orientation Error (deg)** | | **Success Rate** |
|---|---|---|---|---|
| $\mathcal{E}_{\text{pos}}$ | $\sigma_{\text{pos}}$ | $\mathcal{E}_{\text{ori}}$ | $\sigma_{\text{ori}}$ | (# reached / total) |
| 0.0593 | 0.0276 | 7.5670 | 3.8485 | 250/250 |

**Table 8.1:** Pose tracking error (mean and standard deviation) and success rate over 250 randomly generated goals.

The position error is centered around measured mean of 5.9 cm with a standard deviation of 2.7 cm, while quaternion based orientation error is centered at 7.5° with standard deviation of 3.8°, showing accurate and repeatable end-effector pose control across the workspace.

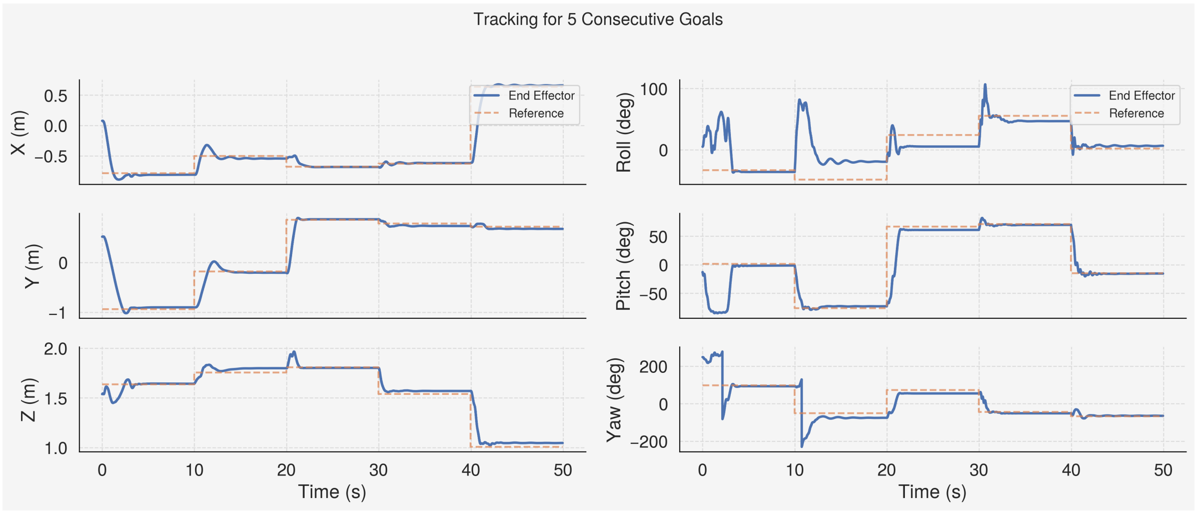


**Figure 8.2:** End-effector pose tracking performance in simulation for 5 goals lasting 10 seconds each

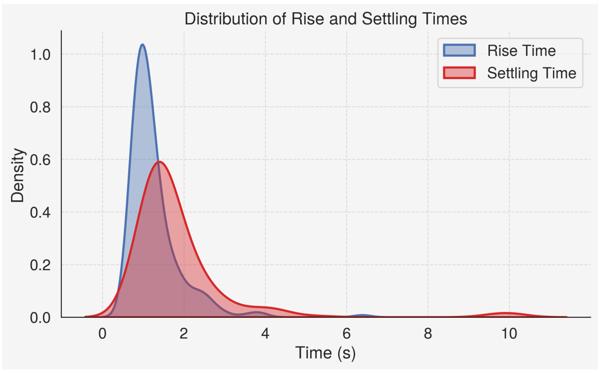


**Figure 8.3:** Rise and Settling time distribution across 250 goal poses tracked

The time-series tracking plot for five consecutive goals (Figure 8.2) shows accurate tracking with minimal overshoot, except when pitch angles reach extremes $(-90°, 90°)$, due to discontinuity in Euler angle representation in gimbal lock. From tracking 250 poses, the end-effector has an average rise time of 1.26 seconds and settle time of 1.86 seconds. Moreover, **our policy reaches every pose, with a success rate of 100%**, demonstrating that it is both robust to varying goal configurations and reliable at achieving poses precisely across the workspace.

### 8.1.2. Payload carrying

| **Payload** | **Position Error (m)** | | **Orientation Error (deg)** | | **Success Rate** |
|---|---|---|---|---|---|
| | $\mathcal{E}_{\text{pos}}$ | $\sigma_{\text{pos}}$ | $\mathcal{E}_{\text{ori}}$ | $\sigma_{\text{ori}}$ | (# reached / total) |
| 50g | 0.078 | 0.057 | 21.82 | 27.59 | 15/15 |
| 100g | 0.104 | 0.063 | 25.54 | 24.38 | 15/15 |
| 150g | 0.211 | 0.287 | 42.04 | 42.53 | 14/15 |
| 200g | 0.229 | 0.181 | 41.47 | 38.80 | 14/15 |

**Table 8.2:** Pose tracking error (mean and standard deviation) and success rate for various payloads.

We evaluate load carrying pose control performance by deploying our policy in the deployment simulator and tracking a set of $N = 15$ goal poses with payloads ranging from 50 grams to 200 grams. To simulate carrying payloads, we increase the end-effector mass with additional 50 grams to 200 grams and track same poses, with 10 seconds of hold per pose to evaluate steady state performance. Goals are sampled from the same workspace as the previous task - $X \in [-1, 1]$, $Y \in [-1, 1]$, $Z \in [1, 2]$ with Roll $\in [-90°, 90°]$, Pitch $\in [-90°, 90°]$ and Yaw $\in [-120°, 120°]$.

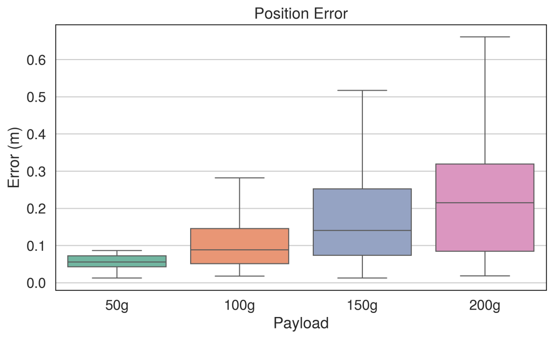


**Figure 8.4:** Position error with different payloads

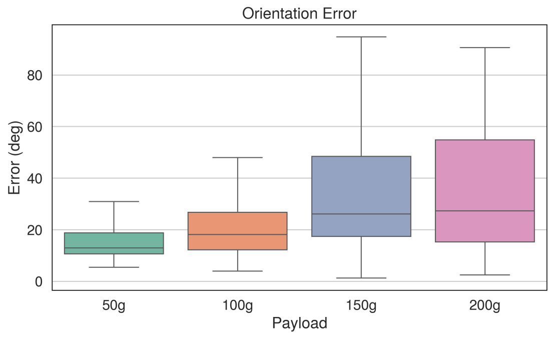


**Figure 8.5:** Orientation error with different payloads

We observe steady increase in both mean pose errors and variance. The performance on position error is within 10 cm for payloads ranging from 50 g to 100 g, while we observe larger deterioration in orientation tracking beyond these payloads. The large orientation errors are likely due to differences in joint dynamics and control frequencies. During training, the arm PID controller operates at 300 Hz, however the simulation loop is fixed to 333 Hz, as the Gazebo control plugin accepts timestep values only up to 3 decimal places (0.003 s).

The policy is trained on payloads up to 120 g during training, yet it **generalizes beyond its training distribution** when deployed in simulation and carries payloads up to 200 g.

## 8.1.3. Path following

We evaluate path following performance of our policy over a figure-8 (8.7a) and straight line (8.7b) path, while maintaining end-effector orientation. By dynamically updating goal poses at 100 Hz, the policy used for end-effector pose control can be applied without retraining. Performance is evaluated over 5 complete loops of each path.

| Path | Position RMSE (m) | Orientation RMSE (deg) |
|---|---|---|
| Figure-8 | 0.5873 | 10.0629 |
| Straight line | 0.2244 | 6.2898 |

**Table 8.3:** RMSE of position and orientation errors for simulated path following tasks.

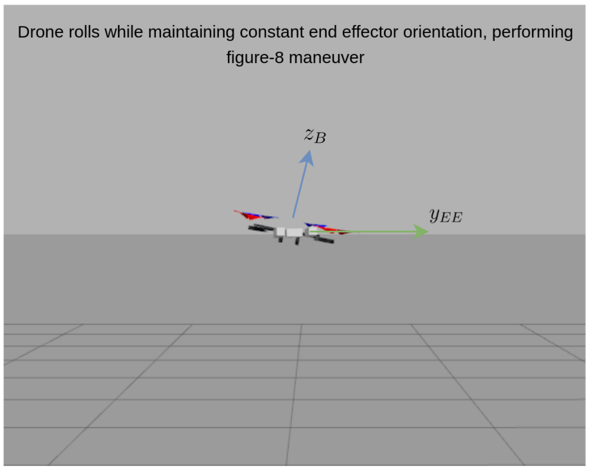
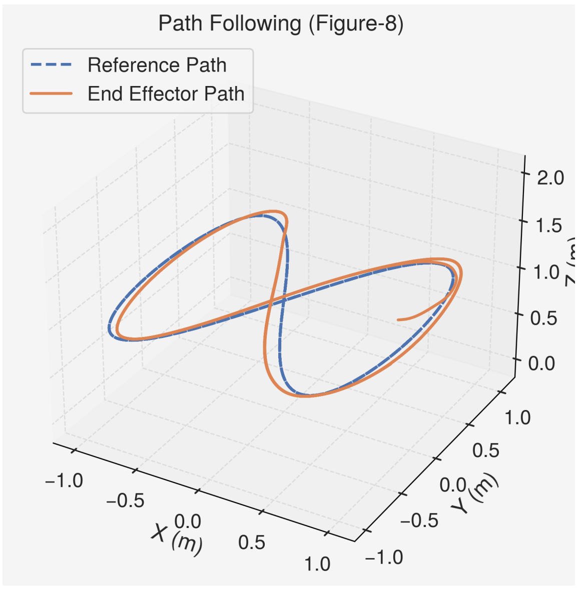
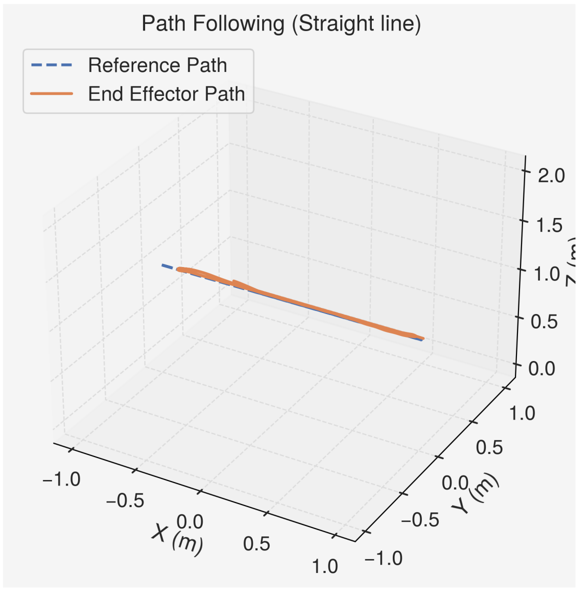

Drone rolls while maintaining constant end effector orientation, performing figure-8 maneuver
$z_B$
$y_{EE}$

**Figure 8.6:** Drone and Manipulator coordination (whole-body control) to maintain constant orientation while following figure-8 path



**(a)** Figure-8 path followed by end-effector in deployment simulator.



**(b)** Straight line path followed by end-effector in deployment simulator

**Figure 8.7:** 3D path following performance in the deployment simulator for two scenarios

In figure-8 path following task, the manipulator has to maintain orientation while coordinating movements with the drone along both degrees of freedom of the joints, **roll - while moving laterally** (See Figure 8.6) and **pitch - when it moves forward and backward**. This dual-axis coordination is more complex than a straight line task, where coordination is limited to a single axis. As a result, figure-8 path following exhibits higher orientation and position tracking errors (Table 8.3).

## 8.2. Ablation results

To study the impact of design choices and validate them, we perform ablations on various components of our training setup. Our primary method of evaluation is using mean of total reward per episode (episode return) across all environments from training runs lasting 2.5 hours. Where necessary, comparison is done on a specific component of total reward such as position, orientation or action smoothing episode returns.

In some ablations, where episode returns are not sufficient to comment on the impact or to gain additional insight, we deploy both the baseline and ablated agents in the deployment simulator and evaluate them by comparing pose tracking performance using previously defined metrics (See 7.2.2).

### 8.2.1. Observation space ablations

#### Joint position

We remove joint positions from the observation space and observe a significant drop in orientation learning (Figure 8.8).

By removing joint state information, the environment transitions from fully observable Markov Decision Process (MDP) to Partially Observable Markov Decision Process (POMDP). In the ablated setup, joint positions are not explicitly provided, but are still internally used to compute end-effector pose via forward kinematics for the remaining observation vector. Theoretically, the agent has enough information to calculate joint positions from end-effector and drone pose, but must do so through internal computation.

As a result, the policy must allocate some capacity to infer the hidden state, reducing sample efficiency and increasing complexity of learning actions for aligning orientation. The observed reduction in orientation reward suggests that explicitly observable joint positions significantly help achieve orientation control by making the environment fully observable.

#### Drone position

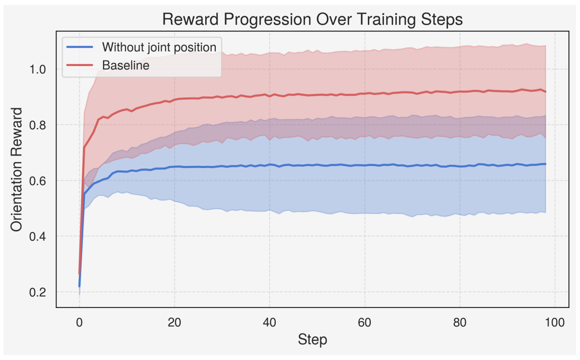


**Figure 8.8:** Comparison of orientation rewards without joint positions in observation space

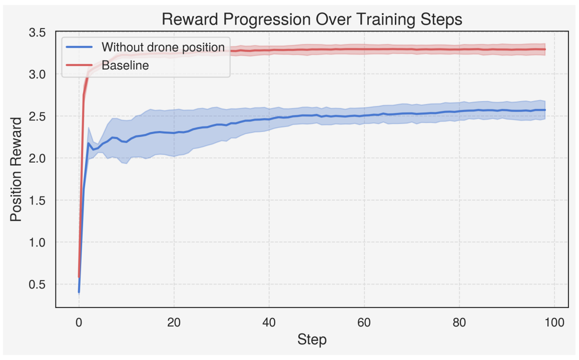


**Figure 8.9:** Comparison of position rewards without drone position in observation space

In the observation space, we include drone position information by expressing goal position in drone frame. By removing this information, we observe substantial decrease in position tracking reward (Figure 8.9), while other reward terms remain unaffected.

Consistent with the previous ablation, removing drone position information also transitions the environment to a POMDP. In this case as well, the missing state can be inferred via inverse kinematics using the joint positions, however requires additional computation and increases task difficulty. Thus, explicit drone position information plays a crucial role in making our system states fully observable.

From a system-level perspective, effects of above ablations align with the control decomposition:

- Position control is primarily governed by the drone base.
- Orientation control is largely handled by the arm.

Therefore, removing drone information impairs position tracking performance, while removing joint information impairs orientation performance.

## 8.2.2. Action space and Low-level controller choices

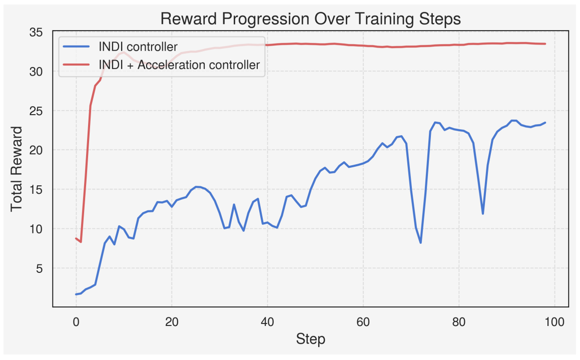


**Figure 8.10:** Comparison of total reward between low-level controller choices

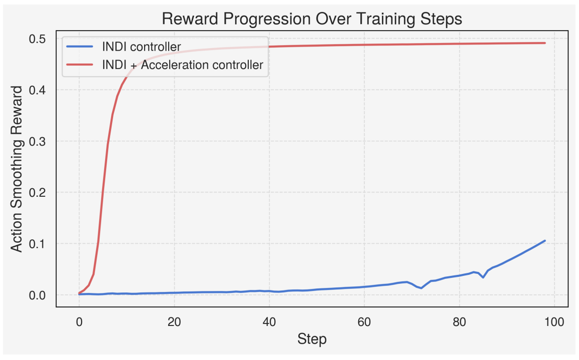


**Figure 8.11:** Comparison of action smoothing reward between low-level controller choices

### Effect of Acceleration controller

Training our agent only with an INDI inner-loop controller by sending combined thrust and reference angular velocities, led to a policy with oscillatory behavior as a result of difficulty in learning to smooth drone control inputs, observed by the lower action smoothing reward at the end of a training loop (Fig 8.11). Adding an acceleration controller and fully utilizing the action space results in improved pose control and faster convergence with higher rewards (Fig 8.10).

The combined effect of low total rewards with significantly lower action smoothing rewards indicates that the agent struggles with exploration when using only the INDI controller. Using only the INDI controller eventually converges, however training takes roughly four times longer than using both acceleration and INDI controller. This might be attributed to several factors:

1. The acceleration controller provides richer control authority, by using linear accelerations for both thrust calculation and attitude control. However, INDI only uses the reference body rates to control attitude, reducing policy's ability to express coordinated position and attitude control.
2. Moreover, only using INDI removes a level of abstraction and giving the policy direct or low-level control. This aligns with [20], showing low-level interfaces can be detrimental for RL performance.

### 8.2.3. Domain randomization

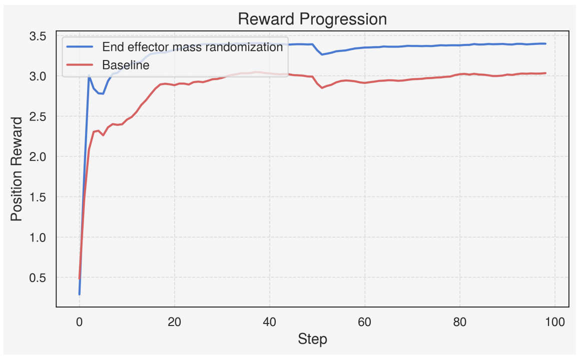


**Figure 8.12:** Position tracking reward comparison with end-effector mass randomization

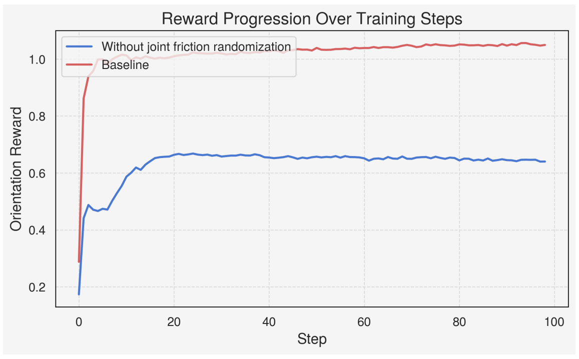


**Figure 8.13:** Orientation tracking reward comparison without friction randomization

#### End-effector mass

We randomize end-effector mass between 0.5 to 5 times its default mass of 30 g, corresponding to 15 g to 150 g.

As seen in Figure 8.12, we observe a higher position reward with end-effector mass randomization. This improvement can be attributed to exploration induced by mass variation. When encountered with varying mass, the agent is forced to develop more robust control actions. This mass randomization acts like controlled exploration for the agent and prevents it from fitting to a single mass configuration, ultimately improving position tracking. Additionally, we deploy both policies in the simulator and compare pose tracking performance on $N = 20$ poses while carrying a payload of 100 g. Without mass randomization the policy exhibited oscillatory and unstable behavior, and crashed before reaching all poses. Therefore, we reduced P gains of our joints identically in both conditions.

From figure 8.14, it is evident that randomizing end-effector mass improves position tracking performance.

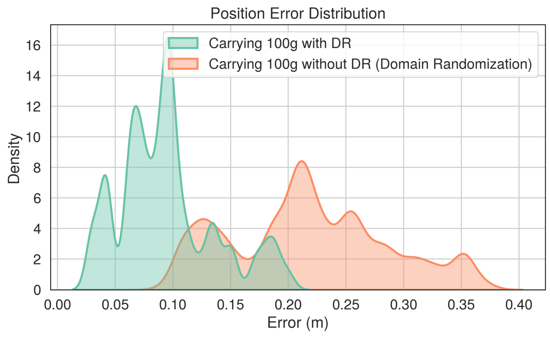


**Figure 8.14:** Position tracking error comparison with end-effector mass randomization

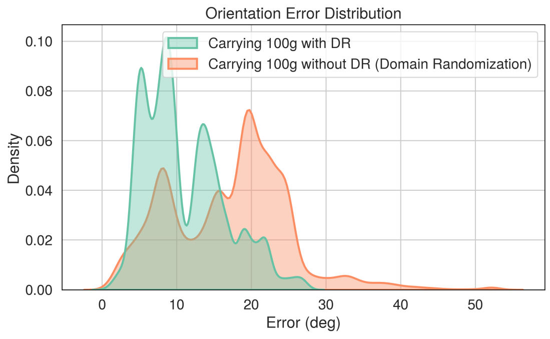


**Figure 8.15:** Orientation tracking error comparison with end-effector mass randomization

#### Joint friction:

By removing joint friction randomization, we observe a substantial drop in orientation learning rewards.

Similar to the previous ablation, the setup with constant friction overfits to a narrow set of end-effector mass and joint stiffness values. When these properties are randomized with constant friction, a stick and slip motion is observed, resulting in unpredictable arm dynamics and reduced orientation learning, as seen in Figure 8.13. Because friction models in real-world are typically unknown, these results motivate randomizing joint friction during training.

### 8.2.4. Reward design choices

#### Action smoothing

Including action smoothing penalty during training is crucial for stable transfer performance. Without this term, the agent generates oscillating actions, frequently saturating actuators and causing the end-effector to oscillate near the goal as shown in Figure 8.16.

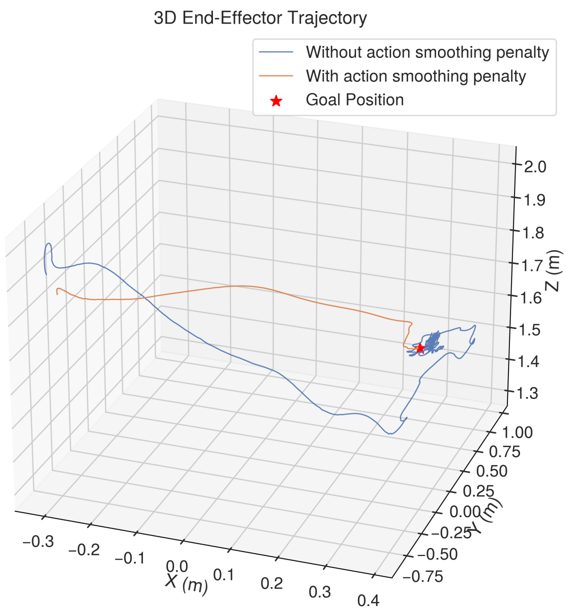


**Figure 8.16:** End-effector trajectory of agents deployed in deployment simulator, trained with and without action smoothing

The observed tracking error across $N = 20$ poses, is slightly lower (Figure 8.17) for the agent trained without action smoothing, likely due to higher agility and faster response from the quadrotor base when it's control inputs are not smoothed. However, this comes at the cost of stability, as the end-effector and quadrotor, both oscillate when they reach the goal pose, as seen by the blue cluster in Figure 8.16, which is unfavorable for transfer.

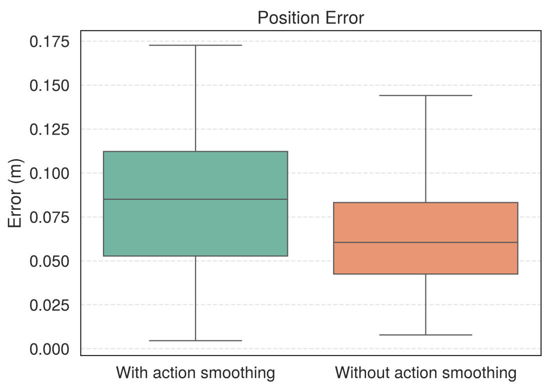


**Figure 8.17:** Position error of agents deployed in simulation tracking 20 poses

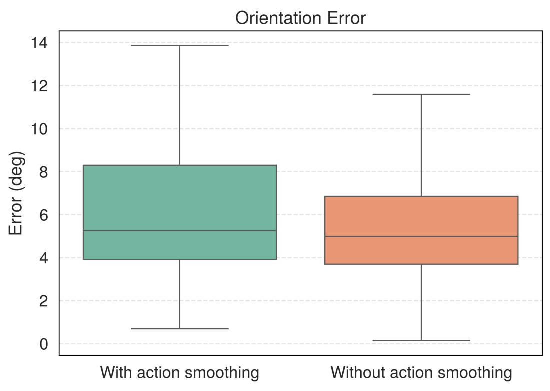


**Figure 8.18:** Orientation error of agents deployed in simulation tracking 20 poses

### 8.2.5. Conclusion

Above ablations validate key choices we make in action space, observation space, rewards and domain randomization. Based on their results, we highlight some takeaways that might be applicable to training RL-based controllers for aerial manipulators and related domains -

1. **Full state observability improves performance:** While the agent can learn with partial observability, explicitly providing critical state variables such as, joint positions or goal in drone frame, ensures full observability, resulting in higher training rewards.
2. **Higher-level control abstractions accelerate learning:** Direct low-level control using RL can be detrimental during both deployment and training, as observed by 4x longer training time with INDI controller. Introducing abstractions such as a higher level controller can help reduce training time and improve final performance.
3. **Domain randomization enhances robustness and generalization:** Randomizing parameters such as payload mass and friction improves agent's ability to generalize beyond the training distribution and helps avoid overfitting to a single configuration. This acts like a form of controlled exploration and improving task performance.
4. **Action smoothing rewards are essential for transfer:** Penalizing abrupt changes in actions leads to smoother trajectories and minimizes frequent actuator saturation. While action smoothing might decrease agility, it improves orientation performance and allows reliable transfer.

These findings guided the development of the proposed methodology and training a functional policy. In the next chapter, we further validate our choices and evaluate controller performance through real-world experiments.

# 9

# Real-world results

In this chapter, we validate our method in practical conditions by performing experiments in real-world. Like experiments performed in deployment simulator, we evaluate performance using the metrics (7.2.2) and tasks (7.2.1) defined earlier - end-effector pose control, load-carrying pose control and path following.

## 9.1. Real-world results

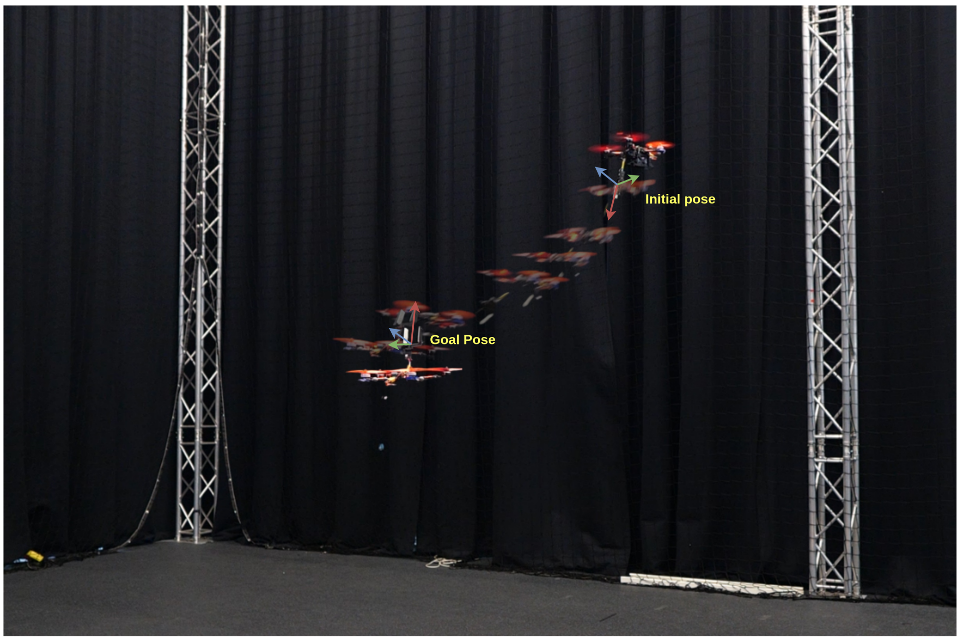


**Figure 9.1:** End-effector pose control performed by policy running onboard in real-world (video link).

### 9.1.1. End-effector Pose control

We validate end-effector pose control on physical system by sending $N = 10$ randomly generated poses $(x, y, z, r, p, y)$. The commanded poses are sampled from the same bounded workspace as in simulation - $X \in [-1, 1]$, $Y \in [-1, 1]$, $Z \in [1, 2]$ with Roll $\in [-90°, 90°]$, Pitch $\in [-90°, 90°]$ and Yaw $\in [-120°, 120°]$. Each pose is maintained for 10 seconds, giving time for convergence. The scale of this experiment is reduced considering experimental time and safety constraints, however serves as a validation for simulation results.

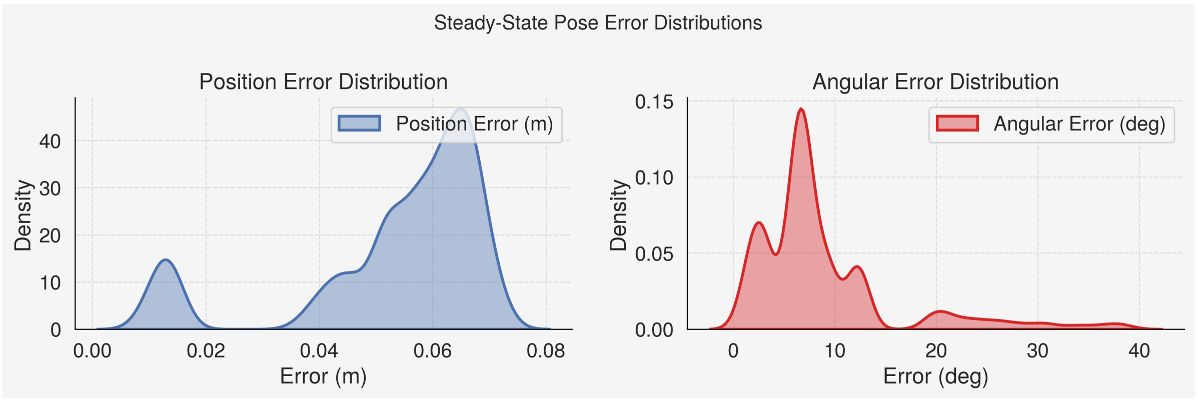


**Figure 9.2:** Pose error distribution of tracking 10 poses in real-world

| **Position Error (m)** | | **Orientation Error (deg)** | | **Success Rate** |
|---|---|---|---|---|
| $\mathcal{E}_{\text{pos}}$ | $\sigma_{\text{pos}}$ | $\mathcal{E}_{\text{ori}}$ | $\sigma_{\text{ori}}$ | (# reached / total) |
| 0.0536 | 0.0166 | 8.8078 | 7.1834 | 10/10 |

**Table 9.1:** Pose tracking error (mean and standard deviation) and success rate over 10 randomly generated goals.

The mean position and orientation errors are 5.3 cm and 8.8°, showing repeatable and accurate pose tracking in real-world, validating our training methodology.

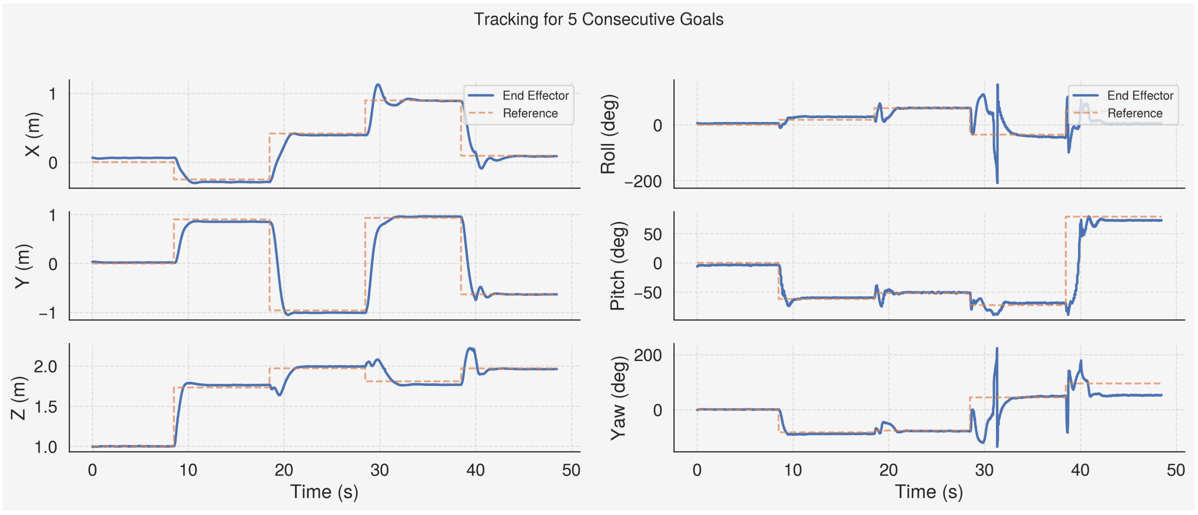


**Figure 9.3:** Pose tracking performance in real-world for 5 goals lasting 10 seconds each

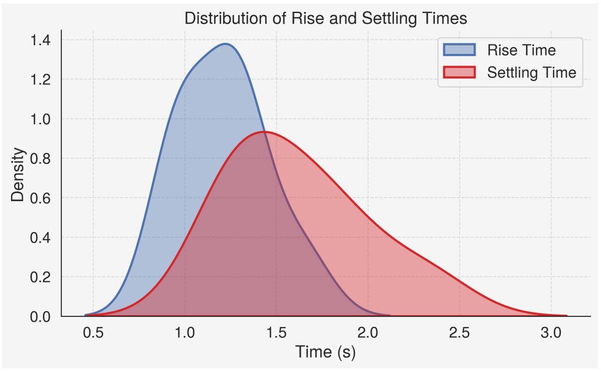


**Figure 9.4:** Rise and Settling time distribution across 10 goals tracked

The time-series tracking plot for five consecutive goals (Figure 9.3) shows accurate tracking with some overshoot in orientation axes. Over 10 poses, the end-effector has an average rise time of 1.26 seconds and settle time of 1.61 seconds. Our policy reliably and precisely achieves poses across the workspace in real-world.

## 9.1.2. Payload carrying

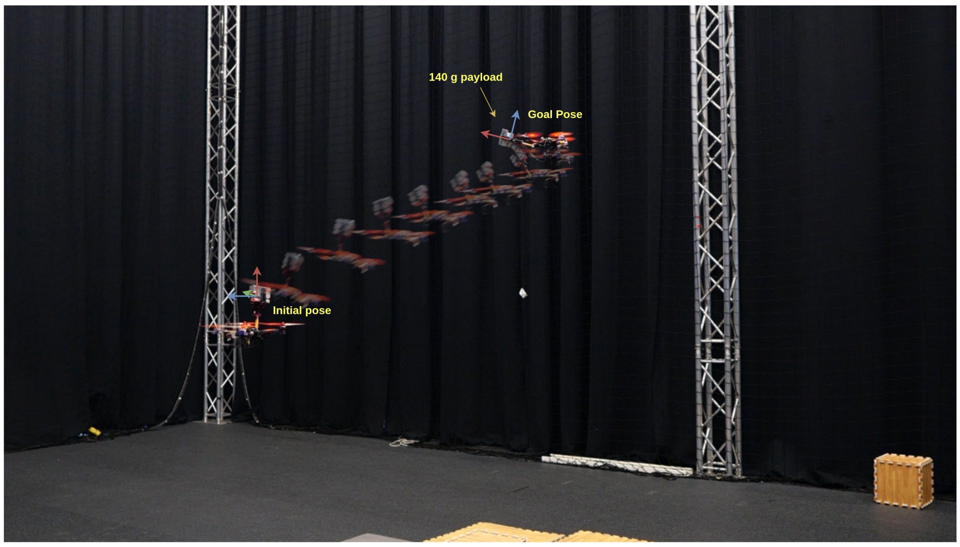


**Figure 9.5:** end-effector pose control while carrying 140g payload, performed by policy running onboard in real-world (video link).

To evaluate end-effector pose control performance while carrying payload we send $N = 7$ randomly generated poses $(x, y, z, r, p, y)$. The commanded poses are sampled from the same bounded workspace as our previous task of pose control without payload. Under similar conditions we compare pose control performance between carrying 50 grams and 140 grams of payload.

| **Category** | **Mean Error** (m / deg) | **Std. Deviation ($\sigma$)** (m / deg) |
|---|---|---|
| 50 g Position ($\mathcal{E}_{\text{pos}}$) | 0.0995 m | 0.0695 m |
| 50 g Orientation ($\mathcal{E}_{\text{ori}}$) | 12.5020° | 3.0619° |
| 140 g Position ($\mathcal{E}_{\text{pos}}$) | 0.0954 m | 0.0505 m |
| 140 g Orientation ($\mathcal{E}_{\text{ori}}$) | 15.7006° | 4.8312° |

**Table 9.2:** Mean and standard deviation of position error and orientation error for 50 g and 140 g payload conditions over the evaluated goal poses in real-world.

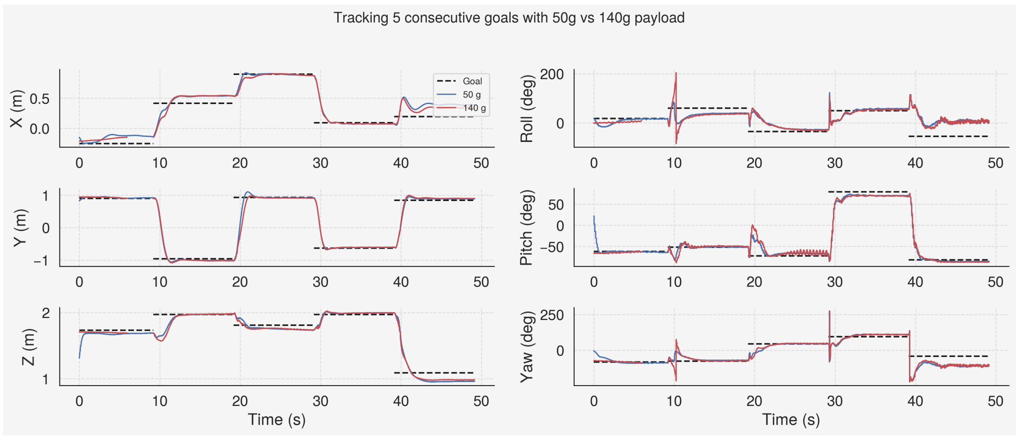


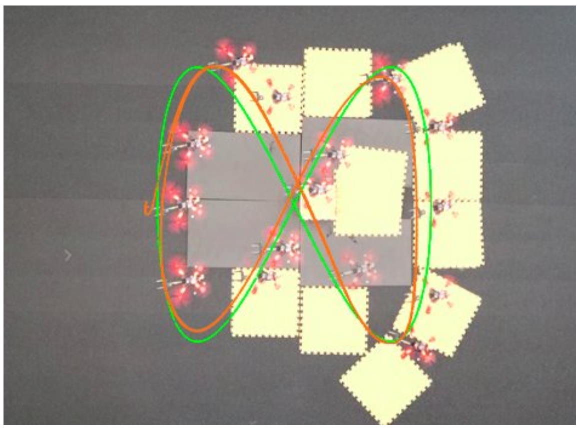
**Figure 9.6:** Pose tracking performance in real-world for 5 goals lasting 10 seconds each, with 50 g and 140 g payload

The average position error while carrying 50 grams is 9.9 cm with similar value for 140 gram scenario. However, the orientation error is 3° more while carrying 140 grams. A possible reason is that position control is primarily governed by quadrotor base, which is less affected by additional payload. Whereas, orientation control is largely handled by the arm, particularly roll and pitch, making it susceptible to load induced disturbances (see roll and pitch responses in Figure 9.6). The abrupt change observed in roll angle is caused by a discontinuity in Euler-angle representation, which occurs due to gimbal lock (see other dimensions represented in A.2).

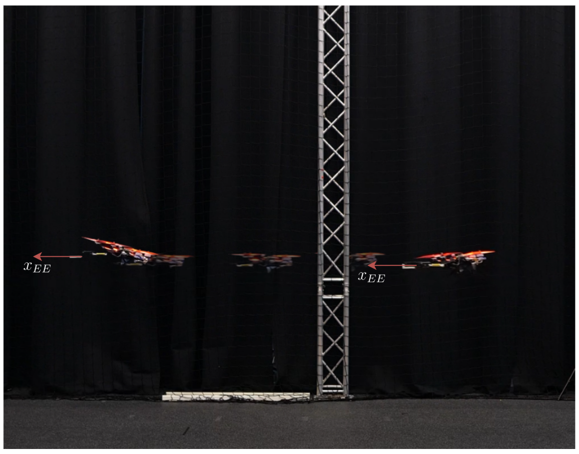
Apart from minor oscillations in the arm, our policy demonstrates capability of carrying payloads while accurately maintaining poses with position errors of approximately 10 cm and orientation errors of 15°. Moreover, the **policy generalizes beyond its training distribution in real-world**, by carrying a payload of 140 grams, despite being trained only on loads up to 120 grams. Tests on higher payloads were avoided due to limited gripper size and safety considerations.

### 9.1.3. Path following

We evaluate performance on path following task by commanding the end-effector to track a figure 8 path and a straight line path, both while maintaining constant end-effector orientation. The figure 8 path tests the system's ability to coordinate motion in both degrees of freedom of the arm, while maintaining the commanded pose. The straight line serves as a simpler baseline to maintain orientation by primarily requiring pitch axis coordination between arm and the drone (visible in Figure 9.7b).



**(a)** Top view of end-effector following a figure-8 path, with reference path colored in green (video link).



**(b)** Side view of end-effector following a straight line while maintaining constant orientation (video link).

**Figure 9.7:** Path-following performance in two real-world scenarios.

| Path | Position RMSE (m) | Orientation RMSE (deg) |
|---|---|---|
| Figure-8 | 0.8167 | 14.3915 |
| Straight line | 0.1960 | 5.5607 |

**Table 9.3:** RMSE of position and orientation errors for the two path following tasks.

As seen in Table 9.3, the RMSE of figure 8 path is 0.82 m in position and 14° in orientation compared to lower RMSE in the straight line scenario with 0.19 m and 5.5°, reflecting the policy's ability to track relatively simple paths with greater accuracy. These results are consistent with observations from the deployment simulator, where figure-8 required continuous roll-pitch coordination, leading to larger deviations.

These results demonstrate that the policy is able to **successfully follow complex paths while maintaining orientation, despite not being explicitly trained for path following**.

## 9.2. Summary

We validate our method by evaluating control performance on three tasks - end-effector pose control, load-carrying pose control and path following, in both simulation and real-world deployments. Table 9.4 summarizes the performance for all evaluated tasks (Section 7.2.1).

| **Task** | **Sim** | | **Real-world** | |
|---|---|---|---|---|
| | Pos (m) | Ori (°) | Pos (m) | Ori (°) |
| Pose control | 0.060 | 7.5 | 0.053 | 8.8 |
| Payload (50 g) | 0.078 | 21.0 | 0.099 | 12.0 |
| Payload (140 g) | 0.132 | 25.7 | 0.095 | 15.0 |
| Payload (200 g) | 0.220 | 41.0 | – | – |
| Figure-8 path (RMSE) | 0.587 | 10.1 | 0.817 | 14.4 |
| Straight line path (RMSE) | 0.224 | 6.3 | 0.196 | 5.6 |

**Table 9.4:** Summary of simulation vs. real-world performance across tasks.

In end-effector pose control task, we evaluate performance across $N = 250$ poses in the simulation and $N = 10$ poses in real-world and observe comparable performance, showing accurate and repeatable pose tracking with reliable transfer.

In the payload-carrying pose control task, we evaluate performance with payloads ranging from 40 g to 200 g in simulation, on $N = 15$ poses. In real-world, we limit to 50 g and 140 g masses on $N = 7$ poses due to gripper limitations and safety concerns. Interestingly, robustness to weight is higher in real-world with minimal increase in pose errors between 50 g and 140 g payloads. The large orientation errors in simulation are due to differences in joint dynamics and control frequencies.

For path following, we evaluate performance in both simulation and real-world over a figure-8 and straight line path. Higher errors are observed on figure-8 path in both deployments due to demands in multi-axis coordination, compared to more simple single-axis coordination in straight line path following.

Through ablations, we find that high-level controller abstractions accelerate learning and that action smoothing and domain randomization ensure reliable sim-to-sim and sim-to-real transfer.

# 10 Conclusion

This thesis addressed the challenge of whole body control of underactuated aerial manipulator equipped with a two-degree-of-freedom manipulator arm. Aerial manipulation is complex due to coupled dynamics of the drone platform and manipulator, as well as influence of external disturbances. Traditional model-based control strategies rely on precise system identification and degrade with model inaccuracies, unmodeled disturbances, or with need of rapid adaptation. To overcome these limitations, this thesis explores reinforcement learning-based control, using PPO (Proximal Policy Optimization) to learn robust control policies for aerial manipulation using interaction data from simulation.

As a precursor to aerial manipulation, we define end-effector pose control as our primary task. We demonstrate that a learned PPO-based policy can achieve 100% of commanded poses (with an average 5.3 cm and 9° of position and orientation error) in the reachable workspace and validate it through experiments in simulation and real-world. Next, we evaluate our policy's robustness to shift in weight distribution and external disturbances by carrying a payload using the end-effector. The policy demonstrates strong generalization capabilities, maintaining stability and accurate pose control under conditions not encountered during training, by carrying payloads up to 140 grams in real-world experiments. Results from real-world experiments are consistent with policy's performance in simulation and in some cases, also show improvement, indicating policy transfer without degradation.

We extend our evaluation to continuous path tracking capability while maintaining orientation, serving as a bridge to perform precise contact based manipulation tasks in the future. Experiments with both straight line and figure-8 path tracking confirmed that the learned whole-body control policy coordinated motion between the manipulator and quadrotor base to maintain orientation. This validates that the RL-based policy not only handles discrete pose commands but also generalizes to continuous coordinated path tracking without any modifications in the method.

We perform ablation studies to gain additional insights in our design choices. Domain randomization of end-effector mass reduced positional error, acting as a form of controlled exploration and increasing robustness. Removing intermediate control abstractions and giving the policy direct access to low-level control was detrimental. This resulted in poorer final performance and with longer convergence time compared to including a higher level control abstraction. This finding is consistent with prior literature on detrimental effect of using direct low-level control [20] and its degraded sim-to-real transferability. Additionally, action smoothing reward was found to be significantly important to avoid actuator saturation and ensure reliable transfer.

In conclusion, this thesis **developed and validated a robust whole-body control policy for an underactuated aerial manipulator**. The methodology develops a policy demonstrating end-effector pose control capabilities with high accuracy, robustness to disturbances, real-time capability (inference time of 0.18 ms) and whole-body coordination, meeting defined requirements in Section 3.2. The findings and methodology lay a foundation to extending RL-based control to contact rich aerial manipulation tasks, with potential applications in industrial inspection and assembly operations.

### 10.0.1. Limitations

Although the proposed training method is model-free and avoids the complexity of deriving an accurate system model in comparison to model-based methods, this complexity is not entirely eliminated, it is shifted to improving simulation fidelity. In RL, the absence of an accurate system model is compensated by need of a simulation that accurately reflects the real-world conditions to collect interaction data. This step is non-trivial (more in Section A.0.1), given the difference in actuator properties, body inertia, low-level controllers and other properties that need modeling to simulate real-world conditions. For successful sim-to-real transfer, the simulation needs to accurately reflect real-world conditions before domain randomization can improve robustness. Moreover, our method is not entirely model-free. The quadrotor model is used by low-level controllers, which is a simplified model of our entire system, while the learned policy captures the coupling between the arm and the quadrotor.

Unlike model-based methods, reinforcement learning does not provide any mathematical safety guarantees. While our policy demonstrated robustness in varying conditions, the lack of formal safety proofs mean that unwanted behaviors could still emerge. In the future, hybrid methods of combining RL and controllers with safety guarantees or switching between controllers can be considered, while performing aerial manipulation in uncontrolled environments.

The policy is trained and evaluated in an environment with perfect state information. Our simulation and real-world lab environment provide noise free state estimates, rarely achievable outside ideal settings. In practice, if state estimates arrive from variety of sensors, they might include noise, latency and drift. These conditions might further reveal limitations of our policy on robustness and generalization capabilities.

### 10.0.2. Future work

This work achieves an important precursor to aerial manipulation - robust end-effector pose control. The methodology and design choices presented can be extended to perform precise contact base manipulation tasks such as push-slide, pick-place or tool use, by including object states in the observation [7].

For performing precise contact rich tasks, the control policy could benefit from additional sensing capability such as force feedback, tactile feedback or vision based tracking [15] for obstacle avoidance, object detection and control forces exerted during manipulation.

To deploy such a setup in real-world, state estimation and localization must be achieved without relying on an expensive motion capture setup. Promising approaches include IMU and vision based odometry [37] , visual-inertial fusion [29], or SLAM based methods using multi-modal sensor integration.

We evaluate our policy on path following task, despite not being explicitly trained on it. Future work could explore training methods tailored for path-following to improve tracking and stability. Including information about upcoming path few time steps in the future might improve tracking performance. Combining accurate path-following, with grasping capabilities could enable complex aerial manipulation tasks such as pick-place and knob-turning.

# References

[1] Albert Albers et al. “Semi-autonomous flying robot for physical interaction with environment”. In: *2010 IEEE Conference on Robotics, Automation and Mechatronics*. 2010, pp. 441–446. DOI: 10.1109/RAMECH.2010.5513152.

[2] OpenAI: Marcin Andrychowicz et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. DOI: 10.1177/0278364919887447. eprint: https://doi.org/10.1177/0278364919887447. URL: https://doi.org/10.1177/0278364919887447.

[3] Federico Augugliaro et al. “Building tensile structures with flying machines”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 3487–3492. DOI: 10.1109/IROS.2013.6696853.

[4] Autonomous Multi-Robots Lab (AMR Lab). *Autonomous Multi-Robots Lab (AMR Lab)*. https://autonomousrobots.nl/. Accessed: 2025-08-16. Delft University of Technology, 2025.

[5] Dario Brescianini and Raffaello D’Andrea. “Tilt-Prioritized Quadrocopter Attitude Control”. In: *IEEE Transactions on Control Systems Technology* 28.2 (2020), pp. 376–387. DOI: 10.1109/TCST.2018.2873224.

[6] Maximilian Brunner et al. “A Planning-and-Control Framework for Aerial Manipulation of Articulated Objects”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10689–10696. DOI: 10.1109/LRA.2022.3191178.

[7] Eugenio Cuniato et al. “Learning to Open Doors with an Aerial Manipulator”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2023/10 2023.

[8] ONNX Runtime developers. *ONNX Runtime*. https://onnxruntime.ai/. Version: x.y.z. 2021.

[9] Cora A. Dimmig and Marin Kobilarov. *Non-Prehensile Aerial Manipulation using Model-Based Deep Reinforcement Learning*. 2024. arXiv: 2407.00889 [cs.RO]. URL: https://arxiv.org/abs/2407.00889.

[10] Aleksandra Faust et al. “Automated aerial suspended cargo delivery through reinforcement learning”. In: *Artificial Intelligence* 247 (2017). Special Issue on AI and Robotics, pp. 381–398. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.11.009. URL: https://www.sciencedirect.com/science/article/pii/S0004370214001416.

[11] Philipp Foehn et al. “Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight”. In: *Science Robotics* (2022).

[12] Pål J. From. “An Explicit Formulation of Singularity-Free Dynamic Equations of Mechanical Systems in Lagrangian Form—Part Two: Multibody Systems”. In: *Modeling, Identification and Control* 33.2 (2012), pp. 61–68. DOI: 10.4173/mic.2012.2.3.

[13] Matteo Fumagalli et al. “Developing an Aerial Manipulator Prototype: Physical Interaction with the Environment”. In: *IEEE Robotics  Automation Magazine* 21.3 (2014), pp. 41–50. DOI: 10.1109/MRA.2013.2287454.

[14] Fadri Furrer et al. “Robot Operating System (ROS): The Complete Reference (Volume 1)”. In: ed. by Anis Koubaa. Cham: Springer International Publishing, 2016. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

[15] Gowtham Garimella and Marin Kobilarov. “Towards model-predictive control for aerial pick-and-place”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 4692–4697. DOI: 10.1109/ICRA.2015.7139850.

[16] Danijar Hafner et al. *Mastering Diverse Domains through World Models*. 2024. arXiv: 2301.04104 [cs.AI]. URL: https://arxiv.org/abs/2301.04104.

[17] Mahmoud Hamandi et al. “Design of multirotor aerial vehicles: A taxonomy based on input allocation”. In: *The International Journal of Robotics Research* 40.8-9 (2021), pp. 1015–1044. DOI: 10.1177/02783649211025998. eprint: https://doi.org/10.1177/02783649211025998. URL: https://doi.org/10.1177/02783649211025998.

[18] Jemin Hwangbo et al. “Control of a Quadrotor with Reinforcement Learning”. In: *CoRR* abs/1707.05110 (2017). arXiv: 1707.05110. URL: http://arxiv.org/abs/1707.05110.

[19] Jemin Hwangbo et al. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* 4.26 (2019), eaau5872. DOI: 10.1126/scirobotics.aau5872. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aau5872. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aau5872.

[20] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 10504–10510. DOI: 10.1109/ICRA46639.2022.9811564.

[21] Elia Kaufmann et al. “Deep Drone Acrobatics”. In: *CoRR* abs/2006.05768 (2020). arXiv: 2006.05768. URL: https://arxiv.org/abs/2006.05768.

[22] Jens Kober and Jan Peters. “Reinforcement Learning in Robotics: A Survey”. In: *Learning Motor Skills: From Algorithms to Robot Experiments*. Cham: Springer International Publishing, 2014, pp. 9–67. ISBN: 978-3-319-03194-1. DOI: 10.1007/978-3-319-03194-1_2. URL: https://doi.org/10.1007/978-3-319-03194-1_2.

[23] Christopher Korpela, Matko Orsag, and Paul Oh. “Towards valve turning using a dual-arm aerial manipulator”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 3411–3416. DOI: 10.1109/IROS.2014.6943037.

[24] Vincenzo Lippiello and Fabio Ruggiero. “Exploiting redundancy in Cartesian impedance control of UAVs equipped with a robotic arm”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 3768–3773. DOI: 10.1109/IROS.2012.6386021.

[25] Yen-Chen Liu and Chi-Yu Huang. “DDPG-Based Adaptive Robust Tracking Control for Aerial Manipulators With Decoupling Approach”. In: *IEEE Transactions on Cybernetics* 52.8 (2022), pp. 8258–8271. DOI: 10.1109/TCYB.2021.3049555.

[26] Josep Marti-Saumell et al. “Full-Body Torque-Level Non-linear Model Predictive Control for Aerial Manipulation”. In: *CoRR* abs/2107.03722 (2021). arXiv: 2107.03722. URL: https://arxiv.org/abs/2107.03722.

[27] Wim Meeussen et al. *ros_control: A set of packages for controller interfaces, managers, transmissions, and hardware interfaces*. https://wiki.ros.org/ros_control. Accessed: 2025-08-15. ROS.org, 2013.

[28] Daniel Mellinger et al. “Design, modeling, estimation and control for aerial grasping and manipulation”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 2668–2673. DOI: 10.1109/IROS.2011.6094871.

[29] Elias Mueggler et al. “Continuous-Time Visual-Inertial Trajectory Estimation with Event Cameras”. In: *CoRR* abs/1702.07389 (2017). arXiv: 1702.07389. URL: http://arxiv.org/abs/1702.07389.

[30] Anibal Ollero et al. “Past, Present, and Future of Aerial Robotic Manipulators”. In: *IEEE Transactions on Robotics* 38.1 (2022), pp. 626–645. DOI: 10.1109/TRO.2021.3084395.

[31] OpenAI. *Spinning Up in Deep RL*. https://spinningup.openai.com/en/latest/. Accessed: July 19, 2025. 2018.

[32] OpenAI. *Vector graphic generated with ChatGPT*. Created using OpenAI’s ChatGPT on request by Shlok Deshmukh. 2025.

[33] Matko Orsag et al. “Stability control in aerial manipulation”. In: *2013 American Control Conference*. 2013, pp. 5581–5586. DOI: 10.1109/ACC.2013.6580711.

[34] Matko Orsag et al. “Valve turning using a dual-arm aerial manipulator”. In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014, pp. 836–841. DOI: 10.1109/ICUAS.2014.6842330.

[35] ROBOTIS Co., Ltd. *XC330-M288 Dynamixel e-Manual*. Accessed: 2025-08-10. 2025. URL: https://emanual.robotis.com/docs/en/dxl/x/xc330-m288/.

[36] Nikita Rudin et al. *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*. 2022. arXiv: 2109.11978 [cs.RO]. URL: https://arxiv.org/abs/2109.11978.

[37] Àngel Santamaria-Navarro et al. *Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors*. This work was funded by the project ROBINSTRUCT (TIN2014-58178-R) et al., 2018. DOI: 10.1007/s10514-017-9690-5.

[38] Antonio Serrano-Muñoz et al. “skrl: Modular and Flexible Library for Reinforcement Learning”. In: *CoRR* abs/2202.03825 (2022). arXiv: 2202.03825. URL: https://arxiv.org/abs/2202.03825.

[39] Yunlong Song et al. “Reaching the limit in autonomous racing: Optimal control versus reinforcement learning”. In: *Science Robotics* 8.82 (2023), eadg1462. DOI: 10.1126/scirobotics.adg1462. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.adg1462. URL: https://www.science.org/doi/abs/10.1126/scirobotics.adg1462.

[40] Nicolas Staub et al. “Towards a Flying Assistant Paradigm: the OTHex”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 6997–7002. DOI: 10.1109/ICRA.2018.8460877.

[41] Sihao Sun et al. “A Comparative Study of Nonlinear MPC and Differential-Flatness-based Control for Quadrotor Agile Flight”. In: *IEEE Transactions on Robotics* (2022). DOI: 10.1109/TRO.2022.3177279.

[42] R.S. Sutton and A.G. Barto. “Reinforcement Learning: An Introduction”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 1054–1054. DOI: 10.1109/TNN.1998.712192.

[43] Hideyuki Tsukagoshi et al. “Aerial manipulator with perching and door-opening capability”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 4663–4668. DOI: 10.1109/ICRA.2015.7139845.

[44] Jake Welde, James Paulos, and Vijay Kumar. “Dynamically Feasible Task Space Planning for Underactuated Aerial Manipulators”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3232–3239. DOI: 10.1109/LRA.2021.3051572.

[45] Grady Williams et al. “Information theoretic MPC for model-based reinforcement learning”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1714–1721. DOI: 10.1109/ICRA.2017.7989202.

[46] H. W. Wopereis et al. “Application of substantial and sustained force to vertical surfaces using a quadrotor”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2704–2709. DOI: 10.1109/ICRA.2017.7989314.

[47] Burak Yüksel et al. “Aerial physical interaction via IDA-PBC”. In: *The International Journal of Robotics Research* 38.4 (2019), pp. 403–421. DOI: 10.1177/0278364919835605. eprint: https://doi.org/10.1177/0278364919835605. URL: https://doi.org/10.1177/0278364919835605.

[48] Jack Zeng et al. *Decentralized Aerial Manipulation of a Cable-Suspended Load using Multi-Agent Reinforcement Learning*. 2025. arXiv: 2508.01522 [cs.RO]. URL: https://arxiv.org/abs/2508.01522.

[49] Guangyu Zhang et al. “Robust Control of an Aerial Manipulator Based on a Variable Inertia Parameters Model”. In: *IEEE Transactions on Industrial Electronics* 67.11 (Nov. 2020), pp. 9515–9525. ISSN: 1557-9948. DOI: 10.1109/tie.2019.2956414. URL: http://dx.doi.org/10.1109/TIE.2019.2956414.

# A
# Appendix

### A.0.1. Transfer challenges

Learning based methods rely on extensive data collection and exploration, which is safely obtained using simulations. However, minor differences between environments can induce behaviors learned in simulation that are infeasible in real-world or cause instable response.

We encountered multiple such **sim-to-real** and **sim-to-sim** challenges. We tackle this by either increasing environment fidelity or training the agent to be robust against differences using domain randomization.

1. **High frequency oscillation in arm motion:** We encountered high frequency oscillations in arm after transferring our trained agent to the deployment simulator, due to difference in PID controllers that generate joint torques. The default PID implementation in Isaac Lab uses joint velocity to calculate the D term ($D = -d * v$), however the ros_control plugin in Gazebo uses rate of error ($D = -d * ((e_{t-1} - e_t)/dt)$). This generates a D term that fluctuates and influences torque applied rapidly. Rewriting the implementation in Isaac Lab to match the ros_control plugin, fixes the gap.
2. **Low frequency oscillation in arm motion:** Low frequency oscillations in arm were observed in both simulation and hardware deployment due to differences in friction model mentioned earlier (See 6.2). This makes it impossible to generate same joint responses with matching control inputs and joint properties (As seen in Figure A.1). We resolve this gap by randomizing friction coefficients during training.
3. **Oscillation in quadrotor roll axis:** When deployed in deployment simulation, the quadrotor oscillated in roll axis due to differences in sequence of applying desired rotor thrusts, within a control step. Since neither environments simulate actual rotor spin, thrusts are computed from desired rotor speeds ($f = c_t \omega^2$, where $c_t$ is the thrust coefficient) and applied as forces on each rotor in body frame.

   In the deployment simulator, thrusts were generated using current rotor speeds $\omega_k$, then updated $\omega_{k+1}$ with the motor model using references from INDI.

   In the training environment, we first updated $\omega_{k+1}$ with the same motor model using $\omega_{\text{ref},k}$ from INDI, then applied thrust using $\omega_{k+1}$.

   Although resulting rotor speeds are identical, the deployment simulator introduces one-step delay in thrust application. With this delay the policy encounters a slower actuator response compared to training and oscillates in the deployment simulator. Aligning the thrust update order in both simulators eliminated the oscillation.

### A.0.2. Arm actuator step response

Notice the difference in step responses across all simulators and real-world, making domain randomization of joint properties necessary.

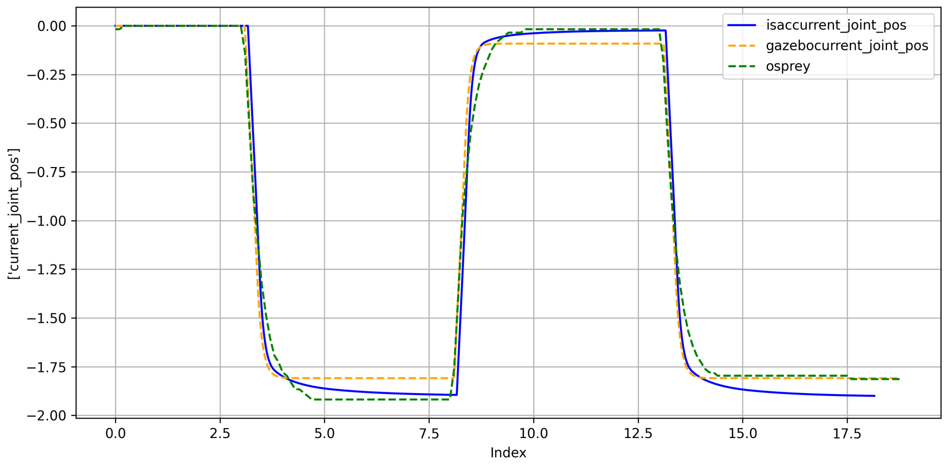


**Figure A.1:** Step response comparison of joints commanded to move the arm along pitch axes, in all - training simulator (*isaaccurrent_joint_pos*), deployment simulator (*gazebocurrent_joint_pos*) and real-world (*osprey*)

### A.0.3. Additional Results

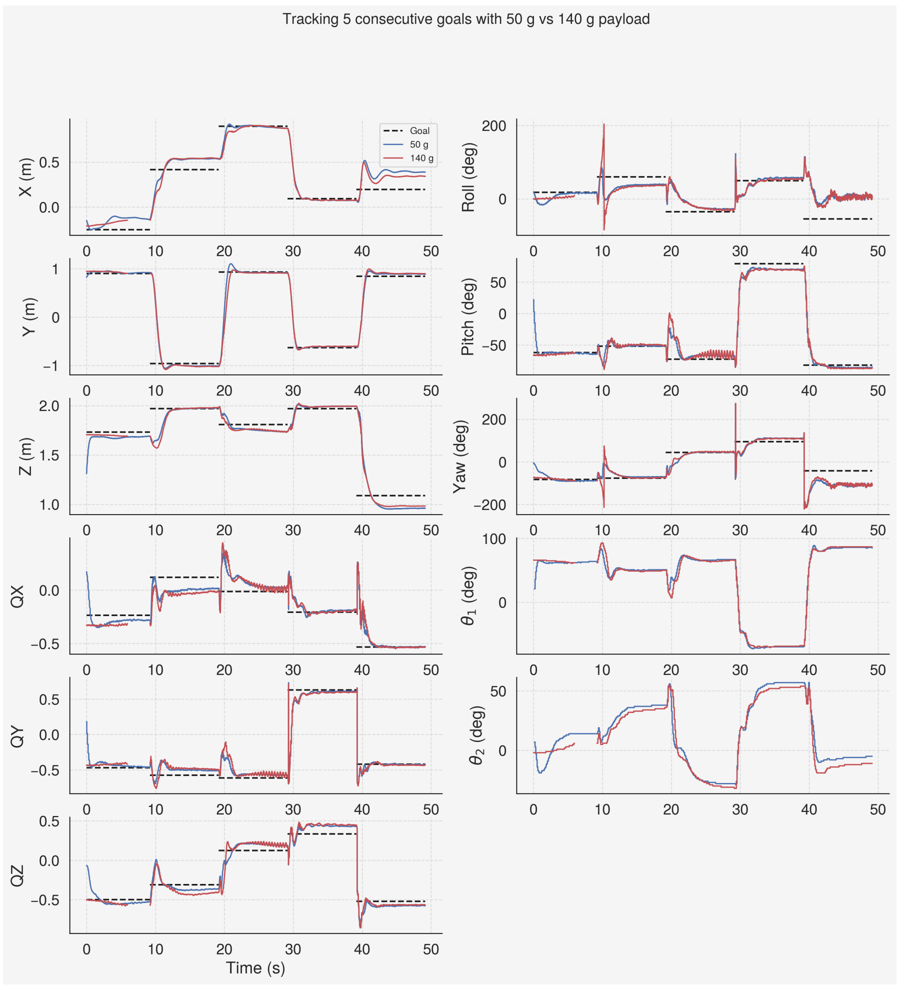


**Figure A.2:** Pose tracking performance in real-world on 5 consecutive goals, with 50 g and 140 g payload.

### A.0.4. PPO Hyperparameters

| | |
|---|---|
| **Policy (Gaussian)** | |
| Network layers | [512, 256, 128] |
| Activations | ELU (all layers) |
| clip_log_std | True |
| log std (min / max / init) | $-20.0\,/\,2.0\,/\,0.0$ |
| **Value (Deterministic)** | |
| Network layers | [512, 256, 128] |
| Activations | ELU (all layers) |
| **PPO agent** | |
| Rollouts | 24 |
| Learning epochs | 5 |
| Mini-batches | 4 |
| Discount factor $\gamma$ | 0.99 |
| GAE $\lambda$ | 0.95 |
| Learning rate | $5 \times 10^{-4}$ |
| Ratio clip $\epsilon$ | 0.2 |
| Value clip | 0.2 |
| Entropy loss scale | 0.005 |
| Value loss scale | 1.0 |
| Grad-norm clip | 1.0 |
| Time-limit bootstrap | True |
| **Preprocessing / Trainer** | |
| State preprocessor | RunningStandardScaler |
| Value preprocessor | RunningStandardScaler |
| Trainer | SequentialTrainer (timesteps = 500,000) |
| Number of environments | 4096 |

**Table A.1:** Key PPO hyperparameters and model configuration (SKRL [38]).

### A.0.5. Manipulator joint parameters

| Parameter | Training sim | Deployment sim | Real world |
|---|---|---|---|
| Control rate (Hz) | 300 | 333 | 300 |
| Effort limit | 1 N-m | 1 N-m | — |
| Velocity limit (rad/s) | 5 | 5 | 10 |
| Tunable friction parameter | 0.04–0.20 (randomized) | 0.10 N-m | — |
| Joint 1 gains $(K_p, K_i, K_d)$ | (1.0, 0.01, 0.3) | (1.0, 0.01, 0.3) | (1.0, 0.0, 0.31) |
| Joint 2 gains $(K_p, K_i, K_d)$ | (1.0, 0.01, 0.1) | (1.0, 0.01, 0.1) | (1.0, 0.0, 0.31) |

**Table A.2:** Arm PID configuration across training simulator, deployment simulator, and real-world experiments.

### A.0.6. Real-world experiment videos

1. End-effector pose control **(Link)**
2. Load-carrying pose control **(Link)**
3. Figure-8 path following (**(Link)**
4. Straight line path following **(Link)**