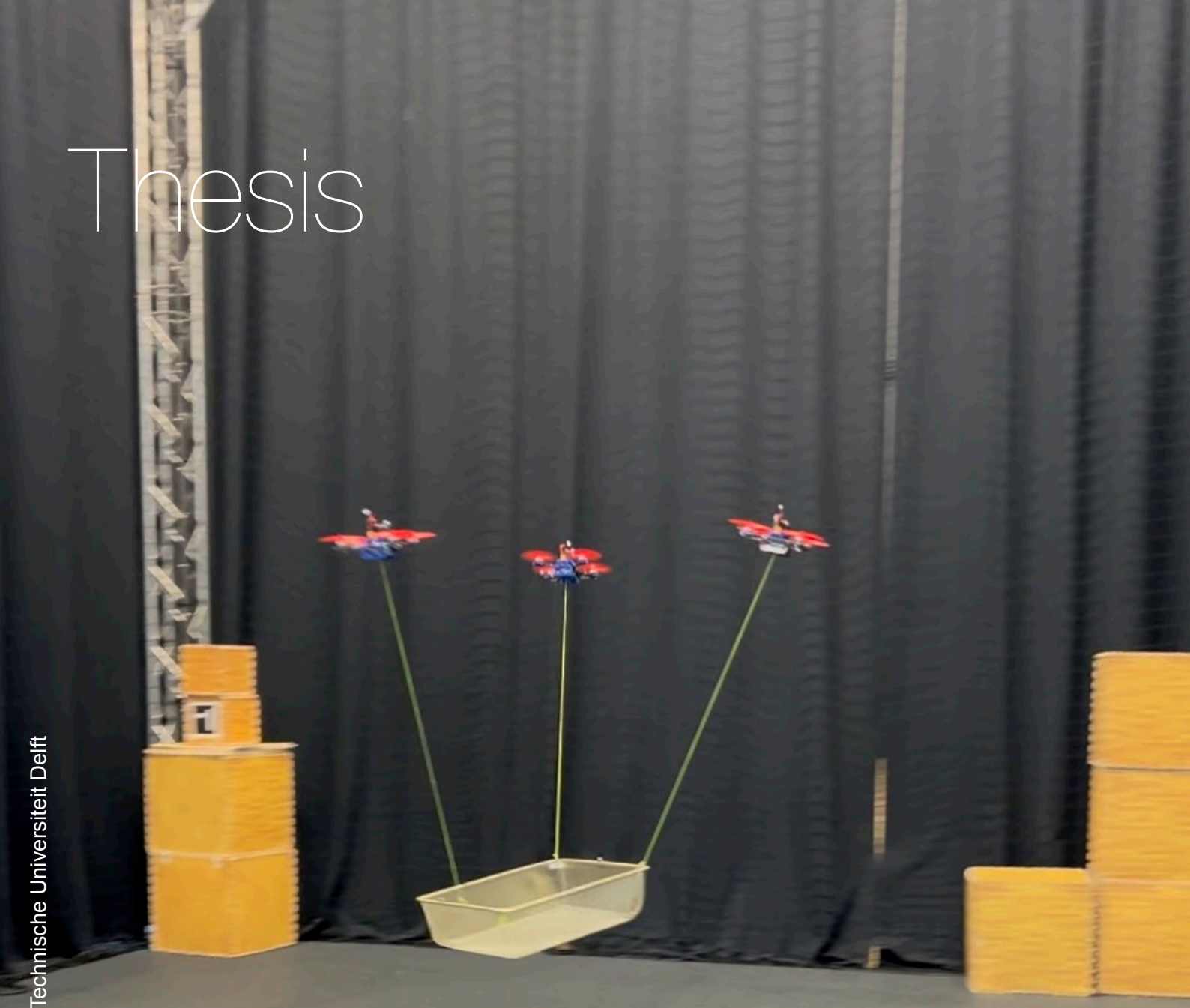
Thesis



Technische Universiteit Delft

Decentralized Real-Time Planning for Multi-UAV Cooperative Manipulation via Imitation Learning

Shantnav Agarwal

TU Delft

Thesis

Decentralized Real-Time Planning for Multi-UAV Cooperative Manipulation via Imitation Learning

by

Shantnav Agarwal

to obtain the degree of Master of Science in Robotics
at the Delft University of Technology,
to be defended publicly on Tuesday, August 28, 2025 at 02:30 PM.

Student number:	5939933	
Project duration:	November 11, 2024 – August 28, 2025	
Thesis committee:	Dr. Sihao Sun,	TU Delft, Daily supervisor
	Prof. Dr. J. A. Morra,	TU Delft, Main supervisor
	Prof. Dr. J. Kober,	TU Delft
	Prof. Dr. J. W. Böhmer,	TU Delft

TU Delft

Acknowledgements

This thesis represents the culmination of two years that have been an incredible journey of exploration and learning. Along the way, I have had the privilege of meeting many inspiring and brilliant individuals whom I would like to thank.

First and foremost, I would like to express my deepest gratitude to Dr. Sihao Sun, whose supervision and support have been paramount in making this thesis possible. I can count numerous instances where his expert guidance and direction enabled me to progress through challenging obstacles. Beyond supervision, Sihao also provided hands-on assistance throughout my real-world experiments.

I would also like to thank Prof. Javier Alonso-Mora for his supervision and for providing me with an incredible research platform. He brought a fresh perspective to my work, offered constructive feedback, and ensured I adhered to manageable timelines to complete this thesis on schedule.

I wish to acknowledge Prof. Julian Kooij for providing me with numerous opportunities throughout my studies at TU Delft. His supervision during my Honours project allowed me to explore the emerging field of acoustics for perception in autonomous vehicles. I would like to express my gratitude to the entire Cognitive Robotics department, where I had the opportunity to interact with and learn from so many talented individuals.

I am especially grateful to my family. I would like to begin by thanking my sister, Anubhuti, who has consistently motivated and counseled me, serving as a constant source of inspiration throughout my life. I extend my heartfelt thanks to my parents for their unwavering belief in me and their support in all my endeavors. I would also like to thank my girlfriend, Surabhi, who has walked alongside me throughout this entire journey.

I would also like to express my gratitude to the numerous friends I made during my time here. Participating in projects and courses together was both joyful and educational. I would particularly like to thank Aleksander, Alexandros, and Jelle, with whom I spent considerable time testing startup ideas and participating in competitions. Last but not least, I would like to thank all my friends back home who supported me and gave advice whenever I needed it.

As my time as a student draws to a close, I reflect on these two years at TU Delft as a period of tremendous growth, learning, and stepping outside my comfort zone. It has been an unforgettable experience that has shaped both my professional trajectory and personal development.

Shantnav Agarwal
Delft, August 2025

Abstract

Collaborative transportation and manipulation of cable-suspended loads by multiple UAVs offer a promising way for expanding UAVs' role in heavy-lifting operations. Existing approaches for collaborative aerial manipulation of a payload along a reference trajectory typically rely either on centralized control architectures or on reliable inter-agent communication. In this work, we propose a novel machine learning–based method for decentralized kinodynamic planning that operates effectively under partial observability and without inter-agent communication. Our method leverages imitation learning to train a decentralized homogenous student policy for each UAV by imitating a centralized kinodynamic motion planner which has access to privileged global observations. The student policy uses physics-informed neural networks that respect the derivative relationships of motion to generate trajectory that are step-wise consistent and guaranteed to be kinematically feasible. During training, the student policies utilize the full trajectory generated by the teacher policy, leading to improved sample efficiency. Therefore, the student policy can be trained in under two hours on modest hardware. We validate our method in both simulation and real-world environments to follow an agile reference trajectory, demonstrating performance comparable to that of centralized approaches.

Contents

List of Figures

1 Introduction

1.1. Background

Unmanned Aerial Vehicles (UAVs) have garnered significant attention in recent years due to their wide-ranging applications across both civilian and military sectors. In the context of the construction industry, UAVs have transformed traditional approaches for project planning and site management by enabling efficient aerial photography, topographic surveying, structural inspections, and real-time safety monitoring. One of the key advantages of UAVs lies in their ability to access hazardous or otherwise inaccessible locations with ease, enhancing both safety and operational efficiency. Furthermore, UAVs present a cost-effective alternative to conventional methods that rely on ground vehicles or manned aircraft, offering reduced operational costs, minimal ground crew requirements, and rapid deployment capabilities. These attributes make UAVs an increasingly vital tool in modern construction workflows as well[1].

The combination of agility, autonomy, and affordability makes UAVs invaluable in construction environments. In particular, heavy-lifting drones have emerged as a more economical and efficient alternative for transporting materials like timber, concrete, steel, or masonry across construction zones [2]. An intriguing example of drone-assisted construction occurred in 2017 in Stuttgart [3], where a drone and robotic arms collaborated to assemble a pavilion using carbon fibre and glass. In this setup, the drone supplied the necessary materials for the robotic arms to perform the assembly.

1.2. Motivation

The limited payload capacity of a single UAV has significantly restricted the broader adoption of UAVs in heavy construction tasks. Because of this limitation, drone-based construction assembly remains in its infancy, primarily limited to experimental projects by academic institutions [4]. To overcome these payload constraints, recent efforts have turned toward leveraging multiple UAVs working in coordination to manipulate and transport heavier loads. Proposed approaches include the use of UAV teams configured with either rigidly attached lifting mechanisms or cable-suspended systems, employing helicopter-style or parallel-propeller configurations [5]. By distributing the weight of the load amongst multiple UAVs, it is possible to lift loads which are beyond the capability of a single UAV. This technique, commonly referred to as Collaborative Aerial Manipulation, promises a cost-effective, adaptive and scalable solution for heavy lifting operations using UAVs.

The use of multiple UAVs for collaborative load manipulation presents substantial technical challenges due to the complexity of the system. In recent years, nonlinear-model based control architectures have been developed for this problem that enable UAVs to fly load along

agile trajectories [6, 7]. While these methods are quite performant, their centralized architecture introduces concerns with scalability and reliability. To operate, these methods require a lot of compute; robust and low-latency communication and precise state estimation for all UAVs involved. These requirements and limitations are significant barriers to the widespread adoption of these methods in the real world. Decentralized controllers have also been developed however their ability to manipulate the full pose of the load along agile trajectories is severely limited. A comprehensive review of the evolution of control strategies proposed in the literature is provided in Chapter 2.

In this thesis we present a solution for *collaborative aerial manipulation* that has performance closer to centralized controllers while operating in a decentralized manner without inter-agent communication.

1.3. Collaborative Aerial Manipulation

Collaborative Aerial Manipulation refers to the coordinated use of multiple aerial robots which jointly transport and manipulate a common object. A representative depiction of this setup is provided in Figure 1.1. The system consists of multiple UAVs, each connected to a rigid *load* via cables. The cables are attached below the centre of mass of each UAV, and at arbitrary points on the load, using universal joints that allow the transmission of forces but not moments.

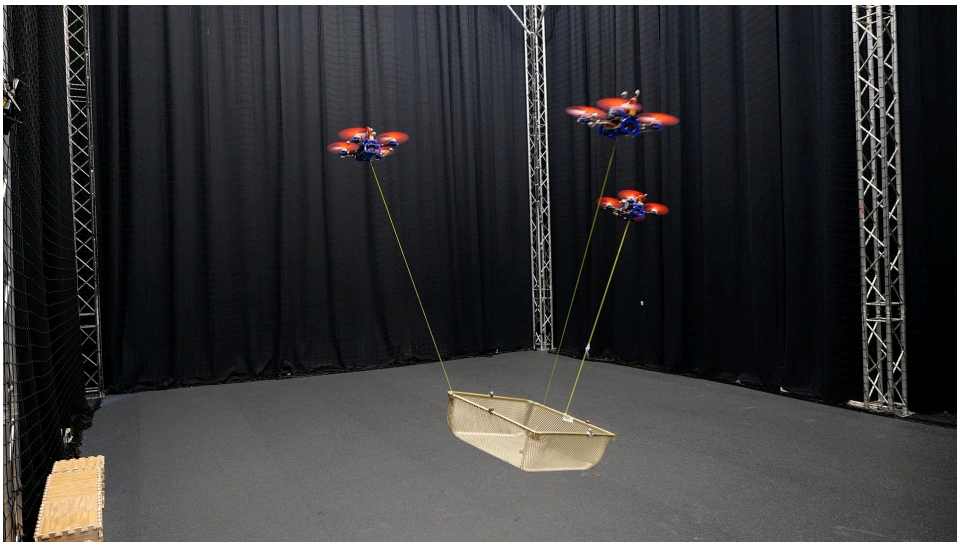


Figure 1.1: Illustration of the multi-drone setup for manipulating a rigid object. The drones are tethered to the object via cables and must work together to move it along a reference trajectory.

Each UAV is modelled as a uni-directional thrust platform, capable of generating thrust solely along its body-fixed positive z_i-axis. The primary objective is for the UAVs to collaboratively control the full pose of the Load—both its position and orientation—so that it tracks a desired trajectory over time.

1.3.1. Decentralized Collaborative Aerial Manipulation

Decentralized control in CAM aims to achieve full-pose control of a Load using multiple UAVs, where each UAV operates independently based on local observations and without direct communication with other UAVs.

This decentralized setting introduces several key constraints and challenges beyond those in CAM:

- **Distributed Control:** Each UAV is governed by an onboard control policy and functions

autonomously, without access to the control inputs or internal states of other UAVs.

- **Local Observations:** Each UAV's control policy relies solely on locally available sensor data, including its own pose and twist, as well as the pose of the Load. No information about the states or actions of other UAVs is accessible.
- **Implicit Coordination:** UAVs must learn to collaborate effectively during the training phase, such that during deployment, coordinated behaviour emerges through physical interactions alone and without any explicit communication channels.

This framework reflects practical limitations in real-world multi-robot systems, where communication constraints and scalability are critical considerations.

1.4. Research Questions

Centralized state-of-the-art methods have demonstrated superior performance in collaborative aerial manipulation compared to decentralized approaches. In this work, we leverage the state-of-the-art centralized expert to train decentralized deep learning policies for the same task, with the objective of narrowing the performance gap between centralized and decentralized methods.

Therefore, the overarching research question in this project is: *Whether it is possible for decentralized student policies to imitate a centralized expert for a highly complex task in a collaborative environment?*. While answering this question, we also encounter the following sub-questions along the way:

- What is a good way within the imitation learning paradigm for the distillation of knowledge from a centralized teacher policy to multiple student agents operating in a decentralized setting?
- In what ways can autonomous agents enhance collaboration and coordination in a decentralized environment without relying on direct communication?
- What design approaches can improve the tracking performance of reinforcement learning (RL) agents?

1.5. Contributions

In literature, state-of-the-art centralised methods have been shown to manipulate the payload along agile trajectories with good pose tracking accuracy. This performance has been beyond the capabilities of decentralized methods until now.

In this thesis, we present a machine learning-based control policy for decentralised collaborative aerial manipulation. We use imitation learning to train this policy from demonstrations collected using a centralised expert. Our proposed policy is decentralized and operates without inter-agent communication. Designed as a kinodynamic planner, the policy can run onboard each individual UAV and leverages only local, egocentric observations to predict its own trajectory.

We use a state-of-the-art centralized nonlinear model predictive controller [7] for collaborative aerial manipulation (CAM) as the expert demonstrator. We use the DAgger algorithm to iteratively collect data in a simulated environment to train our *decentralised student policies*. This policy can then be deployed in the real world without any further fine-tuning.

To the best of our knowledge, this work is the first to develop a communication-free, decentralized control framework for CAM using imitation learning. The trained policies have been validated in simulation and verified through real-world experiments.

2

Related Works

2.1. Cable-driven cooperative aerial manipulation

Numerous control strategies have been proposed for cooperative transportation of suspended payloads using UAV teams. These approaches vary in terms of modelling accuracy, scalability, communication requirements, and capability to regulate the full pose of the payload. Given the focus of this work on decentralized cooperative aerial manipulation, prior methods are categorized into three primary frameworks: centralized approach, decentralized approach with communication, and decentralized approach without communication.

2.1.1. Centralized Approaches

A formation-based geometric control approach was introduced in [8], modelling a point-mass suspended via massless rigid links. This method centrally solves for force equilibrium using full system state information, allowing formation maintenance but neglects payload attitude. Moreover, the approach has only been validated in simulation.

Model Predictive Control (MPC) is commonly used in centralized frameworks due to its ability to handle constraints and optimize control over a receding time horizon. Linear MPC approaches typically model the payload as a point mass and use linearized system dynamics [9, 10]. While computationally efficient, these methods cannot control the payload's orientation and often under perform on dynamic trajectories [11].

Nonlinear MPC (NMPC) techniques have been introduced which enable manipulating the full-pose of the load along agile trajectories. For example, a cascaded NMPC controller has been developed that enables full pose manipulation of the load[6]. However, this controller assumes load dynamics are a magnitude slower than quadrotor dynamics, which reduces control performance on agile trajectories. Recently, in [7], an online kinodynamic motion planner utilizing a load-cable-UAVs model without assumption of timescale separations has been proposed, enabling agile full-pose control.

While centralized controllers have achieved agile and accurate load manipulation, they rely on high-bandwidth, low-latency communication networks with a central coordinator. This assumption is often impractical in real-world scenarios where communication may be unreliable or unavailable. Further, the computational cost of aforementioned approaches grows exponentially with the number of agents thereby limiting scalability. Hence, the community has also explored decentralized methods for cooperative aerial manipulation.

2.1.2. Decentralized Approaches with Communication

A distributed model predictive control framework has been developed which demonstrates performance comparable to centralized MPC while distributing the computational burden across agents[12]. The approach assumes a fully-connected communication graph, wherein each UAV shares its previous control inputs with neighbouring agents and independently optimizes its own input sequence at each iteration. Though validated only in simulation, the framework exhibits scalability and coordination capabilities. While the method is scalable, it still relies on continuous inter-agent communication, motivating the need for decentralized control strategies that operate without explicit communication.

2.1.3. Decentralized Approaches without Communication

A decentralized control scheme combining distance-based formation control with incremental nonlinear dynamic inversion (INDI) is proposed in [13]. UAVs compute local acceleration commands based solely on relative position measurements, eliminating the need for global positioning or inter-agent communication. This method, however, does not regulate the payload's orientation. Master-slave architectures have also been employed, where a designated master UAV generates a reference trajectory, and slave UAVs follow using force-based admittance control [14, 15]. These approaches are limited in their ability to manipulate the payload's full pose and have not been extended beyond two-UAV systems.

2.2. Learning from Privileged Experts

In this project, we use privileged learning [16] to train our decentralized policy. Privileged learning improves policy training by decomposing the process into two stages. First, we develop a teacher policy with access to privileged information and greater computational resources. Such privileges are feasible in simulation or controlled environments but may not be available in real-world settings. These advantages make it easier to train the teacher policy effectively. Next, we train a student policy that operates under real-world constraints and observations. The student learns by imitating the teacher policy (Section 3.1) through online supervision, making the learning process more efficient.

Privileged learning has been widely used in literature and two relevant examples are provided in the following sections.

2.2.1. Quadrupedal Locomotion

Lee et al. [17] trained a quadrupedal robot to operate in rough terrain using privileged learning.

In this approach, a teacher policy was first trained with access to privileged information, i.e. ground-truth terrain geometry and real-time contact dynamics between the robot and the terrain. This information was essential for enabling the policy to effectively master locomotion.

The resulting expert policy was then used to train a student controller via the DAgger algorithm. The student policy, a proprioceptive controller, relied solely on onboard sensors available on the real robot.

Crucially, The authors state *training a rough-terrain locomotion policy directly via reinforcement learning was not successful*.

Their experiments showed that the policy without privileged information failed to achieve stable locomotion in a reasonable amount of time due to sparse reward signals. In contrast, transferring knowledge from the privileged expert to the student policy enabled robust adaptation to challenging real-world environments, demonstrating the effectiveness and practicality of the privileged learning approach.

2.2.2. Acrobatic Drone Control

An example of privileged learning applied in the context of UAVs is the work by Kaufmann et al. [18]. They developed a sensorimotor control policy that could execute acrobatic manoeuvres on a quadcopter. In their case, the teacher policy was an MPC controller which had access to the current state of the quadcopter estimated using motion capture systems.

The student policy in contrast is a sensorimotor controller that uses feature tracks provided by a visual odometer system to ascertain its current state. By distilling the teacher's state-aware expertise into the vision-based student, the system achieved robust acrobatic flight without requiring motion-capture systems or laboratory infrastructure. The policy was successfully deployed in unstructured outdoor environments.

3

Background

3.1. Imitation Learning

Imitation learning is a type of machine learning where an agent learns to perform tasks by mimicking or copying the behaviour of a human or another agent, rather than learning through explicit rewards or trial-and-error [19]. The primary goal of imitation learning is for the agent to observe expert demonstrations and then generalize from those demonstrations to perform similar actions on its own in similar situations.

It can be divided into two main types:

- Behaviour Cloning (BC): In this approach, the agent learns a mapping from states to actions directly by training on the observed data. Essentially, it learns to predict the actions an expert would take in given states.
- Inverse Reinforcement Learning (IRL): Here, the agent tries to infer the underlying reward function that the expert is optimizing based on their behaviour, and then uses reinforcement learning to maximize this inferred reward function.

Imitation learning has been successfully applied in many domains related to robotics. Within the autonomous vehicle industry, manufacturers such as Tesla collect demonstrations from human drivers at scale and then use Behaviour Cloning to train their algorithms. Another benefit of using imitation learning is to instil desired behaviours in the student policy - Humanoid robots are trained using the AMP [20] framework to learn human-like walking behaviour.

In our project, observation & action pairs of the expert are readily available. Therefore, we focus primarily on Behaviour Cloning, and it's derivative algorithms for training the student policy.

3.1.1. Behaviour Cloning

Behaviour Cloning (BC) is a supervised learning approach where a policy π_θ is trained to mimic an expert (teacher) policy π^* by minimizing the discrepancy between their actions. Given a dataset of expert demonstrations $\mathcal{D} = \{(s_i, a_i^*)\}_{i=1}^N$, where s_i represents states and a_i^* the corresponding expert actions, the policy is trained using standard supervised learning to minimize the loss:

$$\mathbb{E}_{(s,a^*)\sim\mathcal{D}}\left[\|\pi_\theta(s) - a^*\|^2\right] \tag{3.1}$$

However, BC suffers from compounding errors — it assumes that states in deployment will follow the distribution seen during training. In reality, small errors in action selection can lead to states that deviate from the expert's trajectory, where the policy has little training data, resulting

in error accumulation. The upper bound on the performance gap between the learned policy and the expert scales as:

$$J(\pi^*) - J(\pi_\theta) = \mathcal{O}(T^2\epsilon) \tag{3.2}$$

where T is the task horizon and ϵ is the per-step imitation error. The quadratic dependence on T makes BC unsuitable for long-horizon tasks.

3.1.2. DAgger

Dataset Aggregation (DAgger) [21] improves upon BC by iteratively collecting new data from the expert to correct errors. Instead of relying solely on the initial dataset, DAgger alternates between training the policy and querying the expert when the learned policy makes mistakes. At iteration i, the dataset is augmented with states encountered by π_{θ_i}, labelled with expert actions $\pi^*(s)$, leading to an expanding dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \cup \{(s, \pi^*(s))\}$. The policy is then re-trained on the growing dataset. This iterative correction reduces error accumulation, improving the performance bound to:

$$J(\pi^*) - J(\pi_\theta) = \mathcal{O}(T\epsilon) \tag{3.3}$$

which is linear in T, a significant improvement over BC. By actively adapting the training set, DAgger ensures the policy learns robustly across a wider distribution of states, making it much more effective for long-horizon tasks.

3.1.3. Centralized Expert Supervises Multi Agents (CESMA)

CESMA is a 2 stage framework that was proposed by Lin et al. [22].

- Centralized Training: A single centralized expert (with full observability) is trained using any RL algorithm (e.g., DDPG, DQN) to solve the multiagent problem in the joint action-observation space.
- Decentralized Execution via Imitation Learning: Independent decentralized agents are trained via imitation learning (e.g., DAgger) to mimic the centralized expert's actions using only local observations.

The authors present a multiagent adaptation of the DAgger algorithm (Section 3.1.2), demonstrating that its original theoretical guarantees remain applicable in collaborative multi-agent settings. Building on this framework, they further extend DAgger to incorporate explicit communication protocols, enabling agents to share observations or policy outputs during training.

Since we consider our policies to be strongly homogeneous (Section 3.4), where policies and reward structures are identical across agents, we refine their multiagent DAgger approach to exploit this symmetry.

After obtaining a centralized expert policy π^*, we initialize a shared agent policy π, which is used by all M agents. We also initialize a dataset D to store observation-label pairs.

As the agents interact with the environment, they collectively observe

$$o = (o_1, \dots, o_M) \tag{3.4}$$

where o_i is the observation for agent i. The expert provides an action label

$$a^* = \pi^*(o) \tag{3.5}$$

with

$$a^* = (a_1^*, \dots, a_M^*), \tag{3.6}$$

where a_i^* is the expert labelled action for agent i. At each time step, we store the pair (o, a^*) in the dataset D.

Once the dataset D has reached a sufficient size, we sample a batch

$$\{(o^{(\beta)}, a^{*,(\beta)})\}_{\beta=1}^{B} \tag{3.7}$$

from D. Since all agents share the same policy π, we construct a unified training set

$$\bigcup_{i=1}^{M} \{(o_i^{(\beta)}, a_i^{*,(\beta)})\}_{\beta=1}^{B} \tag{3.8}$$

and use it to train the shared policy via supervised learning.

3.2. Markov Decision Processes

An MDP is a model of an agent interacting synchronously with the world. The agent takes as input the current state of the environment and based on that executes an action. This action then affects the state of the world which is then presented to the agent again in the next time step. In an MDP framework, it is generally assumed that the agent knows the current state of the environment without any uncertainty [23]. However, the affect of the actions of an agent on the environment are uncertain and introduce stochasticity to the process. An MDP can be summarised as:

- S is a finite set of states of the world.
- A is a finite set of actions.
- $T(s, a, \tilde{s})$ is the state-transition function to get the probability of ending in state $\tilde{s}$ for taking the action a when the world is in state s.
- $R(s, a)$ is the reward function giving the expected immediate reward for taking an action a in state s.

3.3. Partially Observable Markov Decision process

The student policies operate in a decentralized partially observable environment wherein each UAV is equipped with its own student policy. Further, this student policy can only see the local observations of the ego-UAV and cannot communicate with other UAVs. In order to behave effectively in a partially observable world, it is necessary to use the memory of previous observations and actions to aid in the disambiguation of the states of the world. POMDPs provide a systematic framework to model such an environment[23]. A POMDP can be summarised as follows:

- S, A, T and R are analogous to the definitions for an MDP.
- $O(\tilde{s}, a, o)$ is the probability of receiving an observation o when the agent takes an action a and enters state $\tilde{s}$.

3.4. Homogeneous Agents

3.4.1. Introduction

In MultiAgent Reinforcement Learning (MARL), homogeneous agents refer to agents that possess identical properties, capabilities, and policies. They are interchangeable, meaning their

roles and behaviours are symmetric within the environment. This homogeneity simplifies coordination and learning, as each agent follows the same decision-making framework without requiring differentiation in roles or specialized behaviours.

Homogeneous agents have the following characteristics:

1 Shared Policy: Homogeneous agents have the same or nearly identical policies.

2 Identical Observations and Actions: They operate on similar observation spaces and have identical action spaces.

3 Interchangeability: Any agent can perform the role of another without affecting the system's functionality.

4 Scalability: They are suited for scenarios involving large groups of agents, such as swarms or fleets.

Weakly homogeneous agents

In contrast, weakly homogeneous agents only have the same policy architecture but can have different weights [24].

In practical applications, weekly homogeneous agents are particularly useful in environments where agents share identical action spaces and observation modalities, yet operate in distinct roles. A representative example is a football team, where each player can be considered an agent. Although all outfield players possess the same set of actions and observations—thus allowing for a shared policy architecture—they assume different responsibilities based on their positional roles. For instance, attacking players typically position themselves further up the field and focus on goal-scoring opportunities, whereas defenders are primarily tasked with preventing the opposing team from scoring. In such a setting, adopting a weekly homogeneous policy can improve performance by tailoring behaviour to role-specific demands while maintaining architectural consistency. Notably, the goalkeeper constitutes a unique case: unlike outfield players, the goalkeeper can use their hands within a restricted area, which introduces an action set distinct from the others. As such, the goalkeeper's policy would be considered heterogeneous relative to the rest of the team.

Strongly homogeneous agents

Strongly homogeneous agents are policies that have the same architecture and weights [24].

Strongly homogeneous agents are particularly advantageous in multiagent systems where optimal behaviour requires identical policies across agents. This approach is well-suited to environments such as cooperative drone swarms, where agents share identical physical models, dynamics, and constraints. In our framework, the student policies are strongly homogeneous. This assumption is justified by the structure of the centralized Nonlinear Model Predictive Control (NMPC) teacher policy, which enforces uniformity in decision-making due to the agents' shared operational and dynamical characteristics.

3.5. Centralized Training Decentralized Execution (CTDE)

In multiagent reinforcement learning (MARL), the concept of Centralized Training with Decentralized Execution (CTDE) involves training agents using shared information across all agents, while ensuring that each agent's decision-making process relies solely on its own observations. This allows agents to operate independently once deployed. For instance, although training may leverage collective data from all agents to train their policies, each agent's policy is structured to function using only its local inputs, enabling fully decentralized operation in real-time settings. CTDE strikes a balance by combining the advantages of centralized

learning—such as leveraging global context and coordination—with the practical necessity of decentralized action. This framework is especially prevalent in deep MARL approaches because it supports the use of more informative value functions during training by incorporating data unavailable during execution. A common example is multiagent actor-critic methods, where a centralized critic uses the joint observation history to more accurately estimate value functions. Once training concludes, the critic is no longer needed, and each agent acts based on its own locally conditioned policy [24]. A number of algorithms have been developed to use the CTDE paradigm and deliver strong performance in collaborative multiagent test beds - MAPPO [25], MADDPG [26] etc.

Decentralized training and execution involves training agent policies in a completely decentralized fashion, meaning that the policies do not rely on centrally shared information or mechanisms. This approach is particularly well-suited for multiagent reinforcement learning (MARL) in environments where centralized coordination is impractical or impossible. A prime example is the financial market, where individual trading firms operate without access to the internal strategies of their competitors. However, a key challenge of this method lies in the dynamic nature of the environment—since all agents are learning simultaneously, the behaviour of the environment becomes non-stationary, which can hinder stable policy learning [24].

As stated previously, the goal of this research is to approach the problem from a Privileged & Imitation Learning perspective because of it's lower computational requirements and ability to impart desired behaviours. A pure RL approach utilizing CTDE algorithms is a complementary approach to solving this problem. To our knowledge, no existing solutions to this exact problem exist yet.

4

Methodology

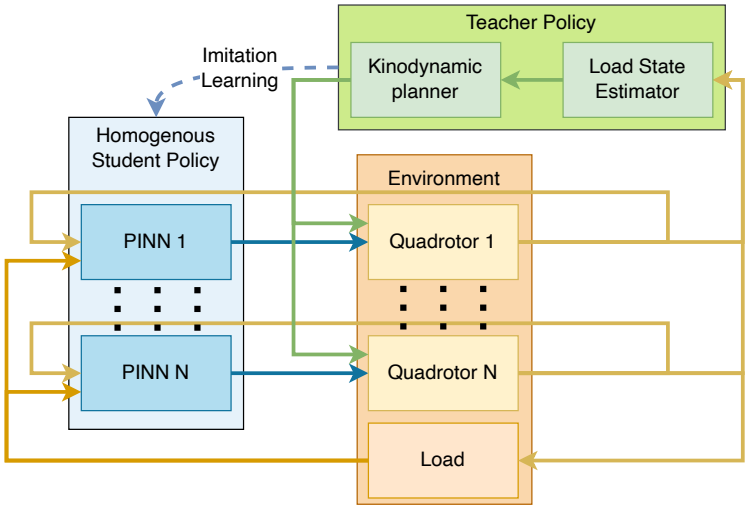


Figure 4.1: A high-level overview of the training process.

We enable decentralised cooperative aerial manipulation by training decentralised student policies to imitate a centralised teacher policy. Following the paradigm introduced in Learning by Cheating [16], we employ a privileged teacher policy to collect demonstrations, which are subsequently used to train the unprivileged student policy. The centralised teacher policy has access to privileged observations, including the complete state information of all UAVs. In contrast, the decentralised student policies learn to produce equivalent actions using only ego-specific local observations. Figure 4.1 provides a representative illustration of this framework. The following sections detail the teacher and student policies, as well as the training environment.

4.1. Notations

We consider a system composed of n unmanned aerial vehicles (UAVs) and a single rigid load body. The pose and twist of the i^{th} UAV are denoted by $[\boldsymbol{p_i} \quad \boldsymbol{q_i}]$ and $[\boldsymbol{v_i} \quad \boldsymbol{\omega_i}]$, respectively, where $\boldsymbol{p_i} \in \mathbb{R}^3$ and $\boldsymbol{q_i} \in \mathbb{S}^3$ represent the position and orientation (quaternion), and $\boldsymbol{v_i}, \boldsymbol{\omega_i} \in \mathbb{R}^3$ denote the linear and angular velocities. We use bold lowercase letters to denote vectors, and bold capitalized letters for matrices; otherwise, variables are scalars.

Similarly, the pose and twist of the Load are represented as $[\boldsymbol{p_L} \quad \boldsymbol{q_L}]$ and $[\boldsymbol{v_L} \quad \boldsymbol{\omega_L}]$, with analogous definitions.

A load-attached reference frame, denoted $\mathcal{L}$, is defined such that its origin coincides with $\boldsymbol{p_L}$, and its orientation is aligned with the inertial frame $\mathcal{I}$.

Each UAV has an associated body-fixed frame $\mathcal{B}_i = [\boldsymbol{O_i} \quad \boldsymbol{x_i} \quad \boldsymbol{y_i} \quad \boldsymbol{z_i}]$, where the origin $\boldsymbol{O_i}$ is located at $\boldsymbol{p_i}$, and the z_i-axis points vertically upward, coinciding with the thrust direction.

Unless otherwise stated, all vectors are expressed in the inertial frame $\mathcal{I}$. Frame-specific quantities are indicated using superscripts when necessary.

4.2. Centralized Teacher Policy

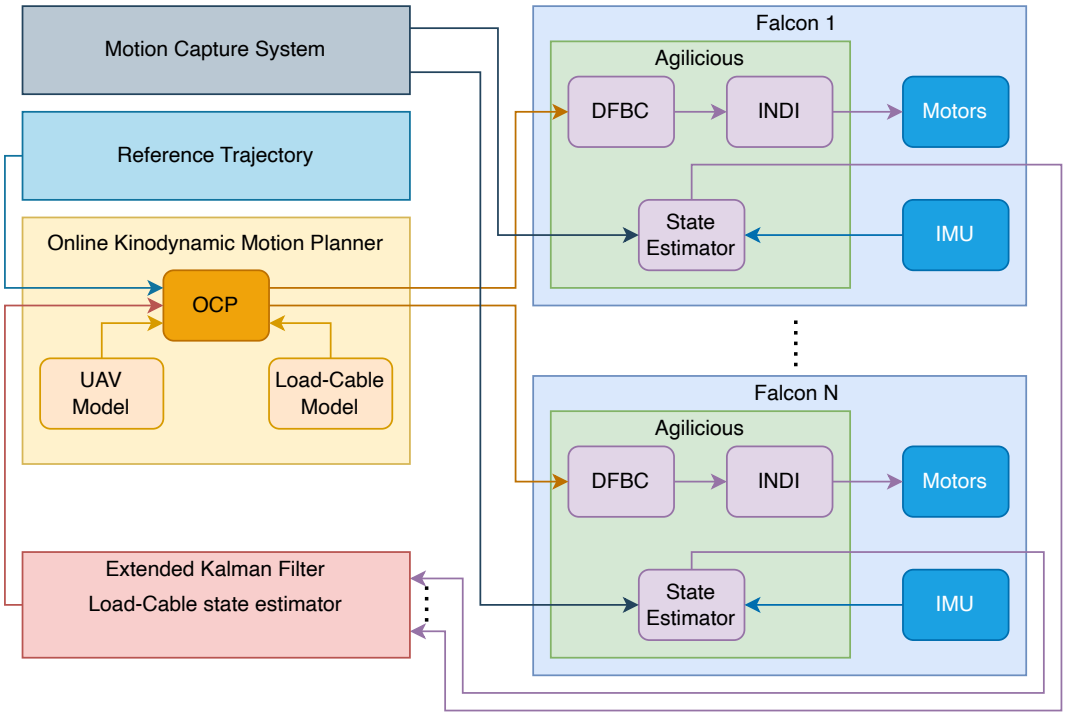


Figure 4.2: The control overview of the teacher policy. It receives as input the state of the environment, Load's reference trajectory and the dynamic model of the environment. As output, it simultaneously produces a receding-horizon trajectory for all UAVs. Each UAV tracks its trajectories using onboard low-level controllers.

We use the planning algorithm introduced in [7] as the teacher policy. This algorithm is a state-of-the-art centralized method for multi-lifting systems. It enables highly agile and robust full-pose control for a cable-suspended load. Given the reference trajectory of the load, the teacher policy solves an optimization-based kinodynamic motion planning problem and generates receding-horizon reference trajectories for each UAV at 10 Hz.

This algorithm consists of a non-linear UAV-load-cable dynamic model. The dynamic model describes the 6-DoF motion of the load as well as the cables attached to it. UAVs are modelled as a standard rigid body which can apply thrust along the z-axis of the body-fixed frame.

The algorithm acts as a centralized kinodynamic motion planner that optimizes a discrete finite-time optimal control problem using the Acados framework [27]. It generates smooth reference trajectories for all quadrotors in a receding horizon fashion while accounting for the

dynamic coupling between the load, cables and the quadrotors.

Note

The trajectories produced by both the teacher and student policies share the same structure, as defined in Equation 4.7. During training, the student policy outputs trajectory components at the same spatio-temporal nodes used by the teacher policy in solving its optimal control problem.

The load pose, twist and the cable directions are estimated using an Extended Kalman Filter. Not only does this filter negate the need for motion capture systems to track the load, but it also estimates the tensions in the cable which are required by the OCP solver.

The receding horizon trajectories generated by the OCP are tracked using low-level controllers on the UAV. The low-level controllers feature a Differential-Flatness Based Controller [28] followed by an Incremental Nonlinear Dynamics Inversion controller [29]

An overview of the teacher policy is provided in Fig. 4.2. This policy was originally implemented as a Simulink model. We reimplemented the policy as Python ROS Node using the Acados framework. This Python-based reimplementation significantly simplified the process of collecting demonstrations across hundreds of randomized trajectories, thereby streamlining the training of student policies. Details of the implementation of the teacher policy are provided in Appendix A.3. We omit a detailed explanation of the teacher policy for readability and refer to the original work [7].

4.3. Decentralised Student Policy

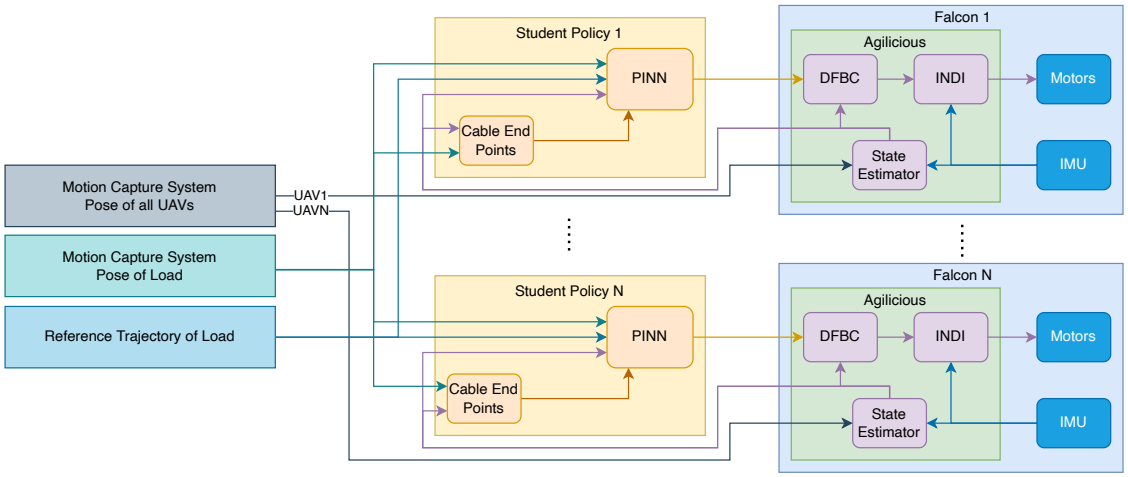


Figure 4.3: The control overview of the student policy. Each policy acts in a decentralised manner. The architecture of the PINNs is shown in Figure 4.4.

While the centralized policy demonstrates high agility and robustness against dynamic model uncertainties, it relies on access to the full state information of all UAVs and the load. In contrast, the student policy learns to generate trajectories for the ego UAV in a decentralised manner, using only the ego UAV's state and the load's pose. Notably, the student policy shares the same low-level controllers as the teacher policy. It is implemented as a strongly homogeneous machine learning policy—meaning all UAVs run identical copies of the policy, with the same architecture and weights [24].

The architecture of the student policy is depicted in Figure 4.4, and an overview of the

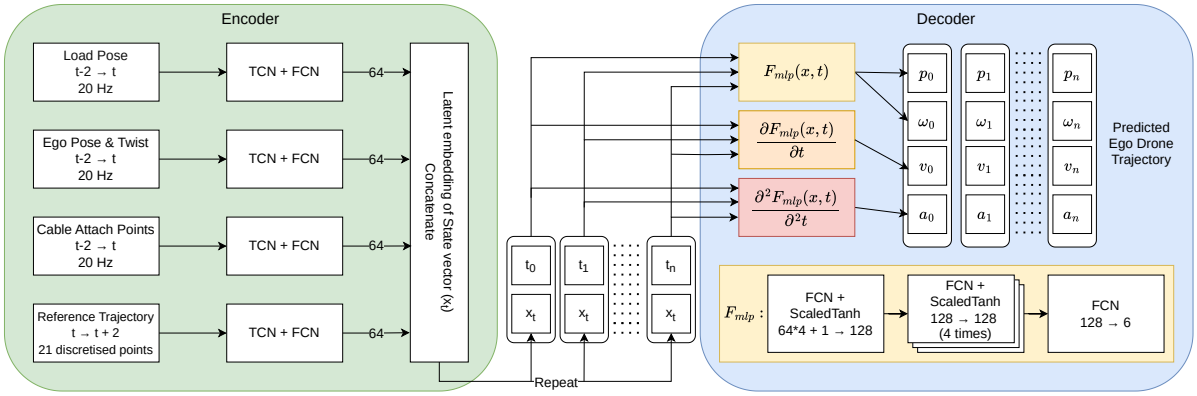


Figure 4.4: The architecture of the student policy is made up of two parts - Encoder (Sec. 4.3.1) & Decoder (Sec. 4.3.2). The Encoder network maps the observation histories to a latent vector x_t. This latent vector is then copied for all nodes in the horizon and passed to the Decoder network.

overall control framework is illustrated in Figure 4.3. The policy operates at a control frequency of 10 Hz, consistent with the teacher policy.

4.3.1. Observation Space

The student policy has access to the following information:

$$\boldsymbol{o}_i = \begin{bmatrix} \boldsymbol{p}_L^{\mathcal{L}} & \boldsymbol{q}_L^{\mathcal{L}} & \boldsymbol{p}_i^{\mathcal{L}} & \boldsymbol{v}_i^{\mathcal{L}} & \boldsymbol{q}_i^{\mathcal{L}} & \boldsymbol{\omega}_i^{\mathcal{L}} & \boldsymbol{p}_{C1,i}^{\mathcal{L}} & \boldsymbol{p}_{C2,i}^{\mathcal{L}} \end{bmatrix} \tag{4.1}$$

The decentralised policy receives the *Pose* of the load as well as the *Pose* and *Twist* of the ego UAV. Notably, the policy does not receive any information about the other UAVs in the team.

To enable generalization across environments, all observations are expressed in the load frame $\mathcal{L}$. This makes the policy invariant to absolute world position allowing it to operate in spaces of arbitrary size.

It is demonstrated in [30] that leveraging load dynamics as implicit communication effectively facilitates cooperative load transportation. Their method encodes the load's state via the positions and velocities of eight bounding-box vertices, where agents' actions alter these dynamics and influence others' strategies, enabling scalable multiagent collaboration.

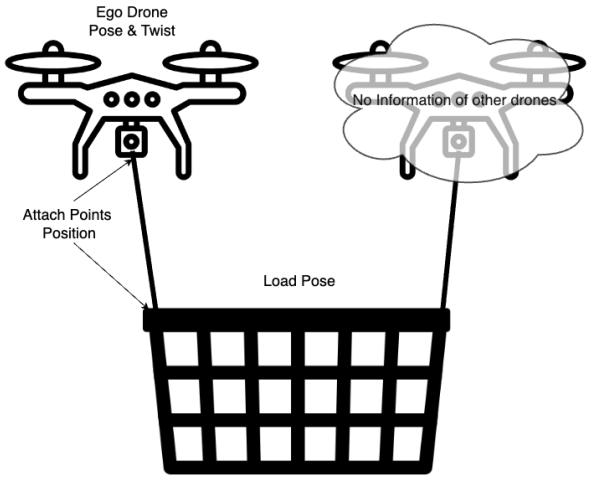


Figure 4.5: Illustration of the observation space for the student policy.

Similarly, we provide the student policy with the position of the ends of the cable on the load ($\boldsymbol{p}_{C1,i}^{\mathcal{L}}$) and on the UAV ($\boldsymbol{p}_{C2,i}^{\mathcal{L}}$) The cable attach points, represented in the load-fixed frame $\mathcal{L}$, convey both the spatial position where each UAV applies force and the direction of that force. The exact location of these points on the load defines the moment arm, which determines the torque generated around the load's centre of mass. This information is crucial for accurately controlling the load's pose during cooperative manipulation. The positions of these points are computed from known attachment positions in the UAV and load frames:

$$\boldsymbol{p}_{C1,i} = \boldsymbol{p}_L + \boldsymbol{R}_L \boldsymbol{\rho}_i^{\mathcal{L}}, \quad \boldsymbol{p}_{C2,i} = \boldsymbol{p}_i + \boldsymbol{R}_i \boldsymbol{r}_i^{\mathcal{B}_i}, \tag{4.2}$$

where $\boldsymbol{R}_L$, $\boldsymbol{R}_i$ is the rotation matrix of the load and the UAV, respectively. $\boldsymbol{\rho}_i^{\mathcal{L}}$ represents the vector from load origin to the i^{th} UAV's cable attach point on the load. Similarly, $\boldsymbol{r}_i^{\mathcal{B}_i}$ represents the vector from the ego UAV's origin to the cable attach point.

Instead of using quaternions, we represent rotations using the 6D continuous formulation proposed by Zhou et al. [31]. This representation avoids gimbal lock and discontinuities inherent in Euler angles or quaternions and therefore improves differentiability and learning stability.

To reason under partial observability and infer the intentions from other agents, the policy uses a history of past observations. Specifically the observation matrix at time t is defined as

$$\boldsymbol{O}_i = \left[\boldsymbol{o}_{i,t}, \boldsymbol{o}_{i,t-dt}, \dots, \boldsymbol{o}_{i,t-Ndt}\right] \tag{4.3}$$

where $\boldsymbol{o}_{i,t}$ is the observation vector of the i-th UAV at time t.

Typically, time parameterized representations of future reference states help the agent develop a deeper understanding of the scenario, leading to improved performance [32]. Therefore, we provide the policy with the reference trajectory for the next 2 seconds at $N = 21$ nodes (defined in Equation 4.12), which are the same references for the teacher policy, thereby ensuring that both the teacher and student get the same reference data as input during training.

$$\boldsymbol{Y} = [\boldsymbol{y}_0 \quad \boldsymbol{y}_1 \dots \boldsymbol{y}_N] \tag{4.4}$$

where

$$\boldsymbol{y}_j = \left[\boldsymbol{p}_L^{\mathcal{L}} \quad \boldsymbol{v}_L^{\mathcal{L}} \quad \boldsymbol{a}_L^{\mathcal{L}} \quad \boldsymbol{q}_L^{\mathcal{L}} \quad \boldsymbol{\omega}_L^{\mathcal{L}}\right]_j, \; j \in \{0, \dots, N\}. \tag{4.5}$$

Here $\boldsymbol{a}_L$ represents the linear acceleration of the load.

To process sequential data of observation histories and reference trajectories, we use Temporal Convolutional Networks (TCN) [33], which offer longer memory, stable gradients, and fast, parallel inference. Then the outputs of the TCN network are passed through a dense layer to create a vector of latent embeddings. The four streams of information as shown in Figure 4.4 are recorded at different time steps and therefore processed using separate TCN networks. The outputs of all TCNs are concatenated into a latent vector $\boldsymbol{x}_i$ that encodes the current state of the system as well as the desired future reference states:

$$\boldsymbol{x}_i = \pi_{\text{encoder}}(\boldsymbol{O}_i, \quad \boldsymbol{Y}) \tag{4.6}$$

4.3.2. Action Space

The student policy, as an online planner, directly generates reference trajectories for the ego UAV. While training, we use the output of the centralized teacher policy's online planner directly as the target for the student policy. Therefore, the action space for the i-th UAV can be represented by the following equations. Note that we omit the subscript i for readability.

$$\boldsymbol{U} = [\boldsymbol{u}_0 \quad \boldsymbol{u}_1 \dots \boldsymbol{u}_N] \tag{4.7}$$

with

$$\boldsymbol{u}_j = [\boldsymbol{p} \quad \boldsymbol{\omega} \quad \boldsymbol{v} \quad \boldsymbol{a}]_j, \quad j \in \{0, \dots, N\} \tag{4.8}$$

where $\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{a}$ represent desired position, velocity and acceleration; $\boldsymbol{\omega}$ represents the desired body rates of the ego UAV. The onboard low-level controller typically employs a differential-flatness-based method to follow the reference trajectory generated by the planner for the UAV. Incorporating higher-order derivatives of the reference trajectory, such as velocity and acceleration, can significantly improve tracking performance [34].

A key challenge in predicting higher-order terms with a learning-based policy is ensuring kinematic consistency. The network may generate trajectories wherein the time derivative of position deviates from the predicted velocity, and likewise, the derivative of velocity from the predicted acceleration. To address this problem, we propose the use of Physics Informed Neural Networks (PINNs) [35] with architectural constraints applied as described below:

$$[\boldsymbol{p}_j, \quad \boldsymbol{\omega}_j] = F_{mlp}(\boldsymbol{x}_i, t_j) \tag{4.9}$$

$$\boldsymbol{v}_j = \frac{\partial F_{mlp}(\boldsymbol{x}_i, t_j)}{\partial t} \tag{4.10}$$

$$\boldsymbol{a}_j = \frac{\partial^2 F_{mlp}(\boldsymbol{x}_i, t_j)}{\partial t^2} \tag{4.11}$$

where t is the instantaneous time and t_j is the timestamp of the j-th node in the UAV reference trajectory calculated using the equation:

$$t_j = t_{j-1} + 0.01 + 0.009 \cdot (j-1), \quad t_0 = t, \quad j \in \{0, \dots, N\} \tag{4.12}$$

We therefore employ PINNs as the decoder (Fig. 4.4) to generate trajectories from latent vectors that are inherently feasible, as they explicitly enforce kinematic constraints, i.e., that velocity is the derivative of position and acceleration is the derivative of velocity.

To facilitate the convergence of student policies while using PINNs, we made several design choices after studying the available literature on effectively training PINNs [36]. First, we implemented moderately deep networks with 5 hidden layers to balance expressive capacity with trainability. Second, we adopted scaled Tanh activation functions between layers, which preserve smooth second-order derivatives critical for PDE residual calculations while adaptively controlling output ranges to mitigate vanishing gradients [37]. Third, we used Xavier initialization to ensure stable forward/backward passes [38].

4.3.3. Loss Function

We employ a loss function based on the mean squared error (MSE) between the predicted values of all four action components, as defined in Equation 4.8. The loss is aggregated across each prediction node of the policy.

The teacher is a model-based policy that generates trajectories which are temporally coherent. Specifically, each trajectory predicted by the teacher always originates from the UAV's current state and is guaranteed to be kinematically valid. Consequently, the student policies implicitly learn to produce trajectories that also begin from the UAV's current location. Kinematic feasibility in the student outputs is enforced through architectural constraints imposed by the physics-informed neural networks (PINNs), as described in Equation 4.9. The temporal consistency of the student-predicted trajectories for real-world experiments is illustrated in Figure 6.3e.

During training, we leverage the outputs of all nodes in the teacher policy at each prediction step to supervise the student policies. Combined with the homogeneous structure of our architecture, this strategy enables highly sample-efficient learning.

4.4. Training Environment

We train the student policies exclusively in simulation. The trained policies are directly deployed in the real world using zero-shot sim-to-real transfer. We employ the DAgger algorithm (Section 3.1.2) for training the student policies. DAgger is an iterative imitation learning algorithm wherein each iteration consists of two primary phases: data collection and policy training. During data collection, either the teacher or the student policy is used to interact with the environment to collect demonstrations. Afterwards the policy is trained to predict the teacher's action given the corresponding observations as input.

In the initial iteration, only the teacher policy is used to determine actions. In subsequent iterations, the probability of using the student policy to control the system is gradually increased. This approach enables the generation of a more diverse dataset that encompasses a wider range of system states, thereby enhancing the robustness of the learnt student policy [21].

4.4.1. Directed randomised perturbations

To further improve the quality and diversity of the training dataset, we introduce randomized external disturbances during the data collection phase. Specifically, at the start of an episode, a random force-torque vector is selected to define the disturbance direction. During each time step of the episode, a wrench with random magnitude is applied along this predefined direction, effectively simulating a consistent yet stochastic perturbation. This method promotes the exploration of a broader range of system dynamics, which contributes to the robustness and generalization capability of the resulting student policy.

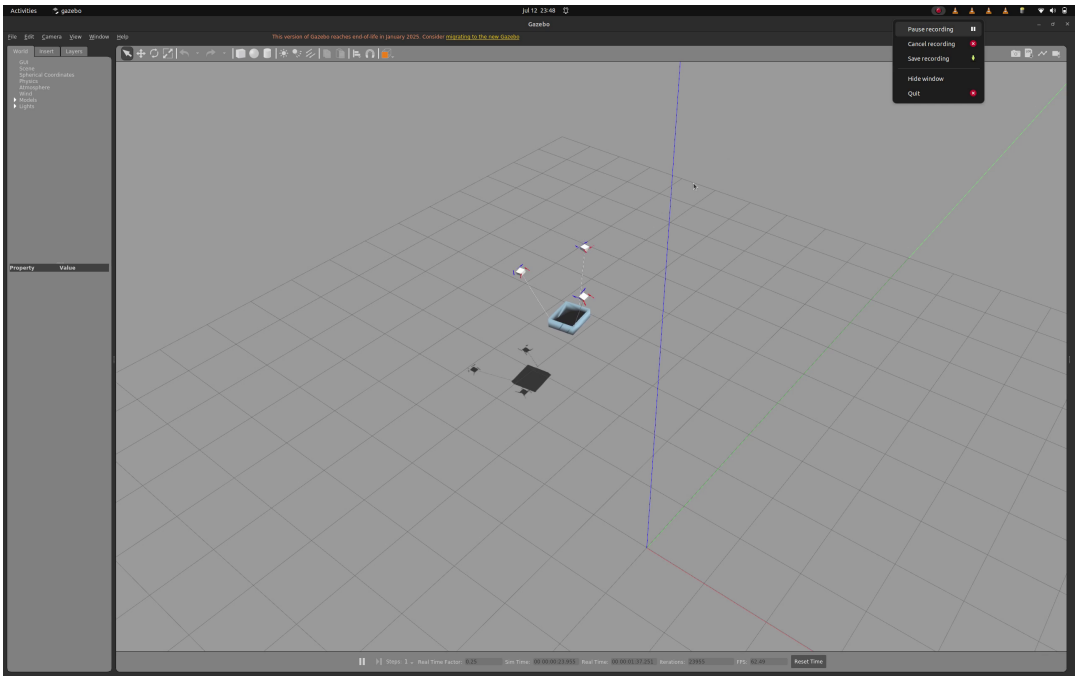


Figure 4.6: The Gazebo simulation environment used for validating the teacher policy and training the student policies.

4.4.2. Technical details of the simulation environment

The main goal for our policy is to achieve sim-to-real transfer. Collecting demonstrations online for fine-tuning the student policies is not feasible. Therefore, to ensure that policies can operate effectively in the real world, a high-fidelity and realistic simulation environment is required.

We primarily considered two simulators compatible with the hardware used for training: MuJoCo and Gazebo. MuJoCo is significantly faster than Gazebo and supports running multiple environments in parallel. In contrast, Gazebo could only run a single environment at a time, and even then at a 0.25 real-time factor.

However, Gazebo is well-supported by the Agilicious flight stack. This allows the same flight stack to be executed in both simulation and the real world, thereby reducing the sim-to-real gap. By contrast, using MuJoCo would have required porting Agilicious to Python to ensure compatibility. Furthermore, the high sample efficiency of imitation learning mitigates the impact of Gazebo's slower simulation speed. Additionally, Gazebo has been found to be particularly suitable for simulations intended for real-world transfer [39]. Based on these trade-offs, we chose to proceed with Gazebo.

An open-source ROS-Gazebo Gymnasium wrapper[1] was used to enable compatibility with Gymnasium environments. An Illustration of the Gazebo simulation environment is shown in Figure 4.6.

[1] `https://github.com/rickstaa/ros-gazebo-gym`

5

Implementation

In this chapter we describe various components we engineered to allow us to conduct real-world experiments safely. We also describe the system design of the teacher and student policies as well as the hardware used for real-world flights. Lastly, we talk about the various desired trajectories on which we test our solution and our method for evaluating the performance of our solution. We also developed a HRI interface consisting of a GUI and controller interface which has been described in Section A.1

5.1. Experimental Setup

5.1.1. System architecture of Teacher Policy

The user starts by selecting the desired trajectory using the GUI. Using the Xbox controller, the user can command the UAVs to take off and go to the starting pose of the trajectory. Once there, the user initializes the teacher policy which starts generating trajectories for the UAVs to follow. The teacher policy is an online kinodynamic motion planner which is implemented using the Acados solver. The Acados solver solves an optimal control problem designed to transport the load along the supplied reference trajectory. The resulting solution from the OCP provides a reference trajectory for each UAV to follow. These trajectories are transmitted over Wi-Fi to the onboard Agilicious flight control stack at a rate of 10 Hz. Concurrently, the predicted trajectory is visualized in RViZ, alongside the current system state estimated via the VICON motion capture system. To enhance operational safety, a dedicated safety module (Section A.2) runs on the ground control laptop. This module can override and issue high-level commands to prevent potential collisions or system failures. An overview of the teacher policy control architecture is illustrated in Figure 5.1.

5.1.2. System architecture of Student Policies

The user interface architecture of the student policy is the same as the teacher. However, the core control pipeline differs significantly. The desired trajectory selected by the user is transmitted to a central student policy controller. This node loads the trajectory and then broadcasts the entire trajectory to the fleet over the ROS Network in one message. Without loss of generality, these trajectories are usually 40 seconds long. A ROS node responsible for running the student policy onboard each UAV receives the entire trajectory. It runs the student policy at 10 Hz and processes the trajectory in 2-second segments based on the current onboard time. Synchronised clocks on the UAVs ensure that the same trajectory segment is processed by the UAVs enabling coordinated behaviour. The output of the student policy is the predicted trajectory for the ego UAV - similar to the teacher policy. This trajectory is transmitted to the

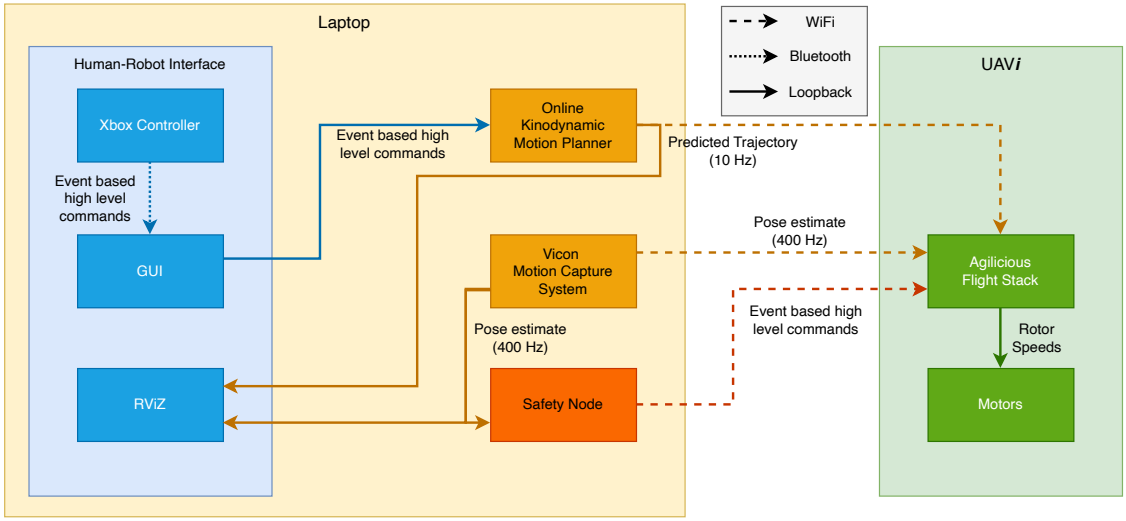


Figure 5.1: System architecture for the Teacher Policy

Agilicious flight stack as well as RViZ for visualisation. The motion capture system and safety node operate in a similar manner to the teacher policy. An overview of the student policy control architecture is illustrated in Figure 5.2.

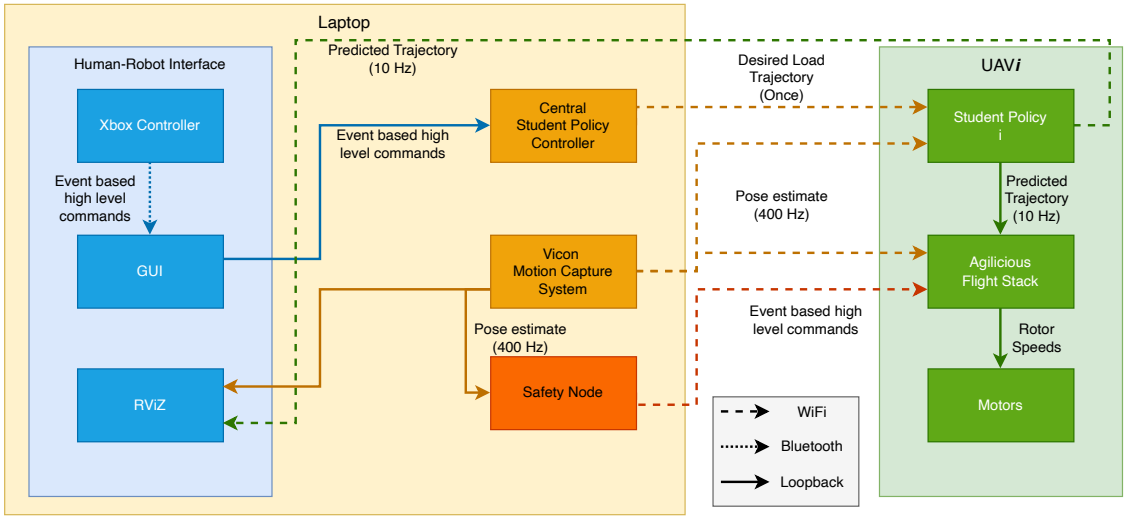


Figure 5.2: System architecture for the Student Policy.

5.1.3. Hardware Setup

The UAV used for practical experiments is called Falcon which is heavily based on the Agilicious [40] open-source platform developed at the Robotics and Perception Group at the University of Zurich. The Falcon is equipped with a Raspberry Pi 5 embedded CPU which is used to run the Agilicious flight stack onboard. The Falcon weighs 0.6 kg while the payload weighs 1.4 kg. Each UAV is attached to the payload by a cable of length 1 m.

The UAV relies on a VICON motion capture system for its state estimation. The payload's pose is also estimated using this system. The VICON system consists of 12 cameras that detect reflective markers attached to the UAVs and payload to ascertain their pose at millimetre

precision. This VICON system is connected to the laptop over a wired Ethernet connection.

A laptop equipped with a Ryzen 7 5800H CPU and an NVIDIA RTX 3060 GPU is used for training the student policies. This laptop is also used for testing the teacher and student policies during real world experiments. The onboard teacher / student policies use the state estimated by VICON to calculate the appropriate trajectories for each UAV. These trajectories are transmitted to each UAV over Wi-Fi which are then tracked by the low-level controllers implemented in the Agilicious flight stack.

The UAVs collaboratively manipulate the pose of the load without communication. To achieve this, the clocks of the onboard computers of the UAVs need to be precisely synchronised. We use Chrony[1], an open-source Network Time Protocol (NTP) to sync the clocks of the UAVs with the laptop. Chrony process on the Raspberry Pi automatically syncs its clock to the laptop on boot and is precise to sub-millisecond.

Figure 5.3 shows the Falcon UAV and the test area.

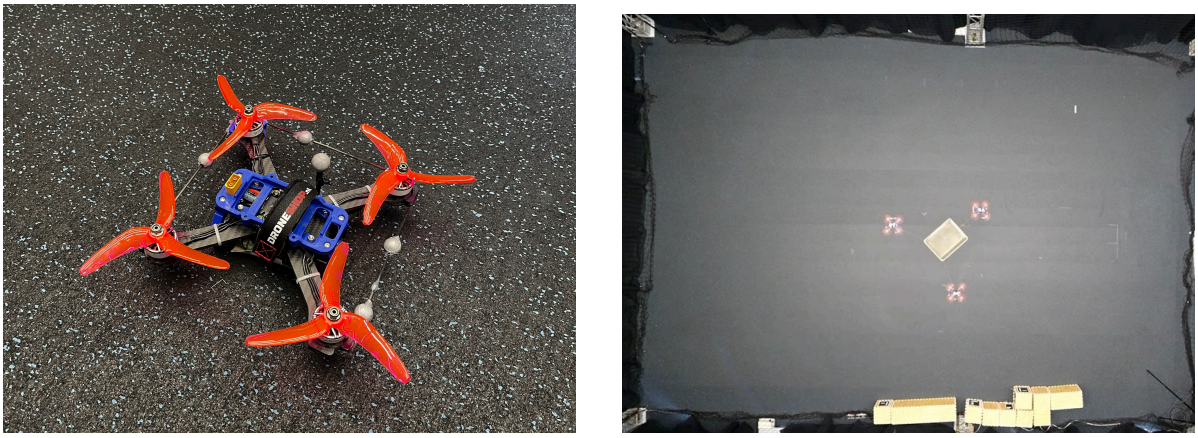


Figure 5.3: Left: The Falcon Drone | Right: Arena used for conducting real-world experiments.

5.2. Trajectories

5.2.1. Position

We train our student policies using a diverse set of trajectories, primarily characterized by three geometric patterns: the *figure-eight, circle, and square*. For all these patterns, trajectory of random sizes are sampled for training. The three trajectories are defined as follows:

Figure-Eight

$$p_x = \hat{x_a}\cos(f_{xy}t') + C; \quad p_y = \hat{y_a}\sin(2f_{xy}t') + C; \quad p_z = \hat{z_a}\sin(f_z t') + C \tag{5.1}$$

Circle

$$p_x = \hat{x_a}\cos(f_{xy}t') + C; \quad p_y = \hat{x_a}\sin(f_{xy}t') + C; \quad p_z = \hat{z_a}\sin(f_z t') + C \tag{5.2}$$

For Figure-Eight & Circle the variables are defined as:

Square

The square trajectory has sides of length $L \in [2, 3]$ m, with rounded corners of curvature radius 0.5 m.

To ensure a smooth start and stop, we formulate t' as follows:

$$t' = \log(1 + e^{k(t-t_0)}) - \log(1 + e^{k(t-t_1)}) \quad | \quad t_0 = t_s + 5 \quad | \quad t_1 = t_e - 5 \tag{5.3}$$

[1] https://chrony-project.org/

Random Variable	Distribution
$\hat{x_a}$	$[1, 2.2] \cup [-2.2, -1]$
$\hat{y_a}$	$[1, 2.2] \cup [-2.2, -1]$
$\hat{z_a}$	$[0.25, 0.75] \cup [1.25, 1.75]$
f_{xy}	$\frac{4}{40}\pi$
f_z	$\left\{\frac{2}{40}\pi, \frac{4}{40}\pi\right\}$

where t_s and t_e represent the start and end time of the trajectory. All trajectories are 40 seconds long.

5.2.2. Orientation

The load's orientation can follow one of three modes: zero-sideslip, fixed orientation, or constant yaw rotation. In the zero-sideslip mode, the load's yaw aligns with the direction of the horizontal velocity vector. In the fixed orientation mode, the load maintains a constant orientation throughout the trajectory. In the constant yaw rotation mode, the yaw rate remains constant, while the roll and pitch vary to ensure zero lateral force in the load's body frame.

We show various trajectories in Figure 5.4.

Beyond position and orientation, the student policies are also provided with linear and angular velocities, as well as linear acceleration. These trajectories are implemented using the CasADi symbolic framework, which enables accurate computation of higher-order derivatives through symbolic differentiation. Each trajectory is designed to start and stop smoothly, ensuring continuity in both position and velocity profiles.

These trajectories are quite agile, featuring velocities of up to 2 m/s and accelerations reaching 2 m/s^2. They also require manipulating the load along all three rotational axes. As illustrated in Figure 5.6 and Figure 5.5, these trajectories cover a wide range of velocities and accelerations.

5.2.3. Zandvoort F1 Track

We further evaluate the proposed method on the Zandvoort Formula 1 circuit. We neglect track elevation changes as well as orientation changes in roll and pitch. The real-world track dimensions are uniformly scaled by a factor of 1:50, resulting in a trajectory length of 88 m with a traversal duration of 150 seconds. The zero-sideslip and constant-orientation configurations are tested for this trajectory. The corresponding velocity distributions are presented in Fig. 5.7, while the track layout is depicted in Fig. 6.11 in the Results chapter.

5.3. Evaluation Metrics

To evaluate the effectiveness of our approach, we quantify the discrepancy between the actual pose of the load and the reference trajectory. This discrepancy, referred to as pose error, is the primary performance metric in our evaluation.

The pose error is composed of two components - position error and attitude error.

5.3.1. Position Error

The position error is defined as the Euclidean distance between the measured position and the reference position. This metric serves as the primary error indicator reported in all time-series analyses presented in the Results chapter. The measured position data is acquired at a sampling frequency of 400 Hz, while the reference trajectory is discretized at 10 Hz. To compute the error signal, the measured position sample with the timestamp closest to each reference trajectory point is selected.

We calculate the Root-Mean-Square-Error of the position error signal to calculate the overall position tracking error for an episode.

5.3.2. Attitude Error

Attitude error is evaluated using quaternion-based orientation representations. The relative orientation (or quaternion error) between the measured orientation q_l and the reference orientation q_r is calculated as:

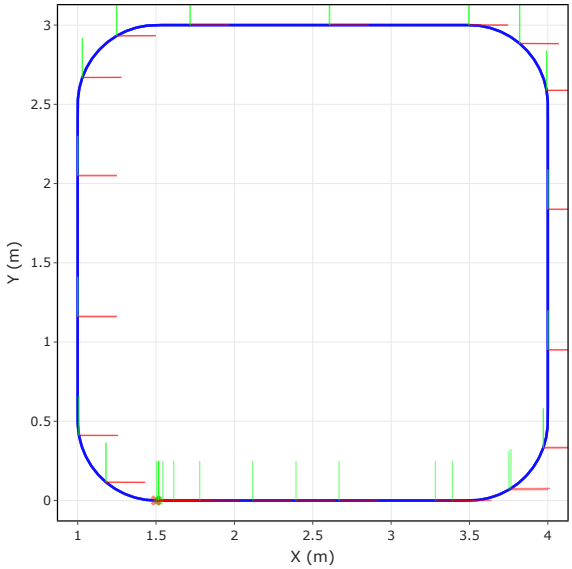
$$q_e = q_r^{-1} \otimes q_l \tag{5.4}$$

The attitude error, $\Delta\alpha$, is then quantified as:

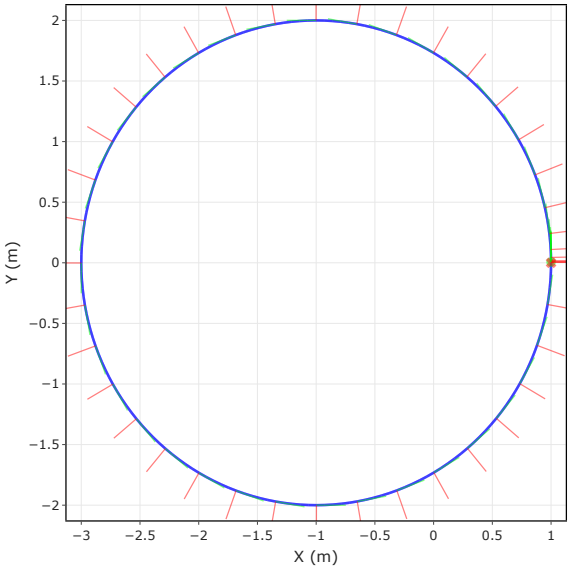
$$\Delta\alpha = 2arccos(q_{e,w})\frac{180}{\pi} \tag{5.5}$$

Here, $\otimes$ denotes quaternion multiplication, and q_r^{-1} is the inverse of the reference quaternion. The advantage of using quaternion error is that it gives us the actual minimum angular distance in radians between two rotations, providing a geometrically meaningful, bounded error metric.

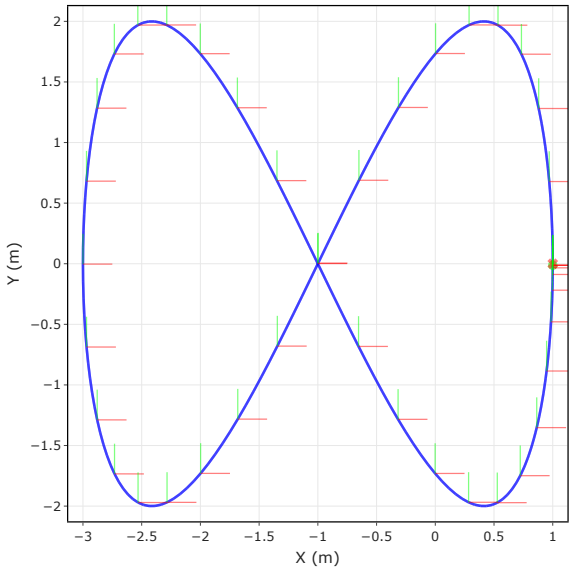
Similar to the position error signal, we use the reference trajectory timestamps for obtaining the orientation error as well. The overall error for an episode is the RMSE of this error signal.



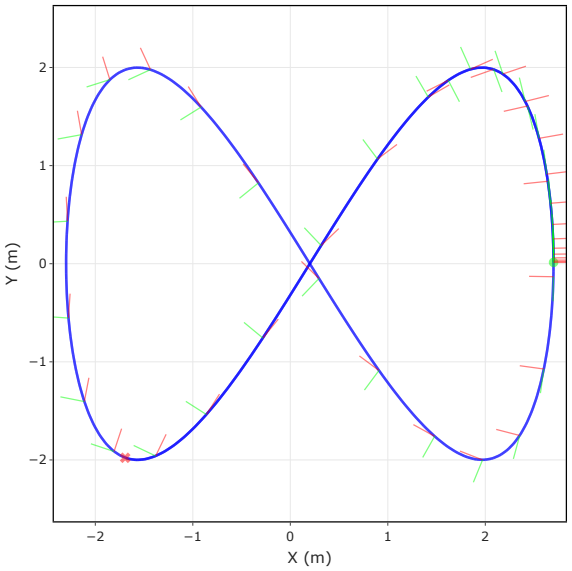
(a) Square constant orientation trajectory: The orientation of the load remains constant



(b) Circle zero-sideslip trajectory: The body Y-axis of the load is aligned with the instantaneous horizontal velocity vector.

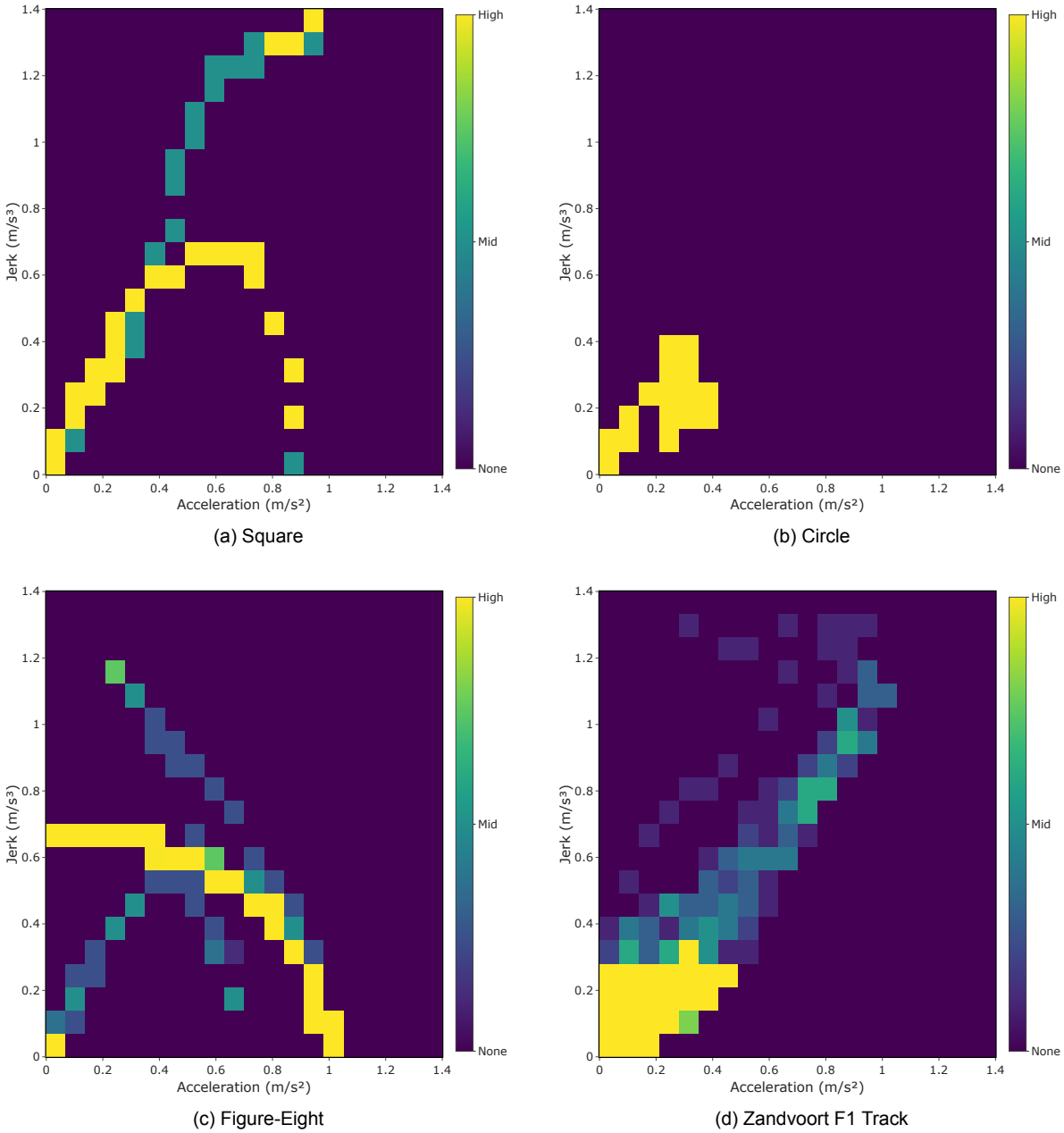


(c) Figure-eight constant orientation trajectory



(d) Figure-eight trajectory used for real world experiments

Figure 5.4: Examples of various reference trajectories for the load used for training and testing. The circle and cross represent the start and end points respectively. The trajectory consists of 2+ laps.



(a) Square

(b) Circle

(c) Figure-Eight

(d) Zandvoort F1 Track

Figure 5.5: Acceleration-Jerk heatmaps for various trajectories. A diverse acceleration–jerk heatmap across training trajectories ensures that the student policy experiences the full spectrum of motion dynamics, from smooth to aggressive manoeuvres. This diversity improves generalization, robustness to disturbances, and transferability to real-world flight conditions.

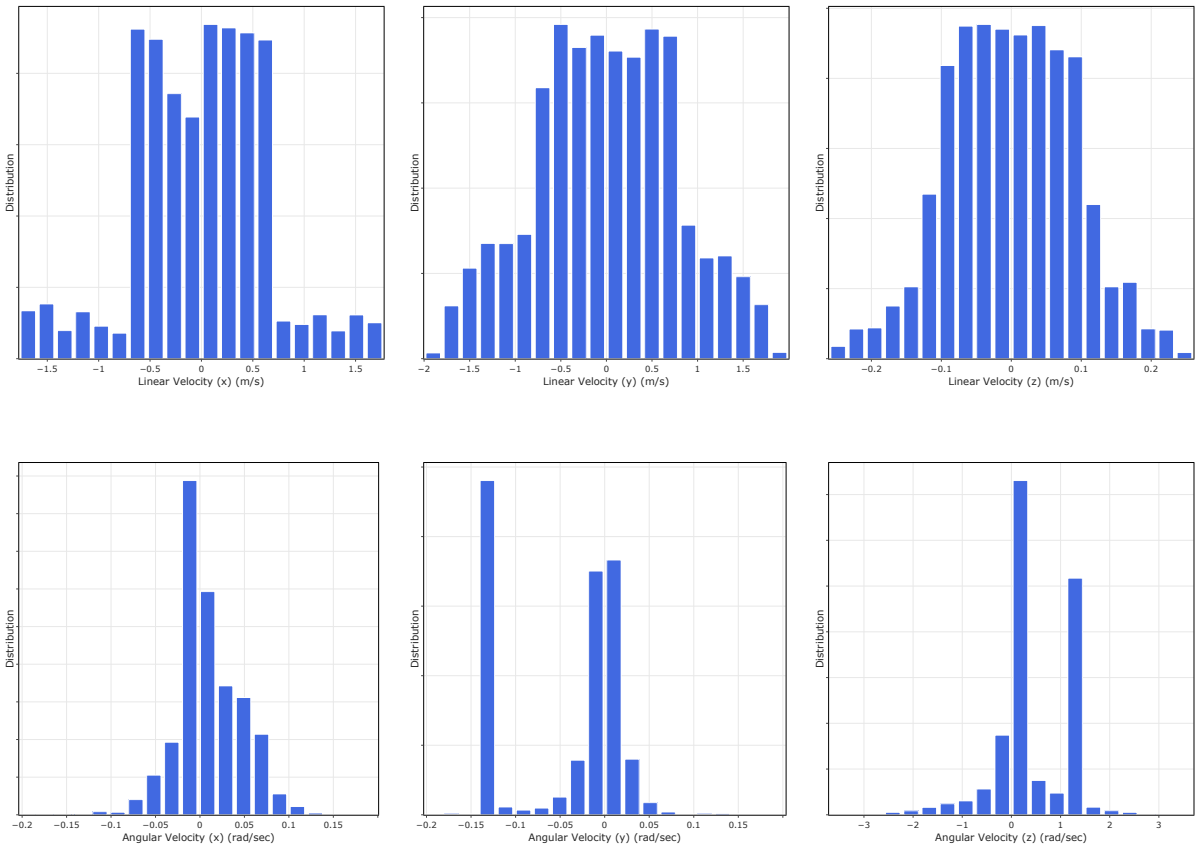


Figure 5.6: Linear (top row) and angular (bottom row) velocity distributions for selected reference trajectories.

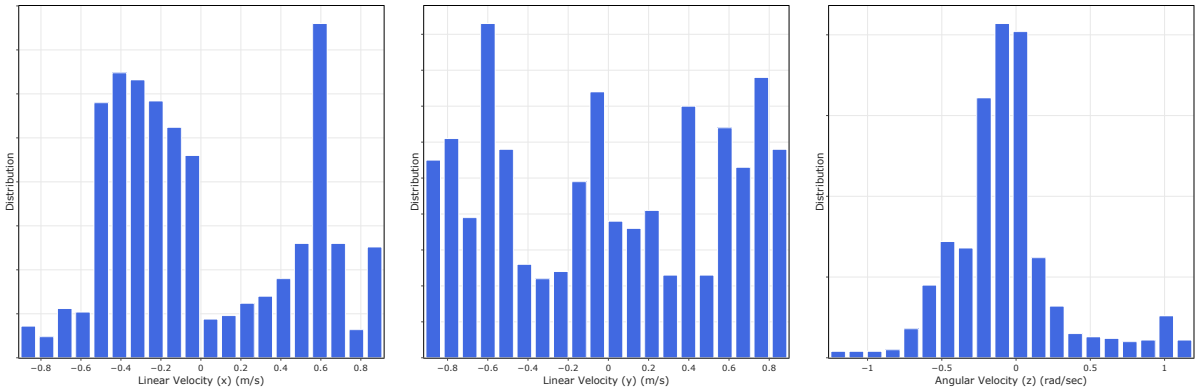


Figure 5.7: Linear (a, b) and angular (c) velocity distributions for the Zandvoort F1 track are shown.

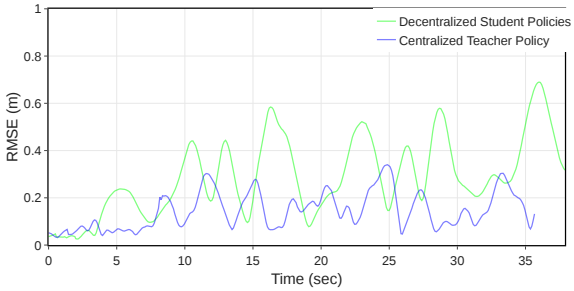
6

Results & Discussion

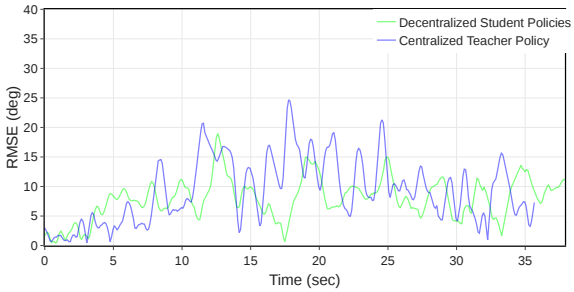
This YouTube playlist contains videos of multiple experiments that have been described in the sections in this chapter.

6.1. Trajectory Tracking - Real world experiments

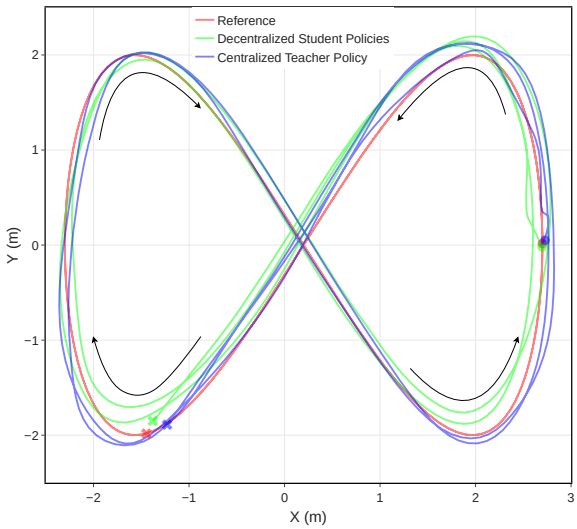
In this section we compare the trajectory tracking performance of the decentralized student policies with the centralized teacher policy. The results are generated using data collected from real world experiments with policies running on the same hardware.



(a) Position tracking error between the measured load pose and the desired reference position, expressed as Euclidean distance in meters.



(b) Orientation tracking error between the measured load pose and the desired reference orientation, computed using the quaternion angular distance.



(c) Top view of the trajectory tracking performance. Circle indicates start of trajectory while cross signifies the end. Two laps of the figure 8 are performed.

Figure 6.1: Comparison of metrics from real-world flights following the figure-eight trajectory. Performance of the teacher and student policy is compared. While the student has slightly better performance in tracking the desired orientation, the teacher has much better position tracking.

The student policy was trained on a figure-eight trajectory with dimensions $\hat{x_a} = 2.2; \quad \hat{y_a} =$

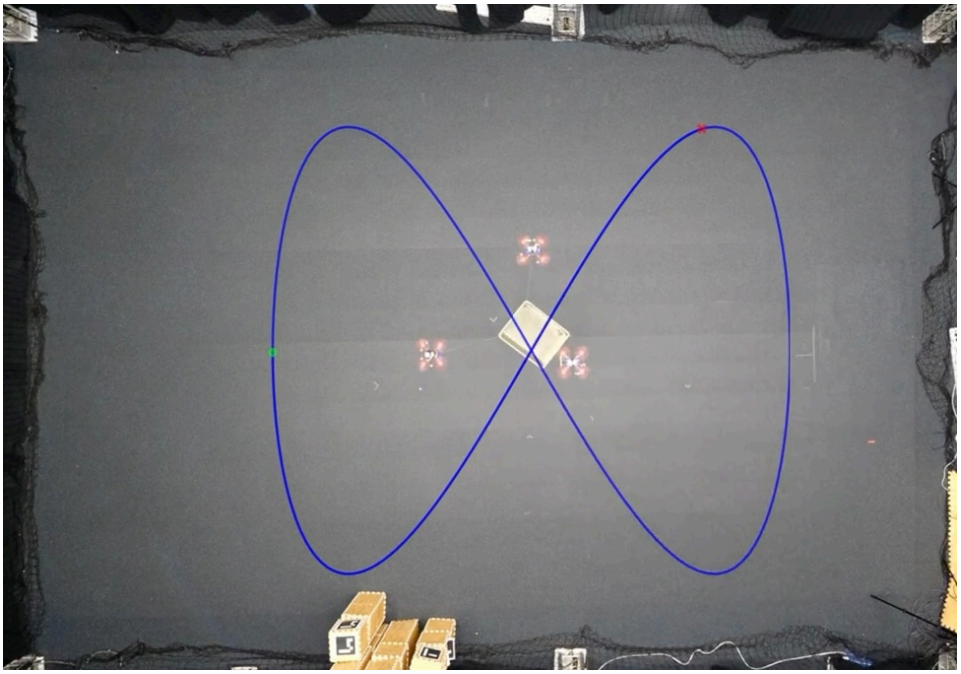


Figure 6.2: Top camera view of the real world experiments. The blue line represents the desired trajectory while circle and cross indicate the start and end of trajectory respectively.

2 meter over 20 DAgger rounds, each consisting of a single episode. During each round, the policy was trained on the aggregated dataset for 16 epochs. After each round, the probability of selecting actions from the student policy was linearly increased from 0 to 1 by round 20. To enhance robustness against real-world perturbations, consistent yet stochastic disturbances, as detailed in Section 4.4.1, were applied to the load during the final 5 episodes of training. The entire training process, including data collection and policy updates was completed within 120 minutes.

The trained policy was then evaluated on the identical figure-eight trajectory in the real world via zero-shot sim-to-real transfer. We performed two flights for both student and teacher policies and report the average of the metrics in Table 6.1. The error signal for position and attitude is shown in Figure 6.1a and 6.1b respectively. We also show the trajectory tracking performance of both policies in Fig 6.1c.

In real-world experiments, the student policy, executed in a fully decentralized manner using only ego-UAV observations, achieves a comparable orientation RMSE to the teacher policy but exhibits a higher position RMSE.

Metric	Teacher Policy	Student Policies
RMSE Posn. (m)	0.171	0.377
RMSE Orient. (°)	10.447	9.335

Table 6.1: Comparison of metrics from real-world flights following the figure-eight trajectory shown in Fig 6.1c. The values are the mean of two flights.

6.2. Inter-agent Distance

One notable limitation of the student policies is the absence of hard guarantees for inter-agent collision avoidance. The teacher policy explicitly enforces minimum separation constraint of 1 m between all drones using a hard constraint in its optimal policy formulation. However, it is not possible to program such constraints in the student policies as they lack direct awareness of

other UAV's positions. In the absence of hard guarantees for maintaining minimum separation, the risk of inter-agent collisions is inherently higher for the decentralized student policy.

The student policy is expected to use the load pose and desired reference trajectory to place UAVs in a safe configuration. Getting the student policies to learn these safe configurations was difficult and required some changes to the student policy and the training method. We describe this in detail in the following subsections:

6.2.1. Load state estimation using an extended Kalman filter

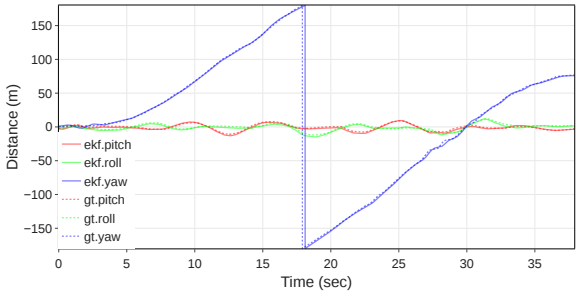
The teacher policy utilizes an extended Kalman filter to determine the pose of the load. This filter uses the instantaneous pose of all UAVs along with the known attachment points of the cables to estimate the cable directions and load pose, twist. In the first version of the student policy (*Student Policy EKF*), we utilized the values of this Kalman filter as the input for the load state. This version took as input the twist of the load state in addition to other observations mentioned in Equation 4.1. While the use of the EKF filter made a part of our solution centralized - the EKF required instantaneous pose of all UAVs; it removed the requirement for measuring the pose of the load - making it more practical for real-world deployment.

This version of the student policies worked well in simulation, however it performed poorly in real world experiments. In real world, the UAVs would progressively come closer to each other as the trajectory progressed. Once the UAVs breached the 0.4 m minimum safe separation between the UAVs, the safety module (described in Section A.2) would intervene and turn off the UAVs to prevent mid-air collisions. This behaviour was not observed in the simulation environment where the UAVs kept a safe distance from each other. We could induce this behaviour in simulation by applying random directed moments on the load.

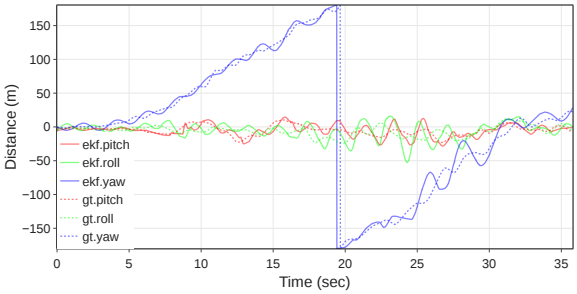
Upon analysing the data collected during real-world experiments, we identified that the primary source of performance degradation was noisy state estimation from the Extended Kalman Filter (EKF). While the EKF provided accurate load orientation estimates in simulation (Figure 6.3a), its performance deteriorated significantly in the real-world deployment due to imperfect UAV state estimation and network-induced delays, resulting in high-frequency noise in the orientation estimates (Figure 6.3b). The student policies did not encounter such noisy measurements during training and were therefore unable to cope with them effectively in real-world deployment. Consequently, the student policy struggled to maintain safe positioning of the ego UAV relative to the load and other UAVs. Additionally, the noise in load state estimation led to instability in the predicted strategies (Figure 6.3d), causing abrupt changes that the low-level onboard controllers could not reliably track.

6.2.2. Load state estimation using motion capture

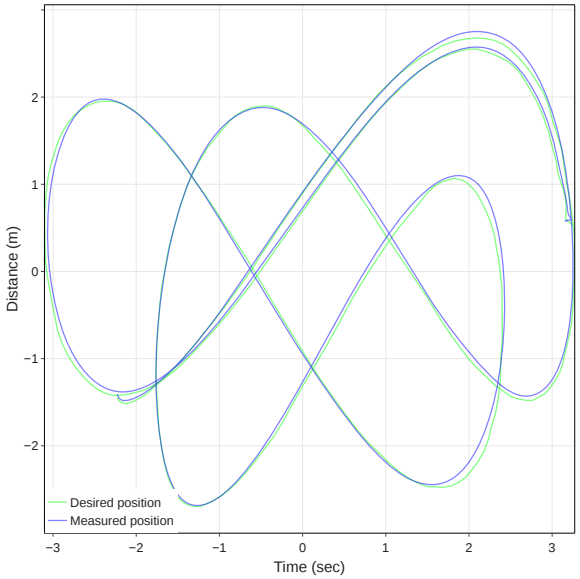
Noisy load pose estimation was attributed to be the main cause of the failure of student policies in the real world. We therefore, changed the architecture of the student policies to use the ground truth pose values estimated using the motion capture system. During training, this ground truth information is provided by the Gazebo simulator. We further add random wrenches on the load during training to make the policies robust and learn to maintain safe inter-agent distances. This second version of the student policy (*Student Policy MoCap*) showed a tremendous improvement in maintaining safe inter-agent distances in many scenarios as shown in Figure 6.4. In all real-world deployments, the UAVs never came closer than 0.8 m to each other.



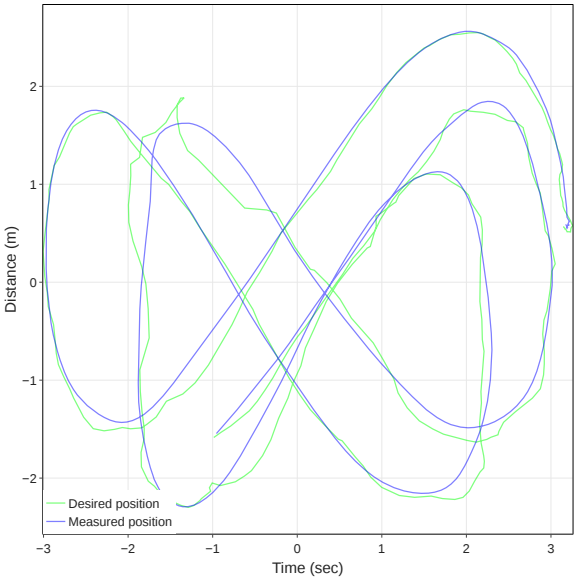
(a) Simulation: Load pose estimation (solid) and ground truth (dot) in simulation environment. In simulation, the EKF works well, and the estimations are accurate.



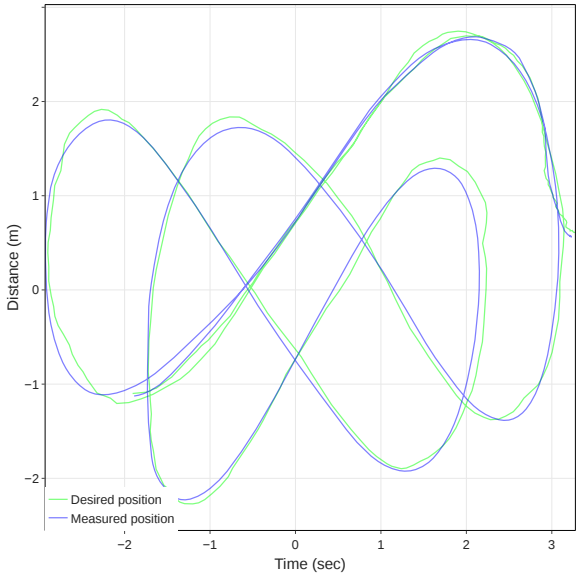
(b) Real World: Pose estimate oscillates around the ground truth pose. The student policies are not robust to this noise in load pose estimation.



(c) Student Policy EKF in Simulation: Desired and measured position of the UAV is shown. Desired position is the first point of trajectory generated at each time step. Predictions are consistent and can be tracked by the onboard controller.

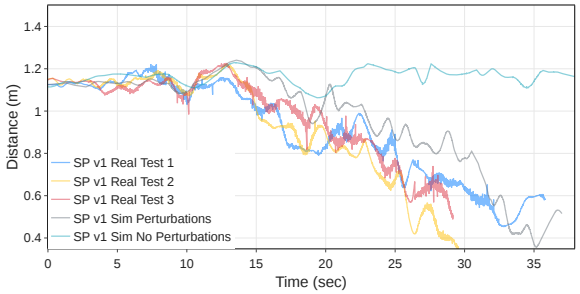


(d) Student Policy EKF in Real World: Policies depend on Load pose information to place UAVs in safe positions. Oscillating Load pose estimations cause the desired positions to be inconsistencies which UAVs cannot track.

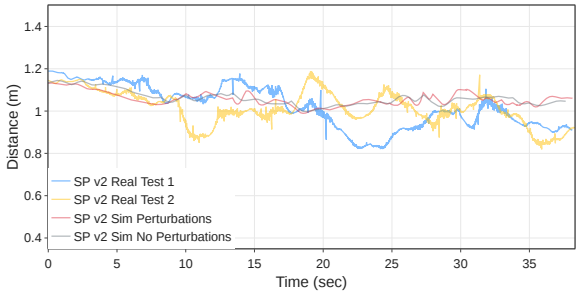


(e) Student Policy MoCap in Real World: Second version of the student policy using ground truth load pose. The trajectory generated by this policy is again consistent and robust to real world perturbations.

Figure 6.3: Left: Performance of the system in simulation | Right: Performance of the system when deployed in the real world | Bottom: Performance of the second version of student policies in real world.



(a) Student Policy EKF: This version of student policies is unable to maintain a safe separation between agents when external disturbances are present.



(b) Student Policy MoCap: Despite lacking awareness of each other's positions, the quadrotors successfully maintain safe distances despite the domain gap between the real world and simulation.

Figure 6.4: Inter-agent distance for various experiments in simulation and the real world. Minimum inter-agent distance is plotted.

Note

Unless stated otherwise, all results presented in this chapter pertain to the *Student Policy MoCap* variant.

6.3. Advantage of PINN over MLP as Decoder

The benefit of using PINNs as decoder is that the trajectories are guaranteed to be consistent and smooth. Consistency comes from the fact that all target trajectories in training data start at ego UAV's pose and therefore the policies learn this behaviour as well. Smoothness is guaranteed by the partial derivative constraints on velocity and acceleration as described in Section 4.3.2.

To demonstrate the smoothness of our proposed architecture, we compare the PINNs with a baseline policy where the *Decoder* network employs a single Multi-layer Perceptron (MLP) to generate all four trajectory components. The trajectories produced by this model are visibly jagged (shown in the supplementary video), lacking the smoothness enforced by the PINN's constraints. Moreover, these methods exhibit notable inconsistencies, wherein the numerical derivative of the predicted position does not align with the corresponding predicted velocity. This discrepancy is illustrated in Fig. 6.5. Table 6.2 presents a comparison of the mean absolute error (MAE) between the numerical derivative of the predicted position and the predicted velocity. The reported values are computed over entire episodes, with each episode comprising approximately 400 trajectory predictions. For each trajectory, only the first 10 time steps are considered, resulting in an aggregate of approximately 4000 measurements used to compute the averages.

Method	X (m/s)	Y (m/s)	Z (m/s)	Overall (m/s)
PINN	0.010	0.030	0.004	0.015
MLP	0.080	0.116	0.048	0.081
Teacher	0.012	0.033	0.003	0.016

Table 6.2: The Mean Absolute Error (MAE) between the numerical derivative of predicted position and the predicted velocity was evaluated for each method. For PINN and the teacher planner, the errors primarily arise from numerical differentiation. In contrast, the MLP exhibits higher errors due to both numerical differentiation and additional inaccuracies stemming from infeasible position and velocity predictions.

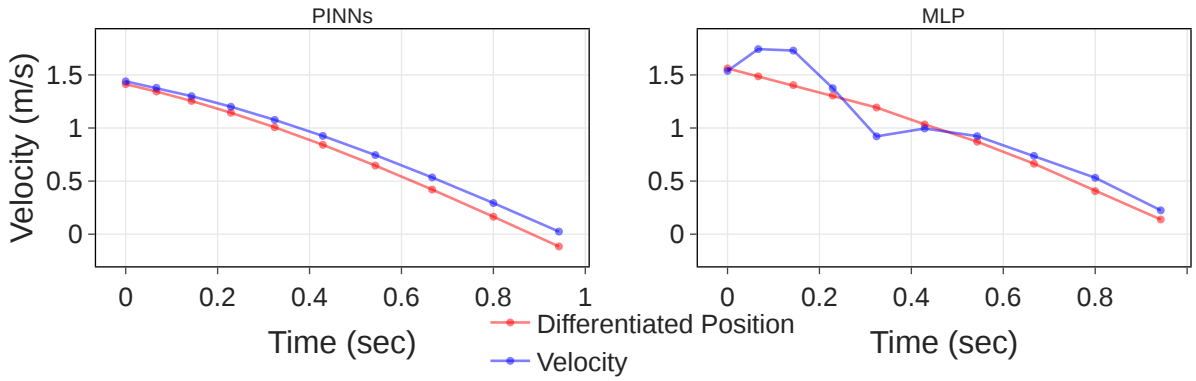


Figure 6.5: Here the trajectory generated by the PINNs is compared with a traditional ML model. Trajectories produced by the PINN are smoother and consistent. The supplementary video shows the smoothness of an entire trajectory.

6.4. Generalizability

6.4.1. Generalize to similar trajectories

We test the ability of the student policies to transport the load along similar but different trajectories. To do this, we train the student policies on two trajectories and then evaluate the resulting policy in simulation on multiple unseen trajectories of *figure-eight*. Training alternated between these two trajectories over 20 DAgger rounds, with perturbations applied during the final 4 rounds. The entire training process for this experiment took 80 minutes.

The performance of the student policies are summarized in Table 6.3. The top view visualization of the student is provided in Figure 6.6. We provide the performance of the teacher policy on these trajectories as a baseline.

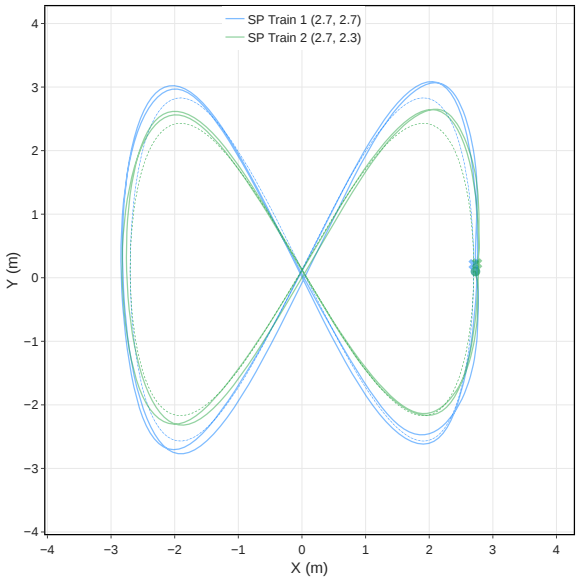
From this experiment we conclude that student policies show limited ability to generalize to very different trajectories. This may be due to the limited number and kind of trajectories used in training.

Ref. Size		**Train**	**Student Policies**		**Teacher Policy**	
$\hat{x_a}$ **(m)**	$\hat{y_a}$ **(m)**	**Dataset**	Posn. (m)	Orient. (°)	Posn. (m)	Orient. (°)
2.7	2.7	Yes	0.166	6.768	0.089	4.111
2.7	2.3	Yes	0.155	5.684	0.091	4.324
2.0	2.0	No	0.377	10.877	0.073	3.397
3.0	3.0	No	0.175	7.531	0.105	5.371
3.5	3.5	No	0.707	12.013	0.192	6.912
2.5	2.5	No	0.205	6.903	0.090	4.196

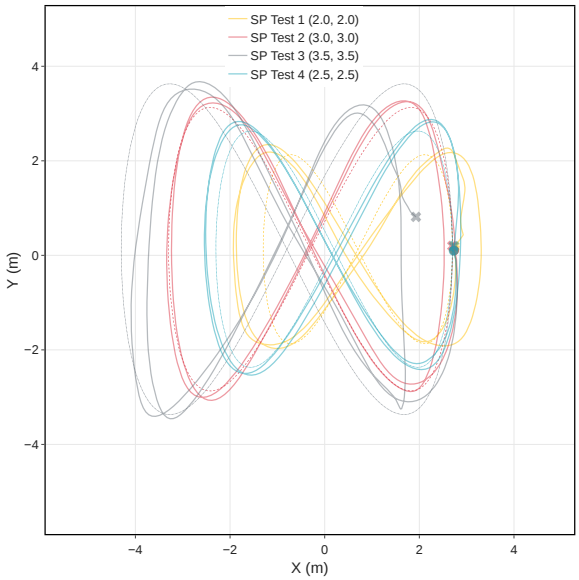
Table 6.3: Comparison of the student and teacher policy performance in simulation for multiple trajectories. The table reports the RMSE for position and orientation. Student policies were trained using the first two trajectories. $\hat{x_a}$ & $\hat{y_a}$ denote the amplitudes of the sine and cosine functions used to generate the trajectories as defined in Section 5.2.

6.4.2. Generalize to dissimilar trajectories

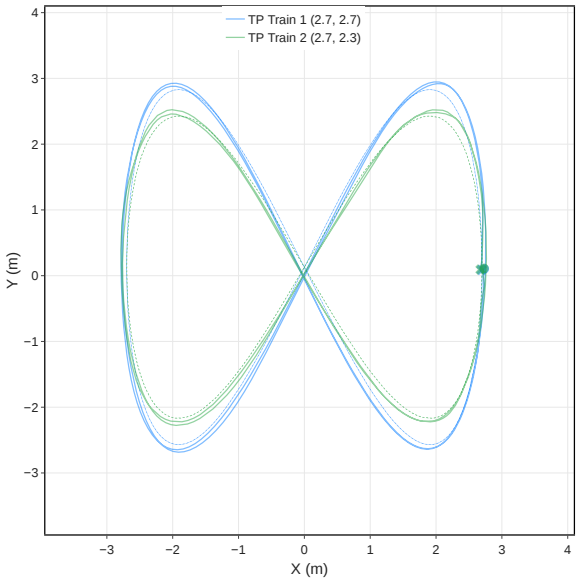
We also train the student policy to learn very different kinds of trajectories—namely, Eight, Circle, and Square. During training, we randomly sample one of these trajectory types at each iteration to encourage the student policy to generalize across diverse motion patterns. These set of trajectories contain a very diverse set of curves with a large range of radii of curvature which needs to be followed by the load. The z value of the trajectories also oscillates with



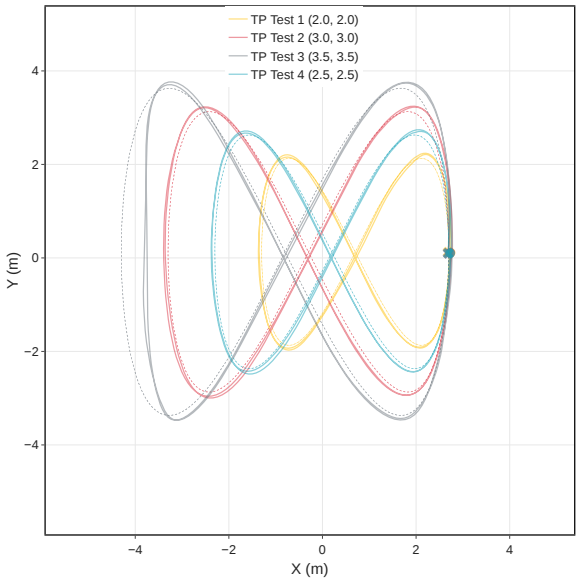
(a) Evaluation of student policies on training trajectories



(b) Evaluation of student policies on testing trajectories



(c) Evaluation of teacher policy on training trajectories



(d) Evaluation of teacher policy on testing trajectories

Figure 6.6: Top-down view of tracking performance of the student and teacher policy. Dotted lines represent reference trajectory. Trajectories start and stop at the same point. Same colours represent same trajectories across figures. The performance of the teacher policy on these trajectories shows the ability of the system to execute these trajectories. The student policies struggle to generalize well to unseen trajectories. For the smallest (yellow) trajectory, the student policy struggles to make sharp turns. On the largest (gray) trajectory, the students struggle to maintain the required speeds leading to large pose tracking error.

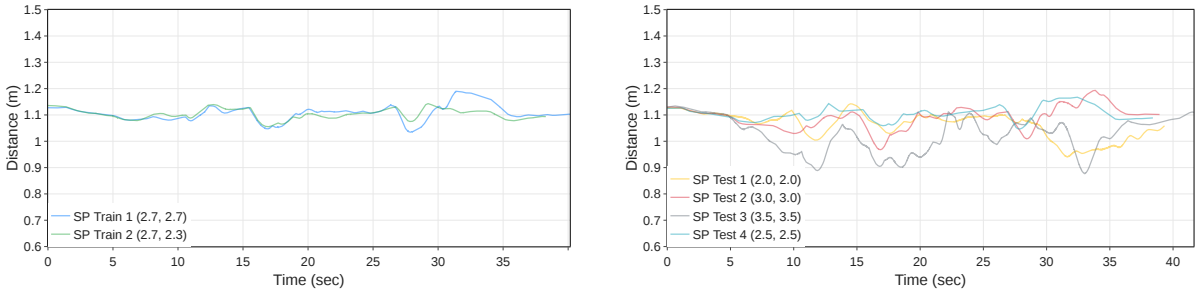


Figure 6.7: Inter-agent distance are shown here. The quadrotors are able to maintain safe inter-agent distances in all tested scenarios despite lacking awareness of the position of other UAVs.

a random amplitude and frequency. The trajectories can also have multiple modes for the desired orientation. These modes further increases the complexity of the task significantly. The different trajectories we use for training are described in Section 5.2.

Constant orientation
Trajectories with constant orientation of the load result in UAV trajectories that are very similar to the given trajectory. The student policies need to focus only on position tracking which leads to lower errors.

Zero-sideslip orientation
To maintain the correct orientation for these trajectories, all the UAVs need to follow very different trajectories. For executing tight turns, the inner UAV needs to slow down or even reverse while the UAV on the outside needs to move fast along an arc. The figure 6.8 shows some examples.

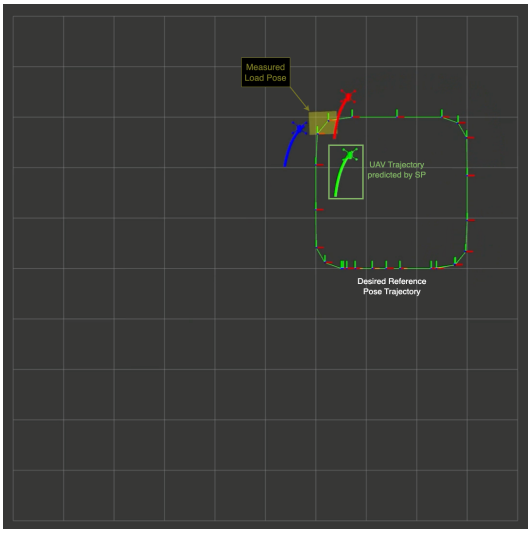
Trajectory	Ref Size (m)	SP (Constant)		SP (Zero-sideslip)		SP (Both)	
		Posn (m)	Orient. (°)	Posn (m)	Orient. (°)	Posn (m)	Orient. (°)
Eight (Constant)	$\hat{x_a}$: 2, $\hat{y_a}$:: 2	0.097	4.443	✗	✗	0.089	4.947
Eight (Zero-sideslip)	$\hat{x_a}$: 2, $\hat{y_a}$:: 2	✗	✗	0.212	12.603	✗	✗
Circle (Constant)	Radius: 2	0.037	2.850	✗	✗	0.049	3.518
Circle (Zero-sideslip)	Radius: 2	✗	✗	0.095	8.293	0.134	10.403
Square (Constant)	Side: 2	0.202	5.475	✗	✗	0.149	6.540
Square (Zero-sideslip)	Side: 2	✗	✗	0.169	14.803	0.181	13.107

Table 6.4: Comparison of performance of various Student Policies on Different Trajectories. ✗ represents instances where inter-agent distance reduced to less than 0.4 m after which the experiment was stopped.

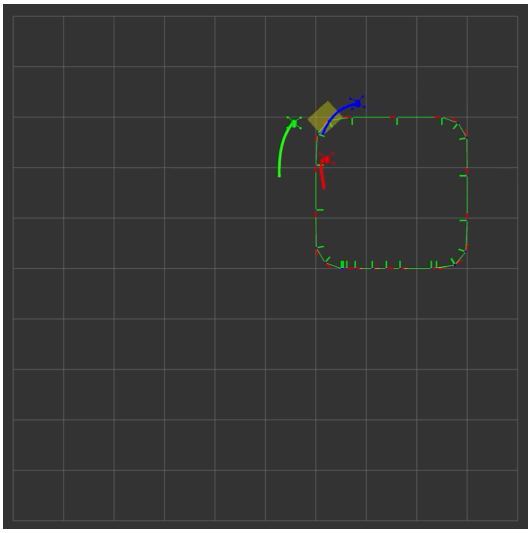
We trained three student policies on these trajectories:

- **Student Policy (Constant)**: Trained on all constant orientation trajectories for 30 rounds. This policy is unable to follow trajectories where the orientation changes.
- **Student Policy (Zero-sideslip)**: Trained on all zero-sideslip trajectories for 30 rounds. This policy is unable to follow trajectories where the orientation is constant.
- **Student Policy (Both)**: Trained on all trajectories with zero-sideslip and constant orientation for 60 rounds. This unified policy can track all trajectories except the figure-eight zero-sideslip path, though it exhibits higher pose tracking error compared to specialized policies.

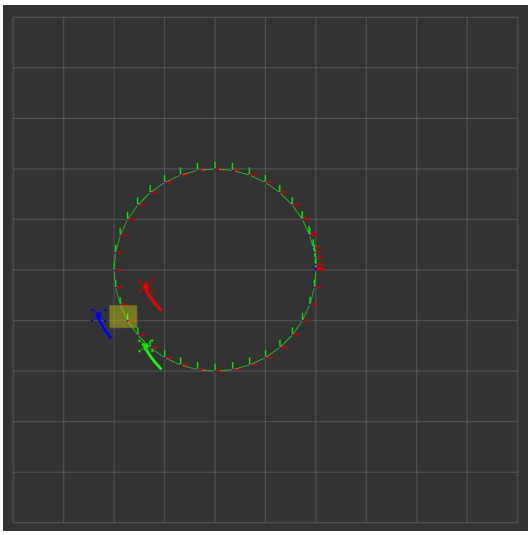
The performance of these models is summarised in Table 6.4. The performance of the Student Policy (Both) is also visualised in Fig. 6.8 and Fig. 6.9a.



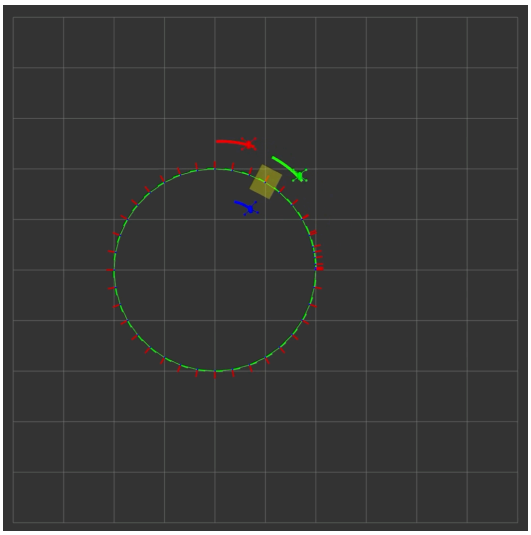
(a) Square constant-orientation trajectory: Student policy for each UAV independently predicts the drone path for the ego UAV. This path is tracked using the onboard low-level controllers on the UAV.



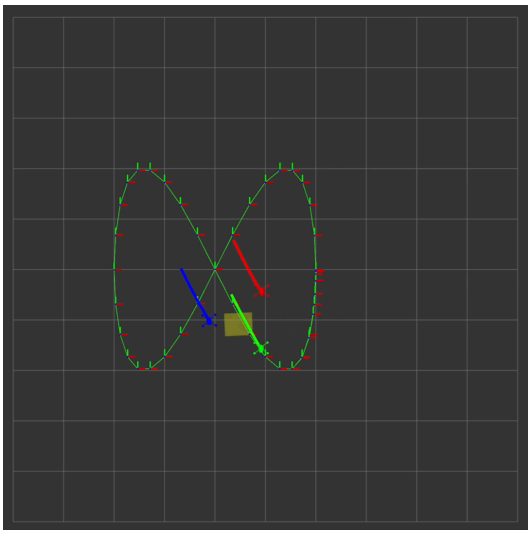
(b) Square zero-sideslip trajectory: The desired body X axis is aligned along velocity. Notice how the inner UAV needs to slow down while the outer UAVs accelerate to track the desired pose.



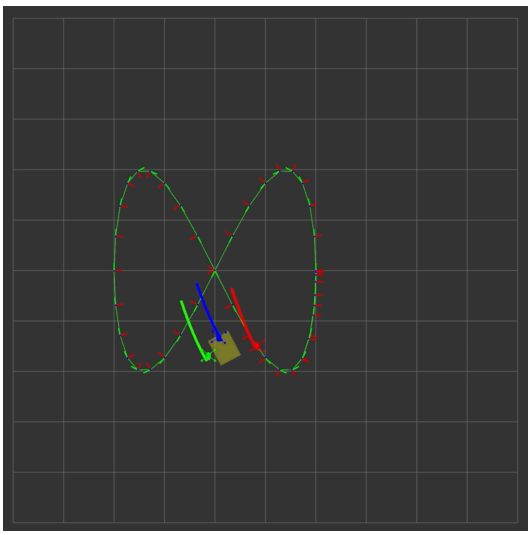
(c) Circle constant orientation: UAVs produce similar trajectories as the orientation of the load is constant.



(d) Circle zero-sideslip trajectory: The formation of the UAV is very different to Figure 6.8c due to the desired orientation.

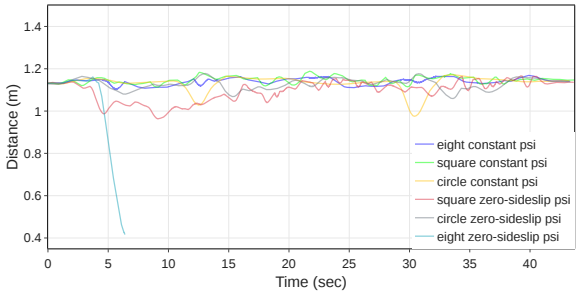


(e) Eight constant orientation: UAVs produce similar trajectories as the orientation of the load is constant.

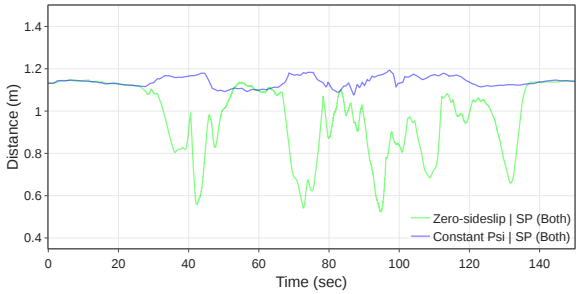


(f) Eight zero-sideslip trajectory: The policy struggles to adapt to the rapid changes in the payload's desired orientation. As a result, the blue and green drones converge too closely, prompting the safety module to intervene.

Figure 6.8: Visualization of the performance Student Policy (Both) on various trajectories. This policy is capable of predicting vastly different UAV trajectory based on the desired reference trajectory.



(a) Student Policy (Both): Inter-agent distances for various constant orientation and zero-sideslip trajectories. The policy maintains safe inter-agent distance for all trajectories except the figure-eight zero-sideslip trajectory.



(b) Student Policy (Both): The inter-agent distances on the Zandvoort F1 track are illustrated here. During zero-sideslip trajectories, the agents frequently come close to one another but consistently maintain a distance of more than 0.4 meters.

Figure 6.9: We show inter-agent distance graphs for the Student Policy (Both) on various trajectories.

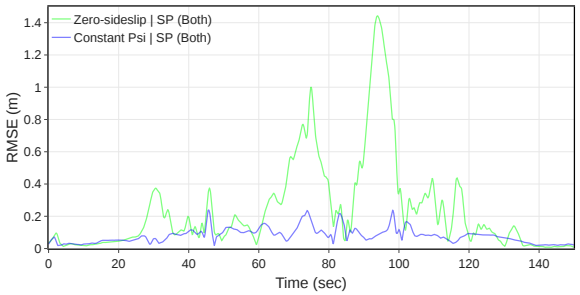
6.4.3. Zandvoort F1 Track

The proposed student policy was evaluated on its capability to transport the payload along the Formula 1 (F1) track located in Zandvoort, Netherlands. This trajectory differs substantially from those encountered during training, exhibiting greater variability in curvature radii, path length, and overall duration.

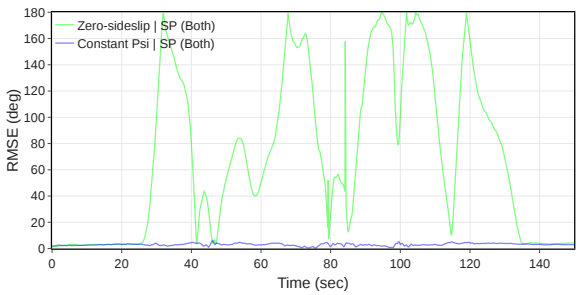
We evaluated the Student Policy (SP (Both)) under both orientation modes: constant orientation and zero-sideslip. The quantitative results are summarized in Table 6.5 and illustrated in Figure 6.10, 6.11.

Metric	Zandvoort (Constant)	Zandvoort (Zero-sideslip)
RMSE Posn. (m)	0.092	0.378
RMSE Orient. (°)	3.157	93.575

Table 6.5: Comparison of metrics from simulated flights around the Zandvoort F1 Track. The Student Policy (Both) was deployed for this experiment without any fine-tuning.



(a) Position tracking error signal



(b) Attitude tracking error signal

Figure 6.10: Time-series of error signals when tracking the Zandvoort F1 Track using the Student Policy (Both).

In the constant orientation mode, the proposed method achieved high position-tracking accuracy, demonstrating the policy's ability to compute appropriate ego-UAV positions for diverse trajectories. The policies exhibit invariance to trajectory scale and duration, as the student processes fixed 2-second trajectory segments at each time step. Furthermore, by expressing observations in the load frame $\mathcal{L}$, the policy is equivariant to the absolute position of the system in the world allowing it to operate in arbitrarily large spaces.

In contrast, in the zero-sideslip mode, the student policy exhibited significantly higher position-tracking error and failed to accurately follow the desired orientation. This suggests

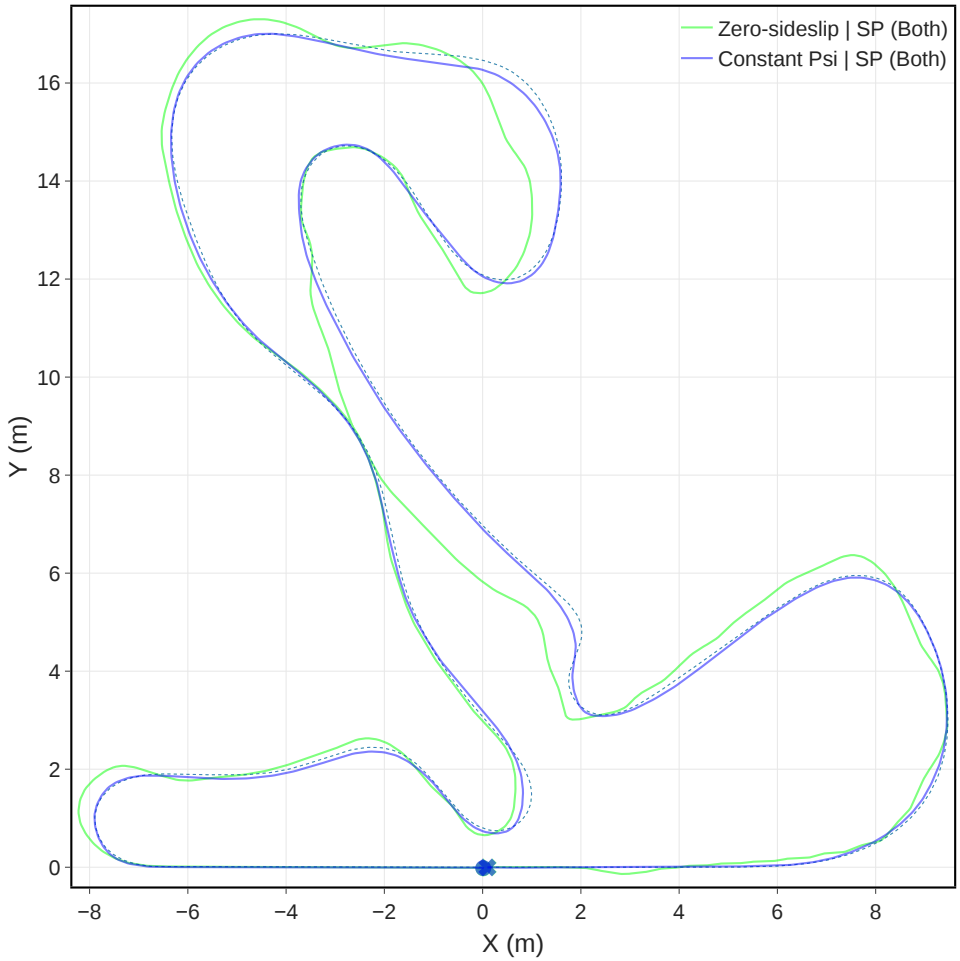


Figure 6.11: Top-down view of tracking performance for the Zandvoort F1 Track using the Student Policy (Both).

that the policy struggles to determine the appropriate UAV positions for novel trajectories involving varying orientations. This is likely due to overfitting to the limited set of three trajectory types used for training as discussed in the preceding section.

6.4.4. Full Pose Control

This section evaluates the student policy's capability to manipulate the complete pose of a payload. The policy receives a desired goal pose represented as a 6-dimensional vector (position and Euler angles) as input. A smooth trajectory is generated between the current payload pose (starting point) and the desired goal pose (endpoint), ensuring zero velocities at both trajectory endpoints. For this experiment, we trained a student policy over 80 DAgger iterations. Each iteration comprised a single episode in which the payload was manipulated from its current starting pose to a randomly selected target pose. The starting pose for each iteration corresponded to the final payload pose achieved in the preceding experiment. The target poses were uniformly sampled within $\pm 15^o$ for roll and pitch angles, with yaw angles sampled across the full range. Consequently, the student policy learned to transport the payload across a diverse set of initial and final pose configurations. All trajectories were executed over a 11-second duration.

Set point Tracking
We compare the student policy's payload manipulation capabilities against the teacher policy, as illustrated in Figure 6.12. Our analysis demonstrates that the student policy successfully manipulates the payload along the generated reference trajectory. Notably, the student policy achieves smoother payload manipulation compared to the teacher policy, which exhibits oscillatory behaviour due to noisy Extended Kalman Filter (EKF) measurements. However, the centralized teacher policy achieves superior accuracy in the final measured pose. The quantitative results are summarised in Table 6.6

Metric	Teacher Policy	Student Policies
RMSE Posn. (m)	0.015	0.097
RMSE Orient. (°)	1.631	3.516

Table 6.6: Comparison of pose tracking performance of the Teacher and Student policies.

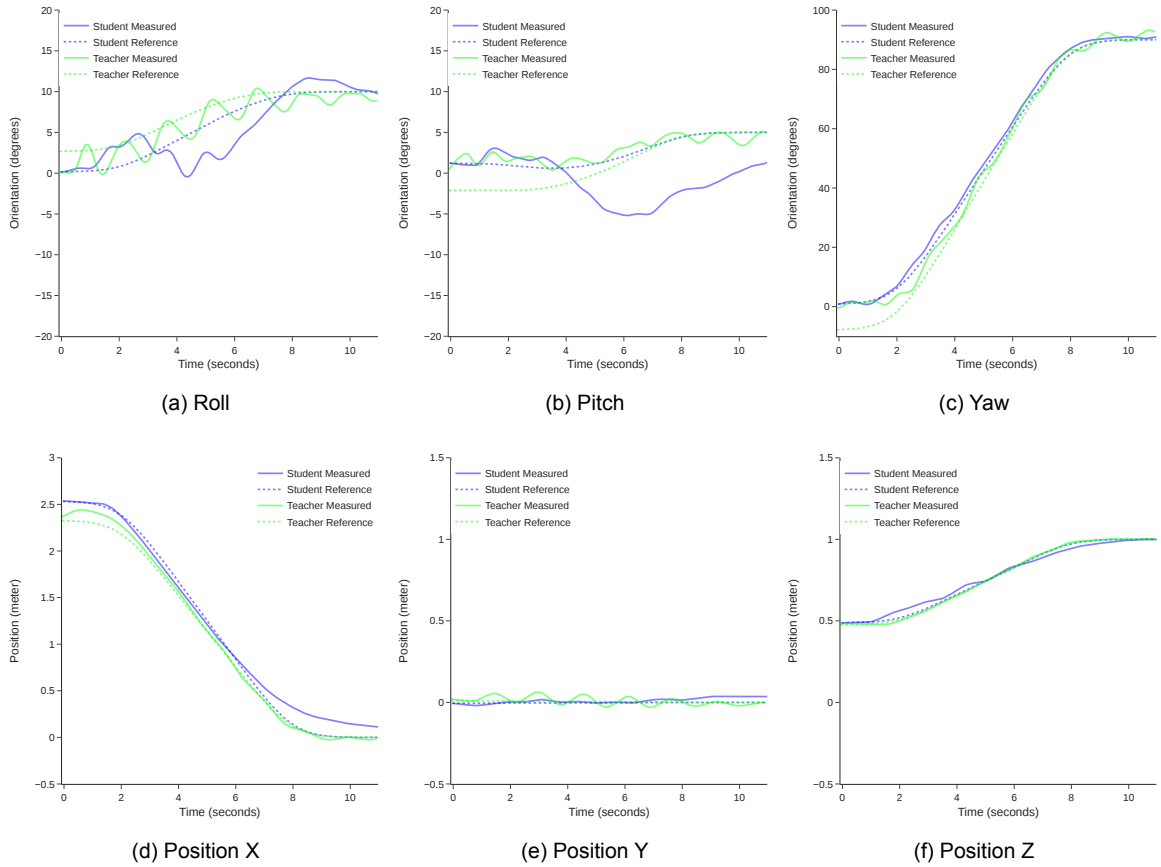
Generalization to Sequential Pose Commands
This experiment evaluates the student policy's ability to navigate through a sequence of randomly genrated poses. Consistent with the training protocol, the initial pose for each subsequent target depends on the previously achieved pose. Consequently, the starting orientation of the payload is not constrained to be horizontal (i.e., zero pitch and roll angles). Table 6.7 presents the performance metrics across multiple experimental trials. The results demonstrate an average position error of 0.058 meters and an average orientation error of 3.590 degrees.

S. No	Type	Start Pose	Goal Pose	Measured End Pose	Error
0	Position:	2.5465, 0.0034, 0.4888	-0.1662, -1.4837, 1.2471	-0.1520, -1.2842, 1.2322	0.2005
	Orientation:	0.7105, 1.1803, 0.3724	-8.8121, -2.7559, 70.3248	-9.6028, -5.4145, 70.5540	2.7884
1	Position:	-0.1212, -1.2827, 1.2262	4.9537, -2.2304, 0.7702	4.9709, -2.2464, 0.7591	0.0259
	Orientation:	-9.5741, -2.9049, 71.6770	-10.4049, -7.8381, 23.8293	-9.7747, -8.5313, 24.0012	0.9363
2	Position:	4.9582, -2.2542, 0.7562	0.9875, -0.6979, 0.8351	0.9979, -0.7218, 0.8349	0.0260
	Orientation:	-8.8006, -8.7949, 25.9951	-14.1062, 9.9981, 43.3958	-16.1230, 9.9351, 45.5272	2.6733
3	Position:	0.9885, -0.7285, 0.8323	-0.2504, -0.5382, 0.7834	-0.2590, -0.5208, 0.7731	0.0220
	Orientation:	-15.5100, 10.1585, 46.2320	9.2304, -3.7758, 72.6969	10.7258, -7.1104, 75.1778	4.3358
4	Position:	-0.2734, -0.5429, 0.7658	2.3408, -1.5267, 0.9428	2.3278, -1.5120, 0.9464	0.0200
	Orientation:	11.4821, -7.4198, 75.1721	5.7869, 11.2529, 65.9417	7.2479, 13.6307, 69.0070	4.3660
5	Position:	2.3254, -1.5146, 0.9465	1.2252, -2.8447, 0.8277	1.1981, -2.7954, 0.8325	0.0565
	Orientation:	7.1448, 13.6192, 69.0242	5.3572, -8.3308, 85.1186	5.9588, -13.9744, 88.3386	6.4663

Table 6.7: Sequential pose manipulation results demonstrating student policy generalization across diverse pose configurations. For each experiment, the table presents the initial pose (derived from the previous experiment's final position), the randomly generated target pose, the measured final pose achieved by the student policy, and the corresponding tracking error. Position coordinates are given in meters [x, y, z], orientation angles in degrees [roll, pitch, yaw], with position errors and orientation errors in the same units respectively.

6.4.5. Conclusion
From these experiments, we deduce the following:

- Our method has sufficient capacity to learn very different kinds of paths that the UAVs may need to follow for complex trajectories. This is evidenced by the *SP (Both)* being able to generate very different trajectories required for zero-sideslip trajectories. Some illustrations are shown in Figure 6.8



(a) Roll (b) Pitch (c) Yaw

(d) Position X (e) Position Y (f) Position Z

Figure 6.12: Full pose tracking performance: Orientation (top row) and Position (bottom row) components for the payload along the reference line trajectory. All plots have different Y axis scales. Note that the initial poses differ slightly between the student and teacher policies, as the teacher policy utilizes Extended Kalman Filter (EKF) measurements to establish the starting pose for trajectory generation.

- Our method is not able to generalize to trajectories that lie outside the domain of the trajectories in the training dataset. This is especially true for trajectories involving novel sequences of orientations. The centralised teacher policy could follow all trajectories tested without any fine-tuning. In contrast, our method needs to be trained from demonstrations on similar trajectories for optimal performance.

- Our solution is limited by the simulation environment - it is not possible to conduct thousands of simulations parallelly on Gazebo. Other simulators purpose built for reinforcement learning applications allow this. A model trained on tens of thousands of random trajectories might be able to generalise better to novel trajectories.

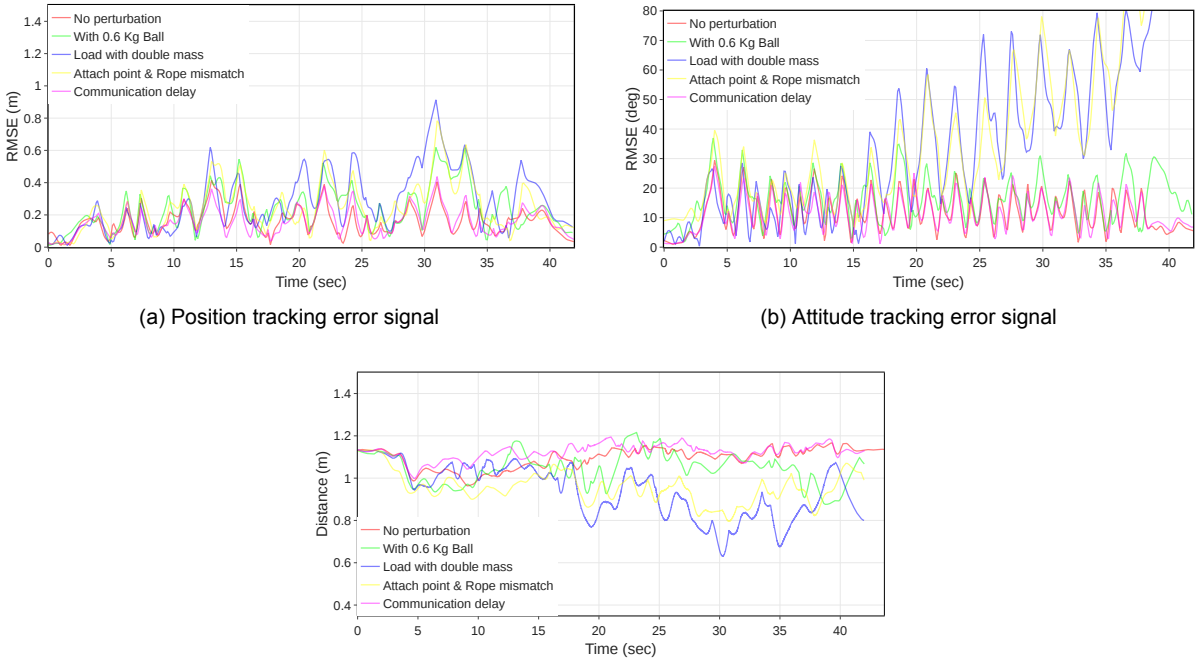
6.5. Robustness

In this section we test our method's ability to handle mismatch in model dynamics and communication delays - scenarios it may encounter in the real world. As a baseline, we take the Student Policy (Both) that was trained to navigate both constant psi and zero side-slip trajectories. We test this policy, without any find tuning, on the Square zero-side slip trajectory in multiple scenarios in Gazebo simulation. These scenarios are described below:

- **Sloshing Load:** A 0.6 kg ball is placed inside the 1.4 kg payload lifted by the UAVs. This ball is free to move relative to the payload structure, significantly altering the inertial

Experiment	Student Policy	RMSE Position (m)	RMSE Orientation (°)
No perturbation	SP (Both)	0.181	13.107
With 0.6 Kg Ball	SP (Both)	0.271	17.795
Load with double mass	SP (Both)	0.334	44.469
Attach point & Rope mismatch	SP (Both)	0.289	47.562
Communication delay	SP (Both)	0.178	12.586

Table 6.8: RMSE for position and orientation under different environment perturbations.



(a) Position tracking error signal

(b) Attitude tracking error signal

(c) Minimum inter-agent distances

Figure 6.13: Performance signals of Student Policy (Both) to measure robustness to various environmental disturbances.

characteristics of the system during flight. The disturbance becomes especially severe when the ball shifts rapidly, impacting the boundaries of the payload basket. Although this introduces unpredictable dynamics, our proposed method demonstrates robustness to these variations, albeit with some degradation in pose tracking accuracy.

- **Mass Variation:** The total payload mass is increased from 1.4 kg to 2.8 kg. The resulting higher inertia makes trajectory tracking particularly challenging, especially during sharp turns. Additionally, the increased cable tensions often disrupt the formation, occasionally bringing the UAVs close to each other. Despite these challenges, our approach remains functional, though with a significant reduction in tracking performance.

- **Attachment Point and Rope Length Variations:** All attachment points on the payload are shifted by a few centimetres. Furthermore, asymmetry is introduced by adjusting the lengths of two out of three suspension ropes: one is shortened by 3.5 cm while another is lengthened by the same amount. Our method exhibits high sensitivity to such changes in attachment geometry, with even small deviations leading to substantial performance degradation in orientation tracking.

- **Communication Delay:** To evaluate robustness to communication latency, we simulated delays in transmitting the student policy's predictions to the Agilicious flight stack.

Specifically, 20% of predictions were dropped, and for the remaining messages, a random delay of up to 30 ms was introduced. Due to our approach of generating receding horizon trajectories with a 2-second time window, degradation in communication did not adversely impact overall system performance.

6.6. Computation Time

Real-time inference is possible on the Raspberry Pi 5 CPU and takes 97 milliseconds. As the policy generates trajectories at 10 Hz, the computation time is just within the available computation budget. However, during experiments, we realized that running both the Agilicious flight control stack and PINN inference on the Raspberry Pi leads to a degradation in the inner control loop update rate. Therefore, we eventually deployed PINNs on a laptop and transmitted trajectories to each UAV through Wi-Fi. The average inference time of the student policy on an NVIDIA RTX 3060 Laptop GPU is 27 milliseconds. The computation time onboard the Raspberry Pi can be drastically reduced by using an AI accelerator along with optimised inference of the ML model.

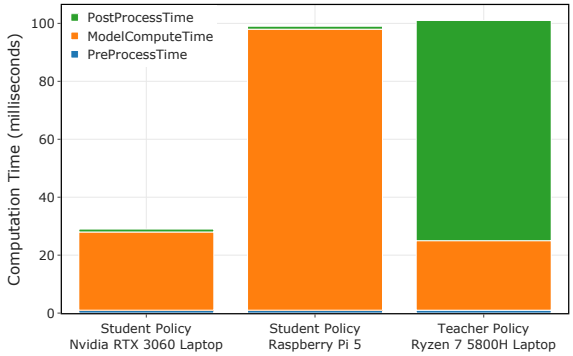


Figure 6.14: Computation time breakdown for teacher and student policies. Total time is divided into three components: (1) ModelComputeTime, representing the time required for inference using either the ML model or the OCP solver; (2) Pre-Processing; and (3) Post-Processing, which involve organizing input/output data for compatibility with the respective algorithms.

In comparison, the teacher policy requires an average computation time of 101 milliseconds (for 3 UAVs) running on a Ryzen 7 5700H CPU. Of this, the nonlinear model predictive control (NMPC) solver accounts for 24 milliseconds, while post-processing operations—such as computing executable drone trajectories from the solver outputs—consume an additional 76 milliseconds. An optimised C++ implementation can considerably reduce the overall computation time of the teacher policy.

Unlike the student policies, the teacher's computational complexity increases exponentially with the number of UAVs. In contrast, the student policy, due to its decentralized and partially observable architecture, exhibits the potential for scalability. By enabling each UAV to independently predict its own trajectory, the framework could support horizontal scaling, making it feasible to accommodate a large number of UAVs without any increase in computation time.

6.7. Training Time

To better understand the time requirements of our learning pipeline, we disaggregate the overall training time into two primary components: data collection and model training. In each round of DAgger, demonstration data is collected from a single episode. This is followed by training the machine learning model on the cumulative dataset of demonstrations collected until now. Each episode yields a 40-second trajectory, and given that our simulator operates

at 0.25× real-time, collecting one trajectory requires approximately 160 seconds. In contrast, training the model is relatively quick in the early stages but becomes progressively slower as the dataset grows. Table 6.9 illustrates the training durations for the various policies introduced in the previous sections.

Name	Rounds	Data Collection Time	Training Time	Total Time
SP (Real)	25	93.89	25.35	119.24
SP (Constant orientation)	30	97.62	104.72	202.34
SP (Zero sideslip)	30	88.76	98.23	186.99
SP (Both)	60	178.67	388.37	567.04

Table 6.9: Data collection, training, and total time for different SP configurations. All values are in minutes. SP (Real) was the policy deployed in real world experiments.

Our use of imitation learning enables efficient policy learning, significantly reducing training time compared to reinforcement learning techniques. By leveraging the full prediction horizon of the expert (teacher) policy, we enhance sample efficiency and accelerate convergence, even when constrained by a slow simulator. Despite these limitations, our method demonstrates robust performance.

Integrating faster, parallelisable simulation environments such as MuJoCo or Isaac Sim would facilitate quicker data collection across a broader set of trajectories. This, in turn, could enable the learning of more generalizable policies capable of adapting to diverse trajectories.

7

Conclusion

In this thesis, we present a deep learning policy capable of manipulating a payload along a desired trajectory while operating in a fully decentralised manner without requiring inter-agent communication. Our approach can execute trajectories that are far more agile and with better performance than any other decentralised method in the literature. This work constitutes the first solution to this problem that leverages modern deep learning–based control methods in conjunction with imitation learning. By learning from a centralised expert, the policy acquires desirable behaviours such as maintaining safe inter-agent distances, while remaining computationally efficient to train, even on modest hardware.

Circling back to the research questions posed at the beginning of this project, our findings demonstrate that employing strongly homogeneous student policies within a CTDE framework constitutes an effective approach for imitation learning. This methodology successfully transferred the capability to manipulate the full payload pose from the centralized model-based teacher policy to decentralized deep-learning based student policies. The primary challenge in achieving collaboration between student policies was maintaining safe inter-agent distances. This implicitly required students to understand their specific role based on their position, attachment to the load, and the provided reference trajectory. We established that effective coordination necessitates students having access to accurate payload pose information alongside data regarding their respective rope attachment points. Furthermore, utilizing the Load frame $\mathcal{L}$ ensured rapid policy convergence while maintaining invariance to trajectory's location in the inertial frame. Finally, providing reference trajectories of appropriate length enabled students to develop long-horizon planning capabilities, ensuring robust performance during trajectory initiation and termination phases as well as through dynamic trajectory segments.

A key factor in enabling real-world deployment is our abstraction of low-level control. The policy outputs high-level trajectories that are then tracked by cascaded low-level controllers. This approach significantly reduces the sim-to-real gap. To support this, we introduced a novel method that uses physics-informed neural networks to generate step-wise consistent and kinematically feasible trajectories. This ensures that the velocity and acceleration terms required by the low-level DFBC controller are accurate, thereby preserving the tracking performance of both the quadrotor and the payload. Furthermore, by using the entire trajectory generated by the online kinodynamic expert for training, the student policy gains the ability to perform the long-term planning needed for agile manoeuvres while improving sample efficiency as well.

We validated our solution through real-world experiments, comparing our decentralised policy against the state-of-the-art centralised expert. While our method showed slightly poorer position tracking performance, it matched the expert in orientation tracking. In simulation, the policy tracked both familiar and novel position sequences accurately, though it struggled with

novel sequences of desired orientations. Robustness testing demonstrated the policy's ability to handle challenging real-world disturbances—including sloshing payloads, large mass variations, attachment point and rope length perturbations, and communication delays—without fine-tuning. While performance degradation was observed—particularly for changes in attachment geometry—the policy maintained functional control in all scenarios, with communication delays having negligible effect due to the receding-horizon design.

7.1. Future Work

This research opens a new door for applying deep learning methods to collaborative aerial manipulation, particularly in a challenging scenario where there is no inter-agent communication. In such a setting, each UAV operates without the knowledge of the positions of other UAVs, making the system susceptible to inter-agent collisions. Future work could explore integrating onboard sensing systems or employing low-frequency communication channels to enhance the agents' awareness of each other. Such improvements would not only mitigate collision risks but also enable safer, decentralized coordination for a broader range of aerial manipulation tasks.

Our solution demonstrates decentralized, communication-free coordination between agents, with the key advantage that its computational complexity is independent of the number of agents in the environment. This property establishes a strong foundation for scaling to large multiagent systems. This is in contrast to centralized or even many decentralized approaches in the literature, where complexity often grows exponentially with the number of agents. A promising research direction would involve training a single "student" policy capable of generalizing to variable numbers of agents, enabling robust performance across a wide range of collaborative configurations.

In this work, we assumed perfect state information for both the UAVs and the payload which is not true for real-world deployments. Existing onboard state estimation techniques for UAVs, as well as the extended Kalman filter (discussed in Section 4.2) approach for payload state estimation, could be incorporated into our framework to bridge this gap. An interesting extension would involve fusing high-frequency ego-pose estimates with low-frequency pose information from other UAVs. As discussed in the preceding chapters, the low-frequency pose information can be obtained using onboard sensing or over a network. Such a solution could pave the way for a scalable decentralised collaborative aerial manipulation which is suitable for real world deployment.

Finally, while our method has successfully executed a variety of trajectories—such as figure-eight, circular, and square patterns—it shows limited ability to generalize to trajectories outside the training set. Although the system can track desired positions accurately, it struggles to manipulate the full pose of the payload in unfamiliar scenarios as demonstrated by the experiments on the Zandvoort F1 Track. This limitation is partly due to constraints of the Gazebo simulator. The slow execution speed of the simulator coupled with being unable to run multiple episodes concurrently limit the amount of novel trajectories on which we can train our solution in a reasonable time. Further, we are not able to introduce any randomness to the system model with this simulator which limits our ability to train robust policies. Transitioning to modern, high-performance simulators such as MuJoCo or IsaacSim could address most of these limitations. These simulators support running hundreds of episodes concurrently while also allowing us to randomly vary environmental parameters to improve policy robustness. This in turn would enable us to collect data on a large scale for thousands of different trajectories. Training policies on this diverse dataset should lead to the development of a more robust and generalizable policies.

A

Appendix

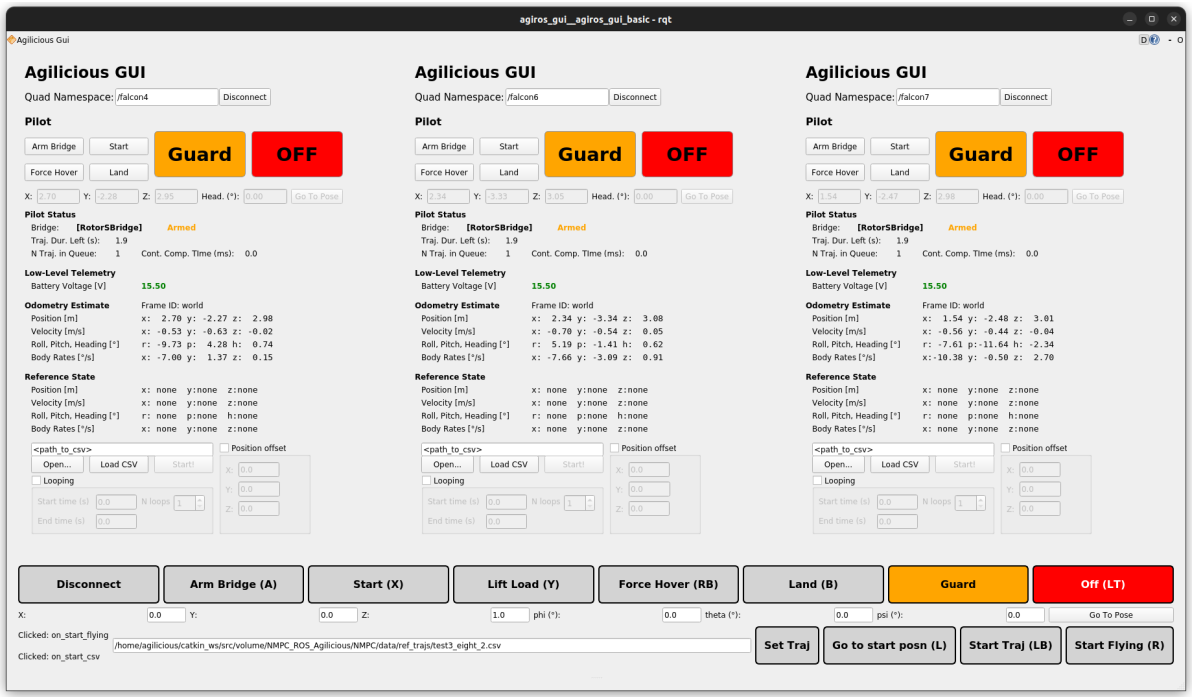
A.1. Human-Robot Interface

The human-robot interface for this system consists of 3 key components:

- **GUI:** A graphical-user interface allows the operator to check the status off all UAVs, including their state (position, velocity), their battery's charge level and the status of the flight controller. This GUI also allows the controller to choose the desired trajectory along which the payload must be transported by the UAVs. Figure A.1a shows the user interface.
- **Controller:** The system can be operated using a handheld controller connected to the ground station via Bluetooth, enabling the operator to issue high-level commands. The controller is equipped with a dead-man switch mechanism: releasing a designated trigger immediately powers down all UAVs, thereby enhancing safety and minimizing the risk of hardware damage. The operator can initiate high-level actions such as activating the Agilicious flight stack, taking off, landing, commanding the UAVs to hover, and shutting them down. Additionally, the controller allows the operator to start the payload transport trajectory. Upon initiation, the system's policy begins generating the corresponding UAV trajectories, which are displayed to the operator for verification. Only after manual confirmation can the UAVs begin following the predicted trajectories. The controller's button mapping is illustrated in Figure A.1b.
- **RViZ:** This interface allows the operator to see a visual of the state of the load and all UAVs. This interface also allows the operator to see the trajectories generated by the teacher or the student policies. This feature is particularly useful as it allows the operator to visually confirm the generated trajectories make sense before enabling UAVs to fly these trajectories thereby giving the operator meaningful control over the system.

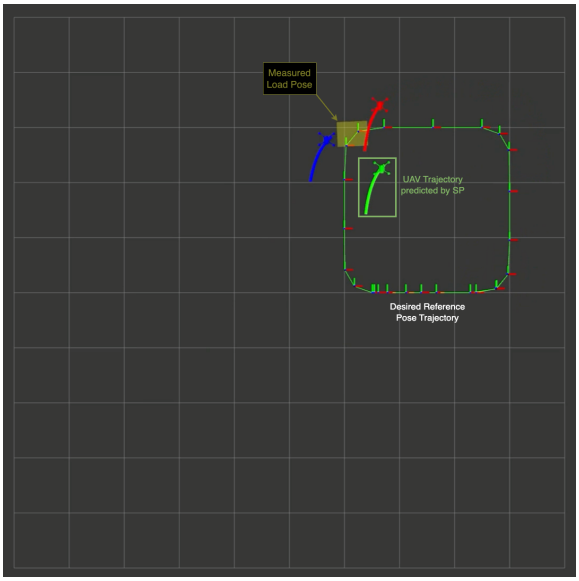
A.2. Safety Module

The safety module is designed to continuously observe the state of the UAVs and the trajectories generated by the policies. Based on this data, the safety node can take measures to prevent collisions or damage to the UAVs.



(a) User interface for controlling the system, extending the original Agilicious GUI for the multi-UAV carrying system. Users can select the desired trajectory in the menu at the bottom.

(b) Xbox controller button mappings. The controller includes a deadman switch that, when released, turns off all UAVs immediately.



(c) RViZ interface showing the current state of the system, including the load, UAVs, and predicted trajectories (red, green, blue lines).

Figure A.1: Human-robot interface: (a) GUI for system control and trajectory selection; (b) Xbox controller for issuing high-level commands; (c) RViZ visualisation of UAV and load states and trajectories.

The safety node has the following features:

- **Position Bounds:** The safety node continuously monitors the pose of all UAVs and turns them off if they go beyond the safe boundaries of the Arena.
- **Velocity and Acceleration Bounds:** The safety node monitors the twist and acceleration of all UAVs and forces them to hover in place if these values are more than the maximum permissible limit.

- **Distance Bounds:** Using the pose measurements of all UAVs, the safety node monitors whether any two drones are coming too close to each other. If UAVs come closer than the minimum permissible distance, the UAVs are turned off.
- **Future Reference Check:** The safety node also monitors the reference trajectories of the UAVs to ensure any of the aforementioned bounds are not violated. In case of violations, the UAVs are forced to hover in place.

For all instances where the safety node intervenes, a message showing the error is also displayed on the GUI shown in Figure A.1a.

A.3. Implementation of the Teacher Policy

To implement the teacher policy for training the student agent, two main options were considered: using the existing Simulink model based on the Acados framework, or re-implementing the model in Python using Acados. Both approaches offered distinct advantages and limitations, which are discussed below.

A.3.1. Option 1: Simulink Model with Acados (ROS Integration)

The first option involved leveraging an existing Simulink model that used Acados for control. This model could be integrated into a ROS-based pipeline, where it would serve as a ROS node. This approach presented some advantages:

- **Ease of Integration with ROS**: The model was already structured for communication over ROS topics or services, allowing for straightforward interaction with the simulation environment or real hardware.
- **Demonstration Collection in Python**: Once the model's outputs were accessible from Python (e.g., via ROS), it became relatively easy to collect demonstration data for imitation learning purposes.

However, several limitations made this option less attractive:

- **Lack of Tight Integration with SB3**: There was no clear mechanism for embedding this ROS-interfaced policy directly into an SB3 training loop as a callable policy, which is essential for the imitation learning pipeline.
- **Poor Scalability and Performance**: It was difficult to run multiple instances of Simulink for each environment running in parallel. In addition, the model was design for inference only and no mechanism was present to reset the model for new episodes. This would limit our ability to generate large volumes of demonstrations.

A.3.2. Option 2: Python-Based Reimplementation Using Acados

The second option was to reimplement the teacher policy entirely in Python, using the Acados solver interface. This approach came with its own trade-offs.

The main challenge was the development effort required:

- **High Initial Complexity**: Re-implementing the nonlinear optimal control problem (OCP) in Python was a significant undertaking.

However, this option offered several compelling advantages:

- **Clarity of Existing Model**: The mathematical formulation and design of the controller were already available in the Simulink version, which could be used as a reference.

- **Integrated Training Pipeline**: A Python-based implementation allowed the teacher and student policies to coexist in the same training process, enabling direct calls and data exchange during training (e.g., for DAgger).
- **Efficient Parallelization**: This implementation could be tailored for fast rollout generation and vectorized environments, which is critical for efficient training.

A.3.3. Final Decision

After evaluating both options, the Python-based Acados implementation was chosen. While it required additional development effort upfront, this approach provided superior flexibility, training performance, and integration capabilities. It enabled the use of vectorized environments and seamless coordination with SB3 RL algorithm, making it the most suitable choice for implementing the teacher policy in an imitation learning setting.

A.4. 2D Collaborative Square Move

A.4.1. Motivation

We started with the hypothesis: Can a centralised teacher policy with full observability be used to transfer to decentralised multiple student policies each operating with partial observability using imitation learning? In literature, Lin et al. have validated part of this statement through their method CESMA (Section 3.1.3). However, they only consider environments with full observability and communication between the agents. The environment we created is a 2D equivalent environment for Collaborative Aerial Manipulation. To test this hypothesis, we started by creating two-dimensional version of Decentralised Collaborative Aerial Manipulation. This environment is computationally fast and allows us to experiment and verify the hypothesis quickly.

A.4.2. Environment

Consider a 2D environment in which two or more agents (blue cylinders) are working together to drag a payload (red cylinder) along a reference trajectory. The agents are attached to the body using rods via a universal joint. The environment can be configured with varying number of agents, a representation of the environment is shown in Figure A.2 for 4 agents. Each blue cylinder can be actuated by applying a force on it's centre of mass along the X/Y direction. The environment is controlled by two types of policies: 1) the centralised teacher policy that jointly controls all the agents; or 2) student policies where each agent is controlled by its own decentralised policy.

To test our hypothesis, we start by training a centralised teacher policy using PPO reinforcement learning algorithm. We then utilize DAgger to train the decentralized student policies using demonstrations collected from the teacher policy. The students operate in a partially observable environment and therefore the agent knows the pose and twist of the payload and itself as well as the reference trajectory.

The student policies observations are:

$$\boldsymbol{o}_i = \begin{bmatrix} \boldsymbol{p}_L^{\mathcal{L}} & \boldsymbol{\psi}_L^{\mathcal{L}} & \boldsymbol{v}_L^{\mathcal{L}} & \dot{\boldsymbol{\psi}}_L^{\mathcal{L}} & \boldsymbol{p}_i^{\mathcal{L}} & \boldsymbol{\psi}_i^{\mathcal{L}} & \boldsymbol{v}_i^{\mathcal{L}} & \dot{\boldsymbol{\psi}}_i^{\mathcal{L}} & \boldsymbol{p}_G^{\mathcal{L}} & \boldsymbol{\psi}_G^{\mathcal{L}} \end{bmatrix} \tag{A.1}$$

The teacher policy operates with the information of all agents and its observation space is defined as:

$$\boldsymbol{o}_t = \bigcup_{i=1}^{N} \boldsymbol{o}_i \tag{A.2}$$

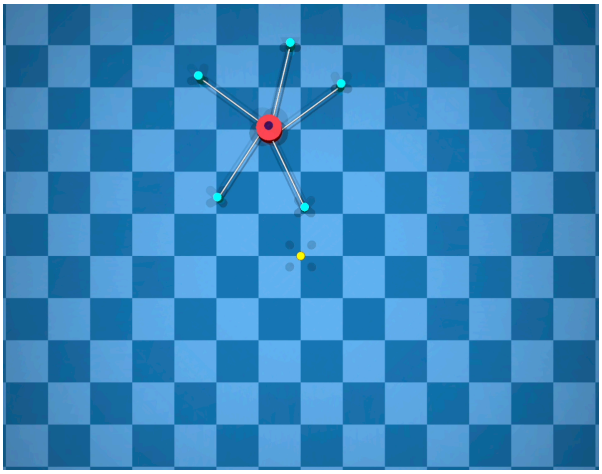


Figure A.2: The 2D environment: Red cylinders represents the payload while blue cylinders are the agents. The instantaneous position of the reference trajectory is shown as the blue sphere. The origin is represented by the yellow sphere.

Here $\boldsymbol{o}_i$ represents the observation of the i$^{\text{th}}$ agent. $p \in \mathbb{R}^2$ represent position in XY plane, $\psi \in \mathbb{S}^1$ represents the yaw angle and $v \in \mathbb{R}^2$ represents velocity. The subscript i, L, G represents the agent, load body and the goal pose. N is the number of agents in the environment.

The observations are provided in the Load body frame as defined in Section 4.1. By providing the observations in the Load frame, we are able to substantially increase the generalisability of the policy since it limits the domain of the observations encountered by the policy. In practice, this change greatly reduced the time to convergence for both the teacher and student policies.

The student policies are strongly homogenous Section 3.4. The action space of the student policy is:

$$\boldsymbol{a}_i = \pi_S(\boldsymbol{o}_i), \quad \boldsymbol{a}_i \in \mathbb{R}^2, \quad \text{with} \quad \boldsymbol{a}_i \in [-1, 1]^2 \tag{A.3}$$

The teacher policy controls all the students simultaneously, and its action space is given by:

$$\boldsymbol{a}_i = \pi_T(\boldsymbol{o}_t), \quad \boldsymbol{a}_t \in \mathbb{R}^{2*N}, \quad \text{with} \quad \boldsymbol{a}_i \in [-1, 1]^{2*N} \tag{A.4}$$

A.4.3. Training Environment

In each new experiment, the body and the goal are randomly spawned within the playing area. The agents are always spawned at a fixed location relative to the body (respecting length limit of the rope). The reward depends on the distance of the body from the goal and the magnitude of force applied by the agents.

Let d be the euclidean distance between the body and the goal at a particular time step and e be the L2 norm of the action signal (all signals are between -1 and 1). The reward at the time step is given by:

$$\text{rew\_time} = 0 \quad \text{if} \quad d < 0.05 \quad \text{otherwise} \quad -1 \tag{A.5}$$

Intuition: No penalty if body is very close to goal. If the distance is within this threshold for 100 steps, environment terminates successfully. Otherwise, counter resets to 0.

$$\text{rew\_goal} = \exp(-d) \tag{A.6}$$

Intuition: Reward increases as the distance between body and goal reduces. Motivates agent to push the body towards goal.

$$\text{rew\_energy} = -0.05 \cdot \exp(e) \tag{A.7}$$

Intuition: Small penalty for applying forces. Motivate agent to stop moving when body is within the threshold distance to goal.

$$\text{rew\_total} = \text{rew\_goal} + \text{rew\_energy} + \text{rew\_time} \tag{A.8}$$

A.4.4. Training agents

PPO Agent

The following parameters were used for training:

- 96 parallel environments are launched for training. This maximizes RAM utilization.
- 300 environment steps for each environment are executed per update. Therefore, rollout buffer is 96*300. This value was chosen based on some intuition from PPO for MARL paper.
- 4 mini batches are used. Batch size is 96*300/4. Again from PPO for MARL paper.
- An MLP policy is used. Last 3 observations are flattened and input to the model. Therefore, input size is 48. Policy is centralized and outputs 4 numbers between -1 and 1.
- The policy is trained for a total of 19,200,000 steps. This takes about 25 minutes to complete.

DAGGER Centralized

- The expert is the PPO agent. Observation (48) and action (4) spaces are exactly same for both PPO and Dagger.
- β starts at 1 and decreases linearly (20 steps). Higher beta equals higher probability of expert carrying out actions in the environment.
- The agent is trained about 25 times. Training is done when data for at least 10 unique episodes is collected with at least 4000 steps.
- Algorithm is trained for a total of 100,000 steps. Takes about 10 minutes to run.

DAGGER Decentralised

- Agents are homogenous and share weights. The expert is the PPO agent.
- Only agent specific information is made available. Observation from past 3 steps is given. Therefore, observation size is 30 and action (output) size is 2.
- The agent is trained about 25 times. Training is done when data for at least 10 unique episodes is collected with at least 4000 steps.
- 2 data points are collected from each step (one for each agent). Therefore, the data for training this agent is double that of the DAGGER centralized.
- Algorithm is trained for a total of 100,000 steps. Takes about 10 minutes to run.

A.4.5. Results

To compare the performance of all 3 approaches, we try the agents on 60 randomly initialised environments. The agents are tested on the same set of *seeds* across 3 metrics:

- Steps: The number of steps taken to complete the experiment. If steps are equal to 1500, the episode failed.
- Reward: Total reward received by the agent as per the aforementioned reward function of the environment.
- Energy: Sum of the L2 norm of the action signal of both the agents.

The PPO agent won 25 times while DAGGER Cent won 20 times and DAGGER Decent won 15 times.

Approach	Energy	Steps	Reward
dagger_decent_out	573.37778575	352.46666667	71.13916271
dagger_cent_out	589.70707957	348.35	78.28487959
ppo_out	600.54234517	351.01666667	80.53787086

Table A.1: Average values of 3 metrics across 60 seeds

Bibliography

[1] S. Habibi, N. Ivaki, and J. Barata, “A systematic literature review of unmanned aerial vehicles for healthcare and emergency services,” 2025. [Online]. Available: https://arxiv.org/abs/2504.08834

[2] S. Goessens, C. Mueller, and P. Latteur, “Feasibility study for drone-based masonry construction of real-scale structures,” *Automation in Construction*, vol. 94, pp. 458–480, Oct. 2018.

[3] N. Pereira da Silva and S. Eloy, *Robotic Construction: Robotic Fabrication Experiments for the Building Construction Industry*. Springer International Publishing, Sep. 2020, pp. 97–109.

[4] C. Picus, “Can drones build your house? - a research on heavy lifting drone applications in the construction industry,” July 2021. [Online]. Available: http://essay.utwente.nl/86550/

[5] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi, “Past, present, and future of aerial robotic manipulators,” *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 626–645, 2022.

[6] G. Li and G. Loianno, “Nonlinear model predictive control for cooperative transportation and manipulation of cable suspended payloads with multiple quadrotors,” in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2023, p. 5034–5041. [Online]. Available: http://dx.doi.org/10.1109/IROS55552.2023.10341785

[7] S. Sun, X. Wang, D. Sanalitro, A. Franchi, M. Tognon, and J. Alonso-Mora, “Agile and cooperative aerial manipulation of a cable-suspended load,” 2025. [Online]. Available: https://arxiv.org/abs/2501.18802

[8] T. Lee, K. Sreenath, and V. Kumar, “Geometric control of cooperating multiple quadrotor uavs with a suspended payload,” in *52nd IEEE Conference on Decision and Control*, 2013, pp. 5510–5515.

[9] G. Tartaglione, E. D’Amato, M. Ariola, P. S. Rossi, and T. A. Johansen, “Model predictive control for a multi-body slung-load system,” *Robotics and Autonomous Systems*, vol. 92, pp. 1–11, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889016305516

[10] I. Moreno Caireta, “Planning and control of a multiple-quadcopter system cooperatively carrying a slung payload in dynamical environments. a centralized model predictive control solution,” Ph.D. dissertation, UPC, Facultat d’Informàtica de Barcelona, Jun 2018. [Online]. Available: http://hdl.handle.net/2117/121147

[11] M. Kamel, M. Burri, and R. Siegwart, “Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[12] J. Wehbeh, S. Rahman, and I. Sharf, “Distributed model predictive control for uavs collaborative payload transport,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 11 666–11 672.

[13] H. G. d. Marina and E. Smeur, “Flexible collaborative transportation by a team of rotorcraft,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 1074–1080.

[14] A. Tagliabue, M. Kamel, S. Verling, R. Siegwart, and J. Nieto, “Collaborative transportation using mavs via passive force control,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5766–5773.

[15] M. Tognon, C. Gabellieri, L. Pallottino, and A. Franchi, “Aerial co-manipulation with cables: The role of internal force for equilibria, stability, and passivity,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2577–2583, 2018.

[16] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, “Learning by cheating,” *CoRR*, vol. abs/1912.12294, 2019. [Online]. Available: http://arxiv.org/abs/1912.12294

[17] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadrupedal locomotion over challenging terrain,” *CoRR*, vol. abs/2010.11251, 2020. [Online]. Available: https://arxiv.org/abs/2010.11251

[18] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, “Deep drone acrobatics,” *CoRR*, vol. abs/2006.05768, 2020. [Online]. Available: https://arxiv.org/abs/2006.05768

[19] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, “A survey of imitation learning: Algorithms, recent developments, and challenges,” 2023. [Online]. Available: https://arxiv.org/abs/2309.02473

[20] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, “AMP: adversarial motion priors for stylized physics-based character control,” *CoRR*, vol. abs/2104.02180, 2021. [Online]. Available: https://arxiv.org/abs/2104.02180

[21] S. Ross, G. J. Gordon, and J. A. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” 2011. [Online]. Available: https://arxiv.org/abs/1011.0686

[22] A. T. Lin, M. J. Debord, K. Estabridis, G. A. Hewer, and S. J. Osher, “CESMA: centralized expert supervises multi-agents,” *CoRR*, vol. abs/1902.02311, 2019. [Online]. Available: http://arxiv.org/abs/1902.02311

[23] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437029800023X

[24] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. [Online]. Available: https://www.marl-book.com

[25] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. M. Bayen, and Y. Wu, “The surprising effectiveness of MAPPO in cooperative, multi-agent games,” *CoRR*, vol. abs/2103.01955, 2021. [Online]. Available: https://arxiv.org/abs/2103.01955

[26] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *CoRR*, vol. abs/1706.02275, 2017. [Online]. Available: http://arxiv.org/abs/1706.02275

[27] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, “acados – a modular open-source framework for fast embedded optimal control,” *Mathematical Programming Computation*, 2021.

[28] J. Ferrin, R. Leishman, R. Beard, and T. McLain, “Differential flatness based control of a rotorcraft for aggressive maneuvers,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2688–2693.

[29] E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon, “Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles,” *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 3, pp. 450–461, 2016. [Online]. Available: https://doi.org/10.2514/1.G001490

[30] J. Gao, Z. Wang, Z. Xiao, J. Wang, T. Wang, J. Cao, X. Hu, S. Liu, J. Dai, and J. Pang, “Coohoi: Learning cooperative human-object interaction with manipulated object dynamics,” 2024. [Online]. Available: https://arxiv.org/abs/2406.14558

[31] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, “On the continuity of rotation representations in neural networks,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5738–5746.

[32] J. Mayer, J. Westermann, J. P. G. H. Muriedas, U. Mettin, and A. Lampe, “Proximal policy optimization for tracking control exploiting future reference information,” *CoRR*, vol. abs/2107.09647, 2021. [Online]. Available: https://arxiv.org/abs/2107.09647

[33] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, “Temporal convolutional networks for action segmentation and detection,” in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 156–165.

[34] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, “A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight,” *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3357–3373, 2022.

[35] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[36] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, “An expert’s guide to training physics-informed neural networks,” 2023. [Online]. Available: https://arxiv.org/abs/2308.08468

[37] R. Gnanasambandam, B. Shen, J. Chung, X. Yue, Zhenyu, and Kong, “Self-scalable tanh (stan): Faster convergence and better generalization in physics-informed neural networks,” 2022. [Online]. Available: https://arxiv.org/abs/2204.12589

[38] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[39] M. Kaup, C. Wolff, H. Hwang, J. Mayer, and E. Bruni, “A review of nine physics engines for reinforcement learning research,” 2024. [Online]. Available: https://arxiv.org/abs/2407.08590

[40] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza, “Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight,” *Science Robotics*, vol. 7, no. 67, p. eabl6259, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abl6259

Decentralized Real-Time Planning for Multi-UAV Cooperative Manipulation via Imitation Learning

Shantnav Agarwal[1], Javier Alonso-Mora[1], and Sihao Sun[1]

***Abstract*— Existing approaches for transporting and manipulating cable-suspended loads using multiple UAVs along reference trajectories typically rely on either centralized control architectures or reliable inter-agent communication. In this work, we propose a novel machine learning–based method for decentralized kinodynamic planning that operates effectively under partial observability and without inter-agent communication. Our method leverages imitation learning to train a decentralized student policy for each UAV by imitating a centralized kinodynamic motion planner with access to privileged global observations. The student policy generates smooth trajectories using physics-informed neural networks that respect the derivative relationships in motion. During training, the student policies utilize the full trajectory generated by the teacher policy, leading to improved sample efficiency. Moreover, each student policy can be trained in under two hours on a standard laptop. We validate our method in both simulation and real-world environments to follow an agile reference trajectory, demonstrating performance comparable to that of centralized approaches.**

I. Introduction

Unmanned aerial vehicles (UAVs) have gained significant traction across domains such as surveillance, agriculture, and infrastructure inspection due to their agility and versatility. However, their limited payload capacity restricts their effectiveness in applications involving the transportation of heavy or bulky objects which is common in construction and large-scale logistics. A scalable and cost-effective solution to this limitation is cable-suspended cooperative aerial manipulation [1], where multiple UAVs cooperatively transport and control a cable-suspended payload. This method enables full pose manipulation of objects whose weight may exceed the capacity of a single UAV.

Numerous control strategies have been proposed for cooperative transportation of suspended payloads using UAV teams. These approaches vary in terms of modeling accuracy, scalability, communication requirements, and capability to regulate the full pose of the payload. Given the focus of this work on decentralized cooperative aerial manipulation, prior methods are categorized into three primary frameworks: centralized control, decentralized control with communication, and decentralized control without communication.

1) Centralized Approaches: A formation-based geometric control approach was introduced in [2], modeling a point-mass suspended via massless rigid links. This method centrally solves for force equilibrium using full system state information, allowing formation maintenance but neglects payload attitude. Moreover, the approach has only been validated in simulation.

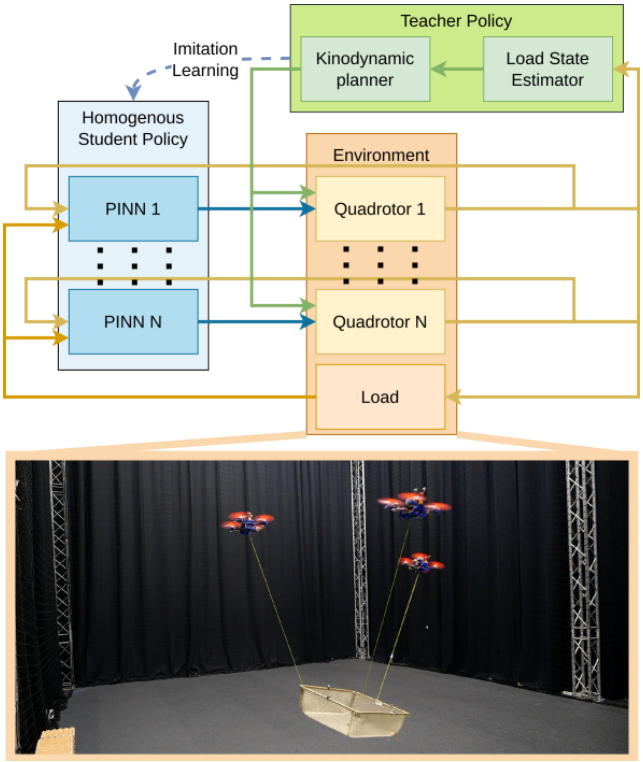


Fig. 1: We enable decentralized cooperative aerial manipulation through student policies that operate independently using only the ego UAV's state and the pose of the load. These student policies are trained via imitation learning from a centralized teacher policy with privileged observations, including the full state of the other UAVs and the load. The policy has been tested in real-world environments, where three UAVs cooperatively manipulate a cable-suspended load.

Model Predictive Control (MPC) is commonly used in centralized frameworks due to its ability to handle constraints and optimize control over a receding time horizon. Linear MPC approaches typically model the payload as a point mass and use linearized system dynamics [3], [4]. While computationally efficient, these methods cannot control the payload's orientation and often under perform on dynamic trajectories [5].

Nonlinear MPC (NMPC) techniques have been introduced that allow manipulating the full-pose of the load along agile trajectories. For example, a cascaded NMPC controller has

[1] Authors are with the Department for Cognitive Robotics, ME, Delft University of Technology, Delft, Netherlands

been developed that enables full pose manipulation of the load[6] . However, this controller assumes load dynamics are a magnitude slower than quadrotor dynamics, which reduces control performance on agile trajectories. Recently, in [7], an online kinodynamic motion planner utilizing a load-cable-UAVs model without assumption of time-scale separations has been proposed, enabling agile full-pose control.

While centralized controllers have achieved agile and accurate load manipulation, they rely on high-bandwidth, low-latency communication networks with a central coordinator. This assumption is often impractical in real-world scenarios where communication may be unreliable or unavailable. Further, the computational cost of aforementioned approaches grows exponentially with the number of agents thereby limiting scalability. Hence, the community has also explored decentralized methods for cooperative aerial manipulation.

2) Decentralized Approaches with Communication: A distributed model predictive control framework has been developed which demonstrates performance comparable to centralized MPC while distributing the computational burden across agents[8]. The approach assumes a fully-connected communication graph, wherein each UAV shares its previous control inputs with neighboring agents and independently optimizes its own input sequence at each iteration. Though validated only in simulation, the framework exhibits scalability and coordination capabilities. While the method is scalable, it still relies on continuous inter-agent communication, motivating the need for decentralized control strategies that operate without explicit communication.

3) Decentralized Approaches without Communication: A decentralized control scheme combining distance-based formation control with incremental nonlinear dynamic inversion (INDI) is proposed in [9]. UAVs compute local acceleration commands based solely on relative position measurements, eliminating the need for global positioning or inter-agent communication. This method, however, does not regulate the payload's orientation. Master-slave architectures have also been employed, where a designated master UAV generates a reference trajectory, and slave UAVs follow using force-based admittance control [10], [11]. These approaches are limited in their ability to manipulate the payload's full pose and have not been extended beyond two-UAV systems.

While decentralized approaches without inter-agent communication offer benefits like scalability and robustness, they struggle to achieve agile and precise full-pose manipulation of a payload. To address this challenge, we present a decentralized, communication-free online kinodynamic motion planner for cooperative aerial manipulation. Our method enables multiple autonomous UAVs to perform agile and accurate full-pose control of a rigid object. Each UAV operates independently, relying solely on local observations and the object's reference trajectory, to generate it's own receding-horizon trajectories. This trajectory is then followed by low-level trajectory tracking controllers deployed onboard the UAV. To achieve this, each UAV is equipped with a neural network policy trained through imitation learning from a privileged expert [12]. A centralized receding-horizon kinodynamic motion planner serves as the teacher policy [7], guiding the training of the strongly homogeneous [13] decentralized student policies. We assume that each UAV can accurately estimate both its own state and the state of the payload. To the best of our knowledge, this is the first work to develop a decentralized, communication-free planner for cooperative cable-suspended aerial manipulation. We have validated our approach in both simulation and real-world experiments.

II. Problem Statement

A. Notations

We consider a system composed of n unmanned aerial vehicles (UAVs) and a single rigid load body. The pose and twist of the i^{th} UAV are denoted by $[\boldsymbol{p}_i \quad \boldsymbol{q}_i]$ and $[\boldsymbol{v}_i \quad \boldsymbol{\omega}_i]$, respectively, where $\boldsymbol{p}_i \in \mathbb{R}^3$ and $\boldsymbol{q}_i \in \mathbb{S}^3$ represent the position and orientation (quaternion), and $\boldsymbol{v}_i, \boldsymbol{\omega}_i \in \mathbb{R}^3$ denote the linear and angular velocities. We use bold lowercase letters to denote vectors, and bold capitalized letters for matrices; otherwise, variables are scalars.

Similarly, the pose and twist of the load are represented as $[\boldsymbol{p}_L \quad \boldsymbol{q}_L]$ and $[\boldsymbol{v}_L \quad \boldsymbol{\omega}_L]$, with analogous definitions.

A load-attached reference frame, denoted $\mathcal{L}$, is defined such that its origin coincides with $\boldsymbol{p}_L$, and its orientation is aligned with the inertial frame $\mathcal{I}$.

Each UAV has an associated body-fixed frame $\mathcal{B}_i = [\boldsymbol{O}_i \quad \boldsymbol{x}_i \quad \boldsymbol{y}_i \quad \boldsymbol{z}_i]$, where the origin $\boldsymbol{O}_i$ is located at $\boldsymbol{p}_i$, and the z_i-axis points vertically upward, coinciding with the thrust direction.

Unless otherwise stated, all vectors are expressed in the inertial frame $\mathcal{I}$. Frame-specific quantities are indicated using superscripts when necessary.

B. Cooperative Aerial Manipulation

cooperative aerial manipulation refers to the coordinated use of multiple aerial robots to jointly transport and manipulate a common object. A representative depiction of this setup is provided in Figure 1. The system consists of multiple UAVs, each connected to a rigid load via cables. The cables are attached below the center of mass of each UAV, and at arbitrary points on the load, using universal joints that allow the transmission of forces but not moments.

Each UAV is modeled as a unidirectional thrust platform, capable of generating thrust solely along its body-fixed positive z_i-axis. The primary objective is for the UAVs to collaboratively control the full pose of the load—both its position and orientation—so that it tracks a desired trajectory over time.

C. Fully Decentralized Cooperative Aerial Manipulation

Fully decentralized cooperative aerial manipulation aims to achieve full-pose control of a load using multiple UAVs, where each UAV operates independently based on local observations and without direct communication with other UAVs.

This decentralized setting introduces several key constraints and challenges - *Distributed Control:* Each UAV

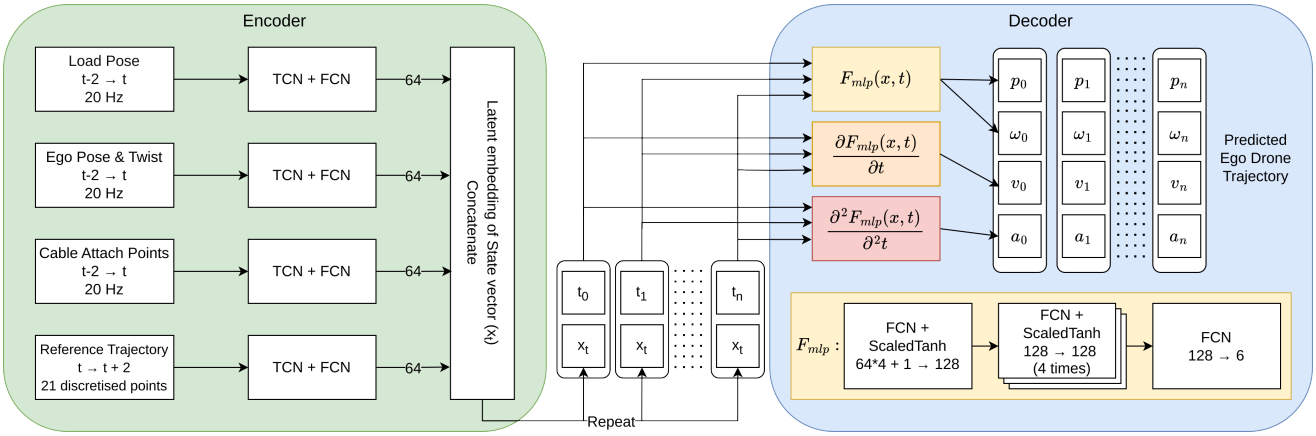


Fig. 2: The architecture of the student policy is made up of two parts - Encoder (Sec. III-B.1) & Decoder (Sec. III-B.2). The Encoder network maps the observation histories to a latent vector x_t. This latent vector is then copied for all nodes in the horizon and passed to the Decoder network.

is governed by an onboard control policy and functions autonomously, without access to the control inputs or internal states of other UAVs. *Local Observations:* Each UAV's planning and control policy relies solely on locally available sensor data, including its own pose and twist, as well as the pose of the load. No information about the states or actions of other UAVs is accessible. *Implicit Coordination:* UAVs must learn to collaborate effectively during the training phase, such that during deployment, coordinated behavior emerges through physical interactions alone and without any explicit communication channels. This framework reflects practical limitations in real-world multi-robot systems, where communication constraints, scalability, and robustness to failure are critical considerations.

III. Method

This section introduces the proposed control policy, including details on the training procedure, observation & action space as well as the architecture of the learned policy.

A. Teacher Policy

We use the planning algorithm introduced in [7] as the teacher policy, which is the state-of-the-art centralized method for the multi-lifting system for full-pose control of a cable-suspended load, with high agility and robustness. Given the reference trajectory of the load, the teacher policy solves an optimization-based kinodynamic motion planning problem and generates receding-horizon reference trajectories for each UAV at 10 Hz. The reference trajectories are then followed by low-level controllers deployed on each UAV, using a flatness-based controller and INDI [14] to compensate for external forces and torques from the cable.

B. Student Policy

While the centralized policy demonstrates high agility and robustness against dynamic model uncertainties, it relies on access to the full state information of all UAVs and the load. In contrast, the student policy learns to generate trajectories for the ego UAV in a decentralized manner, using only the ego UAV's state and the load's pose. Notably, the student policy shares the same low-level controllers as the teacher policy. It is implemented as a strongly homogeneous machine learning policy—meaning all UAVs run identical copies of the policy, with the same architecture and weights [13]. The architecture of the student policy is depicted in Figure 2, and an overview of the overall control framework is illustrated in Figure 1. The policy operates at a control frequency of 10 Hz, consistent with the teacher policy.

1) Observation Space: The student policy has access to the following information:

$$\boldsymbol{o}_i = \begin{bmatrix} \boldsymbol{p}_L^{\mathcal{L}} & \boldsymbol{q}_L^{\mathcal{L}} & \boldsymbol{p}_i^{\mathcal{L}} & \boldsymbol{v}_i^{\mathcal{L}} & \boldsymbol{q}_i^{\mathcal{L}} & \boldsymbol{\omega}_i^{\mathcal{L}} & \boldsymbol{p}_{C1,i}^{\mathcal{L}} & \boldsymbol{p}_{C2,i}^{\mathcal{L}} \end{bmatrix} \quad (1)$$

The decentralized policy receives the *Pose* of the load as well as the *Pose* and *Twist* of the ego UAV. Notably, the policy does not receive any information about the other UAVs in the team.

To enable generalization across environments, all observations are expressed in the load frame $\mathcal{L}$. This makes the policy invariant to absolute world position allowing it to operate in spaces of arbitrary size.

It is demonstrated in [15] that leveraging load dynamics as implicit communication effectively facilitates cooperative load transportation. Their method encodes the load's state via the positions and velocities of eight bounding-box vertices, where agents' actions alter these dynamics and influence others' strategies, enabling scalable multi-agent collaboration.

Similarly, we provide the student policy with the position of the ends of the cable on the load ($\boldsymbol{p}_{C1,i}^{\mathcal{L}}$) and on the UAV ($\boldsymbol{p}_{C2,i}^{\mathcal{L}}$) The cable attach points, represented in the load-fixed frame $\mathcal{L}$, convey both the spatial position where each UAV applies force and the direction of that force. The exact location of these points on the load defines the moment arm, which determines the torque generated around the load's center of mass. This information is crucial for accurately controlling the load's pose during cooperative manipulation. The positions of these points are computed from known

attachment positions in the UAV and load frames:

$$\boldsymbol{p}_{C1,i} = \boldsymbol{p}_L + \boldsymbol{R}_L \boldsymbol{\rho}_i^{\mathcal{L}}, \quad \boldsymbol{p}_{C2,i} = \boldsymbol{p}_i + \boldsymbol{R}_i \boldsymbol{r}_i^{\mathcal{B}_i}, \tag{2}$$

where $\boldsymbol{R}_L$, $\boldsymbol{R}_i$ is the rotation matrix of the load and the UAV, respectively. $\boldsymbol{\rho}_i^{\mathcal{L}}$ represents the vector from load origin to the i^{th} UAV's cable attach point on the load. Similarly, $\boldsymbol{r}_i^{\mathcal{B}_i}$ represents the vector from the the ego UAV's origin to the cable attach point.

Instead of using quaternions, we represent rotations using the 6D continuous formulation proposed by Zhou et al. [16]. This representation avoids gimbal lock and discontinuities inherent in Euler angles or quaternions and therefore improves differentiability and learning stability.

To reason under partial observability and infer the intentions from other agents, the policy uses a history of past observations. Specifically the observation matrix at time t is defined as

$$\boldsymbol{O}_i = [\boldsymbol{o}_{i,t}, \boldsymbol{o}_{i,t-dt}, \ldots, \boldsymbol{o}_{i,t-Ndt}] \tag{3}$$

where $\boldsymbol{o}_{i,t}$ is the observation vector of the i-th UAV at time t.

Typically, time parameterized representations of future reference states help the agent develop a deeper understanding of the scenario, leading to improved performance [17]. Therefore, we provide the policy with the reference trajectory for the next 2 seconds at $N = 21$ nodes, which are the same references for the teacher policy, thereby ensuring that both the teacher and student get the same reference data as input during training.

$$\boldsymbol{Y} = [\boldsymbol{y}_0 \quad \boldsymbol{y}_1 \ldots \boldsymbol{y}_N] \tag{4}$$

where

$$\boldsymbol{y}_j = \begin{bmatrix} \boldsymbol{p}_L^{\mathcal{L}} & \boldsymbol{v}_L^{\mathcal{L}} & \boldsymbol{a}_L^{\mathcal{L}} & \boldsymbol{q}_L^{\mathcal{L}} & \boldsymbol{\omega}_L^{\mathcal{L}} \end{bmatrix}_j, \; j \in \{0, \ldots, N\}. \tag{5}$$

Here $\boldsymbol{a}_L$ represents the linear acceleration of the load.

To process sequential data of observation histories and reference trajectories, we use Temporal Convolutional Networks (TCN), which offer longer memory, stable gradients, and fast, parallel inference. Then the outputs of the TCN network are passed through a dense layer to create a vector of latent embeddings. The four streams of information as shown in Figure 2 are recorded at different time steps and therefore processed using separate TCN networks. The outputs of all TCNs are concatenated into a latent vector $\boldsymbol{x}_i$ that encodes the current state of the system as well as the desired future reference states:

$$\boldsymbol{x}_i = \pi_{\text{encoder}}(\boldsymbol{O}_i, \quad \boldsymbol{Y}) \tag{6}$$

2) Action Space: The student policy, as an online planner, directly generates reference trajectories for the ego UAV. In the training time, we use the output of the centralized teacher policy's online planner directly as the target for the student policy. Therefore, the action space for the i-th UAV can be represented by the following equations. Note that we omit the subscript i for readability.

$$\boldsymbol{U} = [\boldsymbol{u}_0 \quad \boldsymbol{u}_1 \ldots \boldsymbol{u}_N] \tag{7}$$

with

$$\boldsymbol{u}_j = \begin{bmatrix} \boldsymbol{p} & \boldsymbol{\omega} & \boldsymbol{v} & \boldsymbol{a} \end{bmatrix}_j, \quad j \in \{0, \ldots, N\} \tag{8}$$

where $\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{a}$ represent desired position, velocity and acceleration; $\boldsymbol{\omega}$ represents the desired body rates of the ego UAV. The onboard low-level controller typically employs a differential-flatness-based method to follow the reference trajectory generated by the planner for the UAV. Incorporating higher-order derivatives of the reference trajectory, such as velocity and acceleration, can significantly improve tracking performance [18]. A key challenge in predicting higher-order terms with a learning-based policy is ensuring kinematic consistency. The network may generate trajectories wherein the time derivative of position deviates from the predicted velocity, and likewise, the derivative of velocity from the predicted acceleration. To address this problem, we propose the use of Physics Informed Neural Networks (PINNs) with architectural constraints applied as described below:

$$[\boldsymbol{p}_j, \quad \boldsymbol{\omega}_j] = F_{mlp}(\boldsymbol{x}_i, t_j) \tag{9}$$

$$\boldsymbol{v}_j = \frac{\partial F_{mlp}(\boldsymbol{x}_i, t_j)}{\partial t} \tag{10}$$

$$\boldsymbol{a}_j = \frac{\partial^2 F_{mlp}(\boldsymbol{x}_i, t_j)}{\partial t^2} \tag{11}$$

where t is the instantaneous time and t_j is the timestamp of the j-th node in the UAV reference trajectory calculated using the equation:

$$t_j = t_{j-1} + 0.01 + 0.009 \cdot (j-1), \quad t_0 = t \tag{12}$$

We therefore employ PINNs as the decoder (Fig. 2) to generate trajectories from latent vectors that are inherently feasible, as they explicitly enforce kinematic constraints, i.e., that velocity is the derivative of position and acceleration is the derivative of velocity.

C. Training environment

The student policy is exclusively trained within the Gazebo (RotorS) [19] simulation environment, and the resulting policy is subsequently deployed in the real world without any modification. The training process employs privileged learning [12] in conjunction with the Dataset Aggregation (DAgger) algorithm [20]. DAgger is an iterative imitation learning algorithm wherein each iteration consists of two primary phases: data collection and policy training. During data collection, either the teacher or the student policy is used to interact with the environment to collect demonstrations. Afterwards the policy is trained to predict the teacher's action given the corresponding observations as input. In the initial iteration, only the teacher policy is used to determine actions. In subsequent iterations, the probability of using the student policy to control the system is gradually increased. This approach enables the generation of a more diverse dataset that encompasses a wider range of system states, thereby enhancing the robustness of the learned student policy [20].

To further improve the quality and diversity of the training dataset, we introduce randomized external disturbances

during the data collection phase. Specifically, at the start of an episode, a random force-torque vector is selected to define the disturbance direction. During each time step of the episode, a force with random magnitude is applied along this predefined direction, effectively simulating a consistent yet stochastic perturbation. This method promotes the exploration of a broader range of system dynamics, which contributes to the robustness and generalization capability of the resulting policy.

IV. Results

A. Environment for practical experiments

The proposed algorithm is validated through real-world experiments involving cooperative manipulation of a 1.4 kg rigid-body payload using three quadrotors, each with a mass of 0.6 kg. The payload is suspended by three cables, each 1 m in length, connected to distinct attachment points on the payload to facilitate full pose control. The opposing ends of the cables are affixed 0.03 m below the center of gravity (CoG) of each quadrotor. The aerial platforms are customized from the Agilicious open-source hardware framework [21], with onboard computation for low-level controllers handled by Raspberry Pi 5 mini PCs.

The student policy for each UAV is deployed as an independent ROS node on the aforementioned laptop. The predicted trajectories are transmitted to the Agilicious flight controller onboard each UAV via a ROS topic over Wi-Fi at 10 Hz. We use a motion capture system for high-precision state estimation of both the UAVs and the suspended payload.

B. Pose Trajectory Tracking

The student policy was trained on a figure-eight trajectory with dimensions 2.2 m $\times$ 2 m over 20 DAgger rounds, each consisting of a single episode. After each round, the probability of selecting actions from the student policy was linearly increased from 0 to 1 by round 20. During each round, the policy was trained on the aggregated dataset for 16 epochs. To enhance robustness against real-world perturbations, consistent yet stochastic disturbances, as detailed in Section III-C, were applied to the load during the final 5 episodes of training. The entire training process, including data collection and policy updates, was executed on the same laptop used for deployment and completed within 80 minutes. The trained policy was then evaluated on the identical figure-eight trajectory in the real world via zero-shot sim-to-real transfer. We perform two flights for both student and teacher policies on the same hardware and report the average of the metrics in Table I. We also show the trajectory tracking performance of both in Fig 3. In real-world experiments, the student policy, executed in a fully decentralized manner using only ego-UAV observations, achieves a comparable orientation RMSE to the teacher policy but exhibits a higher position RMSE.

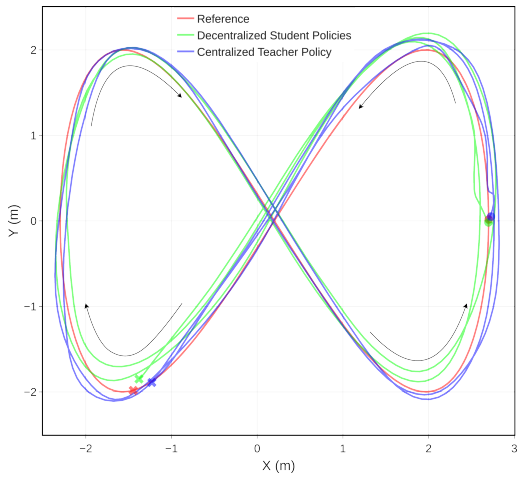


Fig. 3: Top view of the trajectory tracking performance. Circle indicates start of trajectory while cross signfies the end. Two laps of the figure 8 are performed.

Next, we train the student policy in a similar manner, but using two distinct trajectories. Training alternated between these two trajectories over 20 DAgger rounds, with perturbations applied during the final 4 rounds. The resulting policy was then evaluated in simulation on multiple unseen trajectories. The results are summarized in Table II.

Metric	Teacher Policy	Student Policies
RMSE Posn. (m)	0.171	0.377
RMSE Orient. (°)	10.447	9.335

TABLE I: Comparison of metrics from real-world flights following the figure-eight trajectory shown in Fig 3. The values are the means of the two flights.

Ref. Size		Train	Student Policies		Teacher Policy	
X(m)	Y(m)	Dataset	Posn. m	Orient. °	Posn. m	Orient. °
2.7	2.7	Yes	0.166	6.768	0.089	4.111
2.7	2.3	Yes	0.155	5.684	0.091	4.324
2.0	2.0	No	0.377	10.877	0.073	3.397
3.0	3.0	No	0.175	7.531	0.105	5.371
3.5	3.5	No	0.707	12.013	0.192	6.912
2.5	2.5	No	0.205	6.903	0.090	4.196

TABLE II: Comparison of the student and teacher policy performance in simulation for multiple trajectories. The table reports the RMSE for position and orientation. Student policies were trained using the first two trajectories. X and Y denote the amplitudes of the sine and cosine functions used to generate the trajectories.

C. Advantage of PINN over MLP as Decoder

A benefit of using PINNs as decoder is that the trajectories are guaranteed to be consistent and smooth. Consistency comes from the fact that in all target trajectories in training data start at ego UAV's pose and therefore the policies learn this behavior as well. Smoothness are guaranteed by the partial derivative constraints on velocity and acceleration. We compare the PINN architecture with a baseline policy where the *Decoder* network employs a single Multi-layer Perceptron (MLP) to generate all four trajectory components. The trajectories produced by this model are visibly jagged (shown in the supplementary video), lacking the smoothness enforced by the PINN's constraints. Moreover, they exhibit inconsistencies, where the integral of velocity does not match the predicted position as highlighted in Fig 4 & Table III

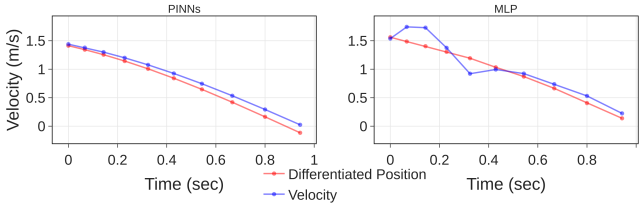


Fig. 4: Here the trajectory generated by the PINNs is compared with a traditional ML model. Trajectories produced by the PINN are smoother and consistent. The supplementary video shows the smoothness of an entire trajectory.

Method	X (m/s)	Y (m/s)	Z (m/s)	Overall (m/s)
PINN	0.010	0.030	0.004	0.015
MLP	0.080	0.116	0.048	0.081
Teacher	0.012	0.033	0.003	0.016

TABLE III: The Mean Absolute Error (MAE) between the numerical derivative of predicted position and the predicted velocity was evaluated for each method. For PINN and the teacher planner, the errors primarily arise from numerical differentiation. In contrast, the MLP exhibits higher errors due to both numerical differentiation and additional inaccuracies stemming from infeasible position and velocity predictions.

D. Computational Time

Real-time inference is possible on the Raspberry Pi 5 CPU and takes 97 milliseconds, as the planner generates trajectories at 10 Hz. However, during experiments, we realized that running both the Agilicious flight control stack and PINN inference on the Raspberry Pi leads to a degradation in the inner control loop update rate. Therefore, we eventually deployed PINNs on a laptop and sent them to each UAV through Wi-Fi. The average inference time of the student policy is 27 milliseconds, on a NVIDIA RTX 3060 Laptop GPU. In comparison, the teacher policy requires an average computation time of 15.3 milliseconds (for 3 UAVs) running on an Intel Core i7-13700H CPU according to [7], and its computational complexity increases exponentially with the number of UAVs. In contrast, the student policy, due to its decentralized and partially observable architecture, exhibits the potential for scalability. By enabling each UAV to independently predict its own trajectory, the framework could support horizontal scaling, making it feasible to accommodate a large number of UAVs without any increase in computation time.

V. Discussions

The proposed student policy demonstrates the capability to control the full pose of a load along agile reference trajectories without relying on networked communication. Trained entirely in simulation, it achieves zero-shot sim-to-real transfer thanks to the onboard-deployed trajectory tracking controller. Despite being a decentralized machine learning policy, it retains several key benefits of the teacher policy: *Sample Efficiency*, as the entire prediction horizon is leveraged during training; *System Dynamics*, since the learned trajectories are physically consistent by design due to architectural constraints; *Receding Horizon Planning*, which can increase the situational awareness of human operators and aid the development of safety modules; and *Lower Frequency*, as the trajectory-based structure requires execution at lower update rates, allowing for larger models and reduced computational load.

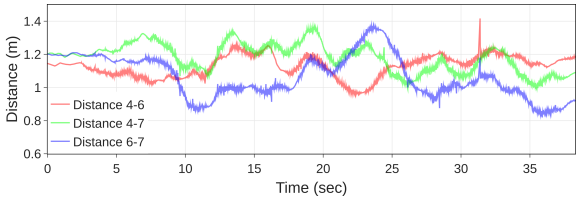


Fig. 5: Despite lacking awareness of each other's positions, the quadrotors successfully maintain safe inter-agent distances

One notable limitation of the student policies is the absence of hard guarantees for inter-agent collision avoidance. Unlike the teacher policy, which explicitly enforces minimum separation constraints, the decentralized policy employed in our method lacks direct awareness of other UAVs' positions. Consequently, the risk of inter-agent collisions is inherently higher for the decentralized student policy. While the student policy learns to approximate safe behavior, primarily by imitating the teacher and placing UAVs in safe configurations relative to the current load pose and reference trajectory, it does not explicitly account for neighboring agents. In real-world experiments, occasional breaches of the 1 m minimum separation threshold were observed; however, the UAVs typically recovered and reestablished safe distances, as illustrated by the trajectory data in Fig. 5. Enhancing the system with onboard sensing for inter-UAV perception and obstacle detection represents a promising avenue for future research, potentially enabling safer decentralized coordination in aerial manipulation scenarios.

The generalizability of the proposed method also remains for further exploration. Simulation results demonstrate that the proposed policy generalizes to variations in the size of the figure-eight trajectory, including turns that require load accelerations of up to 2 m/s^2. However, the student policy has only been tested on trajectories that are similar to the ones in the training dataset. A deeper analysis of this method's capability to track a broader class of trajectories remains an open challenge and is left for future work.

VI. Conclusion

In this work, we have designed and evaluated a fully decentralized real-time planning method for cooperative aerial manipulation of a cable-suspended load using multiple UAVs, without inter-UAV communications. Our method leverages imitation learning to train a student policy by imitating a centralized teacher policy with privileged information of the environment. We have experimentally validated the proposed planning method, showing its effectiveness in real-world flights.

ACKNOWLEDGEMENT

Acknowledgement. We would like to thank Maurits Pfaff, Kseniia Khomenko for the support in practical experiments.

REFERENCES

[1] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi, "Past, present, and future of aerial robotic manipulators," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 626–645, 2022. DOI: 10.1109/TRO.2021.3084395.

[2] T. Lee, K. Sreenath, and V. Kumar, "Geometric control of cooperating multiple quadrotor uavs with a suspended payload," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 5510–5515. DOI: 10.1109/CDC.2013.6760757.

[3] G. Tartaglione, E. D'Amato, M. Ariola, P. S. Rossi, and T. A. Johansen, "Model predictive control for a multi-body slung-load system," *Robotics and Autonomous Systems*, vol. 92, pp. 1–11, 2017, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2017.02.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889016305516.

[4] I. Moreno Caireta, "Planning and control of a multiple-quadcopter system cooperatively carrying a slung payload in dynamical environments. a centralized model predictive control solution," Ph.D. dissertation, UPC, Facultat d'Informàtica de Barcelona, 2018. [Online]. Available: http://hdl.handle.net/2117/121147.

[5] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[6] G. Li and G. Loianno, "Nonlinear model predictive control for cooperative transportation and manipulation of cable suspended payloads with multiple quadrotors," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2023, 5034–5041. DOI: 10.1109/iros55552.2023.10341785. [Online]. Available: http://dx.doi.org/10.1109/IROS55552.2023.10341785.

[7] S. Sun, X. Wang, D. Sanalitro, A. Franchi, M. Tognon, and J. Alonso-Mora, *Agile and cooperative aerial manipulation of a cable-suspended load*, 2025. arXiv: 2501.18802 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2501.18802.

[8] J. Wehbeh, S. Rahman, and I. Sharf, "Distributed model predictive control for uavs collaborative payload transport," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 11 666–11 672. DOI: 10.1109/IROS45743.2020.9341541.

[9] H. G. d. Marina and E. Smeur, "Flexible collaborative transportation by a team of rotorcraft," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 1074–1080. DOI: 10.1109/ICRA.2019.8794316.

[10] A. Tagliabue, M. Kamel, S. Verling, R. Siegwart, and J. Nieto, "Collaborative transportation using mavs via passive force control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5766–5773. DOI: 10.1109/ICRA.2017.7989678.

[11] M. Tognon, C. Gabellieri, L. Pallottino, and A. Franchi, "Aerial co-manipulation with cables: The role of internal force for equilibria, stability, and passivity," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2577–2583, 2018. DOI: 10.1109/LRA.2018.2803811.

[12] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," *CoRR*, vol. abs/1912.12294, 2019. arXiv: 1912.12294. [Online]. Available: http://arxiv.org/abs/1912.12294.

[13] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. [Online]. Available: https://www.marl-book.com.

[14] E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 3, pp. 450–461, 2016. DOI: 10.2514/1.G001490. [Online]. Available: https://doi.org/10.2514/1.G001490.

[15] J. Gao, Z. Wang, Z. Xiao, *et al.*, *Coohoi: Learning cooperative human-object interaction with manipulated object dynamics*, 2024. arXiv: 2406.14558 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2406.14558.

[16] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5738–5746. DOI: 10.1109/CVPR.2019.00589.

[17] J. Mayer, J. Westermann, J. P. G. H. Muriedas, U. Mettin, and A. Lampe, "Proximal policy optimization for tracking control exploiting future reference information," *CoRR*, vol. abs/2107.09647, 2021. arXiv: 2107.09647. [Online]. Available: https://arxiv.org/abs/2107.09647.

[18] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3357–3373, 2022.

[19] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Robot operating system (ros): The complete reference (volume 1)," in A. Koubaa, Ed. Cham: Springer International Publishing, 2016, ch. 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

[20] S. Ross, G. J. Gordon, and J. A. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning*, 2011. arXiv: 1011.0686 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1011.0686.

[21] P. Foehn, E. Kaufmann, A. Romero, *et al.*, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science Robotics*, vol. 7, no. 67, eabl6259, 2022. DOI: 10.1126/scirobotics.abl6259. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abl6259. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abl6259.