Neural-Swarm2: Planning and Control of Heterogeneous Multirotor Swarms using Learned Interactions

Guanya Shi, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung

***Abstract*—We present *Neural-Swarm2*, a learning-based method for motion planning and control that allows heterogeneous multirotors in a swarm to safely fly in close proximity. Such operation for drones is challenging due to complex aerodynamic interaction forces, such as downwash generated by nearby drones and ground effect. Conventional planning and control methods neglect capturing these interaction forces, resulting in sparse swarm configuration during flight. Our approach combines a physics-based nominal dynamics model with learned Deep Neural Networks (DNNs) with strong Lipschitz properties. We make use of two techniques to accurately predict the aerodynamic interactions between heterogeneous multirotors: i) spectral normalization for stability and generalization guarantees of unseen data and ii) heterogeneous deep sets for supporting any number of heterogeneous neighbors in a permutation-invariant manner without reducing expressiveness. The learned residual dynamics benefit both the proposed interaction-aware multi-robot motion planning and the nonlinear tracking control design because the learned interaction forces reduce the modelling errors. Experimental results demonstrate that *Neural-Swarm2* is able to generalize to larger swarms beyond training cases and significantly outperforms a baseline nonlinear tracking controller with up to three times reduction in worst-case tracking errors.**

***Index Terms*—Aerial systems, deep learning in robotics, multi-robot systems, multi-robot motion planning and control**

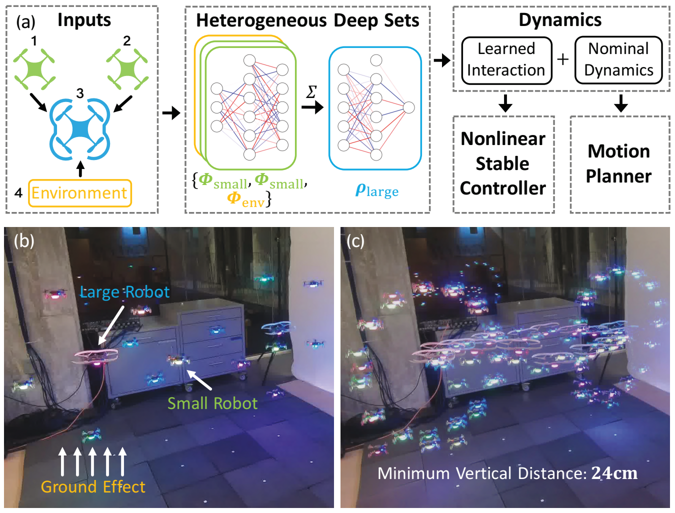


Fig. 1. We learn complex interaction between multirotors using heterogeneous deep sets and design an interaction-aware nonlinear stable controller and a multi-robot motion planner (a). Our approach enables close-proximity flight (minimum vertical distance 24 cm) of heterogeneous aerial teams (16 robots) with significant lower tracking error compared to solutions that do not consider the interaction forces (b,c).

I. Introduction

The ongoing commoditization of unmanned aerial vehicles (UAVs) requires robots to fly in much closer proximity to each other than before, which necessitates advanced planning and control methods for large aerial swarms [1, 2]. For example, consider a search-and-rescue mission where an aerial swarm must enter and search a collapsed building. In such scenarios, close-proximity flight enables the swarm to navigate the building much faster compared to swarms that must maintain large distances from each other. Other important applications of close-proximity flight include manipulation, search, surveillance, and mapping. In many scenarios, heterogeneous teams with robots of different sizes and sensing or manipulation capabilities are beneficial due to their significantly higher adaptability. For example, in a search-and-rescue mission, larger UAVs can be used for manipulation tasks or to transport goods, while smaller ones are more suited for exploration and navigation.

The authors are with California Institute of Technology, USA. {gshi, whoenig, xshi, yyue, sjchung}@caltech.edu.
The video is available at https://youtu.be/Y02juH6BDxo.
The work is funded in part by Caltech's Center for Autonomous Systems and Technologies (CAST) and the Raytheon Company.

A major challenge of close-proximity control and planning is that small distances between UAVs create complex aerodynamic interactions. For instance, one multirotor flying above another causes the so-called downwash effect on the lower one, which is difficult to model using conventional model-based approaches [3]. Without accurate downwash interaction modeling, a large safety distance between vehicles is necessary, thereby preventing a compact 3-D formation shape, e.g., 60 cm for the small Crazyflie 2.0 quadrotor (9 cm rotor-to-rotor) [4]. Moreover, the downwash is sometimes avoided by restricting the relative position between robots in the 2-D horizontal plane [5]. For heterogeneous teams, even larger and asymmetric safety distances are required [6]. However, the downwash for two small Crazyflie quadrotors hovering 30 cm on top of each other is only 9 g, which is well within their thrust capabilities, and suggests that proper modeling of downwash and other interaction effects can lead to more precise motion planning and dense formation control.

In this paper, we present a learning-based approach, *Neural-Swarm2*, which enhances the precision, safety, and density of close-proximity motion planning and control of heterogeneous multirotor swarms. In the example shown in Fig. 1, we safely operate the same drones with vertical distances less than half of those of prior work [4]. In particular, we train deep neural networks (DNNs) to predict the residual interaction forces that are not captured by the nominal models of free-space aerodynamics. To the best of our knowledge, this is the first model for aerodynamic interactions between two or more multirotors in flight. Our DNN architecture supports heterogeneous inputs in a permutation-invariant manner without reducing the expressiveness. The DNN only requires relative positions and velocities of neighboring multirotors as inputs, similar to the existing collision-avoidance techniques [7], which enables fully-decentralized computation. We use the predicted interaction forces to augment the nominal dynamics and derive novel methods to directly consider them during motion planning and as part of the multirotors' controller.

From a learning perspective, we leverage and extend two state-of-the-art tools to derive effective DNN models. First, we extend deep sets [8] to the heterogeneous case and prove its representation power. Our novel encoding is used to model interactions between heterogeneous vehicle types in an index-free or permutation-invariant manner, enabling better generalization to new formations and a varying number of vehicles. The second is spectral normalization [9], which ensures the DNN is Lipschitz continuous and helps the DNN generalize well on test examples that lie outside the training set. We demonstrate that the interaction forces can be computationally efficiently and accurately learned such that a small 32-bit microcontroller can predict such forces in real-time.

From a planning and control perspective, we derive novel methods that directly consider the predicted interaction forces. For motion planning, we use a two-stage approach. In the first stage, we extend an existing kinodynamic sampling-based planner for a single robot to the interaction-aware multi-robot case. In the second stage, we adopt an optimization-based planner to refine the solutions of the first stage. Empirically, we demonstrate that our interaction-aware motion planner both avoids dangerous robot configurations that would saturate the multirotors' motors and reduces the tracking error significantly. For the nonlinear control we leverage the Lipschitz continuity of our learned interaction forces to derive stability guarantees similar to our prior work [10, 11]. The controller can be used to reduce the tracking error of arbitrary desired trajectories, including ones that were not planned with an interaction-aware planner.

We validate our approach using two to sixteen quadrotors of two different sizes, and we also integrate ground effect and other unmodeled dynamics into our model, by viewing the physical environment as a special robot. To our knowledge, our approach is the first that models interactions between two or more multirotor vehicles and demonstrates how to use such a model effectively and efficiently for motion planning and control of aerial teams.

II. Related Work

The aerodynamic interaction force applied to a single UAV flying near the ground (ground effect), has been modeled analytically [12–14]. In many cases, the ground effect is not considered in typical multirotor controllers and thus increases the tracking error of a multirotor when operating close to the ground. However, it is possible to use ground effect prediction in real-time to reduce the tracking error [10, 14].

The interaction between two rotor blades of a single multirotor has been studied in a lab setting to optimize the placement of rotors on the vehicle [15]. However, it remains an open question how this influences the flight of two or more multirotors in close proximity. Interactions between two multirotors can be estimated using a propeller velocity field model [3]. Unfortunately, this method is hard to generalize to the multi-robot or heterogeneous case and it only considers the stationary case, which is inaccurate for real flights.

The use of DNNs to learn higher-order residual dynamics or control actions is gaining attention in the areas of control and reinforcement learning settings [10, 16–22]. For swarms, a common encoding approach is to discretize the whole space and employ convolutional neural networks (CNNs), which yields a permutation-invariant encoding. Another common encoding for robot swarms is a Graphic Neural Network (GNN) [23, 24]. GNNs have been extended to heterogeneous graphs [25], but it remains an open research question how such a structure would apply to heterogeneous robot teams. We extend a different architecture, which is less frequently used in robotics applications, called deep sets [8]. Deep sets enable distributed computation without communication requirements. Compared to CNNs, our approach: i) requires less training data and computation; ii) is not restricted to a pre-determined resolution and input domain; and iii) directly supports the heterogeneous swarm. Compared to GNNs, we do not require any direct communication between robots. Deep sets have been used in robotics for homogeneous [11] and heterogeneous [26] teams. Compared to the latter [26], our heterogeneous deep set extension has a more compact encoding and we prove its representation power.

For motion planning, empirical models have been used to avoid harmful interactions [2, 4, 6, 27, 28]. Typical safe boundaries along multi-vehicle motions form ellipsoids [4] or cylinders [6] along the motion trajectories. Estimating such shapes experimentally would potentially lead to many collisions and dangerous flight tests and those collision-free regions are in general conservative. In contrast, we use deep learning to estimate the interaction forces accurately in heterogeneous multi-robot teams. This model allows us to directly control the magnitude of the interaction forces to accurately and explicitly control the risk, removing the necessity of conservative collision shapes.

We generalize and extend the results of our prior work [11] as follows. i) We derive *heterogeneous deep sets* to extend to the heterogeneous case and prove its expressiveness, which also unifies the approach with respect to single-agent residual dynamics learning (e.g., learning the ground effect for improved multirotor landing [10]) by regarding the environment as a special neighbor. ii) We present a novel two-stage method to use the learned interaction forces for multi-robot motion planning. iii) We explicitly compensate for the delay in motor speed commands in our position and attitude controllers, resulting in stronger experimental results for both our baseline and *Neural-Swarm2*. iv) Leveraging i)-iii), this paper presents a result of close-proximity flight with minimum vertical distance 24 cm (the prior work [4] requires at least 60 cm as the safe distance) of a 16-robot heterogeneous team in challenging tasks (e.g., the 3-ring task in Fig. 1(b,c)). Our prior work [11] only demonstrated 5-robot flights in relatively simple tasks.

III. Problem Statement

Neural-Swarm2 can generally apply to any robotic system and we will focus on multirotor UAVs in this paper. We first present single multirotor dynamics including interaction forces modeled as disturbances. Then, we generalize these dynamics for a swarm of multirotors. Finally, we formulate our objective as a variant of an optimal control problem and introduce our performance metric.

A. Single Multirotor Dynamics

A single multirotor's state comprises of the global position $\mathbf{p} \in \mathbb{R}^3$, global velocity $\mathbf{v} \in \mathbb{R}^3$, attitude rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$, and body angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$. Its dynamics are:

$$\dot{\mathbf{p}} = \mathbf{v}, \qquad m\dot{\mathbf{v}} = m\mathbf{g} + \mathbf{R}\mathbf{f}_u + \mathbf{f}_a, \tag{1a}$$

$$\dot{\mathbf{R}} = \mathbf{R}\mathbf{S}(\boldsymbol{\omega}), \qquad \mathbf{J}\dot{\boldsymbol{\omega}} = \mathbf{J}\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}_u + \boldsymbol{\tau}_a, \tag{1b}$$

$$\boldsymbol{\eta} = \mathbf{B}_0\mathbf{u}, \qquad \dot{\mathbf{u}} = -\lambda\mathbf{u} + \lambda\mathbf{u}_c, \tag{1c}$$

where m and $\mathbf{J}$ denote the mass and inertia matrix of the system, respectively; $\mathbf{S}(\cdot)$ is a skew-symmetric mapping; $\mathbf{g} = [0; 0; -g]$ is the gravity vector; and $\mathbf{f}_u = [0; 0; T]$ and $\boldsymbol{\tau}_u = [\tau_x; \tau_y; \tau_z]$ denote the total thrust and body torques from the rotors, respectively. The output wrench $\boldsymbol{\eta} = [T; \tau_x; \tau_y; \tau_z]$ is linearly related to the control input $\boldsymbol{\eta} = \mathbf{B}_0\mathbf{u}$, where $\mathbf{u} = [n_1^2; n_2^2; \dots; n_M^2]$ is the squared motor speeds for a vehicle with M rotors and $\mathbf{B}_0$ is the actuation matrix. A multirotor is subject to additional disturbance force $\mathbf{f}_a = [f_{a,x}; f_{a,y}; f_{a,z}]$ and disturbance torque $\boldsymbol{\tau}_a = [\tau_{a,x}; \tau_{a,y}; \tau_{a,z}]$. We also consider a first order delay model in (1c), where $\mathbf{u}_c$ is the actual command signal we can directly control, and λ is the scalar time constant of the delay model.

Our model creates additional challenges compared to other exisiting multirotor dynamics models (e.g., [28]). The first challenge stems from the effect of delay in (1c). The second challenge stems from disturbance forces $\mathbf{f}_a$ in (1a) and disturbance torques $\boldsymbol{\tau}_a$ in (1b), generated by the interaction between other multirotors and the environment.

B. Heterogeneous Swarm Dynamics

We now consider N multirotor robots. We use $\mathbf{x}^{(i)} = [\mathbf{p}^{(i)}; \mathbf{v}^{(i)}; \mathbf{R}^{(i)}; \boldsymbol{\omega}^{(i)}]$ to denote the state of the i^{th} multirotor. We use $\mathbf{x}^{(ij)}$ to denote the *relative* state component between robot i and j, e.g., $\mathbf{x}^{(ij)} = [\mathbf{p}^{(j)} - \mathbf{p}^{(i)}; \mathbf{v}^{(j)} - \mathbf{v}^{(i)}; \mathbf{R}^{(i)}\mathbf{R}^{(j)\top}]$.

We use $\mathcal{I}(i)$ to denote the type of the i^{th} robot, where robots with identical physical parameters such as m, $\mathbf{J}$, and $\mathbf{B}_0$ are considered to be of the same type. We assume there are $K \le N$ types of robots, i.e., $\mathcal{I}(\cdot)$ is a surjective mapping from $\{1, \cdots, N\}$ to $\{\text{type}_1, \cdots, \text{type}_K\}$. Let $\mathbf{r}^{(i)}_{\text{type}_k}$ be the set of the relative states of the type_k neighbors of robot i:

$$\mathbf{r}^{(i)}_{\text{type}_k} = \{\mathbf{x}^{(ij)} \mid j \in \text{neighbor}(i) \text{ and } \mathcal{I}(j) = \text{type}_k\}. \tag{2}$$

The neighboring function $\text{neighbor}(i)$ is defined by an interaction volume function $\mathcal{V}$. Formally, $j \in \text{neighbor}(i)$ if $\mathbf{p}^{(j)} \in \mathcal{V}(\mathbf{p}^{(i)}, \mathcal{I}(i), \mathcal{I}(j))$, i.e., robot j is a neighbor of i if the position of j is within the interaction volume of i. In this paper, we design $\mathcal{V}$ as a cuboid based on observed interactions in experiments. The ordered sequence of all relative states grouped by robot type is

$$\mathbf{r}^{(i)}_{\mathcal{I}} = \left(\mathbf{r}^{(i)}_{\text{type}_1}, \mathbf{r}^{(i)}_{\text{type}_2}, \cdots, \mathbf{r}^{(i)}_{\text{type}_K}\right). \tag{3}$$

The dynamics of the i^{th} multirotor can be written in compact form:

$$\dot{\mathbf{x}}^{(i)} = \boldsymbol{\Phi}^{(i)}(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}) + \begin{bmatrix} \mathbf{0} \\ \mathbf{f}_a^{(i)}(\mathbf{r}^{(i)}_{\mathcal{I}}) \\ \mathbf{0} \\ \boldsymbol{\tau}_a^{(i)}(\mathbf{r}^{(i)}_{\mathcal{I}}) \end{bmatrix}, \tag{4}$$

where $\boldsymbol{\Phi}^{(i)}(\mathbf{x}^{(i)}, \mathbf{u}^{(i)})$ denotes the nominal dynamics of robot i, and $\mathbf{f}_a^{(i)}(\cdot)$ and $\boldsymbol{\tau}_a^{(i)}(\cdot)$ are the unmodeled force and torque of the i^{th} robot that are caused by interactions with neighboring robots or the environment (e.g., ground effect and air drag).

Robots with the same type have the same nominal dynamics and unmodeled force and torque:

$$\boldsymbol{\Phi}^{(i)}(\cdot) = \boldsymbol{\Phi}^{\mathcal{I}(i)}(\cdot), \mathbf{f}_a^{(i)}(\cdot) = \mathbf{f}_a^{\mathcal{I}(i)}(\cdot), \boldsymbol{\tau}_a^{(i)}(\cdot) = \boldsymbol{\tau}_a^{\mathcal{I}(i)}(\cdot)\ \forall i. \tag{5}$$

Note that the homogeneous case covered in our prior work [11] is a special case where $K = 1$, i.e., $\boldsymbol{\Phi}^{(i)}(\cdot) = \boldsymbol{\Phi}(\cdot)$, $\mathbf{f}_a^{(i)}(\cdot) = \mathbf{f}_a(\cdot)$, and $\boldsymbol{\tau}_a^{(i)}(\cdot) = \boldsymbol{\tau}_a(\cdot)\ \forall i$.

Our system is heterogeneous in three ways: i) different robot types have heterogeneous nominal dynamics $\boldsymbol{\Phi}^{\mathcal{I}(i)}$; ii) different robot types have different unmodeled $\mathbf{f}_a^{\mathcal{I}(i)}$ and $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$; and iii) the neighbors of each robot belong to K different sets.

We highlight that our heterogeneous model not only captures different types of robot, but also different types of environmental interactions, e.g., ground effect [10] and air drag. This is achieved in a straightforward manner by viewing the physical environment as a special robot type. We illustrate this generalization in the following example.

Example 1 (small and large robots, and the environment)**.** *We consider a heterogeneous system as depicted in Fig. 1(a). Robot* 3 *(large robot) has three neighbors: robot 1 (small), robot 2 (small) and environment 4. For robot* 3*, we have*

$$\mathbf{f}_a^{(3)} = \mathbf{f}_a^{\text{large}}(\mathbf{r}^{(3)}_{\mathcal{I}}) = \mathbf{f}_a^{\text{large}}(\mathbf{r}^{(3)}_{\text{small}}, \mathbf{r}^{(3)}_{\text{large}}, \mathbf{r}^{(3)}_{\text{env}}),$$
$$\mathbf{r}^{(3)}_{\text{small}} = \{\mathbf{x}^{(31)}, \mathbf{x}^{(32)}\},\ \mathbf{r}^{(3)}_{\text{large}} = \emptyset,\ \mathbf{r}^{(3)}_{\text{env}} = \{\mathbf{x}^{(34)}\}$$

and a similar expression for $\boldsymbol{\tau}_a^{(3)}$.

C. Interaction-Aware Motion Planning & Control

Our goal is to move the heterogeneous team of robots from their start states to goal states, which can be framed as the following optimal control problem:

$$\min_{\mathbf{u}^{(i)},\mathbf{x}^{(i)},t_f} \sum_{i=1}^{N} \int_0^{t_f} \|\mathbf{u}^{(i)}(t)\| dt \tag{6}$$
$$\text{s.t.} \begin{cases} \text{robot dynamics (4)} & i \in [1,N] \\ \mathbf{u}^{(i)}(t) \in \mathcal{U}^{\mathcal{I}(i)};\ \mathbf{x}^{(i)}(t) \in \mathcal{X}^{\mathcal{I}(i)} & i \in [1,N] \\ \|\mathbf{p}^{(ij)}\| \geq r^{(\mathcal{I}(i)\mathcal{I}(j))} & i<j,\ j \in [2,N] \\ \|\mathbf{f}_a^{(i)}\| \leq f_{a,\max}^{\mathcal{I}(i)};\ \|\boldsymbol{\tau}_a^{(i)}\| \leq \tau_{a,\max}^{\mathcal{I}(i)} & i \in [1,N] \\ \mathbf{x}^{(i)}(0) = \mathbf{x}_s^{(i)};\ \mathbf{x}^{(i)}(t_f) = \mathbf{x}_f^{(i)} & i \in [1,N] \end{cases}$$

where $\mathcal{U}^{(k)}$ is the control space for type_k robots, $\mathcal{X}^{(k)}$ is the free space for type_k robots, $r^{(lk)}$ is the minimum safety distance between type_l and type_k robots, $f_{a,\max}^{(k)}$ is the maximum endurable interaction force for type_k robots, $\tau_{a,\max}^{(k)}$ is the maximum endurable interaction torque for type_k robots, $\mathbf{x}_s^{(i)}$ is the start state of robot i, and $\mathbf{x}_f^{(i)}$ is the desired state of robot i. In contrast with the existing literature [6], we assume a tight spherical collision model and bound the interaction forces directly, eliminating the need of manually defining virtual collision shapes. For instance, larger $f_{a,\max}^{\mathcal{I}(i)}$ and $\tau_{a,\max}^{\mathcal{I}(i)}$ will yield denser and more aggressive trajectories. Also note that the time horizon t_f is a decision variable.

Solving (6) in real-time in a distributed fashion is intractable due to the exponential growth of the decision space with respect to the number of robots. Thus, we focus on solving two common subproblems instead. First, we approximately solve (6) offline as an interaction-aware motion planning problem. Second, we formulate an interaction-aware controller that minimizes the tracking error online. This controller can use both predefined trajectories and planned trajectories from the interaction-aware motion planner.

Since interaction between robots might only occur for a short time period with respect to the overall flight duration but can cause significant deviation from the nominal trajectory, we consider the worst tracking error of any robot in the team as a success metric:

$$\max_{i,t} \|\mathbf{p}^{(i)}(t) - \mathbf{p}_d^{(i)}(t)\|, \tag{7}$$

where $\mathbf{p}_d^{(i)}(t)$ is the desired trajectory for robot i. Note that this metric reflects the worst error out of *all* robots, because different robots in a team have various levels of task difficulty. For example, the two-drone swapping task in Fig. 8 is very challenging for the bottom drone due to the downwash effect, but relatively easier for the top drone. Minimizing (7) implies improved tracking performance and safety of a multirotor swarm during tight formation flight.

IV. Learning of Swarm Aerodynamic Interaction

We employ state-of-the-art deep learning methods to capture the unknown (or residual) dynamics caused by interactions of heterogeneous robot teams. In order to use the learned functions effectively for motion planning and control, we require that the DNNs have strong Lipschitz properties (for stability analysis), can generalize well to new test cases, and use compact encodings to achieve high computational and statistical efficiency. To that end, we introduce *heterogeneous deep sets*, a generalization of regular deep sets [8], and employ spectral normalization [9] for strong Lipschitz properties.

In this section, we will first review the homogeneous learning architecture covered in prior work [8, 11]. Then we will generalize them to the heterogeneous case with representation guarantees. Finally, we will introduce spectral normalization and our data collection procedures.

A. Homogeneous Permutation-Invariant Neural Networks

Recall that in the homogeneous case, all robots are with the same type (type_1). Therefore, the input to functions $\mathbf{f}_a$ or $\boldsymbol{\tau}_a$ is a single set. The permutation-invariant aspect of $\mathbf{f}_a$ or $\boldsymbol{\tau}_a$ can be characterized as:

$$\mathbf{f}_a(\mathbf{r}^{(i)}_{\text{type}_1}) = \mathbf{f}_a(\pi(\mathbf{r}^{(i)}_{\text{type}_1})),\ \boldsymbol{\tau}_a(\mathbf{r}^{(i)}_{\text{type}_1}) = \boldsymbol{\tau}_a(\pi(\mathbf{r}^{(i)}_{\text{type}_1}))$$

for any permutation π. Since the aim is to learn the function $\mathbf{f}_a$ and $\boldsymbol{\tau}_a$ using DNNs, we need to guarantee that the learned DNN is permutation-invariant. Therefore, we consider the following "deep sets" [8] architecture to approximate homogeneous $\mathbf{f}_a$ and $\boldsymbol{\tau}_a$:

$$\begin{bmatrix} \mathbf{f}_a(\mathbf{r}^{(i)}_{\text{type}_1}) \\ \boldsymbol{\tau}_a(\mathbf{r}^{(i)}_{\text{type}_1}) \end{bmatrix} \approx \boldsymbol{\rho}\left(\sum_{\mathbf{x}^{(ij)} \in \mathbf{r}^{(i)}_{\text{type}_1}} \boldsymbol{\phi}(\mathbf{x}^{(ij)}) \right) := \begin{bmatrix} \hat{\mathbf{f}}_a^{(i)} \\ \hat{\boldsymbol{\tau}}_a^{(i)} \end{bmatrix}, \tag{8}$$

where $\boldsymbol{\phi}(\cdot)$ and $\boldsymbol{\rho}(\cdot)$ are two DNNs. The output of $\boldsymbol{\phi}$ is a hidden state to represent "contributions" from each neighbor, and $\boldsymbol{\rho}$ is a nonlinear mapping from the summation of these hidden states to the total effect.

Obviously the network architecture in (8) is permutation-invariant due to the inner sum operation. We now show that this architecture is able to approximate any continuous permutation-invariant function. The following Lemma 1 and Theorem 2 are adopted from [8] and will be used and extended in the next section for the heterogeneous case.

Lemma 1. *Define $\bar{\boldsymbol{\phi}}(z) = [1; z; \cdots; z^M] \in \mathbb{R}^{M+1}$ as a mapping from $\mathbb{R}$ to $\mathbb{R}^{M+1}$, and $\mathcal{X} = \{[x_1; \cdots; x_M] \in [0,1]^M | x_1 \leq \cdots \leq x_M\}$ as a subset of $[0,1]^M$. For $\mathbf{x} = [x_1; \cdots; x_M] \in \mathcal{X}$, define $\mathbf{q}(\mathbf{x}) = \sum_{m=1}^{M} \bar{\boldsymbol{\phi}}(x_m)$. Then $\mathbf{q}(\mathbf{x}) : \mathcal{X} \to \mathbb{R}^{M+1}$ is a homeomorphism.*

Proof. The proof builds on the Newton-Girard formulae, which connect the moments of a sample set (sum-of-power) to the elementary symmetric polynomials (see [8]). □

Theorem 2. *Suppose $h(\mathbf{x}) : [0,1]^M \to \mathbb{R}$ is a permutation-invariant continuous function, i.e., $h(\mathbf{x}) = h(x_1, \cdots, x_M) = h(\pi(x_1, \cdots, x_M))$ for any permutation π. Then there exist continuous functions $\bar{\rho} : \mathbb{R}^{M+1} \to \mathbb{R}$ and $\bar{\boldsymbol{\phi}} : \mathbb{R} \to \mathbb{R}^{M+1}$ such that*

$$h(\mathbf{x}) = \bar{\rho}\left(\sum_{m=1}^{M} \bar{\boldsymbol{\phi}}(x_m) \right), \quad \forall \mathbf{x} \in [0,1]^M.$$

Proof. We choose $\bar{\phi}(z) = [1; z; \cdots; z^M]$ and $\bar{\rho}(\cdot) = h(\mathbf{q}^{-1}(\cdot))$, where $\mathbf{q}(\cdot)$ is defined in Lemma 1. Note that since $\mathbf{q}(\cdot)$ is a homeomorphism, $\mathbf{q}^{-1}(\cdot)$ exists and it is a continuous function from $\mathbb{R}^{M+1}$ to $\mathcal{X}$. Therefore, $\bar{\rho}$ is also a continuous function from $\mathbb{R}^{M+1}$ to $\mathbb{R}$, and $\bar{\rho}\left(\sum_{m=1}^{M} \bar{\phi}(x_m)\right) = \bar{\rho}(\mathbf{q}(\mathbf{x})) = h(\mathbf{q}^{-1}(\mathbf{q}(\mathbf{x}))) = h(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$. Finally, note that for any $\mathbf{x} \in [0,1]^M$, there exists some permutation π such that $\pi(\mathbf{x}) \in \mathcal{X}$. Then because both $\bar{\rho}(\mathbf{q}(\mathbf{x}))$ and $h(\mathbf{x})$ are permutation-invariant, we have $\bar{\rho}(\mathbf{q}(\mathbf{x})) = \bar{\rho}(\mathbf{q}(\pi(\mathbf{x}))) = h(\pi(\mathbf{x})) = h(\mathbf{x})$ for all $\mathbf{x} \in [0,1]^M$. □

Theorem 2 focuses on scalar valued permutation-invariant continuous functions with hypercubic input space $[0,1]^M$, i.e., each element in the input set is a scalar. In contrast, our learning target function $[\mathbf{f}_a; \boldsymbol{\tau}_a]$ in (8) is a vector valued function with a bounded input space, and each element in the input set is also a vector. However, Theorem 2 can be generalized in a straightforward manner by the following corollary.

Corollary 3. *Suppose* $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(M)}$ *are* M *bounded vectors in* $\mathbb{R}^{D_1}$*, and* $h(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)})$ *is a continuous permutation-invariant function from* $\mathbb{R}^{M\times D_1}$ *to* $\mathbb{R}^{D_2}$*, i.e.,* $h(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)}) = h(\mathbf{x}^{\pi(1)}, \cdots, \mathbf{x}^{\pi(M)})$ *for any permutation* π*. Then* $h(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)})$ *can be approximated arbitrarily close in the proposed architecture in (8).*

Proof. First, there exists a bijection from the bounded vector space in $\mathbb{R}^{D_1}$ to $[0,1]$ after discretization, with finite but arbitrary precision. Thus, Theorem 2 is applicable. Second, we apply Theorem 2 D_2 times and stack D_2 scalar-valued functions to represent the vector-valued function with output space $\mathbb{R}^{D_2}$. Finally, because DNNs are universal approximators for continuous functions [29], the proposed architecture in (8) can approximate any $h(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)})$ arbitrarily close. □

B. Heterogeneous K-Group Permutation-Invariant DNN

Different from the homogeneous setting, the inputs to functions $\mathbf{f}_a^{\mathcal{I}(i)}$ and $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$ in (5) are K different sets. First, we define *permutation-invariance* in the heterogeneous case. Intuitively, we expect that the following equality holds:

$$\mathbf{f}_a^{\mathcal{I}(i)}(\mathbf{r}^{(i)}_{\text{type}_1}, \cdots, \mathbf{r}^{(i)}_{\text{type}_K}) = \mathbf{f}_a^{\mathcal{I}(i)}(\pi_1(\mathbf{r}^{(i)}_{\text{type}_1}), \cdots, \pi_K(\mathbf{r}^{(i)}_{\text{type}_K}))$$

for any permutations $\pi_1, \cdots, \pi_K$ (similarly for $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$). Formally, we define K-group permutation invariance as follows.

Definition 1 (K-group permutation invariance)**.** *Let* $\mathbf{x}^{(k)} = [x_1^{(k)}; \cdots; x_{M_k}^{(k)}] \in [0,1]^{M_k}$ *for* $1 \le k \le K$*, and* $\mathbf{x} = [\mathbf{x}^{(1)}; \cdots; \mathbf{x}^{(K)}] \in [0,1]^{M_K}$*, where* $M_K = \sum_{k=1}^{K} M_k$*.* $h(\mathbf{x}) : \mathbb{R}^{M_K} \to \mathbb{R}$ *is* K*-group permutation-invariant if*

$$h([\mathbf{x}^{(1)}; \cdots; \mathbf{x}^{(K)}]) = h([\pi_1(\mathbf{x}^{(1)}); \cdots; \pi_K(\mathbf{x}^{(K)})])$$

for any permutations $\pi_1, \pi_2, \cdots, \pi_K$*.*

For example, $h(x_1, x_2, y_1, y_2) = \max\{x_1, x_2\} + 2 \cdot \max\{y_1, y_2\}$ is a 2-group permutation-invariant function, because we can swap x_1 and x_2 or swap y_1 and y_2, but if we interchange x_1 and y_1 the function value may vary. In addition, the $\mathbf{f}_a^{\text{large}}$ function in Example 1 is a 3-group permutation-invariant function.

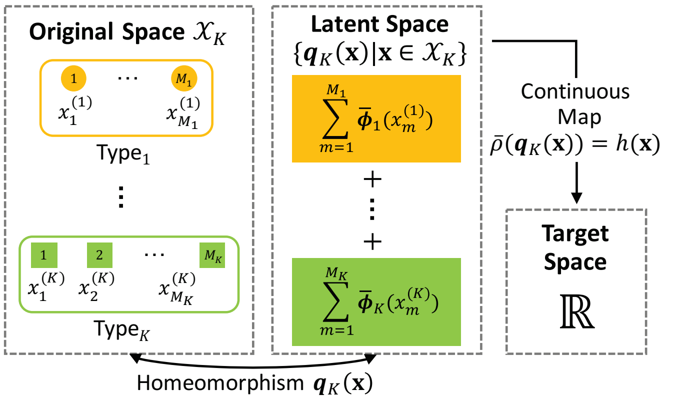

Original Space $\mathcal{X}_K$
1
M_1
$x_1^{(1)}$
$x_{M_1}^{(1)}$
Type$_1$
1
2
M_K
$x_1^{(K)}$
$x_2^{(K)}$
$x_{M_K}^{(K)}$
Type$_K$
Latent Space $\{\boldsymbol{q}_K(\mathbf{x})|\mathbf{x} \in \mathcal{X}_K\}$
$\sum_{m=1}^{M_1} \bar{\boldsymbol{\phi}}_1(x_m^{(1)})$
$\sum_{m=1}^{M_K} \bar{\boldsymbol{\phi}}_K(x_m^{(K)})$
Continuous Map
$\bar{\rho}(\boldsymbol{q}_K(\mathbf{x})) = h(\mathbf{x})$
Target Space
$\mathbb{R}$
Homeomorphism $\boldsymbol{q}_K(\mathbf{x})$

Fig. 2. Illustration of Theorem 4. We first find a homeomorphism $\mathbf{q}_K(\cdot)$ between the original space and the latent space, and then find a continuous function $\bar{\rho}(\cdot)$ such that $\bar{\rho}(\mathbf{q}_K(\cdot)) = h(\cdot)$.

Similar to Lemma 1, in order to handle ambiguity due to permutation, we define $\mathcal{X}_{M_k} = \{[x_1; \cdots; x_{M_k}] \in [0,1]^{M_k} | x_1 \le \cdots \le x_{M_k}\}$ and

$$\mathcal{X}_K = \{[\mathbf{x}^{(1)}; \cdots; \mathbf{x}^{(K)}] \in [0,1]^{M_K} | \mathbf{x}^{(k)} \in \mathcal{X}_{M_k}, \forall k\}.$$

Finally, we show how a K-group permutation-invariant function can be approximated via the following theorem.

Theorem 4. $h(\mathbf{x}) : [0,1]^{M_K} \to \mathbb{R}$ *is a* K*-group permutation-invariant continuous function if and only if it has the representation*

$$\begin{aligned} h(\mathbf{x}) &= \bar{\rho}\left(\sum_{m=1}^{M_1} \bar{\phi}_1(x_m^{(1)}) + \cdots + \sum_{m=1}^{M_K} \bar{\phi}_K(x_m^{(K)})\right) \\ &= \bar{\rho}\left(\sum_{k=1}^{K}\sum_{m=1}^{M_k} \bar{\phi}_k(x_m^{(k)})\right), \quad \forall \mathbf{x} \in [0,1]^{M_K} \end{aligned}$$

for some continuous outer and inner functions $\bar{\rho} : \mathbb{R}^{K+M_K} \to \mathbb{R}$ *and* $\bar{\phi}_k : \mathbb{R} \to \mathbb{R}^{K+M_K}$ *for* $1 \le k \le K$*.*

Proof. The sufficiency follows from that $h(\mathbf{x})$ is K-group permutation-invariant by construction. For the necessary condition, we need to find continuous functions $\bar{\rho}$ and $\{\bar{\phi}_k\}_{k=1}^K$ given h. We define $\bar{\phi}_k(x) : \mathbb{R} \to \mathbb{R}^{K+M_K}$ as

$$\bar{\phi}_k(x) = \left[\mathbf{0}_{M_1}; \cdots; \mathbf{0}_{M_{k-1}}; \begin{bmatrix} 1 \\ x \\ \vdots \\ x^{M_k} \end{bmatrix}; \mathbf{0}_{M_{k+1}}; \cdots; \mathbf{0}_{M_K}\right]$$

where $\mathbf{0}_{M_k} = [0; \cdots; 0] \in \mathbb{R}^{M_k+1}$. Then

$$\mathbf{q}_K(\mathbf{x}) = \sum_{k=1}^{K}\sum_{m=1}^{M_k} \bar{\phi}_k(x_m^{(k)})$$

is a homeomorphism from $\mathcal{X}_K \subseteq \mathbb{R}^{M_K}$ to $\mathbb{R}^{K+M_K}$ from Lemma 1. We choose $\bar{\rho} : \mathbb{R}^{K+M_K} \to \mathbb{R}$ as $\bar{\rho}(\cdot) = h(\mathbf{q}_K^{-1}(\cdot))$ which is continuous, because both $\mathbf{q}_K^{-1}$ and h are continuous. Then $\bar{\rho}(\mathbf{q}_K(\mathbf{x})) = h(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}_K$. Finally, because i) for all $\mathbf{x} = [\mathbf{x}^{(1)}; \cdots; \mathbf{x}^{(K)}]$ in $[0,1]^{M_K}$ there exist permutations $\pi_1, \cdots, \pi_K$ such that $[\pi_1(\mathbf{x}^{(1)}); \cdots; \pi_K(\mathbf{x}^{(K)})] \in \mathcal{X}_K$; and ii) both $\bar{\rho}(\mathbf{q}_K(\mathbf{x}))$ and $h(\mathbf{x})$ are K-group permutation-invariant, we have $\bar{\rho}(\mathbf{q}_K(\mathbf{x})) = h(\mathbf{x})$ for $\mathbf{x} \in [0,1]^{M_K}$. □

Figure 2 depicts the key idea of Theorem 4. Moreover, we provide a 2-group permutation-invariant function example to highlight the roles of ϕ and ρ in the heterogeneous case.

Example 2 (2-group permutation-invariant function)**.** *Consider* $h(x_1, x_2, y_1, y_2) = \max\{x_1, x_2\} + 2 \cdot \max\{y_1, y_2\}$, *which is* 2*-group permutation-invariant. Then we define* $\boldsymbol{\phi}_x(x) = [e^{\alpha x}; xe^{\alpha x}; 0; 0]$, $\boldsymbol{\phi}_y(y) = [0; 0; e^{\alpha y}; ye^{\alpha y}]$ *and* $\rho([a; b; c; d]) = b/a + 2 \cdot d/c$. *Note that*

$$\begin{aligned}&\rho(\boldsymbol{\phi}_x(x_1) + \boldsymbol{\phi}_x(x_2) + \boldsymbol{\phi}_y(y_1) + \boldsymbol{\phi}_y(y_2)) \\ =&\frac{x_1 e^{\alpha x_1} + x_2 e^{\alpha x_2}}{e^{\alpha x_1} + e^{\alpha x_2}} + 2 \cdot \frac{y_1 e^{\alpha y_1} + y_2 e^{\alpha y_2}}{e^{\alpha y_1} + e^{\alpha y_2}},\end{aligned}$$

which is asymptotically equal to $\max\{x_1, x_2\} + 2 \cdot \max\{y_1, y_2\}$ *as* $\alpha \to +\infty$.

Similar to the homogeneous case, Theorem 4 can generalize to vector-output functions with a bounded input space by applying the same argument as in Corollary 3. We propose the following *heterogeneous deep set* structure to model the heterogeneous functions $\mathbf{f}_a^{\mathcal{I}(i)}$ and $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$:

$$\begin{aligned}&\begin{bmatrix}\mathbf{f}_a^{\mathcal{I}(i)}(\mathbf{r}^{(i)}_{\text{type}_1}, \cdots, \mathbf{r}^{(i)}_{\text{type}_K}) \\ \boldsymbol{\tau}_a^{\mathcal{I}(i)}(\mathbf{r}^{(i)}_{\text{type}_1}, \cdots, \mathbf{r}^{(i)}_{\text{type}_K})\end{bmatrix} \\ \approx&\boldsymbol{\rho}_{\mathcal{I}(i)}\left(\sum_{k=1}^{K} \sum_{\mathbf{x}^{(ij)} \in \mathbf{r}^{(i)}_{\text{type}_k}} \boldsymbol{\phi}_{\mathcal{I}(j)}(\mathbf{x}^{(ij)})\right) := \begin{bmatrix}\hat{\mathbf{f}}_a^{(i)} \\ \hat{\boldsymbol{\tau}}_a^{(i)}\end{bmatrix}.\end{aligned} \tag{9}$$

Example 3 (Use of 3-group permutation-invariant function for multirotors)**.** *For example, in the heterogeneous system provided by Example 1 (as depicted in Fig. 1(a)), we have*

$$\begin{aligned}&\begin{bmatrix}\mathbf{f}_a^{(3)} \\ \boldsymbol{\tau}_a^{(3)}\end{bmatrix} = \begin{bmatrix}\mathbf{f}_a^{\text{large}}(\mathbf{r}^{(3)}_{\text{small}}, \mathbf{r}^{(3)}_{\text{large}}, \mathbf{r}^{(3)}_{\text{env}}) \\ \boldsymbol{\tau}_a^{\text{large}}(\mathbf{r}^{(3)}_{\text{small}}, \mathbf{r}^{(3)}_{\text{large}}, \mathbf{r}^{(3)}_{\text{env}})\end{bmatrix} \\ \approx&\boldsymbol{\rho}_{\text{large}}\left(\boldsymbol{\phi}_{\text{small}}(\mathbf{x}^{(31)}) + \boldsymbol{\phi}_{\text{small}}(\mathbf{x}^{(32)}) + \boldsymbol{\phi}_{\text{env}}(\mathbf{x}^{(34)})\right),\end{aligned}$$

for the large robot 3, where $\boldsymbol{\phi}_{\text{small}}$ *captures the interaction with the small robot 1 and 2, and* $\boldsymbol{\phi}_{\text{env}}$ *captures the interaction with the environment 4, e.g., ground effect and air drag.*

The structure in (9) has many valuable properties:

- **Representation ability.** Since Theorem 4 is necessary and sufficient, we do not lose approximation power by using this constrained framework, i.e., any K-group permutation-invariant function can be learned by (9). We demonstrate strong empirical performance using relatively compact DNNs for $\boldsymbol{\rho}_{\mathcal{I}(i)}$ and $\boldsymbol{\phi}_{\mathcal{I}(j)}$.
- **Computational and sampling efficiency and scalability.** Since the input dimension of $\boldsymbol{\phi}_{\mathcal{I}(j)}$ is always the same as the single vehicle case, the feed-forward computational complexity of (9) grows linearly with the number of neighboring vehicles. Moreover, the number of neural networks ($\boldsymbol{\rho}_{\mathcal{I}(i)}$ and $\boldsymbol{\phi}_{\mathcal{I}(j)}$) we need is $2K$, which grows linearly with the number of robot types. In practice, we found that one hour flight data is sufficient to accurately learn interactions between two to five multirotors.
- **Generalization to varying swarm size.** Given learned $\boldsymbol{\phi}_{\mathcal{I}(j)}$ and $\boldsymbol{\rho}_{\mathcal{I}(i)}$ functions, (9) can be used to predict interactions for any swarm size. In other words, we can accurately model swarm sizes (slightly) larger than those used for training. In practice, we found that our model can give good predictions for five multirotor swarms, despite only being trained on one to three multirotor swarms. Theoretical analysis on this generalizability is an interesting future research direction.

C. Spectral Normalization for Robustness and Generalization

To improve a property of robustness and generalizability of DNNs, we use spectral normalization [9] for training optimization. Spectral normalization stabilizes a DNN training by constraining its Lipschitz constant. Spectrally-normalized DNNs have been shown to generalize well, which is an indication of stability in machine learning. Spectrally-normalized DNNs have also been shown to be robust, which can be used to provide control-theoretic stability guarantees [10, 30]. The bounded approximation error assumption (Assumption 1) in our control stability and robustness analysis (Sec. VI-B) also relies on spectral normalization of DNNs.

Mathematically, the Lipschitz constant $\|\mathbf{g}\|_{\text{Lip}}$ of a function $\mathbf{g}(\cdot)$ is defined as the smallest value such that:

$$\forall\, \mathbf{x}, \mathbf{x}': \ \|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}')\|_2 / \|\mathbf{x} - \mathbf{x}'\|_2 \le \|\mathbf{g}\|_{\text{Lip}}.$$

Let $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta})$ denote a ReLU DNN parameterized by the DNN weights $\boldsymbol{\theta} = \mathbf{W}_1, \cdots, \mathbf{W}_{L+1}$:

$$\mathbf{g}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{W}_{L+1}\sigma(\mathbf{W}_L\sigma(\cdots\sigma(\mathbf{W}_1\mathbf{x})\cdots)), \tag{10}$$

where the activation function $\sigma(\cdot) = \max(\cdot, 0)$ is called the element-wise ReLU function. In practice, we apply the spectral normalization to the weight matrices in each layer after each batch gradient descent as follows:

$$\mathbf{W}_i \leftarrow \mathbf{W}_i / \|\mathbf{W}_i\|_2 \cdot \gamma^{\frac{1}{L+1}}, i \in [1, L+1], \tag{11}$$

where $\|\mathbf{W}_i\|_2$ is the maximum singular value of $\mathbf{W}_i$ and γ is a hyperparameter. With (11), $\|\mathbf{g}\|_{\text{Lip}}$ will be upper bounded by γ. Since spectrally-normalized $\mathbf{g}$ is $\gamma-$Lipschitz continuous, it is robust to noise $\Delta\mathbf{x}$, i.e., $\|\mathbf{g}(\mathbf{x} + \Delta\mathbf{x}) - \mathbf{g}(\mathbf{x})\|_2$ is always bounded by $\gamma\|\Delta\mathbf{x}\|_2$. In this paper, we apply the spectral normalization on both the $\boldsymbol{\phi}_{\mathcal{I}(j)}(\cdot)$ and $\boldsymbol{\rho}_{\mathcal{I}(i)}(\cdot)$ DNNs in (9).

D. Curriculum Learning

Training DNNs in (9) to approximate $\mathbf{f}_a^{\mathcal{I}(i)}$ and $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$ requires collecting close formation flight data. However, the downwash effect causes the nominally controlled multirotors (without compensation for the interaction forces) to move apart from each other. Thus, we use a curriculum/cumulative learning approach: first, we collect data for two multirotors without a DNN and learn a model. Second, we repeat the data collection using our learned model as a feed-forward term in our controller, which allows closer-proximity flight of the two vehicles. Third, we repeat the procedure with increasing number of vehicles, using the current best model.

Note that our data collection and learning are independent of the controller used and independent of the $\mathbf{f}_a^{\mathcal{I}(i)}$ or $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$ compensation. In particular, if we actively compensate for the learned $\mathbf{f}_a^{\mathcal{I}(i)}$ or $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$, this will only affect $\boldsymbol{\eta}$ in (1a) and not the observed $\mathbf{f}_a^{\mathcal{I}(i)}$ or $\boldsymbol{\tau}_a^{\mathcal{I}(i)}$.

V. Interaction-Aware Multi-Robot Planning

We approximately solve (6) offline by using two simplifications: i) we plan sequentially for each robot, treating other robots as dynamic obstacles with known trajectories, and ii) we use double-integrator dynamics plus learned interactions. Both simplifications are common for multi-robot motion planning with applications to multirotors [2, 31]. Such a motion planning approach can be easily distributed and is complete for planning instances that fulfill the *well-formed infrastructure* property [32]. However, the interaction forces (9) complicate motion planning significantly, because the interactions are highly nonlinear and robot dynamics are not independent from each other anymore.

For example, consider a three robot team with two small and one large robot as in Fig. 1(a). Assume that we already have valid trajectories for the two small robots and now plan a motion for the large robot. The resulting trajectory might result in a significant downwash force for the small robots if the large robot flies directly above the small ones. This strong interaction might invalidate the previous trajectories of the small robots or even violate their interaction force limits $f_{a,\max}^{\text{small}}$ and $\tau_{a,\max}^{\text{small}}$. Furthermore, the interaction force is asymmetric and thus it is not sufficient to only consider the interaction force placed on the large robot. We solve this challenge by directly limiting the change of the interaction forces placed on all neighbors when we plan for a robot. This concept is similar to trust regions in sequential optimization [33].

The simplified state is $\mathbf{x}^{(i)} = [\mathbf{p}^{(i)}; \mathbf{v}^{(i)}; \hat{\mathbf{f}}_a^{(i)}]$ and the simplified dynamics (4) become:

$$\dot{\mathbf{x}}^{(i)} = \mathbf{f}^{(i)}(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}) = \begin{bmatrix} \mathbf{v}^{(i)} \\ \mathbf{u}^{(i)} + \hat{\mathbf{f}}_a^{(i)} \\ \dot{\hat{\mathbf{f}}}_a^{(i)} \end{bmatrix}. \tag{12}$$

These dynamics are still complex and nonlinear because of $\hat{\mathbf{f}}_a^{(i)}$, which is the learned interaction force represented by DNNs in (9). We include $\hat{\mathbf{f}}_a^{(i)}$ in our state space to simplify the enforcement of the bound on the interaction force in (6).

We propose a novel hybrid two-stage planning algorithm, see Algorithm 1, leveraging the existing approaches while still highlighting the importance of considering interactive forces/torques in the planning. The portions of the pseudocode in Algorithm 1 that significantly differ from the existing methods to our approach are highlighted. In Stage 1, we find initial feasible trajectories using a kinodynamic sampling-based motion planner. Note that any kinodynamic planner can be used for Stage 1. In Stage 2, we use sequential convex programming (SCP) [2, 33, 34] to refine the initial solution to reach the desired states exactly and to minimize our energy objective defined in (6). Intuitively, Stage 1 identifies the homeomorphism class of the solution trajectories and fixes t_f, while Stage 2 finds the optimal trajectories to the goal within that homeomorphism class. Both stages differ from similar methods in the literature [2], because they need to reason over the coupling of the robots caused by interaction forces $\hat{\mathbf{f}}_a^{(i)}$.

Algorithm 1: Interaction-aware motion planning

```
Input: $\mathbf{x}_0^{(i)}, \mathbf{x}_f^{(i)}, \Delta t$
Result: $\mathcal{X}_{\text{sol}}^{(i)} = \left(\mathbf{x}_0^{(i)}, \mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \dots, \mathbf{x}_{T^{(i)}}^{(i)}\right)$,
        $\mathcal{U}_{\text{sol}}^{(i)} = \left(\mathbf{u}_0^{(i)}, \mathbf{u}_1^{(i)}, \mathbf{u}_2^{(i)}, \dots, \mathbf{x}_{T^{(i)}-1}^{(i)}\right)$
▷ Stage 1: Find duration $t_f$ and initial trajectories that are close to the goal state
1  $c^{(i)} \leftarrow \infty, \mathcal{X}_{\text{sol}}^{(i)} \leftarrow (), \mathcal{U}_{\text{sol}}^{(i)} \leftarrow ()\ \forall i \in \{1, \dots, N\}$
2  repeat
3      foreach $i \in$ RandomShuffle($\{1, \dots, N\}$) do
4          $\mathcal{T} = (\{\mathbf{x}_0^{(i)}\}, \emptyset)$
5          repeat
6              $\mathbf{x}_{\text{rand}} \leftarrow$ UniformSample($\mathcal{X}^{\mathcal{I}(i)}$)
7              $\mathbf{x}_{\text{near}} \leftarrow$ FindClosest($\mathcal{T}, \mathbf{x}_{\text{rand}}$)
8              $\mathbf{u}_{\text{rand}} \leftarrow$ UniformSample($\mathcal{U}^{\mathcal{I}(i)}$)
9              $\mathbf{x}_{\text{new}}, c \leftarrow$ Propagate($\mathbf{x}_{\text{near}}, \mathbf{u}_{\text{rand}}, \Delta t$,
                   $\{\mathcal{X}_{\text{sol}}^{(j)} | j \neq i\}$)
10             if StateValid($\mathbf{x}_{\text{new}}, \{\mathcal{X}_{\text{sol}}^{(j)} | j \neq i\}$) and $c \leq c^{(i)}$
               then
11                 Add($\mathcal{T}, \mathbf{x}_{\text{near}} \rightarrow \mathbf{x}_{\text{new}}$)
12                 if $\|\mathbf{x}_{\text{new}} - \mathbf{x}_f^{(i)}\| \leq \varepsilon$ then
13                     $c^{(i)} \leftarrow c$
14                     $\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)} \leftarrow$ ExtractSolution($\mathcal{T}, \mathbf{x}_{\text{new}}$)
15                     break
16 until TerminationCondition1()
17 $\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)} \leftarrow$ PostProcess($\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)}$)
▷ Stage 2: Refine trajectories sequentially; Based on SCP
18 repeat
19     foreach $i \in$ RandomShuffle($\{1, \dots, N\}$) do
20         $\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)} \leftarrow$ SolveCP(Eq. (16), $\{\mathcal{X}_{\text{sol}}^{(i)} | \forall i\}, \{\mathcal{U}_{\text{sol}}^{(i)} | \forall i\}$)
21 until Converged()
```

A. Stage 1: Sampling-Based Planning using Interaction Forces

For Stage 1, any kinodynamic single-robot motion planner can be extended. For the coupled multi-robot setting in the present paper, we modify AO-RRT [35], which is is a meta-algorithm that uses the rapidly-exploring random tree (RRT) algorithm as a subroutine.

Sampling-Based Planner: Our adaption of RRT (Lines 3 to 15 in Algorithm 1) works as follows. First, a random state $\mathbf{x}_{\text{rand}}$ is uniformly sampled from the state space (Line 6) and the closest state $\mathbf{x}_{\text{near}}$ that is already in the search tree $\mathcal{T}$ is found (Line 7). This search can be done efficiently in logarithmic time by employing a specialized data structured such as a kd-tree [36] and requires the user to define a distance function on the state space. Second, an action is uniformly sampled from the action space (Line 8) and the dynamics (4) are forward propagated for a fixed time period Δt using $\mathbf{x}_{\text{near}}$ as the initial condition, e.g., by using the Runge-Kutta method (Line 9). Note that this forward propagation directly considers the learned dynamics $\hat{\mathbf{f}}_a^{(i)}$. Third, the new state $\mathbf{x}_{\text{new}}$ is checked for validity with respect to i) the state space (which includes

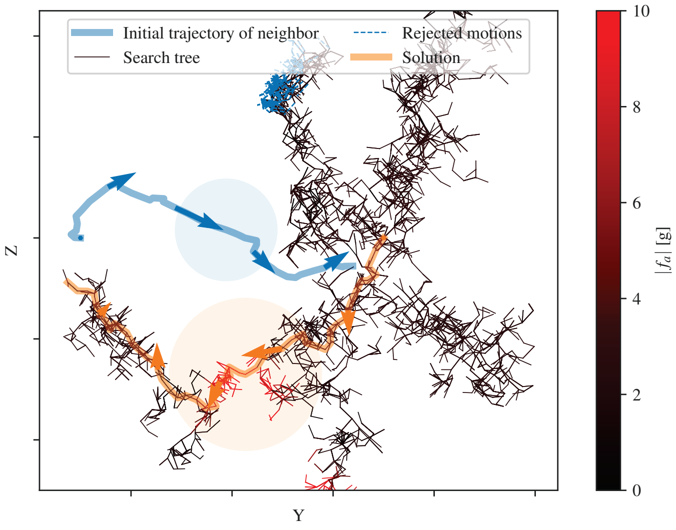


Fig. 3. Example for Stage 1 of our motion planning with learned dynamics. Here, we have an initial solution for a small (blue) robot and plan for a large (orange) robot. The created search tree of the large robot is color-coded by the magnitude of the interaction force on the orange robot. During the search, we reject states that would cause a significant change in the interaction force for the blue robot (edges in blue).

$\hat{\mathbf{f}}_a^{(i)}$), ii) collisions with other robots, and iii) change and bound of the neighbor's interaction forces (Line 10). The first validity check ensures that the interaction force of the robot itself is bounded, while the third check is a trust region and upper bound for the neighbor's interaction forces.

If $\mathbf{x}_{\mathrm{new}}$ is valid, it is added as a child node of $\mathbf{x}_{\mathrm{near}}$ in the search tree $\mathcal{T}$ (Line 11). Finally, if $\mathbf{x}_{\mathrm{new}}$ is within an ε-distance to the goal $\mathbf{x}_f^{(i)}$, the solution can be extracted by following the parent pointers of each tree node starting from $\mathbf{x}_{\mathrm{new}}$ until the root node $\mathbf{x}_0^{(i)}$ is reached (Line 15).

We note that our RRT steering method departs from ones in the literature which either sample Δt, use a best-control approximation of the steer method in RRT, or use a combination of both Δt-sampling and best-control approximation [35]. We are interested in a constant Δt for our optimization formulation in Stage 2. In this case, a best-control approximation would lead to a probabilistic incomplete planner [37]. We adopt a technique of goal biasing where we pick the goal state rather than $\mathbf{x}_{\mathrm{rand}}$ in fixed intervals, in order to improve the convergence speed.

While RRT is probabilistically complete, it also almost surely converges to a suboptimal solution [38]. AO-RRT remedies this shortcoming by planning in a state-cost space and using RRTs sequentially with a monotonically decreasing cost bound. The cost bound $c^{(i)}$ is initially set to infinity (Line 1) and the tree can only grow with states that have a lower cost associated with them (Line 10). Once a solution is found, the cost bound is decreased accordingly (Line 13) and the search is repeated using the new cost bound (Line 2). This approach is asymptotically optimal, but in practice the algorithm is terminated based on some condition, e.g., a timeout or a fixed number of iterations without improvements (Line 16).

Modification of Sampling-Based Planner: We extend AO-RRT to a sequential interaction-aware multi-robot planner by adding $\hat{\mathbf{f}}_a^{(i)}$ and time to our state space and treating the other robots as dynamic obstacles. As cost, we use a discrete approximation of the objective in (6). For each AO-RRT outer-loop iteration with a fixed cost bound, we compute trajectories sequentially using a random permutation of the robots (Line 3). When we check the state for validity (Line 10), we also enforce that the new state is not in collision with the trajectories of the other robots and that their interaction forces are bounded and within a trust region compared to their previous value, see Fig. 3 for visualization. Here, the red edges show motions that cause large ($\approx 10\,$g) but admissible interaction forces on the orange robot, because the blue robot flies directly above it. The blue edges are candidate edges as computed in Line 9 and are not added to the search tree, because their motion would cause a violation of the interaction force trust region of the blue robot (condition in Line 10). Once the search tree contains a path to the goal region, a solution is returned (orange path).

The output of the sequential planner (Line 15) is a sequence of states $\mathcal{X}_{\mathrm{sol}}^{(i)}$ and actions $\mathcal{U}_{\mathrm{sol}}^{(i)}$, each to be applied for a duration of Δt. Note that the sequences might have different lengths for each robot. Implicitly, the sequences also defines t_f. Furthermore, the first element of each sequence is the robots' start state and the last element is within a ε-distance of the robots' goal state. We postprocess this sequence of states to make it an appropriate input for the optimization, e.g., for uniform length (Line 17). In practice, we found that repeating the last state and adding null actions, or (virtual) tracking of the computed trajectories using a controller are efficient and effective postprocessing techniques.

Other sampling-based methods can be used as foundation of the first stage as well, with similar changes in sequential planning, state-augmentation to include the interaction forces, state-validity checking, and postprocessing.

B. *Stage 2: Optimization-Based Motion Planning*

We employ sequential convex programming (SCP) for optimization. SCP is a local optimization method for nonconvex problems that leverages convex optimization. The key concept is to convexify the nonconvex portions of the optimization problem by linearizing around a prior solution. The resulting convex problem instance is solved and a new solution obtained. The procedure can be repeated until convergence criteria are met. Because of the local nature of this procedure, a good initial guess is crucial for high-dimensional and highly nonlinear system dynamics. In our case, we use the searched trajectories from Stage 1 in Sec. V-A as the initial guess.

We first adopt a simple zero-order hold temporal discretization of the dynamics (12) using Euler integration:

$$\mathbf{x}_{k+1}^{(i)} = \mathbf{x}_k^{(i)} + \dot{\mathbf{x}}_k^{(i)} \Delta t. \tag{13}$$

Second, we linearize $\dot{\mathbf{x}}_k^{(i)}$ around prior states $\bar{\mathbf{x}}_k^{(i)}$ and actions $\bar{\mathbf{u}}_k^{(i)}$:

$$\dot{\mathbf{x}}_k^{(i)} \approx \mathbf{A}_k(\mathbf{x}_k^{(i)} - \bar{\mathbf{x}}_k^{(i)}) + \mathbf{B}_k(\mathbf{u}_k^{(i)} - \bar{\mathbf{u}}_k^{(i)}) + \mathbf{f}^{(i)}(\bar{\mathbf{x}}_k^{(i)}, \bar{\mathbf{u}}_k^{(i)}), \tag{14}$$

where $\mathbf{A}_k$ and $\mathbf{B}_k$ are the partial derivative matrices of $\mathbf{f}^{(i)}$ with respect to $\mathbf{x}_k^{(i)}$ and $\mathbf{u}_k^{(i)}$ evaluated at $\bar{\mathbf{x}}_k^{(i)}, \bar{\mathbf{u}}_k^{(i)}$. Because we encode $\hat{\mathbf{f}}_a^{(i)}$ using fully-differentiable DNNs, the partial derivatives can be efficiently computed analytically, e.g., by using `autograd` in PyTorch [39].

Third, we linearize $\hat{\mathbf{f}}_a^{(j)}$ around our prior states $\bar{\mathbf{x}}_k^{(i)}$ for all neighboring robots $j \in \text{neighbor}(i)$:

$$\hat{\mathbf{f}}_a^{(j)} \approx \mathbf{C}_k^{(j)}(\mathbf{x}_k^{(i)} - \bar{\mathbf{x}}_k^{(i)}) + \hat{\mathbf{f}}_a^{(j)}(\mathbf{r}_{\mathcal{I}}^{(i)}(\bar{\mathbf{x}}_k^{(i)})), \tag{15}$$

where $\mathbf{C}_k^{(j)}$ is the derivative matrix of $\hat{\mathbf{f}}_a^{(j)}$ (the learned interaction function of robot j, represented by DNNs) with respect to $\mathbf{x}_k^{(i)}$ evaluated at $\bar{\mathbf{x}}_k^{(i)}$; and $\mathbf{r}_{\mathcal{I}}^{(i)}(\bar{\mathbf{x}}_k^{(i)})$ is the ordered sequence of relative states as defined in (3) but using the fixed prior state $\bar{\mathbf{x}}_k^{(i)}$ rather than decision variable $\mathbf{x}_k^{(i)}$ in (2).

We now formulate a convex program, one per robot:

$$\min_{\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)}} \sum_{t=0}^{T} \|\mathbf{u}_k^{(i)}\|^2 + \lambda_1 \|\mathbf{x}_T^{(i)} - \mathbf{x}_f^{(i)}\|_\infty + \lambda_2 \delta \tag{16}$$

subject to:

$$\begin{cases} \text{robot dynamics (13) and (14)} & i \in [1, N] \\ \mathbf{u}_k^{(i)} \in \mathcal{U}^{\mathcal{I}(i)} & i \in [1, N] \\ \mathbf{x}_k^{(i)} \in \mathcal{X}_\delta^{\mathcal{I}(i)} & i \in [1, N], \delta \geq 0 \\ \langle \bar{\mathbf{p}}_k^{(ij)}, \mathbf{p}_k^{(i)} - \bar{\mathbf{p}}_k^{(i)} \rangle \geq r^{(\mathcal{I}(i)\mathcal{I}(j))} \|\bar{\mathbf{p}}_k^{(ij)}\|_2 & i < j, j \in [2, N] \\ \mathbf{x}_0^{(i)} = \mathbf{x}_s^{(i)} & i \in [1, N] \\ |\mathbf{C}_k^{(j)}(\mathbf{x}_k^{(i)} - \bar{\mathbf{x}}_k^{(i)})| \leq b_{fa} & i < j, j \in [2, N] \\ |\mathbf{x}_k^{(i)} - \bar{\mathbf{x}}_k^{(i)}| \leq \mathbf{b}_x;\ |\mathbf{u}_k^{(i)} - \bar{\mathbf{u}}_k^{(i)}| \leq \mathbf{b}_u & i \in [1, N] \end{cases}$$

where $\mathcal{X}_\delta^{\mathcal{I}(i)}$ is the state space increased by δ in each direction, the linearized robot dynamics are similar to [33, 40], and the convexified inter-robot collision constraint is from [2]. We use soft constraints for reaching the goal (with weight λ_1) and the state space (with weight λ_2), and trust regions around $\bar{\mathbf{x}}_k^{(i)}$, $\bar{\mathbf{u}}_k^{(i)}$, and the neighbors' interaction forces for numerical stability. Interaction forces are constrained in (16) because $\hat{\mathbf{f}}_a^{(i)}$ is part of the state space $\mathcal{X}^{\mathcal{I}(i)}$.

We solve these convex programs sequentially and they converge to a locally optimal solution [2]. For the first iteration, we linearize around the trajectory computed during Stage 1 of our motion planner while subsequent iterations linearize around the solution of the previous iteration (Lines 18 to 21 in Algorithm 1). It is possible to implement Algorithm 1 in a distributed fashion similar to prior work [2].

VI. Interaction-Aware Tracking Controller

Given arbitrary desired trajectories, including ones that have not been computed using the method presented in Sec. V, we augment existing nonlinear position and attitude controllers for multirotors [41, 42] that account for interaction forces and torques and compensate motor delays.

A. Tracking Control Law with Delay Compensation

We use a typical hierarchical structure as shown in Fig. 4 for controlling multirotor robots. Given the desired 3D position trajectory $\mathbf{p}_d^{(i)}(t)$ for robot i, we define a reference velocity

$$\mathbf{v}_r^{(i)} = \dot{\mathbf{p}}_d^{(i)} - \mathbf{\Lambda}_p^{(i)} \tilde{\mathbf{p}}^{(i)}, \tag{17}$$

with position error $\tilde{\mathbf{p}}^{(i)} = \mathbf{p}^{(i)} - \mathbf{p}_d^{(i)}$ and gain matrix $\mathbf{\Lambda}_p^{(i)} \succ 0$. The position controller is defined by the desired thrust vector

$$\begin{aligned} \mathbf{f}_d^{(i)} = &-m^{(i)}\mathbf{g} + m^{(i)}\dot{\mathbf{v}}_r^{(i)} - \hat{\mathbf{f}}_a^{(i)} \\ &- \left(\mathbf{K}_v^{(i)} + m\mathbf{\Gamma}_v^{(i)}\right)\tilde{\mathbf{v}}^{(i)} - \mathbf{K}_v^{(i)}\mathbf{\Gamma}_v^{(i)} \int \tilde{\mathbf{v}}^{(i)}, \end{aligned} \tag{18}$$

where $\tilde{\mathbf{v}}^{(i)} = \mathbf{v}^{(i)} - \mathbf{v}_r^{(i)}$ is the velocity error, and $\mathbf{K}_v^{(i)}$, $\mathbf{\Gamma}_v^{(i)}$ are positive definite gain matrices. From (1a), by setting $\mathbf{R}^{(i)}\mathbf{f}_u^{(i)} = \mathbf{f}_d^{(i)}$, we compute the total desired thrust $T_d^{(i)} = \mathbf{f}_d^{(i)} \cdot \hat{k}$ and the desired attitude $\mathbf{R}_d^{(i)}$ [10]. We convert the error rotation matrix $\tilde{\mathbf{R}}^{(i)} = \mathbf{R}_d^{(i)\top}\mathbf{R}^{(i)}$ to a constrained quaternion error $\tilde{\mathbf{q}}^{(i)} = [\tilde{q}_0^{(i)}, \tilde{\mathbf{q}}_v^{(i)}]$, and define the reference angular rate as

$$\boldsymbol{\omega}_r^{(i)} = \tilde{\mathbf{R}}^{(i)\top}\boldsymbol{\omega}_d^{(i)} - \mathbf{\Lambda}_q^{(i)}\tilde{\mathbf{q}}_v^{(i)}. \tag{19}$$

We use the following nonlinear attitude controller from [41] with interaction torque compensation:

$$\begin{aligned} \boldsymbol{\tau}_d^{(i)} = &\mathbf{J}^{(i)}\dot{\boldsymbol{\omega}}_r^{(i)} - \mathbf{J}^{(i)}\boldsymbol{\omega}^{(i)} \times \boldsymbol{\omega}_r^{(i)} - \hat{\boldsymbol{\tau}}_a^{(i)} \\ &- \mathbf{K}_\omega^{(i)}\tilde{\boldsymbol{\omega}}^{(i)} - \mathbf{\Gamma}_\omega^{(i)} \int \tilde{\boldsymbol{\omega}}^{(i)}. \end{aligned} \tag{20}$$

$\mathbf{K}_\omega^{(i)}$ and $\mathbf{\Gamma}_\omega^{(i)}$ are positive definite gain matrices on angular rate error $\tilde{\boldsymbol{\omega}}^{(i)} = \boldsymbol{\omega}^{(i)} - \boldsymbol{\omega}_r^{(i)}$ and its integral, respectively.

From (18) and (20), the desired output wrench for the i-th robot $\boldsymbol{\eta}_d^{(i)} = \left[T_d^{(i)}; \boldsymbol{\tau}_d^{(i)}\right]$ must be realized through a delayed motor signal $\mathbf{u}_c^{(i)}$ from (1c). Here, we implement a simple yet effective method to compensate for motor delay [43]:

$$\mathbf{u}_c^{(i)} = \mathbf{B}_0^{(i)+}\left(\boldsymbol{\eta}_d^{(i)} + \frac{\dot{\boldsymbol{\eta}}_d^{(i)}}{\lambda^{(i)}}\right), \tag{21}$$

where the actuation matrix $\mathbf{B}_0^{(i)}$ and delay constant $\lambda^{(i)}$ are determined a priori. We consider the multirotor to be fully or over-actuated, thus $(\cdot)^+$ denotes either the inverse or right pseudo-inverse. $\dot{\boldsymbol{\eta}}_d^{(i)}$ can be obtained through numerical differentiation [43].

B. Analysis of Stability and Robustness

The robust position and attitude controllers (18) and (20) can handle bounded model disturbance [10, 41]. Here, we make the same assumption as in [10, 11], that the learning errors and their rates of change are upper bounded.

Assumption 1 (Bounded approximation error of DNNs)**.** *We denote the approximation errors between the learned model in (9) and the true unmodeled dynamics for interaction force and torque as* $\boldsymbol{\epsilon}_f = \mathbf{f}_a^{(i)} - \hat{\mathbf{f}}_a^{(i)}$ *and* $\boldsymbol{\epsilon}_\tau = \boldsymbol{\tau}_a^{(i)} - \hat{\boldsymbol{\tau}}_a^{(i)}$*, respectively. For each robot, we assume* $\boldsymbol{\epsilon}_f$ *and* $\boldsymbol{\epsilon}_\tau$ *are uniformly upper bounded. Formally,* $\sup_{\mathbf{x}^{(i)} \in \mathcal{X}^{\mathcal{I}(i)}} \|\boldsymbol{\epsilon}_f\| = \bar{\epsilon}_f^{(i)}$ *and* $\sup_{\mathbf{x}^{(i)} \in \mathcal{X}^{\mathcal{I}(i)}} \|\boldsymbol{\epsilon}_\tau\| = \bar{\epsilon}_\tau^{(i)}$*. Furthermore, we assume the time derivative of errors are upper bounded as well, i.e* $\sup_{\mathbf{x}^{(i)} \in \mathcal{X}^{\mathcal{I}(i)}} \|\dot{\boldsymbol{\epsilon}}_f\| = \bar{d}_f^{(i)}$ *and* $\sup_{\mathbf{x}^{(i)} \in \mathcal{X}^{\mathcal{I}(i)}} \|\dot{\boldsymbol{\epsilon}}_\tau\| = \bar{d}_\tau^{(i)}$*.*

Note that when the number of agents are fixed, the $\bar{\epsilon}_f^{(i)}$, $\bar{\epsilon}_\tau^{(i)}$, $\bar{d}_f^{(i)}$, and $\bar{d}_\tau^{(i)}$ can all be derived from the Lipschitz constant

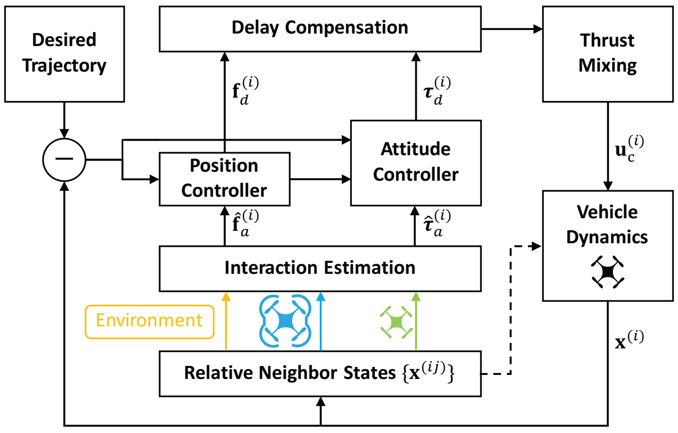


Fig. 4. Hierarchy of control and planning blocks with information flow for commands and sensor feedback. We use different colors to represent heterogeneous neighbors. Note that the neighbors will influence the vehicle dynamics (dashed arrow).

in (9) for spectrally-normalized DNNs [9, 30], under standard training data distribution assumptions. It is common to assume such bounded approximation errors in learning-based control, e.g., [10, 18–20]. Under Assumption 1, we can show the stability of the position and attitude controllers using results from [11, 41]:

Theorem 5. *For the position controller defined in (17) and (18) under Assumption 1, the position tracking error $\|\tilde{\mathbf{p}}^{(i)}\|$ converges exponentially to an error ball:*

$$\lim_{t\to\infty} \|\tilde{\mathbf{p}}^{(i)}\| = \frac{\bar{d}_f^{(i)}}{\lambda_{\min}(\boldsymbol{\Lambda}_p^{(i)})\lambda_{\min}(\boldsymbol{\Gamma}_v^{(i)})\lambda_{\min}(\mathbf{K}_v^{(i)})} \quad (22)$$

Proof. We select sliding variables: $\mathbf{s}_1 = \dot{\tilde{\mathbf{p}}}^{(i)} + \boldsymbol{\Lambda}_p^{(i)}\tilde{\mathbf{p}}^{(i)}$ and $\mathbf{s}_2 = m\dot{\mathbf{s}}_1 + \mathbf{K}_v\mathbf{s}_1$. Then, (18) can be written as

$$\mathbf{f}_d^{(i)} = -m^{(i)}\mathbf{g} + m^{(i)}\dot{\mathbf{v}}_r^{(i)} - \hat{\mathbf{f}}_a^{(i)} - \mathbf{K}_v^{(i)}\mathbf{s}_1 - \boldsymbol{\Gamma}_v^{(i)}\int \mathbf{s}_2.$$

Applying to (1a), we can get closed-loop dynamics $\dot{\mathbf{s}}_2 + \boldsymbol{\Gamma}_v^{(i)}\mathbf{s}_2 = \dot{\boldsymbol{\epsilon}}_f$. Thus combining hierarchical linear systems of $\dot{\mathbf{s}}_1$ and $\dot{\mathbf{s}}_2$, we can easily arrive at (22). □

Theorem 6. *For the attitude controller defined in (19) and (20) under Assumption 1, the attitude tracking error $\|\tilde{\mathbf{q}}_v^{(i)}\|$ converges exponentially to an error ball determined by $\bar{d}_\tau^{(i)}$.*

Proof. Applying (20) to (1b), we get closed-loop dynamics

$$\mathbf{J}^{(i)}\dot{\tilde{\boldsymbol{\omega}}}^{(i)} - \mathbf{J}^{(i)}\boldsymbol{\omega}^{(i)} \times \tilde{\boldsymbol{\omega}}^{(i)} - \mathbf{K}_\omega^{(i)}\tilde{\boldsymbol{\omega}}^{(i)} - \boldsymbol{\Gamma}_\omega^{(i)}\int \tilde{\boldsymbol{\omega}}^{(i)} = \boldsymbol{\epsilon}_\tau.$$

Following the proof structure for Theorem 2 in [41], we can derive an ultimate bound for $\|\tilde{\mathbf{q}}_v^{(i)}\|$ determined by $\bar{d}_\tau^{(i)}$. □

With motor delay, we can state the following result for stabilizing the output wrench error $\tilde{\boldsymbol{\eta}}^{(i)} = \boldsymbol{\eta}^{(i)} - \boldsymbol{\eta}_d^{(i)}$ with (21), assuming the motor delay constant is obtained from testing.

Theorem 7. *For robot i, the controllers (18), (20) and (21) will exponentially stabilize the augmented states of position, attitude and output wrench error: $[\tilde{\mathbf{p}}^{(i)}; \tilde{\mathbf{v}}^{(i)}; \tilde{\mathbf{q}}_v^{(i)}; \tilde{\boldsymbol{\omega}}^{(i)}; \tilde{\boldsymbol{\eta}}^{(i)}]$.*

Proof. (18) and (20) stabilizes $[\tilde{\mathbf{p}}^{(i)}; \tilde{\mathbf{v}}^{(i)}; \tilde{\mathbf{q}}_v^{(i)}; \tilde{\boldsymbol{\omega}}^{(i)}]$ exponentially from Theorems 5 and 6. Thus by Theorem 3.1 from [43], it follows that the augmented states is also exponentially stabilized by (21). □

With small modelling errors on $\lambda^{(i)}$, the controller (21) can robustly cancel out some effects from delays and improve tracking performance in practice. Furthermore, it can handle not only first-order motor delay (1c), but also signal transport delays [43]. In case of the small quadrotors used in our experiments, such delays are on the same order of magnitude as the motor delay, thus making (21) essential for improving the control performance.

VII. Experiments

We use quadrotors based on Bitcraze Crazyflie 2.0/2.1 (CF). Our small quadrotors are Crazyflie 2.X, which are small (9 cm rotor-to-rotor) and lightweight (34 g) products that are commercially available. Our large quadrotors use the Crazyflie 2.1 as control board on a larger frame with brushed motors (model: Parrot Mini Drone), see Table I for a summary of physical parameters. We use the Crazyswarm [44] package to control multiple Crazyflies simultaneously. Each quadrotor is equipped with a single reflective marker for position tracking at 100 Hz using a motion capture system. The nonlinear controller, extended Kalman filter, and neural network evaluation are running on-board the STM32 microcontroller.

For the controller, we implement the delay compensation (21) in the following way: i) we numerically estimate $\dot{T}_d^{(i)}$ as part of the position controller (18), and ii) we approximate $\dot{\boldsymbol{\tau}}_d^{(i)}$ by adding the additional term $-\mathbf{K}_{\dot{\omega}}^{(i)}\dot{\boldsymbol{\omega}}^{(i)}$ to the attitude controller (20), where $\dot{\boldsymbol{\omega}}^{(i)}$ is numerically estimated and $\mathbf{K}_{\dot{\omega}}^{(i)}$ is a positive definite gain matrix. We found that the other terms of $\dot{\boldsymbol{\tau}}_d^{(i)}$, i.e. the time-derivative of (20), are negligible for our use-case. The baseline controller is identical (including the chosen gains) to our proposed controller except that the interaction force for the baseline is set to zero. The baseline controller is much more robust and efficient than the well-tuned nonlinear controller in the Crazyswarm package, which cannot safely execute the close-proximity flight shown in Fig. 1(c) and requires at least 60 cm safety distance [4].

For data collection, we use the μSD card extension board and store binary encoded data roughly every 10 ms. Each dataset is timestamped using the on-board microsecond timer and the clocks are synchronized before takeoff using broadcast radio packets. The drift of the clocks of different Crazyflies can be ignored for our short flight times (less than 2 min).

A. Calibration and System Identification of Different Robots

Prior to learning the residual terms $\mathbf{f}_a^{(i)}$ and $\boldsymbol{\tau}_a^{(i)}$, we first calibrate the nominal dynamics model $\boldsymbol{\Phi}^{(i)}(\mathbf{x}, \mathbf{u})$. We found that existing motor thrust models [45, 46] are not very accurate, because they only consider a single motor and ignore the effect of the battery state of charge. We calibrate each Crazyflie by mounting the whole quadrotor upside-down on a load cell (model TAL221) which is directly connected

TABLE I
SYSTEM IDENTIFICATION OF THE USED QUADROTORS.

	Small	Large
Weight	34 g	67 g
Max Thrust	65 g	145 g
Diameter	12 cm	19 cm
λ	16	16
$\psi_1(\hat{p},\hat{v})$	$11.09-39.08\hat{p}-9.53\hat{v}+20.57\hat{p}^2+38.43\hat{p}\hat{v}$	$44.1.0-122.51\hat{p}-36.18\hat{v}+53.11\hat{p}^2+107.68\hat{p}\hat{v}$
$\psi_2(\hat{f},\hat{v})$	$0.5+0.12\hat{f}-0.41\hat{v}-0.002\hat{f}^2-0.043\hat{f}\hat{v}$	$0.56+0.06\hat{f}-0.6\hat{v}-0.0007\hat{f}^2-0.015\hat{f}\hat{v}$
$\psi_3(\hat{p},\hat{v})$	$-9.86+3.02\hat{p}-26.72\hat{v}$	$-29.91+8.1\hat{p}+65.2\hat{v}$

to the Crazyflie via a custom extension board using a 24-bit ADC (model HX711). The upside-down mounting avoids contamination of our measurements with downwash-related forces. We use a 100 g capacity load cell for the small quadrotor and a 500 g capacity load cell for the large quadrotor. We randomly generate desired PWM motor signals (identical for all 4 motors) and collect the current battery voltage, PWM signals, and measured force. We use this data to find three polynomial functions: ψ_1, ψ_2, and ψ_3. The first $\hat{f}=\psi_1(\hat{p},\hat{v})$ computes the force of a single rotor given the normalized PWM signal $\hat{p}$ and the normalized battery voltage $\hat{v}$. This function is only required for the data collection preparation in order to compute $\mathbf{f}_a^{(i)}$. The second $\hat{p}=\psi_2(\hat{f},\hat{v})$ computes the required PWM signal $\hat{p}$ given the desired force $\hat{f}$ and current battery voltage $\hat{v}$. Finally, $\hat{f}_{\max}=\psi_3(\hat{p},\hat{v})$ computes the maximum achievable force $\hat{f}_{\max}$, given the current PWM signal $\hat{p}$ and battery voltage $\hat{v}$. The last two functions are important at runtime for outputting the correct force as well as for thrust mixing when motors are saturated [47].

We use the same measurement setup with the load cell to establish the delay model of $T_d^{(i)}$ with a square wave PWM signal. While the delay model is slightly asymmetric in practice, we found that our symmetric model (1c) is a good approximation. All results are summarized in Table I. We use the remaining parameters ($\mathbf{J}$, thrust-to-torque ratio) from the existing literature [46].

B. Data Collection

Recall that in (9), we need to learn $2K$ neural networks for K types of robots. In our experiments, we consider two types of quadrotors (small and large) and also the environment (mainly ground effect and air drag), as shown in Example 1. Therefore, we have 5 neural networks to be learned:

$$\boldsymbol{\rho}_{\text{small}},\ \boldsymbol{\rho}_{\text{large}},\ \boldsymbol{\phi}_{\text{small}},\ \boldsymbol{\phi}_{\text{large}},\ \boldsymbol{\phi}_{\text{env}}, \tag{23}$$

where we do not have $\boldsymbol{\rho}_{\text{env}}$ because the aerodynamical force acting on the environment is not interesting for our purpose. To learn these 5 neural networks, we fly the heterogeneous swarm in 12 different scenarios (see Table II) to collect labeled $\mathbf{f}_a^{(i)}$ and $\boldsymbol{\tau}_a^{(i)}$ data for each robot. For instance, Example 1 (as depicted in Fig. 1(a)) corresponds to the "$\{\mathrm{S},\mathrm{S}\}\rightarrow\mathrm{L}$" scenario in Table II, where the large robot has two small robots and the environment as its neighbors.

We utilize two types of data collection tasks: random walk and swapping. For random walk, we implement a simple reactive collision avoidance approach based on artificial potentials on-board each Crazyflie [48]. The host computer randomly selects new goal points within a cube for each vehicle at a fixed frequency. These goal points are used as an attractive force, while neighboring drones contribute a repulsive force. For swapping, the drones are placed in different horizontal planes on a cylinder and tasked to move to the opposite side. All the drones are vertically aligned for one time instance, causing a large interaction force. The random walk data helps us to explore the whole space quickly, while the swapping data ensures that we have data for a specific task of interest. Note that for both random walk and swapping, the drones also move close to the ground, to collect sufficient data for learning the ground effect. The collected data covers drone flying speeds from 0 to 2 m/s, where 7% are with relatively high speeds ($\geq$0.5 m/s) to learn the aerodynamic drag. For both task types, we varied the scenarios listed in Table II.

To learn the 5 DNNs in (23), for each robot i in each scenario, we collect the timestamped states $\mathbf{x}^{(i)}=[\mathbf{p}^{(i)};\mathbf{v}^{(i)};\mathbf{R}^{(i)};\boldsymbol{\omega}^{(i)}]$. We then compute $\mathbf{y}^{(i)}$ as the observed value of $\mathbf{f}_a^{(i)}$ and $\boldsymbol{\tau}_a^{(i)}$. We compute $\mathbf{f}_a^{(i)}$ and $\boldsymbol{\tau}_a^{(i)}$ using (4), where the nominal dynamics $\boldsymbol{\Phi}^{(i)}$ is calculated based on our system identification in Sec. VII-A. With $\boldsymbol{\Phi}^{(i)}$, $\mathbf{y}^{(i)}$ is computed by $\dot{\mathbf{x}}^{(i)}-\boldsymbol{\Phi}^{(i)}$, where $\dot{\mathbf{x}}^{(i)}$ is estimated by the five-point numerical differentiation method. Note that the control delay $\lambda^{(i)}$ is also considered when we compute $\mathbf{f}_a^{(i)}$ and $\boldsymbol{\tau}_a^{(i)}$. Our training data consists of sequences of $\left(\{\mathbf{r}^{(i)}_{\text{type}_1},\cdots,\mathbf{r}^{(i)}_{\text{type}_K}\},\mathbf{y}^{(i)}\right)$ pairs, where $\mathbf{r}^{(i)}_{\text{type}_k}=\{\mathbf{x}^{(ij)}|j\in\text{neighbor}(i)\text{ and }\mathcal{I}(j)=\text{type}_k\}$ is the set of the relative states of the type-k neighbors of i. We have the following loss function for robot i in each scenario (see Table II for the detailed model structure in each scenario):

$$\left\|\boldsymbol{\rho}_{\mathcal{I}(i)}\Big(\sum_{k=1}^{K}\sum_{\mathbf{x}^{(ij)}\in\mathbf{r}^{(i)}_{\text{type}_k}}\boldsymbol{\phi}_{\mathcal{I}(j)}(\mathbf{x}^{(ij)})\Big)-\mathbf{y}^{(i)}\right\|_2^2, \tag{24}$$

and we stack all the robots' data in all scenarios and train on them together. There are 1.4 million pairs in the full dataset.

In practice, we found the unmodeled torque $\|\boldsymbol{\tau}_a^{(i)}\|$ is very small (smaller by two orders of magnitude than the feedback term $\mathbf{K}_\omega^{(i)}\tilde{\boldsymbol{\omega}}^{(i)}$ in the attitude controller (20)), so we only learn $\mathbf{f}_a^{(i)}$. We compute the relative states from our collected data as $\mathbf{x}^{(ij)}=[\mathbf{p}^{(j)}-\mathbf{p}^{(i)};\mathbf{v}^{(j)}-\mathbf{v}^{(i)}]\in\mathbb{R}^6$ (i.e., relative position and relative velocity both in the world frame), since the attitude information $\mathbf{R}$ and $\boldsymbol{\omega}$ are not dominant for $\mathbf{f}_a^{(i)}$. If the type of neighbor j is "environment", we set $\mathbf{p}^{(j)}=\mathbf{0}$ and $\mathbf{v}^{(j)}=\mathbf{0}$. In this work, we only learn the z-component of $\mathbf{f}_a^{(i)}$ since we found the other two components, x and y, are much smaller and less varied, and do not significantly alter the nominal dynamics. In data collection, the rooted means and standard deviations of the squared values of $f_{a,x}$, $f_{a,y}$, and

TABLE II
12 SCENARIOS FOR DATA COLLECTION.

Scenario	S	$\mathrm{S} \to \mathrm{S}$	$\mathrm{L} \to \mathrm{S}$	$\{\mathrm{S},\mathrm{S}\} \to \mathrm{S}$
Model	$\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}})$	$\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}} + \boldsymbol{\phi}_{\mathrm{small}})$	$\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}} + \boldsymbol{\phi}_{\mathrm{large}})$	$\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}}+\boldsymbol{\phi}_{\mathrm{small}}+\boldsymbol{\phi}_{\mathrm{small}})$
Scenario	$\{\mathrm{S},\mathrm{L}\} \to \mathrm{S}$	$\{\mathrm{L},\mathrm{L}\} \to \mathrm{S}$	L	$\mathrm{S} \to \mathrm{L}$
Model	$\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}}+\boldsymbol{\phi}_{\mathrm{small}}+\boldsymbol{\phi}_{\mathrm{large}})$	$\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}}+\boldsymbol{\phi}_{\mathrm{large}}+\boldsymbol{\phi}_{\mathrm{large}})$	$\boldsymbol{\rho}_{\mathrm{large}}(\boldsymbol{\phi}_{\mathrm{env}})$	$\boldsymbol{\rho}_{\mathrm{large}}(\boldsymbol{\phi}_{\mathrm{env}} + \boldsymbol{\phi}_{\mathrm{small}})$
Scenario	$\mathrm{L} \to \mathrm{L}$	$\{\mathrm{S},\mathrm{S}\} \to \mathrm{L}$	$\{\mathrm{S},\mathrm{L}\} \to \mathrm{L}$	$\{\mathrm{L},\mathrm{L}\} \to \mathrm{S}$
Model	$\boldsymbol{\rho}_{\mathrm{large}}(\boldsymbol{\phi}_{\mathrm{env}} + \boldsymbol{\phi}_{\mathrm{large}})$	$\boldsymbol{\rho}_{\mathrm{large}}(\boldsymbol{\phi}_{\mathrm{env}}+\boldsymbol{\phi}_{\mathrm{small}}+\boldsymbol{\phi}_{\mathrm{small}})$	$\boldsymbol{\rho}_{\mathrm{large}}(\boldsymbol{\phi}_{\mathrm{env}}+\boldsymbol{\phi}_{\mathrm{small}}+\boldsymbol{\phi}_{\mathrm{large}})$	$\boldsymbol{\rho}_{\mathrm{large}}(\boldsymbol{\phi}_{\mathrm{env}}+\boldsymbol{\phi}_{\mathrm{large}}+\boldsymbol{\phi}_{\mathrm{large}})$

$f_{a,z}$ are $1.6 \pm 2.5, 1.2 \pm 2.2, 5.0 \pm 8.9$ grams, respectively (for reference, the weights of the small and large drones are 34 g and 67 g). Therefore, the output of our learning model in (9) is a scalar to approximate the z-component of the unmodeled force function $\mathbf{f}_a^{(i)}$.

C. Learning Results and Ablation Analysis

Each scenario uses a trajectory with a duration around 1000 seconds. For each scenario, we equally split the total trajectory into 50 shorter pieces, where each one is about 20 seconds. Then we randomly choose 80% of these 50 trajectories for training and 20% for validation.

Our DNN functions of $\boldsymbol{\phi}$ ($\boldsymbol{\phi}_{\mathrm{small}}, \boldsymbol{\phi}_{\mathrm{large}}, \boldsymbol{\phi}_{\mathrm{env}}$) have four layers with architecture $6 \to 25 \to 40 \to 40 \to H$, and our $\boldsymbol{\rho}$ DNNs ($\boldsymbol{\rho}_{\mathrm{small}}, \boldsymbol{\rho}_{\mathrm{large}}$) also have $L = 4$ layers, with architecture $H \to 40 \to 40 \to 40 \to 1$. We use the ReLU function as the activation operator, and we use PyTorch [39] for training and implementation of spectral normalization (see Sec. IV-C) of all these five DNNs. During training we iterate all the data 20 times for error convergence.

Note that H is the dimension of the hidden state. To study the effect of H on learning performance, we use three different values of H and the mean validation errors for each H are shown in Table III. Meanwhile, we also study the influence of the number of layers by fixing $H = 20$ and changing L, which is the number of layers of all $\boldsymbol{\rho}$ nets and $\boldsymbol{\phi}$ nets. For $L = 3$ or $L = 5$, we delete or add a $40 \to 40$ layer for all $\boldsymbol{\rho}$ nets and $\boldsymbol{\phi}$ nets, before their last layers. We repeat all experiments three times to get mean and standard deviation. As depicted in Table III, we found that the average learning performance (mean validation error) is not sensitive to H, but larger H results in higher variance, possibly because using a bigger hidden space (larger H) leads to a more flexible encoding that is harder to train reliably. In terms of the number of layers, four layers are significantly better than five (which tends to overfit data), and slighter better than three. To optimize performance, we finally choose $H = 20$ and use four-layer neural networks, which can be efficiently evaluated on-board. We notice that H and L are the most important parameters, and the learning performance is not sensitive to other parameters such as the number of weights in intermediate layers.

Figure 5 depicts the prediction of $f_{a,z}$, trained with flight data from the 12 scenarios listed in Table II. The color encodes the magnitude of $\hat{f}_{a,z}$ for a single small multirotor positioned at different global (y, z) coordinates. The big/small black drone icons indicate the (global) coordinates of neighboring big/small multirotors, and the dashed line located at $z = 0$ represents the ground. All quadrotors are in the same x-plane.

TABLE III
ABLATION ANALYSIS. TOP: $L = 4$ AND H VARIES. BOTTOM: $H = 20$ AND L VARIES. THE ERROR IS THE MEAN SQUARED ERROR (MSE) BETWEEN $f_{a,z}$ PREDICTION AND THE GROUND TRUTH.

H	10	20	40
Validation Error	6.70±0.05	6.42±0.18	6.63±0.35

L	3	4	5
Validation Error	6.52±0.17	6.42±0.18	7.21±0.28

For example, in Fig. 5(e), one large quadrotor is hovering at $(y = -0.1, z = 0.5)$ and one small quadrotor is hovering at $(y = 0.1, z = 0.5)$. If we place a third small quadrotor at $(y = 0, z = 0.3)$, it would estimate $\hat{f}_{a,z} = -10\,\mathrm{g}$ as indicated by the red color in that part of the heatmap. Similarly, in Fig. 5(a) the small multirotor only has the environment as a special neighbor. If the small multirotor is hovering at $(y = 0, z = 0.05)$, it would estimate $\hat{f}_{a,z} = 5\,\mathrm{g}$, which is mainly from the ground effect. All quadrotors are assumed to be stationary except for Fig. 5(d), where the one neighbor is moving at 0.8 m/s.

We observe that the interaction between quadrotors is non-stationary and sensitive to relative velocity. In Fig. 5(d), the vehicle's neighbor is moving, and the prediction becomes significantly different from Fig. 5(c), where the neighbor is just hovering. To further understand the importance of relative velocity, we retrain neural networks neglecting relative velocity and the mean squared validation error degrades by 18%, from 6.42 to 7.60. We can also observe that the interactions are not a simple superposition of different pairs. For instance, Fig. 5(g) is significantly more complex than a simple superposition of Fig. 5(a) plus three (b), i.e., $\boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{env}}) + \boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{small}}) + \boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{small}}) + \boldsymbol{\rho}_{\mathrm{small}}(\boldsymbol{\phi}_{\mathrm{small}})$. The maximum gap between Fig. 5(g) and the superposition version is 11.4 g. Moreover, we find that the ground effect and the downwash effect from a neighboring multirotor interact in an intriguing way. For instance, in Fig. 5(b), the downwash effect is "mitigated" as the vehicle gets closer to the ground. Finally, we observe that the large quadrotors cause significantly higher interaction forces than the small ones (see Fig. 5(e)), which further emphasizes the importance of our heterogeneous modeling.

Note that in training we only have data from 1-3 vehicles (see Table II). Our approach can generalize well to a larger swarm system. In Fig. 5, predictions for a 4-vehicle team (as shown in Fig. 5(g,h)) are still reliable. Moreover, our models work well in real flight tests with 5 vehicles (see Fig. 9) and even 16 vehicles (see Fig. 1).

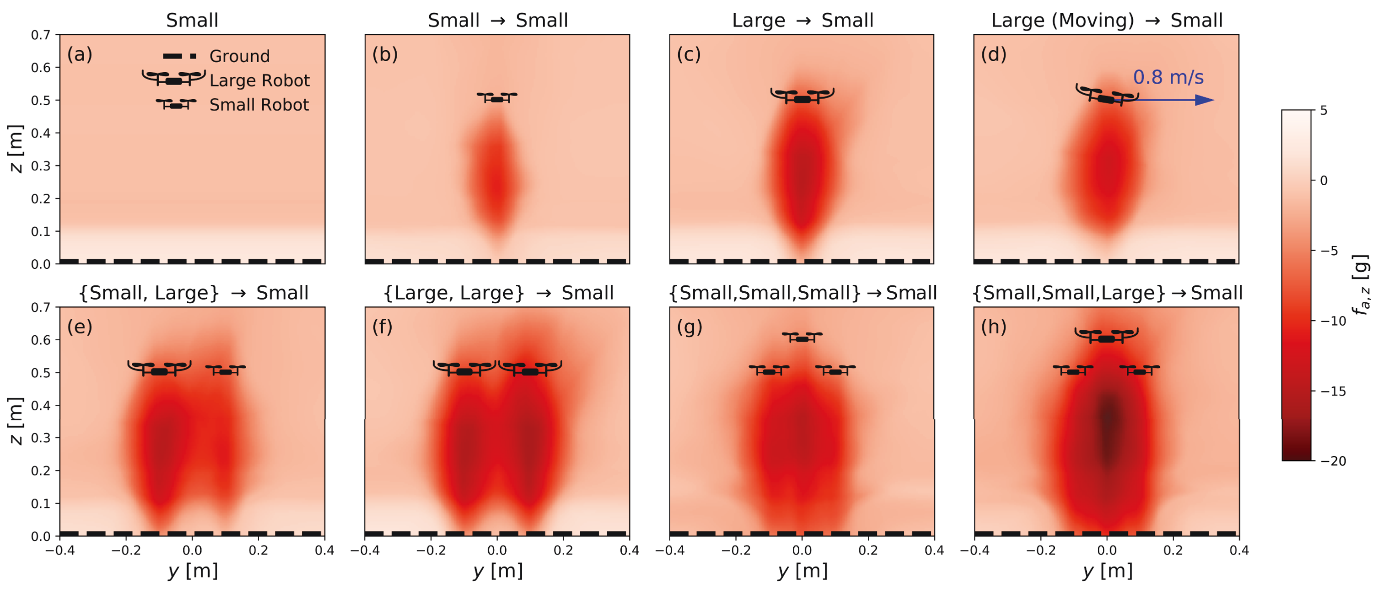


Fig. 5. $f_{a,z}$ prediction from the trained $\{\boldsymbol{\rho}_{\text{small}}, \boldsymbol{\rho}_{\text{large}}, \boldsymbol{\phi}_{\text{small}}, \boldsymbol{\phi}_{\text{large}}, \boldsymbol{\phi}_{\text{env}}\}$ networks. Each heatmap gives the prediction of $f_{a,z}$ of a vehicle in different horizontal and vertical (global) positions. The (global) position of neighboring drones are represented by drone icons. See Sec. VII-C for details.

D. Motion Planning with Aerodynamics Coupling

We implement Algorithm 1 in Python using PyTorch 1.5 [39] for automatic gradient computation, CVXPY 1.0 [49] for convex optimization, and GUROBI 9.0 [50] as underlying solver. To simulate the tracking performance of the planned trajectories, we also implement a nonlinear controller, which uses the planned controls as feed-forward term. We compare trajectories that were planned with a learned model of $f_{a,z}$ with trajectories without such a model (i.e., $f_{a,z} = 0$) using Algorithm 1 with identical parameters. At test time, we track the planned trajectories with our controller, and forward propagate the dynamics with our learned model of $f_{a,z}$.

We visualize an example in Fig. 6, where two small and one large robots are tasked with exchanging positions. We focus on the 2D case in the yz-plane to create significant interaction forces between the robots. The first stage of Algorithm 1 uses sampling-based motion planning to identify the best homeomorphism class where the small multirotors fly on top of the large multirotor (the interaction forces would require more total energy the other way around). However, the robots do not reach their goal state exactly and motions are jerky (Fig. 6, left). The second stage uses SCP to refine the motion plan such that robots reach their goal and minimize the total control effort (Fig. 6, middle). The planned trajectory can be tracked without significant error and the interaction forces are very small for the two small quadrotors and within the chosen bound of 10 g for the large quadrotor. We compare this solution to one where we do not consider the interaction forces between robots by setting $f_{a,z} = 0$ in Algorithm 1. The planned trajectories tend to be shorter (Fig. 6, right, dashed lines) in that case. However, when tracking those trajectories, significant tracking errors occur and the interaction forces are outside their chosen bounds of 5 g for the small multirotors.

We empirically evaluated the effect of planning with and without considering interaction forces in several scenarios, see Fig. 7. We found that ignoring the interaction forces results in significant tracking errors in all cases (top row). While this tracking error could be reduced when using our interaction-aware control law, the interaction forces are in some cases significantly over their desired limit. For example, in the small/large, small/small/large, and large/large cases, the worst-case interaction forces were consistently nearly double the limit (red line, bottom row). In practice, such large disturbances can cause instabilities or even a total loss of control, justifying the use of an interaction-aware motion planner.

E. Control Performance in Flight Tests

We study the flight performance improvements on swapping tasks with varying number of quadrotors. For each case, robots are initially arranged in a circle when viewed from above but at different z-planes and are tasked with moving linearly to the opposite side of the circle in their plane. During the swap, all vehicles align vertically at one point in time with vertical distances of 0.2 m to 0.3 m between neighbors. The tasks are similar, but not identical to the randomized swapping tasks used in Sec. VII-B because different parameters (locations, transition times) are used.

Our results are summarized in Fig. 8 for various combinations of two and three multirotors, where we use “XY2Z” to denote the swap task with robots of type X and Y at the top and a robot of type Z at the bottom. We compute a box plot with median (green line) and first/third quartile (box) of the maximum z-error (repeated over 6 swaps). In some cases, the downwash force was so large that we upgraded the motors of the small quadrotor to improve the best-case thrust-to-weight ratio to 2.6. Such modified quadrotors are indicated as “S*”. We also verified that the x- and y-error distributions are similar across the different controllers and omit those numbers for brevity.

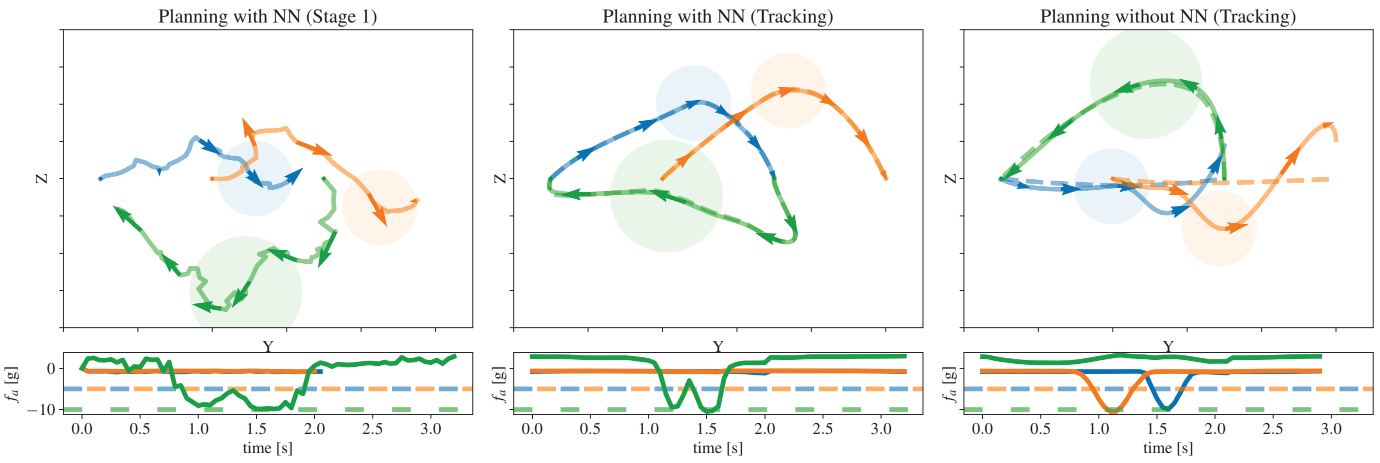


Fig. 6. Example motion planning result for a three-robot swapping task in 2D (blue and orange: small robots; green: large robot). Top row: yz-state space plot, where the arrows indicate the velocities every second, and the circles show the robot collision boundary shape at the middle of the task. Bottom row: interaction force for each robot over time (dashed: desired limit per robot). Left: Sampling-based motion planning with neural network to compute trajectories where the large robots moves below the small robots. Middle: Refined trajectories using SCP (dashed) and tracked trajectories (solid). Right: Planned trajectories when ignoring interaction forces (dashed) and tracked trajectories (solid). In this case, a dangerous configuration is chosen where the large robot flies on top of the small robots, exceeding their disturbance limits of 5 g.

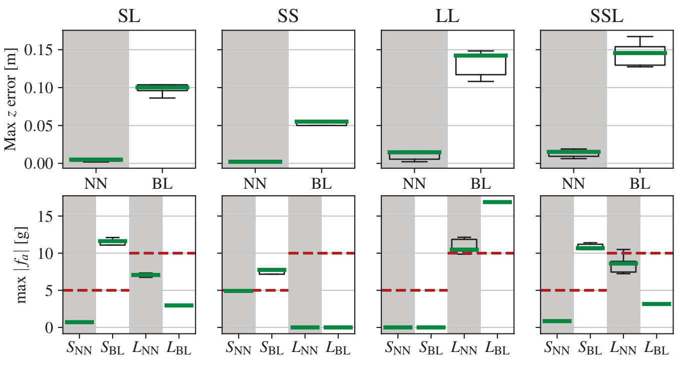


Fig. 7. Motion planning results for different scenarios (e.g., SSL refers to two small robots and one large robot) comparing planning without neural network (BL) and planning with neural network (NN) over 5 trials. Top: Worst-case tracking error. Ignoring the interaction force can result in errors of over 10 cm. Bottom: Worst-case interaction force for small and large quadrotors. The baseline has significant violations of the interaction force bounds, e.g., the SL case might create interaction forces greater than 10 g for the small quadrotor.

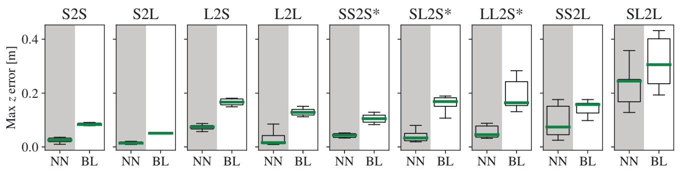


Fig. 8. Flight test results comparing our solution with learned interaction compensation (NN) with the baseline (BL) in different scenarios. For each case, robots are initially arranged in a circle when viewed from above but at different z-planes and are tasked with moving linearly to the opposite side of the circle in their plane. For each swap, we compute the worst-case z-error of the lowest quadrotor and plot the data over six swaps.

Our controller improves the median z-error in all cases and in most cases this improvement is statistically significant. For example, in the "L2S" case, where a large multirotor is on top of a small multirotor for a short period of time, the median z-error is reduced from 17 cm to 7 cm.

To estimate the limits of our learning generalization, we

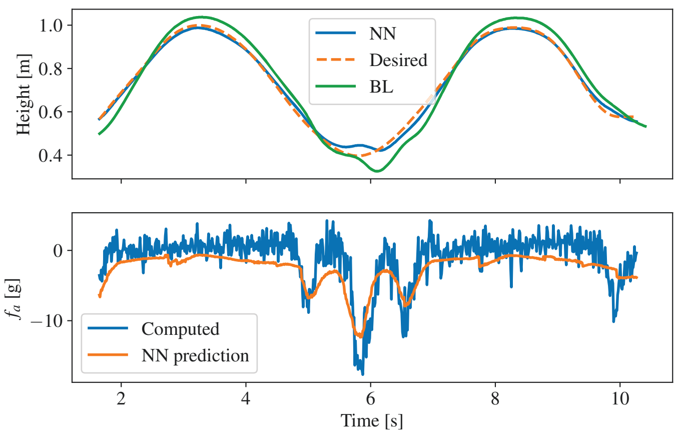


Fig. 9. Generalization to a team of five multirotors. Three small multirotor move in a vertical ring and two large multirotor move in a horizontal ring. The maximum z-error of a small multirotor in the vertical ring with powerful motors is reduced from 10 cm to 5 cm and f_a is predicted accurately.

test our approach on larger teams. First, we consider a team of five robots, where two large robots move on a circle in the horizontal plane and three small robots move on circle in the vertical plane such that the two circles form intertwined rings. In this case, the $f_{a,z}$ prediction is accurate and the maximum z-error can be reduced significantly using our neural network prediction, see Fig. 9 for an example. Second, we consider a team of 16 robots moving on three intertwined rings as shown in Fig. 1(b,c). Here, two large and four small robots move on an ellipsoid in the horizontal plane, and five robots move on circles in different vertical planes. In this case, robots can have significantly more neighbors (up to 15) compared to the training data (up to 2), making the prediction of $f_{a,z}$ relatively less accurate. However, the maximum z-error of a small multirotor in one of the vertical rings with powerful motors is still reduced from 15 cm to 10 cm.

We note that a conceptually-simpler method is to estimate and compensate for $f_{a,z}$ online without learning. However, online estimation will not only introduce significant delays, but also be very noisy especially in close-proximity flight. Our learning-based method has no delay (because it directly *predicts* $f_{a,z}$ at the current time step), and considerably mitigates the noise due to the use of spectral normalization and delay-free filtering in the training process. In experiments, we observe that the online estimation and compensate method would quickly crash the drone.

VIII. Conclusion

In this paper, we present *Neural-Swarm2*, a learning-based approach that enables close-proximity flight of heterogeneous multirotor teams. Compared to previous work, robots can fly much closer to each other safely, because we accurately predict the interaction forces caused by previously unmodeled aerodynamic vehicle interactions. To this end, we introduce *heterogeneous deep sets* as an efficient and effective deep neural network architecture that only relies on relative positions and velocities of neighboring vehicles to learn the interaction forces between multiple quadrotors. Our architecture also allows to model the ground effect and other unmodeled dynamics by viewing the physical environment as a special neighboring robot. To our knowledge, our approach provides the first model of interaction forces between two or more multirotors.

We demonstrate that the learned interactions are crucial in two applications of close-proximity flight. First, they can be used in multi-robot motion planning to compute trajectories that have bounded disturbances caused by neighboring robots and that consider platform limitations such as maximum thrust capabilities directly. The resulting trajectories enable a higher robot density compared to existing work that relies on conservative collision shapes. Second, we can compute the interaction forces in real-time on a small 32-bit microcontroller and apply them as additional feed-forward term in a novel delay-compensated nonlinear stable tracking controller. Such an approach enables to reduce the tracking error significantly, if the maximum thrust capabilities of the robots are sufficient.

We validate our approach on different tasks using two to sixteen quadrotors of two different sizes and demonstrate that our training method generalizes well to a varying number of neighbors, is computationally efficient, and reduces the worst-case height error by a factor of two or better.

References

[1] S.-J. Chung, A. A. Paranjape, P. M. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 837–855, 2018.

[2] D. Morgan, G. P. Subramanian, S.-J. Chung, and F. Y. Hadaegh, "Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming," *I. J. Robotics Res.*, vol. 35, no. 10, pp. 1261–1285, 2016.

[3] K. P. Jain, T. Fortmuller, J. Byun, S. A. Mäkiharju, and M. W. Mueller, "Modeling of aerodynamic disturbances for proximity flight of multirotors," in *Int. Conf. Unmanned Aircraft Syst.*, 2019, pp. 1261–1269.

[4] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 856–869, 2018.

[5] X. Du, C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Fast and in sync: Periodic swarm patterns for quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 9143–9149.

[6] M. Debord, W. Hönig, and N. Ayanian, "Trajectory planning for heterogeneous robot teams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 7924–7931.

[7] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Int. Symp. on Robot. Res.*, vol. 70, 2009, pp. 3–19.

[8] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 3391–3401.

[9] P. L. Bartlett, D. J. Foster, and M. Telgarsky, "Spectrally-normalized margin bounds for neural networks," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 6240–6249.

[10] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural Lander: Stable drone landing control using learned dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 9784–9790.

[11] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 3241–3247, 2020.

[12] I. Cheeseman and W. Bennett, "The effect of ground on a helicopter rotor in forward flight," *Aeronautical Research Council Reports And Memoranda*, 1955.

[13] D. Yeo, E. Shrestha, D. A. Paley, and E. M. Atkins, "An empirical model of rotorcrafy uav downwash for disturbance localization and avoidance," in *AIAA Atmospheric Flight Mechanics Conf.*, 2015.

[14] X. Kan, J. Thomas, H. Teng, H. G. Tanner, V. Kumar, and K. Karydis, "Analysis of ground effect for small-scale uavs in forward flight," *IEEE Trans. Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3860–3867, Oct. 2019.

[15] D. Shukla and N. Komerath, "Multirotor drone aerodynamic interaction investigation," *Drones*, vol. 2, no. 4, 2018.

[16] M. O'Connell, G. Shi, X. Shi, and S.-J. Chung, "Meta-learning-based robust adaptive flight control under uncertain wind conditions," *arXiv preprint arXiv:2103.01932*, 2021.

[17] H. M. Le, A. Kang, Y. Yue, and P. Carr, "Smooth imitation learning for online sequence prediction," in *Proc. Int. Conf. Machine Learning*, vol. 48, 2016, pp. 680–688.

[18] A. J. Taylor, V. D. Dorobantu, H. M. Le, Y. Yue, and A. D. Ames, "Episodic learning with control lyapunov functions for uncertain robotic systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 6878–6884.

[19] R. Cheng, A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. Burdick, "Control regularization for reduced variance reinforcement learning," in *Proc. Int. Conf. Machine Learning*, 2019, pp. 1141–1150.

[20] C. D. McKinnon and A. P. Schoellig, "Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks," *IEEE Trans. Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2180–2187, 2019.

[21] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-efficient control policy search using residual dynamics learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 4709–4715.

[22] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6023–6029.

[23] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2008.

[24] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Proc. Conf. Robot Learning*, 2020, pp. 671–682.

[25] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, 2019, pp. 793–803.

[26] B. Rivière, W. Hönig, Y. Yue, and S.-J. Chung, "GLAS: global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning," *IEEE Trans. Robot. Autom. Lett.*, vol. 5, no. 3, pp. 4249–4256, 2020.

[27] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *J. Guid., Control, Dyn.*, vol. 37, no. 6, pp. 1725–1740, 2014.

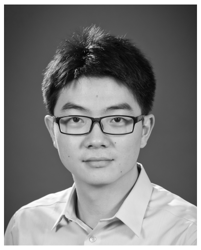
[28] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 477–483.

[29] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[30] A. Liu, G. Shi, S.-J. Chung, A. Anandkumar, and Y. Yue, “Robust regression for safe exploration in control,” in *Learning for Dynamics and Control*, 2020, pp. 608–619.
[31] C. E. Luis and A. P. Schoellig, “Trajectory generation for multiagent point-to-point transitions via distributed model predictive control,” *IEEE Trans. Robot. Autom. Lett.*, vol. 4, no. 2, pp. 375–382, 2019.
[32] M. Čáp, P. Novák, A. Kleiner, and M. Selecký, “Prioritized planning algorithms for trajectory coordination of multiple mobile robots,” *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 835–849, 2015.
[33] R. Foust, S.-J. Chung, and F. Y. Hadaegh, “Optimal guidance and control with nonlinear dynamics using sequential convex programming,” *J. Guid., Control, Dyn.*, vol. 43, no. 4, pp. 633–644, 2020.
[34] Q. T. Dinh and M. Diehl, “Local convergence of sequential convex programming for nonconvex optimization,” in *Recent Advances in Optimization and Its Applications in Engineering*, Springer, 2010, pp. 93–102.
[35] K. Hauser and Y. Zhou, “Asymptotically optimal planning by feasible kinodynamic planning in a state-cost space,” *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1431–1443, 2016.
[36] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[37] T. Kunz and M. Stilman, “Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete,” in *Int. Workshop Algorithmic Foundations of Robotics*, 2015, pp. 233–244.
[38] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *I. J. Robotics Res.*, vol. 30, no. 7, pp. 846–894, 2011.
[39] A. Paszke, S. Gross, F. Massa, *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Proc. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.
[40] Y. K. Nakka, A. Liu, G. Shi, A. Anandkumar, Y. Yue, and S.-J. Chung, “Chance-constrained trajectory optimization for safe exploration and learning of nonlinear systems,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 389–396, 2020.
[41] S. Bandyopadhyay, S.-J. Chung, and F. Y. Hadaegh, “Nonlinear attitude control of spacecraft with a large captured object,” *J. Guid., Control, Dyn.*, vol. 39, no. 4, pp. 754–769, 2016.
[42] X. Shi, K. Kim, S. Rahili, and S.-J. Chung, “Nonlinear control of autonomous flying cars with wings and distributed electric propulsion,” in *Proc. IEEE Conf. Decis. Control*, 2018, pp. 5326–5333.
[43] X. Shi, M. O’Connell, and S.-J. Chung, “Numerical predictive control for delay compensation,” *arXiv preprint arXiv:2009.14450*, 2020.
[44] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian, “Crazyswarm: A large nano-quadcopter swarm,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3299–3304.
[45] Bitcraze. (2015). “Crazyflie 2.0 thrust investigation,” [Online]. Available: https://wiki.bitcraze.io/misc:investigations:thrust (visited on 08/07/2020).
[46] J. Förster, “System identification of the crazyflie 2.0 nano quadrocopter,” en, M.S. thesis, ETH Zurich, Zurich, 2015.
[47] M. Faessler, D. Falanga, and D. Scaramuzza, “Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight,” *IEEE Trans. Robot. Autom. Lett.*, vol. 2, no. 2, pp. 476–482, 2017.
[48] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 1985, pp. 500–505.
[49] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” *J. Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.
[50] L. Gurobi Optimization, *Gurobi optimizer reference manual*, 2020.

Guanya Shi (S’18) is a Ph.D. student at the department of computing and mathematical sciences at the California Institute of Technology, USA. He holds a diploma in mechanical engineering (*summa cum laude*) from Tsinghua University, China (2017). His research focuses on the intersection of machine learning and control theory with the applications in real-world complex systems such as robotics. He was the recipient of several awards, including the Simoudis Discovery Prize and the Qualcomm scholarship.

Wolfgang Hönig (S’15–M’20) is an junior research group leader at TU Berlin, Germany. Previously, he was a postdoctoral scholar at the California Institute of Technology, USA. He received the diploma in Computer Science from TU Dresden, Germany in 2012, and the M.S. and Ph.D. degrees from the University of Southern California (USC), USA in 2016 and 2019, respectively. His research focuses on enabling large teams of physical robots to collaboratively solve real-world tasks, using tools from informed search, optimization, and machine learning. Dr. Hönig has been the recipient of several awards, including Best Paper in Robotics Track for a paper at ICAPS 2016 and the 2019 Best Dissertation Award in Computer Science at USC.



Xichen Shi (S’13–M’20) is a software engineer at Waymo, an autonomous driving vehicle company. He received a Ph.D. in Space Engineering from California Institute of Technology, USA in 2021, and a B.S in Aerospace Engineering (Highest Honors) from University of Illinois at Urbana-Champaign, USA in 2013. His research focuses on intelligent control systems for fixed-wing and multirotor aerial robots.

Yisong Yue is a professor of Computing and Mathematical Sciences at the California Institute of Technology. He was previously a research scientist at Disney Research and a postdoctoral researcher at Carnegie Mellon University. He received a Ph.D. from Cornell University and a B.S. from the University of Illinois at Urbana-Champaign. Dr. Yue’s research interests are centered around machine learning and has been applied to information retrieval, data-driven animation, behavior analysis, protein engineering, and learning-accelerated optimization. Dr. Yue was the recipient of several awards and honors, including the Best Paper Award at ICRA 2020, the Best Student Paper Award at CVPR 2021, the Best Paper Nomination at WSDM 2011, ICDM 2014, SSAC 2017 and RA-L, Best Reviewer at ICLR 2018, and the Okawa Foundation Grant Recipient, 2018.

Soon-Jo Chung (M’06–SM’12) is the Bren Professor of Aerospace and Control and Dynamical Systems and a Jet Propulsion Laboratory Research Scientist in the California Institute of Technology, USA. He was with the faculty of the University of Illinois at Urbana-Champaign during 2009–2016. He received the B.S. degree (*summa cum laude*) in aerospace engineering from the Korea Advanced Institute of Science and Technology, South Korea, in 1998, and the S.M. degree in aeronautics and astronautics and the Sc.D. degree in estimation and control from Massachusetts Institute of Technology, USA, in 2002 and 2007, respectively. His research interests include spacecraft and aerial swarms and autonomous aerospace systems, and in particular, on the theory and application of complex nonlinear dynamics, control, estimation, guidance, and navigation of autonomous space and air vehicles. Dr. Chung was the recipient of the UIUC Engineering Deans Award for Excellence in Research, the Beckman Faculty Fellowship of the UIUC Center for Advanced Study, the U.S. Air Force Office of Scientific Research Young Investigator Award, the National Science Foundation Faculty Early Career Development Award, a 2020 Honorable Mention for the IEEE RA-L Best Paper Award, and three Best Conference Paper Awards from the IEEE and AIAA. He is an Associate Editor of IEEE T-AC and AIAA JGCD. He was an Associate Editor of IEEE T-RO, and the Guest Editor of a Special Section on Aerial Swarm Robotics published in the IEEE T-RO. He is an Associate Fellow of AIAA.