Learning to Navigate in Turbulent Flows with Aerial Robot Swarms: A Cooperative Deep Reinforcement Learning Approach

Diego Patiño[1], Siddharth Mayya[2], Juan Calderon[3], Kostas Daniilidis[1] and David Saldaña[4]

***Abstract*— Aerial operation in turbulent environments is a challenging problem due to the chaotic behavior of the flow. This problem is made even more complex when a team of aerial robots is trying to achieve coordinated motion in turbulent wind conditions. In this paper, we present a novel multi-robot controller to navigate in turbulent flows, decoupling the trajectory-tracking control from the turbulence compensation via a nested control architecture. Unlike previous works, our method does not learn to compensate for the air-flow at a specific time and space. Instead, our method learns to compensate for the flow based on its effect on the team. This is made possible via a deep reinforcement learning approach, implemented via a Graph Convolutional Neural Network (GCNN)-based architecture, which enables robots to achieve better wind compensation by processing the spatial-temporal correlation of wind flows across the team. Our approach scales well to large robot teams —as each robot only uses information from its nearest neighbors—, and generalizes well to robot teams larger than seen in training. Simulated experiments demonstrate how information sharing improves turbulence compensation in a team of aerial robots and demonstrate the flexibility of our method over different team configurations.**

I. Introduction

Aerial vehicles naturally have to operate in environments with windy conditions. The wind field directly affects the vehicle's motion, potentially leading it outside its desired trajectory or even to crash. Navigating in windy conditions is even more difficult when air-flow is turbulent, presenting a chaotic behavior with hard-to-predict changes in pressure and flow velocity. This challenge is exacerbated in aerial multi-robot scenarios where a team of robots has to perform coordinated tasks which might require staying within communication range without colliding with one another. However, operating multi-robot systems in turbulent environments is highly relevant to reducing delivery and transportation delays, as well as supporting search and rescue operations during natural disasters from storms, tornadoes, and hurricanes.

The existing robotics literature has studied the problem of navigation flows, relying on assumptions to make the problem tractable. While some approaches assume a known (static or dynamic) wind field, e.g., [1], [2], [3], other methods learn an association between a *location* in the environment and the effect of the flow [4], [5]. These are relevant limitations because it does not allow the robots to reuse their learned information in turbulent flows where such associations are constantly evolving or being faced with a new or unknown environment.

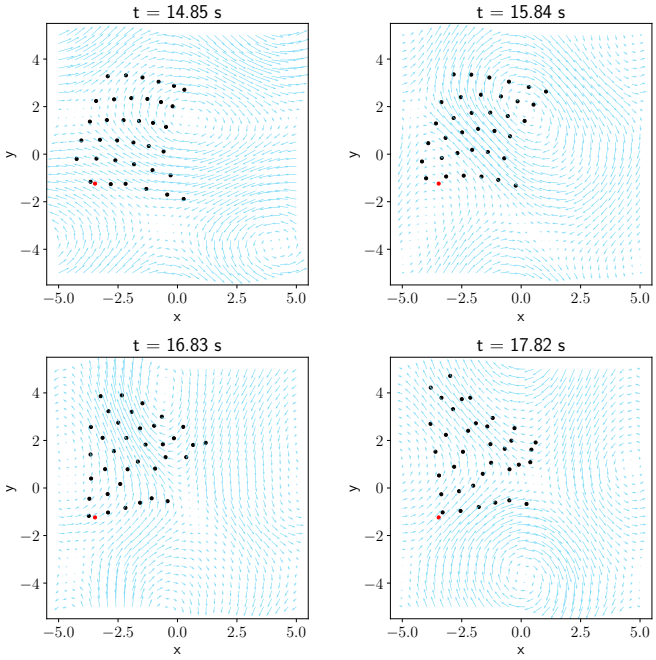


Fig. 1: A team of 36 robots navigating in turbulent wind. The robots are trying to maintain a square formation using only a trajectory-tracking controller. Blue arrows show the wind vector field. The red dot shows the target location of the bottom-left robot in the formation.

In Fig. 1, we show an aerial multi-robot system operating in a turbulent wind flow. This figure illustrates a key observation: *sensory information sharing* between the robots can provide valuable information to improve the robots' turbulence compensation in the absence of predictive wind-flow maps. For example, the approach of a new wind front could be detected by a robot, which can then relay pertinent information to other robots to better compensate the wind. This essentially occurs because the rapid fluctuations in wind velocity and direction inherent to turbulent winds are spatio-temporally correlated across the region.

The primary contribution of this paper is a novel method for trajectory tracking in turbulent flows using multiple aerial vehicles equipped with sensors to measure wind pressure and relative distance to other robots. Specifically, our method leverages structured information sharing over a graph where robots represent nodes and communication between robots

[1] D. Patiño and Kostas Daniilidis are with the GRASP Lab., University of Pennsylvania, PA, USA: {diegopc, kostas }@cis.upenn.edu

[2] S. Mayya is with Amazon Robotics, North Reading, MA, USA: mayya.siddharth@gmail.com. This work is not related to Amazon.

[3] J. Calderon is with Universidad Santo Tomas, Colombia, and Bethune Cookman University, FL, USA: calderonj@cookman.edu

[4] D. Saldaña is with the Autonomous and Intelligent Robotics Laboratory (AIRLab), Lehigh University, PA, USA: saldana@lehigh.edu

represents edges. To ensure generality over qualitatively different turbulent flows, we develop a deep reinforcement learning approach, implemented via a Graph Convolutional Neural Network (GCNN). Our approach learns to fuse and transform sensory information received from neighbors [6] in order to compensate for wind forces.

Crucially, our method does not need to learn to map between a specific location and the wind flow. Instead, it leverages spatio-temporal correlations (as described by the Navier-Stokes equations [7]) in wind flow between team members. Our method ensures that the learned information will not be associated with a specific training environment or trajectory. Furthermore, this ensures a decoupling between the nominal trajectory tracking controller and the controller for turbulence compensation.

Our approach is scalable due to the use of the GCNN because each robot only uses the information from its on-board sensors and the information of its neighbors in the communication graph. Our experiments demonstrate this scalability as well as the efficacy of the proposed approach. These experiments also offer insights into how the learned models leverage shared information among the robots for effective turbulence compensation.

Related Work: The original robotic navigation problem in windy environments was proposed by Zermelo in 1931 [8]. When modeling the flow as a vector field, some works assume that the flow is known and quasi-static, i.e., does not change in time and space. These works focus on developing planning methods for static vector fields [1], [2], and spatio-temporal dynamic fields [3]. However, those methods rely on knowing the vector field at the planning stage, which is unpredictable for turbulent flows.

For unknown static flows, the works in [9], [10] design robot navigation strategies that drive the robot to sweep the environment and create a map of the flow. In [11], the authors designed an adaptive controller for a quadrotor that models the flow as two parts: 1) a time-varying vector field that can be estimated and 2) an unknown speed-bounded flow that is assumed as noise. Flow prediction is also studied and implemented in realistic settings [12], [13]. Similar to the aforementioned works, however, they involve a large number of samples of the environment.

For unknown dynamics of the flow, learning approaches have shown promising results. A safe learning approach for a quadrotor is presented in [14]. Assuming the flow is static, the robot starts in a safe region that can be expanded as the learning process evolves. The work in [4] presents an adaptive flight control that learns how to track a given trajectory on a static flow. A reinforcement learning approach to navigate a static wind field is presented in [5].

As discussed in the introduction, our method does not need to create associations between locations in the environment and the wind flow. Towards this end, we leverage Graph Convolutional Neural Networks (GCNNs) [6], [15]. They are effective at modeling associations within a graph and have been applied in a wide range of fields, including multi-robot coordination and decision making e.g., [16], [17].

II. Problem Statement

Robot Team: Consider a team of n aerial robots, denoted by the set $\mathcal{V} = \{1, ..., n\}$. Assuming that all robots are at the same height, we analyze their location and motion on the plane. The position of each robot $i \in \mathcal{V}$ is denoted by $\boldsymbol{r}_i \in \mathbb{R}^2$. We define the state vector by the position and the velocity of the robot, i.e, $\boldsymbol{x}_i = [\boldsymbol{r}_i^\top, \dot{\boldsymbol{r}}_i^\top]^\top$. We assume all robots are homogeneous and have the same mass m. Each robot i can use its local sensors to estimate its state as well as select variables of the environment. Each robot can generate a force vector $\boldsymbol{f}_i \in \mathbb{R}^2$ as control input, i.e.,

$$\boldsymbol{f}_i = \boldsymbol{u}_i. \tag{1}$$

For this formulation, our aerial vehicles can be a fully actuated hexarotor [18] or an under-actuated quadrotor that tilts to generate a force in any direction [19]. Each robot i can exchange messages with its k nearest robots denoted by $\mathcal{N}_i$. At every time step, each robot communicates its state and information from on-board sensors.

Wind field: Our robot team operates in a windy environment $\mathcal{W} \subset \mathbb{R}^2$. We represent the wind's velocity at a time t and a location $\boldsymbol{r}_i \in \mathcal{W}$ as a vector-valued function $\boldsymbol{w} : \mathbb{R}_{\geq 0} \times \mathbb{R}^2 \to \mathbb{R}^2$. The vector field follows the dynamics of a fluid, described by the incompressible Navier-Stokes equations [7]

$$\begin{aligned} \nabla \cdot \boldsymbol{w} &= 0 \\ \dot{\boldsymbol{w}} + \boldsymbol{w} \cdot \nabla \boldsymbol{w} &= -\nabla p + \frac{1}{Re} \nabla^2 \boldsymbol{w}, \end{aligned} \tag{2}$$

where Re is the flow's Reynolds number, and p is the scalar pressure field. The Reynolds number measures the ratio between inertial and viscous forces. It characterizes flow patterns in a fluid, e.g., at low Re, flows tend to be dominated by laminar flow, while at high Re, flows tend to be turbulent. In this work, we focus on turbulent environments with high Reynolds numbers [20], $Re \geq 4 \times 10^3$, in the flow dynamics (2). Note that this type of turbulent environment has not been explored in the mobile robotics literature.

As a robot moves through the air, the wind exerts a drag force on the robot in the fluid's direction [21]. We compute the drag force as

$$\mathbf{f}^{drag} = \frac{1}{2} \rho \|\boldsymbol{w}\|^2 C_d \, A \, \hat{\boldsymbol{w}}, \tag{3}$$

where ρ is the air density, the operator $\|\cdot\|$ is the 2-norm, C_d is the robot's drag coefficient, A is the cross-sectional area, and $\hat{\boldsymbol{w}}$ is a unit vector in the direction of $\boldsymbol{w}$. In this context, the reference area is the orthogonally projected frontal area, i.e., the object's visible area as seen from a point on its line of travel. We assume that the drag coefficient and the air density are constant.

Sensors: The robots in our team do not know the wind field nor any of the coefficients in (3). However, they can use their equipped sensors and noisy measurements to gather information about their surroundings. Each robot is equipped with a pressure sensor, a location sensor, and an inertial measurement unit (IMU). The IMU estimates the robot's linear velocity. The robot can measure the relative distance to

their k-nearest neighbors using any relative location system, e.g., camera, LIDAR, sonar, or time of flight (ToF) sensor.

Robot dynamics: The robot's actuation and the turbulent wind generate linear forces that determine the robot's motion. We model the dynamics of the ith robot using Newton's equation,

$$m\ddot{\boldsymbol{r}}_i = \boldsymbol{u}_i + \mathbf{f}_i^{drag}. \tag{4}$$

Trajectory tracking: The goal for the robot i is to follow a given trajectory $\boldsymbol{x}_t^d = [\boldsymbol{r}_t^{d\top}, \dot{\boldsymbol{r}}_t^{d\top}]^\top$, specified by a desired location $\boldsymbol{r}_i^d$ and desired velocity $\dot{\boldsymbol{r}}_i^d$ in a time interval $[0, t_f]$ [22]. Assuming an environment without wind, i.e., $\mathbf{f}_i^{drag} = \mathbf{0}$ in (4), we can use a classical trajectory-tracking approach that provides exponential stability [23] based on a feed-forward controller,

$$\boldsymbol{u}_i^{tt} = \boldsymbol{K}_p(\boldsymbol{r}_i^d - \boldsymbol{r}_i) + \boldsymbol{K}_d(\dot{\boldsymbol{r}}_i^d - \dot{\boldsymbol{r}}_i) + \ddot{\boldsymbol{r}}_i^d, \tag{5}$$

where $\mathbf{K}_p$ and $\mathbf{K}_d$ are the diagonal gain matrices. The main challenge here is that $\mathbf{f}_i^{drag}$ is not negligible and can drive the robot far away from the given trajectory, thereby making the dynamical system in (4) unstable.

Objective: Our objective is to allow the robot team to track a trajectory while operating in a dynamic, turbulent wind field. We, therefore, need to solve the following:

Problem 1: Given a set of n robots and a trajectory that can be solved with a control policy $\boldsymbol{u}_i^{tt}$, which does not consider turbulence, find a control input $\boldsymbol{u}_i$ such that the robots can perform the given task in a turbulent environment. Our key insight is that although the robots do not know the wind field, each can share its state and sensor measurements with neighboring robots. Sharing information allows each robot to increase its knowledge about the working environment, leading to an action policy that effectively compensates for the wind's drag force.

Note that our approach is independent of the trajectory tracking because we aim to learn the wind patterns independently of the trajectory-tracking controller.

III. Deep Reinforcement Learning Method

Control Strategy: The key to our control strategy is decoupling the trajectory-tracking controller and the wind compensation. Trajectory-tracking controllers already show exponential convergence [19], [23]. However, convergence is not guaranteed when an external force from the wind is added to the dynamics as modeled in (4). To overcome this limitation, we leverage Reinforcement Learning (RL) to design a second controller that compensates for wind disturbances. This new controller forms an inner control loop, as seen in Fig. 2, and assists the trajectory-tracking controller by helping it converge as if operating in a disturbance-free setting.

The force generated by a robot is the combination of an RL-based wind compensation force $\boldsymbol{f}_i^{rl}$ and trajectory tracking force $\boldsymbol{f}_i^{tt}$. So the total force generated by the robot is $\boldsymbol{u}_i = \boldsymbol{f}_i^{rl} + \boldsymbol{f}_i^{tt}$. Substituting the total force in (4), we obtain

$$m\ddot{\boldsymbol{r}}_i = \boldsymbol{f}_i^{rl} + \boldsymbol{f}_i^{tt} + \mathbf{f}_i^{drag}. \tag{6}$$

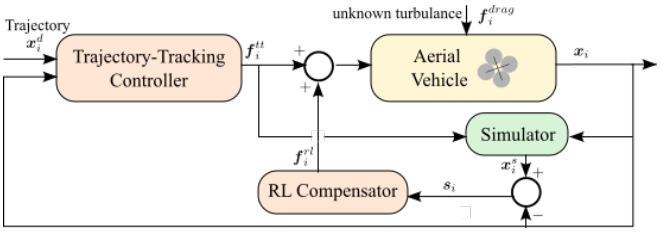


Fig. 2: Control diagram of our proposed method.

We set the trajectory-tracking force to be the control's action from (5), such that $\boldsymbol{f}^{tt} = \boldsymbol{u}_i^{tt}$.

The purpose of the $\boldsymbol{f}_i^{rl}$ is to compensate for the effect of the wind flow, thereby allowing the robots to track their desired trajectory. To this end, let $\mathcal{A}$ be the action space and $\mathcal{S}$ the state space in the RL context. We use a Deep-RL policy – $\boldsymbol{\pi}_i^{\boldsymbol{\theta}}(\boldsymbol{a}_i|\boldsymbol{s}_i)$ – to compute a wind compensation action for each robot. We parametrize the policy with a deep neural network with parameters $\boldsymbol{\theta}$, conditioned on a set of observed variables $\boldsymbol{s}_i \in \mathcal{S}$. Then, we set $\boldsymbol{f}^{rl} = \boldsymbol{a}_i$ where $\boldsymbol{a}_i \sim \boldsymbol{\pi}_i^{\boldsymbol{\theta}}(\boldsymbol{a}_i|\boldsymbol{s}_i)$.

We set the action space $\mathcal{A}$ to be $[-f_{rl}^{max}, f_{rl}^{max}]^2 \subset \mathbb{R}^2$, representing a two-dimensional bounded force. Unlike classical RL methods, the ith robot's policy depends on all the states in the robotic team rather than just $\boldsymbol{s}_i$. This allows our method to use information across robots. Later in this section, we will offer a precise definition of $\mathcal{S}$ and the information-sharing architecture of our RL method.

In our method, we perform sampling and actuating periodically. Consequently, we assume that the time is discrete, i.e., $t = 0, 1, 2, ...$, and the time step Δt is small enough to apply our method in the dynamics equations in (6).

Soft Actor-Critic: We learn $\boldsymbol{\pi}^{\boldsymbol{\theta}}(\boldsymbol{a}_i|\boldsymbol{s}_i)$ using the Soft Actor-Critic algorithm (SAC). SAC is an off-policy Deep Reinforcement Learning (DRL) algorithm based on entropy regularization to trade off exploitation and exploration policies. SAC has demonstrated stability, sample-efficient learning, and optimal policy convergence [24]. The SAC method optimizes $\boldsymbol{\pi}^{\boldsymbol{\theta}}{}_i$ by jointly maximizing its expected reward and its entropy [24], [25]. Incorporating the entropy term into the RL framework casts an optimization problem of the form

$$\pi_i^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma_t\Big(r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_i') + \alpha H\left(\pi(\cdot|\boldsymbol{s}_i)\right)\Big)\right], \tag{7}$$

where $\boldsymbol{s}_i'$ is the state in the next time step after applying the action $\boldsymbol{a}_i$, α is the trade-off coefficient, r is the reward signal, γ is the discount factor, and H is the policy's entropy. The α values control the trade-off between the expected reward and entropy of the policy, balancing exploration and exploitation. Appropriate values of α accelerate the learning process towards the optimal policy and prevent convergence to local minima [24].

Following (7), SAC uses a Deep Q-Learning strategy that incorporates H into a slightly modified version of the Bellman equation for the value function

$$V(\boldsymbol{s}_i) = \mathop{\mathbb{E}}_{\boldsymbol{a}_i\sim\pi}\left[Q(\boldsymbol{s}_i, \boldsymbol{a}_i)\right] + \alpha H\left(\pi(\cdot|\boldsymbol{s}_i)\right) \tag{8}$$

and the Bellman equation for the Q-function

$$Q(\boldsymbol{s}_i, \boldsymbol{a}_i) = \mathop{\mathbb{E}}_{\boldsymbol{s}'_i \sim P} \left[r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) + \gamma V(\boldsymbol{s}'_i) \right], \tag{9}$$

where P is the probability distribution of the future state s'_i.

In practice, SAC estimates three functions: The policy (Actor) and two Q-functions (Critics). First, it approximates the policy as a Gaussian distribution $\boldsymbol{\pi}^{\boldsymbol{\theta}} \sim \mathcal{N}(\mu_{\boldsymbol{\theta}}, \Sigma_{\boldsymbol{\theta}})$. Both $\mu_{\boldsymbol{\theta}}$ and $\Sigma_{\boldsymbol{\theta}}$ are the outputs or a neural network parametrized with $\boldsymbol{\theta}$ and optimized through gradient descent using the re-parametrization trick [26]. Similarly, SAC estimates two Q-functions $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$ as neural networks with parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_1$, respectively. The Q-function networks train by minimizing the objective $J_Q(\boldsymbol{\theta}_i)$

$$\mathop{\mathbb{E}}_{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) \sim \mathcal{D}} \left[\left(Q_{\boldsymbol{\theta}_j}(\boldsymbol{s}_i, \boldsymbol{a}_i) - (r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \gamma V_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2}(\boldsymbol{s}'_i)) \right)^2 \right] \tag{10}$$

over samples taken from a replay buffer $\mathcal{D} = \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ of experience gathered during multiple episodes in the training process. The value function $V_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2}$ is implicitly defined through the Q-function and the policy, as stated in [25]. Similarly, the objective for the Gaussian policy is given by

$$J_\pi(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{s}_i \sim \mathcal{D}, \boldsymbol{a}_i \sim \boldsymbol{\pi}^{\boldsymbol{\theta}}} \left[\alpha \log \boldsymbol{\pi}^{\boldsymbol{\theta}}(\boldsymbol{a}_i | \boldsymbol{s}_i) - \min_{j \in \{1,2\}} Q_{\boldsymbol{\theta}_j}(\boldsymbol{s}_i, \boldsymbol{a}_i) \right]. \tag{11}$$

Note that minimizing (10) is equivalent to finding the Q-function that best approximates the value function V. Analogous, minimizing (11) is equivalent to jointly maximizing the expected reward and the policy's entropy.

In this work, we adapt the SAC method to optimize the ith robot's policy conditioned on all the robot states in the team as opposed to a single agent state.

State space: Our approach does not focus on tracking the trajectory but on learning how to directly compensate for the disturbance experienced by the robots, such that the trajectory-tracking controller can operate freely. For this purpose, we integrate the dynamics in (6) to simulate the robot's dynamics under perfect conditions. In these conditions, there is no drag force and hence no need for wind compensation. Therefore, our RL approach's state $\boldsymbol{s}_i \in \mathcal{S}$ relates to how much the trajectory-tracking state $\boldsymbol{x}_i$ differs from a simulated state $\boldsymbol{x}_i^{sim}$. Note that $\boldsymbol{x}_i^{sim}$ does not consider the wind effect.

Let us denote the trajectory-tracking state of the ith robot at a time t by $\boldsymbol{x}_i[t]$, and its simulated state by $\boldsymbol{x}_i^{sim}[t]$. Using Euler integration, we can predict the disturbance-free state $\boldsymbol{x}_i^{sim}[t]$ using the past state $\boldsymbol{x}_i[t-1]$, and a trajectory-tracking action $\boldsymbol{u}_i^{tt}[t-1]$. We can write the discrete-time dynamics from (6) in matrix form, assuming $\mathbf{f}^{drag} = \boldsymbol{f}^{rl} = 0$, to compute the simulated state at the time t,

$$\boldsymbol{x}_i^{sim}[t] = \boldsymbol{A}\boldsymbol{x}_i[t-1] + \boldsymbol{B}\boldsymbol{u}_i^{tt}[t-1], \tag{12}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \Delta t\, \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \; \boldsymbol{B} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\Delta t}{m}\mathbf{1} \end{bmatrix},$$

and Δt is a small time step. Then, the wind disturbance displacement vector is the difference between the current state $\boldsymbol{x}_i[t]$ and the simulated state $\boldsymbol{x}_i^{sim}[t]$,

$$\boldsymbol{e}_i[t] = \boldsymbol{x}_i^{sim}[t] - \boldsymbol{x}_i[t]. \tag{13}$$

As described in Sec. II, the wind applies a drag force $\mathbf{f}_i^{drag}$ on the robots. This force results from the pressure field gradient plus the friction forces due to air particles as described by (2). Each robot takes noisy measurements of the pressure field p_i at its location to compensate for the effect of these forces.

Finally, we define the state vector $\boldsymbol{s}_i$ for our RL method at each robot i, by concatenating the displacement vector $\boldsymbol{e}_i$, the pressure field value p_i, and the robot's velocity $\dot{\boldsymbol{r}}_i$ such that

$$\boldsymbol{s}_i = \boldsymbol{e}_i \parallel \dot{\boldsymbol{r}}_i \parallel p_i, \tag{14}$$

where $\cdot \parallel \cdot$ is the concatenation operator. We include the robot's velocity because the drag force directly affects this quantity. During training, as discussed more in Sec. IV, we add Gaussian noise to $\boldsymbol{s}_i$ to simulate real-world sensory noise.

Graph Convolutional Neural Network Architecture: The wind flow dynamics in (2) reveal a spatio-temporal correlation for $\boldsymbol{w}$, i.e., the wind velocity at a given location correlates with the wind velocities at nearby areas. Our proposed method takes advantage of the spatial correlation by enabling information sharing between the robotic team members. When we use multiple robots spatially distributed in $\mathcal{W}$, we form a sensing network that indirectly samples information about the effects of the wind on the robots. Consequently, we use this sensing network to improve the action that compensates for the drag force exerted on a robot i with the help of its neighbors $\mathcal{N}_i$.

Since SAC was designed for a single agent, its actor's architecture is a multi-layer perceptron (MLP). An MLP acts only on the individual robot's states $\boldsymbol{s}_i$ to compute the robot's action $\boldsymbol{a}_i$. Hence, the MLP's architecture does not use information from other robotic team members. To model this information exchange explicitly, we design the actor – and the two critics – as Graph Convolutional neural networks (GCNN) [27]. A L-layered GCNN is a type of neural network that can process data represented as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \boldsymbol{H})$ with nodes $\mathcal{N}$, edges $\mathcal{E}$, and a feature set $\boldsymbol{H} = \{\boldsymbol{H}^0, \ldots, \boldsymbol{H}^L\}$. In the context of this paper, the nodes represent robots, and the edges represent the information exchange between them. We present an overview of the full architecture for our GCNN-based actor and the critic in Fig. 3.

At a given layer $l \in [0, ..., L]$, the network computes a feature vector for each robot i, denoted by $\boldsymbol{h}_i^l$, and organizes them into a $n \times c_l$ matrix

$$\boldsymbol{H}^l = [\boldsymbol{h}_1^l, .., \boldsymbol{h}_n^l]^\top.$$

We compute $\boldsymbol{H}^l$ from the previous layer's features following

$$\boldsymbol{H}^{l+1} = \sigma\left(\boldsymbol{H}^l \boldsymbol{\Theta}_1^l + \boldsymbol{A}_{adj} \boldsymbol{H}^l \boldsymbol{\Theta}_2^l \right), \tag{15}$$

where $\boldsymbol{A}_{adj}$ is the adjacency matrix of the graph, $\boldsymbol{\Theta}_1^l$ and $\boldsymbol{\Theta}_2^l$ are learnable weight matrices of size $c_l \times c_{l+1}$, and $\sigma(\cdot)$ is an element-wise non-linear activation function. We set the input features of the network to be a matrix containing all the robot's states defined in (14), such that

$$\boldsymbol{H}^0 = [\boldsymbol{s}_1, ..., \boldsymbol{s}_n]^\top.$$

The operation in (15) denotes a graph convolution operation where a robot's features are updated using information from its neighbors in the graph. However, this operation does not include information about the relative position $\boldsymbol{r}_{ij}$ between robot i and its neighbor j. Without the relative position, the robots do not know where the neighboring robots are located. This makes it difficult to approximate vector quantities such as the pressure gradient in (2). To overcome this limitation, we incorporate the relative position into the convolution operator by concatenating $\boldsymbol{r}_{i,j}$ to the features at each layer right before the weighting and the neighbor aggregation. For simplicity, we will use the per-node notation of (15) to denote the convolution at each robot i. We incorporate the changes to include the relative position and define the layer's features at each robot as

$$\boldsymbol{h}_i^{l+1} = \sigma\left(\boldsymbol{\Theta}_1^l \boldsymbol{h}_i^l + \boldsymbol{\Theta}_2^l \sum_{j \in \mathcal{N}(i)} \left(\boldsymbol{h}_j^l \parallel \boldsymbol{r}_{i,j}\right)\right), \tag{16}$$

where $\boldsymbol{\Theta}_2^l$ is now a $c_{l+1} \times (c_l + 2)$ matrix. The actor's GCNN architecture takes $\boldsymbol{H}^0$ and $\boldsymbol{A}_{adj}$ as inputs, and computes a latent vector representation $\boldsymbol{h}_i^L$ at the last layer L. To decode $\boldsymbol{h}_i^L$ into the robot's action, we pass $\boldsymbol{h}_i^L$ through an small MLP network. We split the MLP's output into $\boldsymbol{\mu}_i^{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_i^{\boldsymbol{\theta}}$, and we use them to parameterize $\boldsymbol{\pi}^{\boldsymbol{\theta}}$ as a normal distribution. Following [24], we set $\boldsymbol{\Sigma}_i^{\boldsymbol{\theta}}$ to be a diagonal matrix. Finally, we use the policy to obtain the action $\boldsymbol{a_i}$.

Each of the critic's architecture follows a similar design with two small modifications since the critic is a function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. First, the critic's output is a single-value function instead of a probability distribution. To model its output properly, we modify the critic's MLP decoder to have a single output neuron rather than $\boldsymbol{\mu}_i^{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_i^{\boldsymbol{\theta}}$. Second, the input space of the critic architecture consists of the robot's action in addition to just the state. Consequently, the input to the critic's GCNN is a feature vector

$$\boldsymbol{H}^{0'} = \left[(\boldsymbol{s}_1 \parallel \boldsymbol{a}_1), ..., (\boldsymbol{s}_n \parallel \boldsymbol{a}_n)\right]^\top.$$

In each architecture, we use a two-layer GCNN with ReLU as the non-linear activation function and two hidden layers of 64 neurons per layer. The MLP decoders are two-layer networks of size 64 and 16, respectively. We add an extra output layer to the decoders to re-shape the network's output to the appropriate size for the actor or critics. The MLP's layers use ReLU as their activation function in the inner layers and a linear activation function for the output layer. Finally, the actor's output is squeezed into the range $[-1, 1]$ using a $\tanh$ function as described in the SAC formulation. In practice, we scale $\boldsymbol{a}_i$ by a preset factor of $\sqrt{2} f_{rl}^{max}$ representing the maximum force that the robots can generate, as discussed after (6).

Reward Signal: The final component of our proposed method is the reward signal. The reward signal at each step tells the SAC how well the robots compensate for the wind's drag force at a given step in the training process.

Recall that we expect the robot team to learn to operate the same as when there is no turbulence. Because turbulence affects the acceleration of the robots, the divergence between

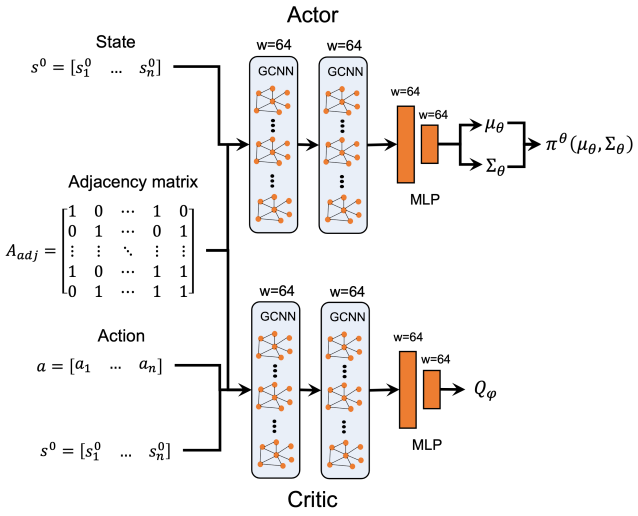


Fig. 3: Team-level architecture of the actor and critic networks used within the proposed RL architecture.

the expected simulated velocity at a robot i and its actual velocity is an appropriate quantity to incorporate into our reward signal. We can do a similar analysis on the divergence between the simulated position and the actual position measured with the robot's instruments. These divergence quantities are captured into the displacement vector $\boldsymbol{e}_i$ in (13). Hence, we define our reward function for our RL method as the L1-norm to weighted displacement,

$$r[t] = -\|\boldsymbol{\beta} \odot \boldsymbol{e}_i\|, \tag{17}$$

with $\odot$ the Hadamard product, and $\boldsymbol{\beta}$ a weight vector rating the importance of each component of $\boldsymbol{e}_i$ in the reward signal.

As the displacement vector between the simulated state and the actual state vector approaches the zero vector, the reward signal becomes less negative. Therefore, learning a policy that maximizes (17) is equivalent to learning an action policy that compensates for the effect of the wind on the robots.

IV. Experiments

We design three experiments to evaluate our method's performance. First, we show that our method allows robots to navigate turbulent wind regimes by independently compensating the wind and tracking the target's trajectory separately. Second, we show that our method is robust to changes in the robot team's configuration, such as neighborhood size and formation size. Third, we demonstrate that the advantages of our method arise from our GCNN-based RL strategy by ablating the GCNN and replacing it with an MLP.

Experimental Setup: We conduct all our experiments in a 2-dimensional square simulation space $\mathcal{W}$ of size 10×10 sq m. We simulate $M = 60$ wind fields $\boldsymbol{w}$ by solving the Navier-Stokes equations inside $\mathcal{W}$, with random initial conditions. Each $\boldsymbol{w}$ is guaranteed to be in a turbulent regime at $Re \geq 4 \times 10^3$. The turbulence intensifies with time in all of our $\boldsymbol{w}$, increasing the Re value as shown in Fig. 4. We control the maximum possible wind speed in each wind simulation and bound it to a value of 60 m/s. We generate the

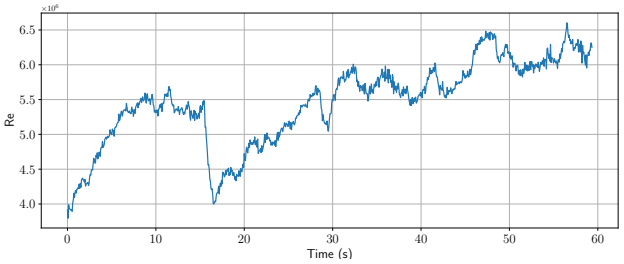


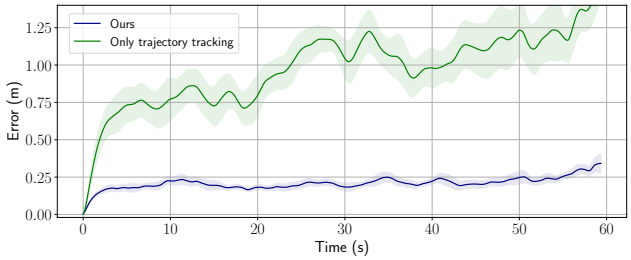
Fig. 4: Reynolds number (Re) evolution. In all of our wind simulations, the value of Re increases as the wind becomes more turbulent.

wind flows using a publicly available Computational Fluid Dynamics (CFD) software [28], [29], [30]. We provide a script to compute simulations along with the project's source code[1]. For each robot, we compute the drag force exerted by the wind as per (3). We set the air density to $\rho = 1.184$ and the drag coefficient to $C_d = 0.47$. Additionally, we assume all of the robots are small spheres of radius $r = 0.1m$ with a cross-sectional area of $A = \pi r^2$. We use lattice formations in all of our experiments at different sizes and chose the lattices' initial location to fit entirely into $\mathcal{W}$.

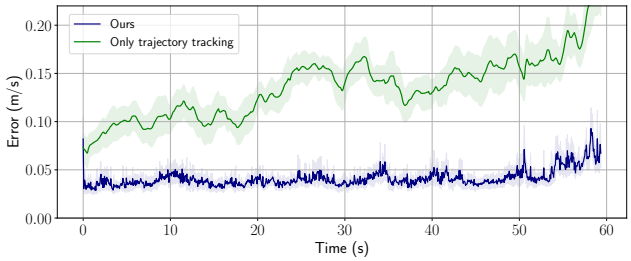
We train all our models on only 50 of the wind simulations and reserve the remaining 10 for testing. We train each RL model for 5×10^6 steps using a replay buffer of 2×10^5. This replay buffer's size ensures the RL model focuses more on recent experiences where the reward is expected to be better. We optimize the SAC's loss functions from (10) and (11) using Adam optimizer [31] with a fix learning rate of 1×10^{-3}. At training, all the episodes have a fixed duration of $T = 60$ s. We set the weights in the reward to $\boldsymbol{\beta} = [1, 1, 10, 10]$. We use the k-nearest neighbor algorithm (knn) to define the graph's adjacency matrix at each time step. In all of our experiments, we start the robot's formation at random locations within $\mathcal{W}$. We report average absolute errors over 20 episodes with corresponding 95% accuracy confidence intervals.

Experiment 1: Wind compensation. In this experiment, we explore the benefits of assisting the trajectory-tracking control from (5) with our RL method to compensate for the force that a turbulent wind field exerts on a robot. To this end, we compute the position and velocity errors at each time t of the trajectory-tracking control with and without the RL wind-compensation strategy. We use a formation size of $n = 25$ robots and a neighborhood size of $k = 12$. We report average errors over 20 episodes and all n robots in the swarm and summarize the results in Fig. 5. Our method (blue curve) shows a statistically significant improvement compared to trajectory tracking only (green curve). Note that our method maintains the position and velocity errors at relatively stable values despite the increase in the turbulence regime described in Fig. 4. From this result, we conclude that our proposed method can capture and compensate for the wind effects that affect the robots, regardless of the intensity and complexity of the wind. Additionally, we report in Fig. 6

[1]https://github.com/dipaco/robot_formation_in_turbulence



(a) Position error



(b) Velocity error

Fig. 5: Our method's performance compared to only the trajectory-tracking control. The curves show the mean error across 20 episodes with corresponding 95% confidence interval.

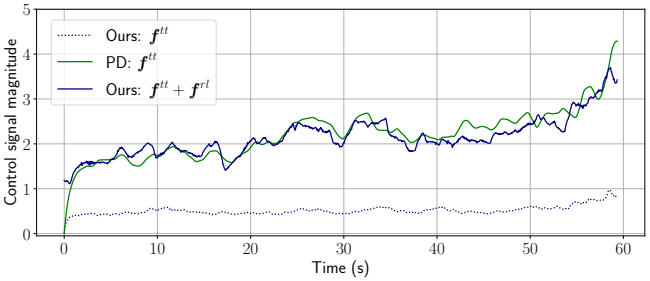


Fig. 6: Magnitude of the control signal. Solid lines show the total action signal for our method (blue) and only the trajectory-tracking control (green). Additionally, we show the tracking-trajectory component of our method (dotted blue).

the magnitude of the total control signal of each robot, averaged over all the robots in the formation. Recall that the total control signal from our proposed method is the sum of trajectory-tracking control and the RL action as per (6). The magnitude of the control signal is associated with the amount of energy the robots use to complete their task. e.g., tracking a trajectory. By comparing the curves in Fig. 6, we conclude that our method achieves significantly lower errors with approximately the same control signal magnitude. Hence, our methods preserve the amount of energy the robots use to fulfill their tasks while achieving better performance. Moreover, we report the trajectory-tracking component of our method (dotted blue) and highlight the smoothness of the curve compared to the trajectory-tracking alone. We conclude that this occurs because the robots can track a target free of perturbations when the RL compensates for the wind's effect.

Experiment 2: Sensitivity Analysis. In this experiment, we test our method's sensitivity regarding two key parameters of our model: the robot's neighborhood size, k, and the number of robots in the team, n.

We investigate the effect of the neighbor size on our method's ability to learn a wind compensation action. To

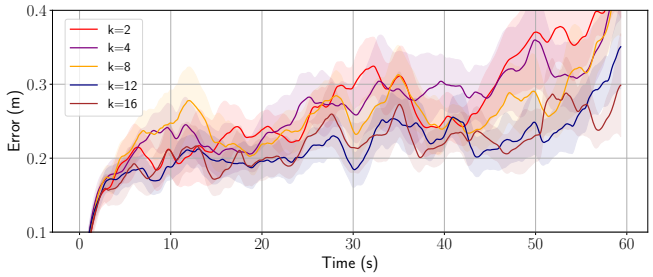


Fig. 7: Sensitivity to the neighborhood size.

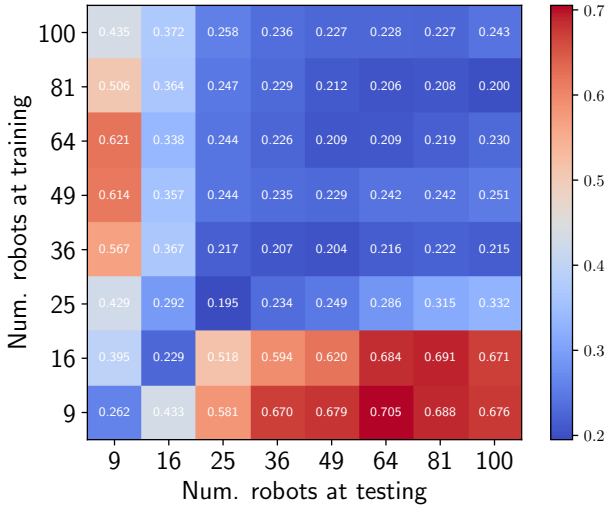


Fig. 8: Sensitivity to the formation size.

this end, we train five models varying the neighborhood size at increasing values of k, such that $k = \{2, 4, 8, 12, 16\}$, and maintain the formation size constant at $n = 25$. We report the average position error of each of these models in Fig. 7. Our results show a decrease in the error when k increases. Note that the error gap between curves with lower values of k and curves with larger k increases with the turbulence intensity (See Fig. 4). We did not observe a significant improvement in performance for models trained with $k > 12$.

We train eight of our RL-based models, varying the training and testing formation size to test our method's sensitivity to the training formation. We use $n^{\text{train}}, n^{\text{test}} \in \{3^2, ..., 10^2\}$ while maintaining the neighborhood size constant at $k = 12$. We report the average position error of each test in Fig. 8. Note that our method scales well to large formation when trained with enough robots without retraining, e.g., $n \geq 25$. The performance decrease in the first two columns results from testing on formations that do not satisfy the neighborhood requirements when training the models, $k = 12$. Similarly, the two first rows in Fig. 8 show a decrease in performance due to training with insufficient robots. In this last scenario, the neighborhood cannot meet the requirements to capture the wind dynamics.

Experiment 3: Ablation Study. We conduct an ablation study to investigate the contribution of our proposed architecture to the overall system. We compare our model with five baselines to highlight the advantages of information sharing in our model. In the first baseline, we replaced the GCNN with an MLP shared across all robots in the team. The MLP

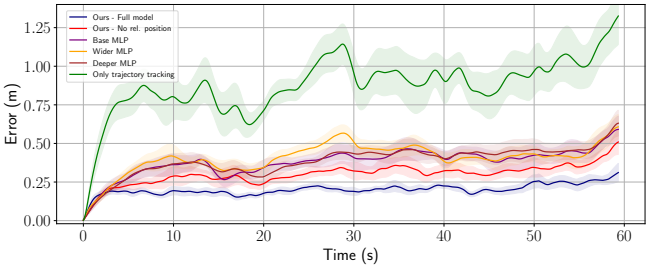


Fig. 9: Ablation study results. Our method (blue curve) shows a statistically significant improvement compared to all baselines.

has the same number of hidden layers and neurons but does not share information with its neighbors. It can only access the features of the nodes in which it is operating. The second baseline is a deeper MLP of four hidden layers. The increase in depth has the effect of approximately doubling the number of weights. Similarly, the third baseline is a wider MLP with a layer width of 128 neurons. Doubling the layer's width increases the number of weights in the base MLP by approximately a factor of four.

We include a fourth baseline to test the ability of our model to learn spatially distributed information from a robot's neighbor. In this baseline, we ablate the inclusion of the relative position $\boldsymbol{r}_{i,j}$ in the convolution definition of (16). By removing the relative position, our GCNN can still share information between a robot i and its neighbors. However, the robot cannot identify where those neighbors are located relative to itself. Finally, the last baseline is the trajectory-tracking controller without our RL wind compensation. Our experiments show that our approach achieves the lowest position error among all methods in the ablation study. We summarize all the ablation experiments in Tab. I and Fig. 9. We report the average position error of each method along the corresponding Re values along an episode. Note that all the MLP-based baselines have similar error curves, despite the significant increase in capacity of the Deeper and Wider MLP. These results demonstrate that the advantages of our method arise from our GCNN-based RL strategy and not from the neural network's size.

Discussion: Navigation in turbulent flows with high levels of turbulence, $Re > 4{\times}10^6$, is a challenging problem. However, these high turbulence levels have not been studied in the state-of-the-art. This scenario is especially challenging for a single robot since its perception of the flow is limited. In this paper, we leveraged multiple robots to navigate high-turbulence flows and evaluate different factors that help understand the difficulties of operation in this type of aggressive environment. Although the spherical robots we presented only exist in simulations, our method can be implemented in actual robots.

V. Conclusion and Future Work

In this paper, we introduced a novel RL-based method to control a team of aerial robots to track a trajectory while working together in a dynamic, turbulent wind field. Our method's strategy decouples the trajectory-tracking controller and wind compensation. So our method can learn to compensate for the wind turbulence independently of the

Method	Position Error						
Time:	$0s$	$10s$	$20s$	$30s$	40	$50s$	$60s$
Re:	3.9×10^6	4.3×10^6	4.4×10^6	5.6×10^6	5.2×10^6	5.3×10^6	6.6×10^6
Base MLP	0.092 ± 0.022	0.362 ± 0.065	0.338 ± 0.048	0.419 ± 0.072	0.414 ± 0.050	0.418 ± 0.073	0.591 ± 0.085
Wider MLP	0.105 ± 0.022	0.418 ± 0.076	0.340 ± 0.046	0.533 ± 0.053	0.405 ± 0.047	0.431 ± 0.051	0.607 ± 0.079
Deeper MLP	$\mathbf{0.089 \pm 0.014}$	0.367 ± 0.068	0.282 ± 0.051	0.438 ± 0.049	0.410 ± 0.039	0.457 ± 0.071	0.631 ± 0.084
Only trajectory tracking	0.282 ± 0.055	0.803 ± 0.156	0.699 ± 0.063	1.020 ± 0.132	0.939 ± 0.131	0.932 ± 0.152	1.325 ± 0.184
Ours - No rel. position	0.092 ± 0.011	0.287 ± 0.058	0.235 ± 0.030	0.325 ± 0.052	0.316 ± 0.043	0.340 ± 0.059	0.509 ± 0.081
Ours - Full model	0.128 ± 0.021	$\mathbf{0.194 \pm 0.027}$	$\mathbf{0.173 \pm 0.014}$	$\mathbf{0.191 \pm 0.025}$	$\mathbf{0.228 \pm 0.017}$	$\mathbf{0.241 \pm 0.041}$	$\mathbf{0.311 \pm 0.059}$

TABLE I: Quantitative results of the ablation study. The table shows the average position error along the duration of an episode. We report the average Re at selected times.

motion controller. Our RL approach allowed us to find an optimal policy to compensate for the wind force via a graph neural network designed to share information among the robotic team members. Our method shows that sharing sensor measurements between nearby robots provides valuable information to improve the robots' turbulence compensation and learn spatially-distributed wind patterns. We demonstrate the advantages of our strategy through several simulations strategically designed to test our method's performance for wind compensation, its scalability to large robot formations, and its parameter sensitivity.

In future work, we want to design and implement a lab testbed to generate air flows with high turbulence levels like the ones presented in this paper. Although this type of testbed has a high cost and complexity, it would allow us to test and extend methods for navigation in high turbulence.

REFERENCES

[1] B. Garau, A. Alvarez, and G. Oliver, "Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an a* approach," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 194–198.

[2] E. Bakolas and P. Tsiotras, "Time-optimal synthesis for the zermelo-markov-dubins problem: the constant wind case," in *Proceedings of the 2010 American Control Conference*. IEEE, 2010, pp. 6163–6168.

[3] M. Otte, W. Silva, and E. Frew, "Any-time path-planning: Time-varying wind field + moving obstacles," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2575–2582.

[4] M. Bisheban and T. Lee, "Geometric adaptive control for a quadrotor uav with wind disturbance rejection," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 2816–2821.

[5] C. Montella and J. R. Spletzer, "Reinforcement learning for autonomous dynamic soaring in shear winds," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3423–3428.

[6] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 128–138, 2020.

[7] C. Foias, O. Manley, R. Rosa, and R. Temam, *Navier-Stokes Equations and Turbulence*, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2001.

[8] E. Zermelo, "Über das navigationsproblem bei ruhender oder veränderlicher windverteilung," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 11, no. 2, pp. 114–124, 1931.

[9] D. Chang, W. Wu, C. R. Edwards, and F. Zhang, "Motion tomography: Mapping flow fields using autonomous underwater vehicles," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 320–336, 2017.

[10] S. Allison, H. Bai, and B. Jayaraman, "Estimating wind velocity with a neural network using quadcopter trajectories," in *AIAA Scitech 2019 Forum*, 2019, p. 1596.

[11] J. Escareño, S. Salazar, H. Romero, and R. Lozano, "Trajectory control of a quadrotor subject to 2d wind disturbances," *Journal of Intelligent & Robotic Systems*, vol. 70, no. 1, pp. 51–63, 2013.

[12] L. Rodriguez Salazar, J. A. Cobano, and A. Ollero, "Small uas-based wind feature identification system part 1: Integration and validation," *Sensors*, vol. 17, no. 1, p. 8, 2017.

[13] G. W. Donnell, J. A. Feight, N. Lannan, and J. D. Jacob, "Wind characterization using onboard imu of suas," in *2018 Atmospheric Flight Mechanics Conference*, 2018, p. 2986.

[14] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2460–2465.

[15] L. Ruiz, F. Gama, and A. Ribeiro, "Graph neural networks: architectures, stability, and transferability," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 660–682, 2021.

[16] J. Blumenkamp, S. Morad, J. Gielis, Q. Li, and A. Prorok, "A framework for real-world multi-robot systems running decentralized gnn-based policies," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8772–8778.

[17] W. Gosrich, S. Mayya, R. Li, J. Paulos, M. Yim, A. Ribeiro, and V. Kumar, "Coverage control in multi-robot systems via graph neural networks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8787–8793.

[18] S. Rajappa, M. Ryll, H. H. Bülthoff, and A. Franchi, "Modeling, control and design optimization for a fully-actuated hexarotor aerial vehicle with tilted propellers," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4006–4013.

[19] T. Lee, M. Leoky, and N. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," *49th IEEE Conference on Decision and Control (CDC), 2010.*, pp. 5420–5425, 2010.

[20] E. T. Gilbert-Kawai and M. D. Wittenberg, *Reynold's number (and turbulent flow)*. Cambridge University Press, 2014, p. 21–23.

[21] C. K. Batchelor and G. Batchelor, *An introduction to fluid dynamics*. Cambridge university press, 2000.

[22] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2520–2525, 2011.

[23] H. Khalil, *Nonlinear Systems*, ser. Pearson Education. Prentice Hall, 2002.

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[25] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *ArXiv*, vol. abs/1812.11103, 2019.

[26] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014.

[27] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *AAAI*, 2019.

[28] P. Augier, A. V. Mohanan, and C. Bonamy, "FluidDyn: A python open-source framework for research and teaching in fluid dynamics by simulations, experiments and data processing," *Journal of Open Research Software*, vol. 7, 2019.

[29] A. V. Mohanan, C. Bonamy, and P. Augier, "FluidFFT: Common API (c++ and python) for fast fourier transform HPC libraries," *Journal of Open Research Software*, vol. 7, 2019.

[30] A. V. Mohanan, C. Bonamy, M. C. Linares, and P. Augier, "FluidSim: Modular, Object-Oriented Python Package for High-Performance CFD Simulations," *Journal of Open Research Software*, vol. 7, 2019.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.