IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Dynamically Feasible Trajectory Generation With Optimization-Embedded Networks for Autonomous Flight

Zhichao Han, Long Xu, *Graduate Student Member, IEEE*, Liuao Pei, *Member, IEEE*, and Fei Gao, *Member, IEEE*

***Abstract*—This paper aims to bridge perception and planning in navigation systems by learning optimal trajectories from depth information in an end-to-end fashion. However, using neural networks as closed-box replacements for traditional modules risks scalability and adaptability. Moreover, such methods often fall short in sufficiently incorporating the robot's dynamic constraints, resulting in trajectories that are either inadequately executable or unexpectedly aggressive, diverging from user expectations. In this paper, we fuse the benefits of conventional methods and neural networks by introducing an optimization-embedded network based on a compact trajectory library. The network distills spatial constraints, which are then applied to model-based spatial-temporal trajectory optimization problem, yielding feasible and optimal solutions. By making the optimization problem differentiable, our model seamlessly approximates the optimal trajectory. Additionally, the introduced regularized trajectory library permits efficient capture of the spatial distribution of optimal trajectories with minimal storage cost, safeguarding multimodal planning features. Benchmarking demonstrates the outstanding performance of our method in trajectory smoothness, success rate, and constraint satisfaction. Real-world flight experiments with an onboard computer showcase the autonomous quadrotor's ability to navigate swiftly through dense forests.**

***Index Terms*—Motion and path planning, integrated planning and learning, aerial systems: applications.**

I. Introduction

THE navigation module plays a pivotal role in the autonomous operation of uncrewed aerial vehicles (UAVs), attracting considerable attention from industry and academia. Traditionally, these modules rely on sensors like depth cameras to perceive the environment, creating occupancy maps and generating environment representations for motion planning, such as Euclidean signed distance fields (ESDF) [1]. Subsequently, motion planning algorithms are applied to compute optimal trajectories that consider initial and final states, along with obstacle avoidance. While this modular approach seems logical in engineering terms, it introduces physical delays and often lacks integration among sub-modules, necessitating extensive manual parameter tuning. In recent times, learning-based navigation [2], [3], [4] has garnered significant interest for its seamless fusion of perception and planning modules. This holistic approach generates trajectories directly from raw sensor data, eliminating the need for explicit mapping. Nevertheless, this methodology heavily leans on the neural network's proficiency, resulting in a closed-box system that presents challenges for debugging and limits the system's adaptability and interpretability. Furthermore, due to task demands or inherent platform constraints, there is a need to impose various custom constraints on the trajectory, such as dynamic constraints, which pose challenges to the network. While researchers often devise sophisticated strategies to address these constraints, these approaches may occasionally compromise optimality or fail to ensure adequate constraint satisfaction, impacting the overall system's completeness.

To address the limitations of existing methods, this work integrates trajectory optimization with neural networks to create an end-to-end visual navigation system that directly generates dynamically feasible spatial-temporal optimal trajectories from depth images. Unlike conventional learning-based methods, we model a spatial-temporal trajectory optimization problem based on a lightweight trajectory representation [5]. Then, we apply implicit function theory [6]—a method for deriving the relationship between the optimal solution and optimization parameters—to enable the differentiability of the optimization process for coupled training. This method not only ensures optimality, but also enhances interpretability and kinematic feasibility. Specifically, rather than naively producing a series of trajectory points, our neural network extracts safe regions from depth images which are subsequently used to construct safety constraints for the trajectory optimization problem. The optimization problem is then efficiently solved (∼1 ms) to obtain a high-quality, feasible trajectory. By differentiating the optimization process, we can directly backpropagate the trajectory evaluation loss, which directs the network's focus towards regions for optimal trajectories. Furthermore, the introduction of

Received 7 July 2025; accepted 19 July 2025. Date of publication 1 August 2025; date of current version 21 August 2025. This article was recommended for publication by Associate Editor and Editor A. Bera upon evaluation of the reviewers' comments. This work was supported by the "Pioneer" and "Leading Goose" R&D Program of Zhejiang under Grant 2024C01170. *(Zhichao Han and Long Xu contributed equally to this work.) (Corresponding author: Fei Gao.)*

The authors are with the State Key Laboratory of Industrial Control Technology, Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China, and also with the Huzhou Institute of Zhejiang University, Huzhou 313000, China (e-mail: zhichaohan@zju.edu.cn; fgaoaa@zju.edu.cn).

Our project page with videos is at https://zju-fast-lab.github.io/e2e_opt/.

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2025.3595077, provided by the authors.

Digital Object Identifier 10.1109/LRA.2025.3595077

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

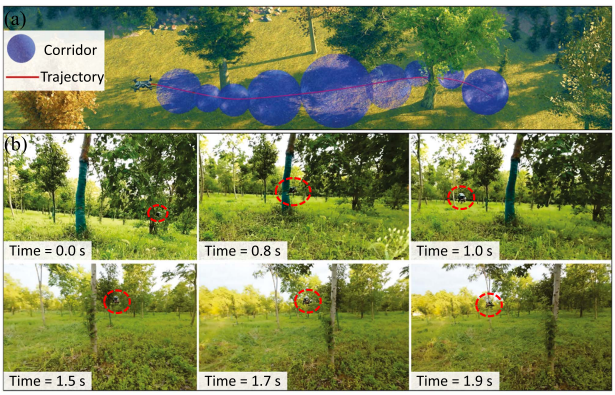


Fig. 1. Quadrotor flight in forests. The top figure illustrates the visualization of the trajectory and corridors. The bottom figure showcases the real high-speed autonomous flight of the quadrotor in the wild.

model-based optimization enhances the scalability and adaptability. For instance, our method can easily be integrated with upstream decision modules to adjust robot behavior online by tweaking optimization parameters without additional training [7].

To fully explore environments and align the multimodal nature of local motion planning, our network outputs a mixture distribution over a regularized lightweight motion primitive library [8], which is represented by a collection of pre-defined local trajectory segments. Based on the probabilities, specific motion primitives are selected and allocated safe feasible spaces, which are subsequently input into the optimization module to generate optimal motion. Fig. 1(a) illustrates the flight corridor-based feasible space [9] and the optimized trajectory. We conduct extensive comparative experiments against various state-of-the-art algorithms across multiple scenarios, including both traditional and learning-based methods. The results demonstrate that our method excels in success rate, smoothness, and constraint satisfaction. Moreover, as shown in Fig. 1(b), we validate our method's practical applicability through real-world tests on a fully autonomous physical platform without relying on external perception and localization. Overall, the main contributions of this paper can be summarized as follows:

- Based on a regularized motion primitive library, we propose a specialized neural network considering the multimodal characteristics of local planning to extract safe guiding regions within the environment, which are then formalized as spatial constraints essential for subsequent optimization problem.
- Based on the compact trajectory representation, we model a spatial-temporal trajectory optimization problem and innovatively embed it differentiably into neural networks for joint training. This design enhances scalability and interpretability while fundamentally ensuring the feasibility of motion kinematics.
- We integrate the strengths of neural networks and numerical optimization to propose a novel end-to-end planning framework. We perform comprehensive simulations to showcase the superiority of our approach. Furthermore, we implement our algorithm on a fully autonomous drone and conduct real-world experiments in intricate environments to validate its practicality.

II. Related Work

A. Classical Motion Planning for UAV

As mainstream approaches for UAV local planning, optimization-based methods [10], [11] leverage gradient information to efficiently converge to feasible trajectories in continuous spaces, balancing quality and time. These methods often require explicit environmental modeling through depth information and manual safety constraint extraction, for example, using ESDF to construct safety constraints [10]. However, constructing ESDF incurs additional computational costs and involves a trade-off between efficiency and accuracy. The ego planner [11] avoids ESDF construction by iteratively generating safe guidance paths for obstacle avoidance gradients, but it lacks convergence guarantees and may get trapped in unsafe local minima in complex environments.

B. Learning-Based Motion Planning for UAV

Learning-based methods [3], [4], [12], [13], [14], [15] emerge as promising approaches in local planning, eliminating the need for explicit mapping and reducing latency. Some methods [3], [4], [12] directly employ neural networks to learn trajectories from observations, relying on supervision from ground truth or predefined losses based on expert knowledge. However, these methods lack theoretical guarantees and may produce trajectories that are kinematically infeasible. Recently, similar to our work, some works [13], [14], [15] combine optimization with neural networks, leveraging the concept of bilayer optimization. Wu et al. [13] utilize networks to learn time allocation for piecewise polynomial trajectories, achieving optimal solutions under ideal conditions. This method, however, requires a known global map and offline convex decomposition of safe regions, making it impractical for vision-based end-to-end local planners. Iplanner [14] is an end-to-end local planner that generates trajectories from depth images. This approach uses closed-form cubic splines to interpolate the points output by the network, approximating a feasible solution to the original trajectory planning problem. Despite being lightweight, this approximation does not strictly guarantee adherence to dynamic constraints, potentially resulting in trajectories that are difficult for the physical platform to execute during high-speed flight, leading to crashes. BarrierNet [15] differentiates the control problem based on barrier functions and is utilized for end-to-end self-driving. However, this method relies on predefined rules and offline fixation of obstacle size and shape, making it suitable for structured vehicle navigation but unsuitable for the unstructured and cluttered flight scenarios we focus on.

III. Spatial-Temporal Trajectory Optimization Formulation

The objective of visual navigation is to find a dynamically feasible trajectory within a safe region $\mathcal{E}_{free}$, guided by depth observations $\boldsymbol{D} \in \mathbb{R}^{H \times W}$, while satisfying initial and terminal

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

state constraints. Based on differential flatness for UAVs [16], we formulate the trajectory planning problem in the flat output space. Consequently, we only need to optimize the flat outputs (center of mass position) and their derivatives, as all other states can be analytically derived from them. Moreover, this modeling approach extends beyond UAVs to other differentially flat systems, such as Ackermann-steered vehicles [17] and fixed-wing aircraft [18]. The trajectory $\xi_{\boldsymbol{Q},T}(t):[0,T]\rightarrow\mathbb{R}^3$ is represented using time-uniform MINCO [5], a special time-uniform piecewise polynomial that adheres to the principle of minimum energy, and is parameterized by trajectory duration $T\in\mathbb{R}^+$ and a series of waypoints $\boldsymbol{Q}=[\boldsymbol{q}_1,\ldots,\boldsymbol{q}_{N-1}]\in\mathbb{R}^{3\times(N-1)}$. N is the number of pieces of the trajectory. With the trajectory representation based on MINCO [5], our parameter space achieves a lower dimensionality compared to traditional piecewise polynomial formulations, while naturally fulfilling initial and terminal state conditions and ensuring high-order continuity at waypoints between adjacent polynomials. These attributes provide a solid foundation for smooth and coherent motion, simultaneously improving the efficiency of trajectory optimization. Subsequently, as shown in Fig. 1. A, we employ flight corridors [9] to model safety constraints, a widely adopted technique in the field of trajectory planning. These flight corridors are typically represented as a sequence of free convex geometries, with safety ensured by constraining the trajectory to remain within these geometries. Consequently, we formulate the bilayer optimization model as follows:

$$\min_{\phi} J = J(\xi^*_{\boldsymbol{Q}^*,T^*}(t)) \tag{1}$$

$$s.t.\quad \mathcal{F}_\phi \in \mathcal{E}_{free}, \tag{2}$$

$$\xi^*_{\boldsymbol{Q}^*,T^*}(t) = \arg\min_{\xi^*_{\boldsymbol{Q},T}(t)\in\mathbb{F}(\phi)} J = J(\xi^*_{\boldsymbol{Q},T}(t)), \tag{3}$$

where ϕ is the parameters of the neural network and $\xi^*_{\boldsymbol{Q}^*,T^*}(t)$ denotes the optimal trajectory derived from the objective J within the constraint set $\mathbb{F}(\phi)$. To improve readability, we omit the subscripts used to parameterize $\xi(t)$ later. J is represented as the control energy and includes a first-order time regularization:

$$J=\int_0^T (\xi^{(u)})^{\mathrm{T}}\xi^{(u)}dt+\rho T, \tag{4}$$

where $\rho\in\mathbb{R}^+$ is the time regularization weight and $u=3$ denotes the derivative order of the control input. In this paper, $\mathcal{F}_\phi$ is modeled as spheres with centers at ζ_ϕ and radii γ_ϕ, which enables straightforward analytical modeling of the safety condition (2), as elaborated in Section V-A. Then, the constraint set $\mathbb{F}(\phi)$ is instantiated as follows:

$$\mathcal{C}(\xi(t),\xi^{(1)}(t),\ldots,\xi^{(u)}(t))\leq 0,\forall t\in[0,T], \tag{5}$$

$$||\xi(t)-\zeta_\phi(t)||_2^2\leq\gamma_\phi^2(t),\forall t\in[0,T]. \tag{6}$$

$\mathcal{C}$ generally refers to various user-defined constraints tailored to specific task scenarios, such as constraints on velocity, acceleration, thrust, torque, and others. Equation (6) serves as a spatial constraint to confine the trajectory within the safe corridors.

This work proposes to leverage neural networks to obtain safety constraints, upon which a differentiable trajectory optimization is performed to generate a feasible, high-quality trajectory. Intuitively, the model-free neural network undertakes the outer optimization problem, extracting the corridor from the depth information, while the model-based trajectory planning performs the inner optimization problem, determining the spatial-temporal optimal trajectory. After obtaining the trajectory, we propagate the gradient related to trajectory quality back to the corridor network through the differentiable trajectory optimization layer, thereby updating the network's parameters. This synergistic process of network updating and trajectory optimization forms a nested, coupled bilevel optimization framework, enabling the network to directly learn the safe regions that are most conducive to high-quality trajectory generation.

IV. End-to-End Trajectory Generation

The planning pipeline is illustrated in Fig. 2. Firstly, the network selects a primitive from the offline established primitive library based on multiple frames of depth images and start-end information, and refines it to obtain a flight corridor. Subsequently, the differentiable trajectory optimization layer models the flight corridor as a spatial constraint for further optimization to obtain a spatio-temporally optimal trajectory. The loss measuring the quality of the trajectory is back-propagated to the network for learning.

A. Neural Network

Firstly, we instantiate the network's output representation. Addressing the time-continuous constraint (5), (6), we discretize each piece of the piecewise polynomial trajectory into λ constraint points and impose constraints at these points to approximately ensure that the original constraints are satisfied. This technique has been demonstrated to be practically effective in numerous studies [5], [17], [19], [20].

$$\begin{aligned}&\mathcal{C}(\xi(t),\xi^{(1)}(t),\ldots,\xi^{(u)}(t))\leq 0,\\&||\xi(t)-\zeta_\phi(t)||_2^2\leq\gamma_\phi^2(t),\forall t\in[0,T]\Longleftrightarrow\\&\mathcal{C}\left(\xi_i\left(\frac{jT}{\lambda N}\right),\xi_i^{(1)}\left(\frac{jT}{\lambda N}\right),\ldots,\xi_i^{(u)}\left(\frac{jT}{\lambda N}\right)\right)\leq 0,\end{aligned} \tag{7}$$

$$||\xi_i\left(\frac{jT}{\lambda N}\right)-\zeta_{\phi,i,j}||_2^2\leq\gamma_{\phi,i,j}^2,\forall i\in\{1,\ldots,N\},\forall j\in\{1,\ldots,\lambda\}. \tag{8}$$

Equation (8) implies that the network needs to assign a safety sphere, parameterized by $\zeta_{\phi,i,j},\gamma_{\phi,i,j}$ to each constraint point $\xi_{i,j}=\xi_i(\frac{jT}{\lambda N})$ along the trajectory, resulting in $N\lambda$ spheres.

In this paper, we design a specialized motion primitive-based network architecture π_ϕ that effectively estimates the optimal trajectory's mixture distribution within the workspace and further refines the corridors. To avoid overfitting due to absolute world frame coordinates, we model the trajectory planning under the robot's current local coordinate system. Moreover, for more stable training, we preprocess image depth values by clipping and normalizing them to a range of 0 to 1, setting the maximum

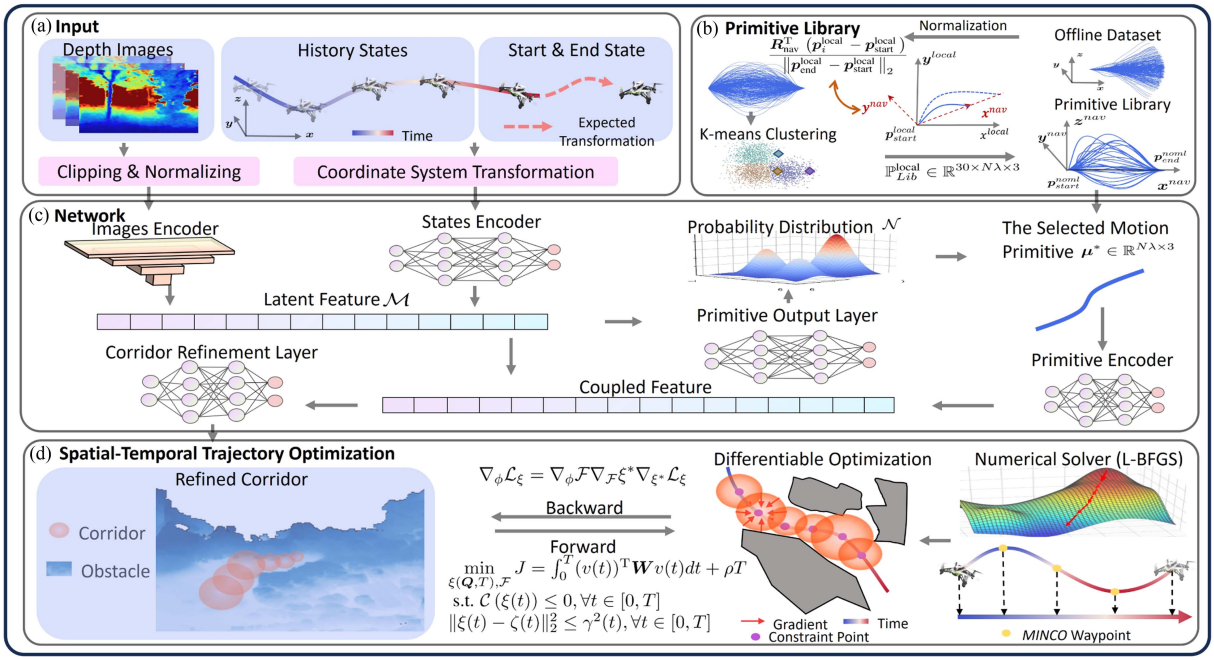


Fig. 2. Planning pipeline. Subfigure a illustrates the preprocessing of input data. Subfigure b presents the construction process of the trajectory library, including regularization and clustering purification of offline trajectories. Subfigure c showcases the network structure, which first selects a suitable primitive to generate a corridor. Subfigure d depicts the trajectory optimization based on the flight corridor, which has been made differentiable for the backpropagation of the gradient.

depth value at 10 meters, as shown in Fig. 2. A. Our network uses residual convolutions to encode 10 depth image frames and multilayer perceptrons (MLPs) to encode the start and end states along with past 10 odometry frames. The outputs are then concatenated to form a latent feature, denoted as $\mathcal{M}$. Assuming we have pre-constructed an offline motion primitive library $\mathbb{M}$ that represents the spatial topology, the latent feature will then be fed into another MLP to output a probability distribution $\mathcal{N}$ over the library. Each motion primitive $\mu \in \mathbb{M}$ is represented by $N\lambda$ points to align with subsequent corridor parameters. To enhance the accuracy, the selected motion primitive μ^* is further coupled with the latent features $\mathcal{M}$ and fed into the final corridor refinement layer, which adjusts each point's position on the motion primitive and assigns the corresponding safety radius to each sphere. Next, we discuss the method of constructing the motion primitive library.

We uniformly discretize the offline-generated UAV trajectories (approximately 300,000) into $N\lambda$ points in the robot's local coordinate system: $\boldsymbol{P}^{local} = [\boldsymbol{p}_1^{local}, \ldots, \boldsymbol{p}_{N\lambda}^{local}] \in \mathbb{R}^{N\lambda\times 3}$. To remove redundant motion primitives and limit the library size, as shown in Fig. 2. B, we define a navigation frame $\boldsymbol{R}_{nav} = [\boldsymbol{x}^{nav}, \boldsymbol{y}^{nav}, \boldsymbol{z}^{nav}] \in \mathbb{R}^{3\times 3}$ based on the body frame, with its origin coinciding with the start point of the body frame and the X-axis pointing towards the endpoint position:

$$\boldsymbol{x}^{nav} = \frac{\boldsymbol{p}_{end}^{local} - \boldsymbol{p}_{start}^{local}}{||\boldsymbol{p}_{end}^{local} - \boldsymbol{p}_{start}^{local}||_2}, \tag{9}$$

$$\boldsymbol{y}^{nav} = \frac{\boldsymbol{e}_3 \times \boldsymbol{x}^{nav}}{||\boldsymbol{e}_3 \times \boldsymbol{x}^{nav}||_2}, \tag{10}$$

$$\boldsymbol{z}^{nav} = \boldsymbol{x}^{nav} \times \boldsymbol{y}^{nav}, \tag{11}$$

where $\boldsymbol{e}_3 = [0,0,1]^{\mathrm{T}}$. p_{start}^{local} and p_{end}^{local} represent the start and end points in the robot's local coordinate system, respectively. By transferring the motion primitives from the local body frame to the navigation frame, we achieve regularization in the direction towards the endpoint. Furthermore, to avoid a large number of redundant motion primitives that are actually very similar in shape but differ in spatial scale during planning, we also normalize the distance to the endpoint. The resulting processed motion primitives $\boldsymbol{P}^{normalized}$ are as follows:

$$\boldsymbol{p}_i^{normalized} = \frac{\boldsymbol{R}_{nav}^{\mathrm{T}}(\boldsymbol{p}_i^{local} - \boldsymbol{p}_{start}^{local})}{||\boldsymbol{p}_{end}^{local} - \boldsymbol{p}_{start}^{local}||_2}, \forall i \in \{1, \ldots, N\lambda\}. \tag{12}$$

Having obtained $M \approx 300,000$ normalized trajectories $\boldsymbol{P}^{normalized}$, our next objective is to downsample these into m elite trajectories while preserving the spatial distribution characteristics of the original set as closely as possible. Here, we employ the K-Means clustering algorithm [21], a well-established machine learning method for partitioning data into representative clusters. Briefly, K-Means iteratively assigns each trajectory to one of m clusters based on its proximity to the cluster centroid and updates these centroids to minimize within-cluster variance. Since each trajectory is represented by $N\lambda$ points, we define the distance between trajectories as the average Euclidean distance between their corresponding points. In our work, we set $m = 30$, yielding 30 cluster centroids that correspond to the elite trajectories forming the motion primitive library.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

B. Differentiable Trajectory Optimization

Before delving into gradient backpropagation, we first reshape the inner optimization problem in (3) and outline its solution method (forward process). Drawing from the approach [17], we ease the original time-continuous constraint using a time-discrete penalty term and transform the initial trajectory planning problem into an unconstrained nonlinear optimization problem:

$$\min_{\xi_{\boldsymbol{Q},T}(t)} L_{\mathcal{F}(\zeta_\phi,\gamma_\phi)} = \int_0^T (\xi^{(u)}(t))^{\mathrm{T}} \xi^{(u)}(t) dt + \rho T + \sum_{i=1}^{N}\sum_{j=1}^{\lambda} w_{\mathcal{C}} \mathrm{L}_1(\mathcal{C}) + w_{\mathcal{F}} \mathrm{L}_1(||\xi_{i,j} - \zeta_{\phi,i,j}||_2^2 - \gamma_{\phi,i,j}^2). \quad (13)$$

Here, $w_{\mathcal{C}}$ and $w_{\mathcal{F}}$ signify the weights of the penalty terms. $||\xi_{i,j} - \zeta_{\phi,i,j}||_2^2 - \gamma_{\phi,i,j}^2$ is the violation of trajectory points within the spatial corridor constraints, simplified as $\mathcal{S}_{i,j}$. $\mathrm{L}_1(\cdot)$ is a first-order relaxation function to guarantee the continuous differentiability and non-negativity of penalty terms [17]. Then, the reformulated problem can be robustly solved by L-BFGS [22].

Defining ξ^* is the optimal solution to (13), and J_ξ is the evaluation loss applied to the trajectory during training, the gradient of the neural network can be computed:

$$\nabla_\phi J_\xi = \nabla_\phi \mathcal{F} \nabla_{\mathcal{F}} \xi^* \nabla_{\xi^*} J_\xi, \quad (14)$$

where all gradient derivations adhere to the denominator layout. Generally, $\nabla_{\xi^*} J_\xi$ can be analytically computed and $\nabla_\phi \mathcal{F}$ is also easily computed using automatic differentiation based on the network structure. Therefore, our main focus is on computing the gradient of the optimal solution with respect to the spatial constraint descriptors $\nabla_{\mathcal{F}} \xi^*$. Due to the use of gradient-based numerical solvers, an intuitive method for estimating parameter gradients, known as unrolling [23], involves maintaining the entire computational graph throughout the iteration process. However, this approach presents significant challenges in terms of memory usage and efficiency. Additionally, it may also face issues related to gradient divergence or vanishing [24]. In this work, since we have already obtained the optimal solution ξ^*, we use the implicit function theory [6] to analytically derive the gradients without the need for explicit unrolling of the entire iteration process. Based on the first-order optimality condition of nonlinear programming [25], the optimal solution of (13) should satisfy the equation:

$$\nabla_\xi L_{\mathcal{F}(\zeta_\phi,\gamma_\phi)}(\xi^*) = 0. \quad (15)$$

Then, we apply the total differential operator to this first-order optimality condition (15):

$$(\nabla_{\xi,\xi} L_{\mathcal{F}(\zeta_\phi,\gamma_\phi)}(\xi^*))^{\mathrm{T}} d\xi^* + (\nabla_{\xi,\zeta_\phi} L_{\mathcal{F}(\zeta_\phi,\gamma_\phi)}(\xi^*))^{\mathrm{T}} d\zeta_\phi + (\nabla_{\xi,\gamma_\phi} L_{\mathcal{F}(\zeta_\phi,\gamma_\phi)}(\xi^*))^{\mathrm{T}} d\gamma_\phi = 0, \quad (16)$$

where each term here can be analytically derived. Subsequently, through matrix operations, we equivalently transform (16) into the following compact form:

$$d\xi^* = (\nabla_{\xi,\xi} L_{\mathcal{F}}(\xi^*))^{-1} \begin{bmatrix} \nabla_{\xi,\zeta_\phi} L_{\mathcal{F}}(\xi^*) \\ \nabla_{\xi,\gamma_\phi} L_{\mathcal{F}}(\xi^*) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} d\zeta_\phi \\ d\gamma_\phi \end{bmatrix}, \quad (17)$$

where $L_{\mathcal{F}} = L_{\mathcal{F}(\zeta_\phi,\gamma_\phi)}$ for brevity. By solving this system of equations, we can analytically obtain the desired Jacobian matrix $\nabla_{\mathcal{F}(\zeta_\phi,\gamma_\phi)} \xi^*$ in (14), which in turn allows us to derive the final parameter gradients $\nabla_\phi J_\xi$:

$$\nabla_\phi J_\xi = \nabla_\phi \mathcal{F} \begin{bmatrix} \nabla_{\xi,\zeta_\phi} L_{\mathcal{F}}(\xi^*) \\ \nabla_{\xi,\gamma_\phi} L_{\mathcal{F}}(\xi^*) \end{bmatrix} (\nabla_{\xi,\xi} L_{\mathcal{F}}(\xi^*))^{-1} \nabla_{\xi^*} J_\xi. \quad (18)$$

V. Implementation Details

A. Loss Function

The training loss function is defined as the expected loss form of the flight corridor policy: $\overline{\mathcal{L}}_\phi = \mathbb{E}_{\mathcal{F}\sim P_\phi(\mathcal{F})}[\mathcal{L}(\mathcal{F})] = \sum_{\mathcal{F}} \mathcal{L}(\mathcal{F}) P_\phi(\mathcal{F})$. As each corridor is uniquely generated by its corresponding primitive, the probability distribution of the corridor is equivalent to the primitive's probability distribution. Hence, the expected loss is instantiated as follows:

$$\overline{\mathcal{L}}_\phi = \sum_{\mu\in\mathbb{M}} P_\phi(\mu) \mathcal{L}(\mathcal{F}(\mu)), \quad (19)$$

$$\mathcal{L}(\mathcal{F}(\mu)) = J(\xi^*(\mathcal{F}(\mu))) + w_s \mathcal{L}_{sf}(\mathcal{F}(\mu)) + w_f \mathcal{L}_{feas}(\mathcal{F}(\mu)). \quad (20)$$

Here, w_s and w_f are the weights corresponding to the respective losses. J is the evaluation metric of the trajectory optimized based on the flight corridor $\mathcal{F}$, defined as a combination of energy and execution time, as described in (4). $\mathcal{L}_{sf}$ represents the safety condition (2) of the corridor, acting as a penalty term during training to push the corridor away from obstacle regions:

$$\mathcal{L}_{sf}(\mathcal{F}) = \sum_{i=1}^{N}\sum_{j=1}^{\lambda} \mathrm{L}_1(\gamma_{i,j} - \mathcal{S}_d(\zeta_{i,j}, \epsilon)), \quad (21)$$

where $\mathcal{S}_d$ is the signed distance from the sphere center to the obstacle, efficiently derived from a precomputed ESDF field ϵ. To prevent the generation of unreasonable safety corridors by the network's initial parameters, which could render the inner optimization problem infeasible, we introduce $\mathcal{L}_{feas}$ to penalize such infeasible corridors.

Upon obtaining the inner optimization output, we check the corridor constraints (6) to assess the degree of violation. If the constraint violation exceeds a predetermined threshold δ, the output corridor is deemed infeasible and a penalty is applied. To mitigate the infeasibility of the corridor, we adjust the sphere by moving its center closer to the detached constraint point $\hat{\xi}_{i,j}$ and appropriately increasing its radius:

$$\mathcal{L}_{feas}(\mathcal{F}) = \sum_{i=1}^{N}\sum_{j=1}^{\lambda} L_1(-\delta + ||\hat{\xi_{i,j}} - \zeta_{i,j}||^2 - \gamma_{i,j}^2). \quad (22)$$

Starting training directly on the original task (18) might steer the network towards unfavorable minima or saddle points. To resolve this, we utilize a curriculum learning strategy, progressively guiding the network in an orderly manner. Initially, we

separate the network's layers, training each on their respective outputs. The primitive output layer calculates the cross-entropy loss between the output and the ground truth labels. Conversely, during training, the corridor refinement layer doesn't select the network's highest probability primitive. Instead, it inputs the ground truth primitive μ^* directly into the model, supervising the resulting corridor's centers and radii by the reference trajectory's constraint points and the signed distance to obstacles, respectively. The losses from both layers are then combined for backpropagation and parameter updates. After convergence, we reconnect the layers for further refinement using $\overline{\mathcal{L}}_\phi$.

B. Data Collection

We generate hundreds of random forest environments and random start-end points to construct navigation tasks. For data collection, we apply a high-resolution hybrid A* frontend to find an initial topological path for backend optimization. Moreover, we maintain a precise ESDF in real time for obstacle avoidance. The local ESDF is set to cover 14 meters forward and backward in the local coordinate system to enhance the trajectory foresight. However, maintaining a large ESDF leads to a significant navigation latency of a few hundred milliseconds, making it unsuitable for high-speed UAV flight scenarios. Therefore, we deliberately adjust the simulator's time to 1/10 of real-time. Although this method reduces data collection efficiency, the server's excellent multi-threading capabilities allow us to collect data in large batches. Even with a target of 400000 data, collection time remains within 10 hours. Due to the potential of falling into infeasible local optima, the collected local trajectories may not always be safe. However, through our unsupervised loss functions and gradually transitioning to a reference-free training strategy, we reduce our method's dependence on reference trajectories and enhance robustness.

VI. Evaluations

We execute all training procedures on an Nvidia RTX 4090 GPU with a batch size of 64, utilizing the Adam optimizer with a learning rate of 1.0e-5. Subsequently, all testing within the simulation environment is conducted on an RTX 2060 GPU, an Intel 10700 CPU, and an Ubuntu 20.04 operating system. For real-world experiments, we deploy them on a fully autonomous quadcopter equipped with a RealSense D430 camera and NVIDIA Jetson Orin NX.

A. Benchmarks

In this section, we compare our approach with two traditional algorithms and a SOTA learning-based method: **1. Fast** [10]: A hierarchical motion planner where kinodynamic search is used to construct an initial solution, followed by using ESDF to enforce safety constraints required for subsequent trajectory optimization. **2. Ego** [11]: A lightweight optimization-based motion planner that utilizes graph search to obtain a collision-free guiding path, guiding trajectories to safe regions, thus eliminating the need for ESDF construction. **3. Iplanner** [14]: A powerful learning-based motion planner that has been validated in numerous real-world experiments. For a comprehensive comparison, we categorize scenarios into low, medium, and high aggressiveness based on varying velocity and acceleration limits of the robot. In the simulation, the map size is set to 50 m $\times$ 50 m, with an obstacle density of approximately 45%. Each environment includes 230 cylindrical obstacles with a radius ranging from 1.2 m to 1.3 m, and 30 circle obstacles with a radius ranging from 0.5 m to 0.7 m. For each scenario, we randomly generate 200 navigation tasks, each approximately 70 meters in distance, within previously unseen environments, as illustrated in Fig. 3(a). Furthermore, for each scenario, the learning-based IPlanner is retrained to achieve optimal performance. For each planning instance, the local target is selected from the straight line pointing towards the destination at a specific distance (12 m) from the current position. The replanning logic for each planner follows a time-triggered approach, with a fixed replanning frequency of 10 Hz.

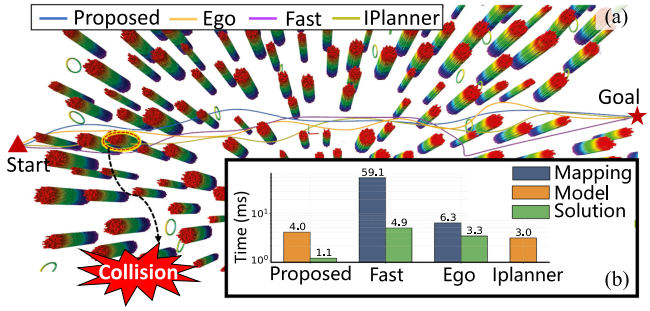


Fig. 3. (a) The simulation scenario and trajectory visualization at high aggressiveness. (b) The computation time for each method. "Mapping" refers to the time taken to convert depth maps into grid maps or ESDF. "Model" represents the time required for model inference, while "Solution" refers to the duration of numerical optimization.

We calculate the computation times for each method, as demonstrated in Fig. 3(b). Simultaneously, we measure kinematic metrics during flight and success rates, as exhibited in Table I. In various settings, our method demonstrates the highest success rate. Although Ego [11] also shows impressive robustness at low aggressiveness, its success rate drops sharply in high aggressiveness scenarios, accompanied by huge average and peak jerk. This phenomenon arises from two key factors. First, the obstacle avoidance depends on a safe path derived from a low-dimensional graph search in positional space, neglecting robot kinematics, which compromises optimality as speed increases. Second, the planner's dependence on prior optimization results as initial values proves less robust at high speeds. Rapidly changing obstacle distributions can render these prior solutions highly suboptimal or significantly unsafe, guiding the trajectory optimization toward an undesirable local optimum. As for the Fast [10], it requires maintaining an ESDF, which significantly increases mapping time, resulting in planning delays that are tens of times longer than our method. This delay reduces the system's responsiveness to unknown environments, particularly affecting performance at high speeds. Additionally, the discrete nature of the search strategy in this method lacks completeness within limited computation time, further reducing the success rate.

TABLE I
QUANTITATIVE BENCHMARKS IN VARIOUS CASES

Aggressiveness	Low $v_{limit}=2m/s, a_{limit}=3m/s^2$				Middle $v_{limit}=5m/s, a_{limit}=6m/s^2$				High $v_{limit}=8m/s, a_{limit}=10m/s^2$			
Methods	Proposed	Fast [10]	Ego [11]	Iplanner [14]	Proposed	Fast [10]	Ego [11]	Iplanner [14]	Proposed	Fast [10]	Ego [11]	Iplanner [14]
M.V (m/s)	2.0 (2.0)	2.0 (2.0)	2.0 (2.0)	**2.3** (**5.4**)	5.0 (5.0)	5.0 (5.0)	5.0 (5.0)	4.8 (4.9)	8.0 (8.0)	7.9 (8.0)	7.5 (8.0)	7.6 (8.0)
M.A (m/s^2)	1.5 (2.0)	1.8 (2.9)	2.2 (3.0)	**3.4** (**14.8**)	6.0 (6.0)	5.9 (6.0)	6.0 (6.0)	**6.5** (**22.5**)	6.5 (8.5)	7.5 (10.0)	9.5 (10.0)	6.8 (**44.3**)
M.JK (m/s^3)	**6.9** (**13.6**)	9.4 (32.1)	15.7 (27.6)	23.9 (124.6)	**29.5** (50.0)	30.9 (**46.1**)	50.1 (89.2)	41.4 (362.7)	**27.5** (**54.4**)	39.4 (71.5)	118.2 (311.9)	51.6 (629.5)
Exe.T (s)	**35.45**	36.33	**35.45**	42.67	13.66	**13.54**	14.67	13.69	10.13	10.50	12.88	**9.28**
Suc.R (%)	**100.0**	92.5	**100.0**	93.0	**97.5**	88.0	91.0	90.5	**93.0**	75.0	78.0	71.5

We record the highest velocity (M.V), acceleration (M.A), and jerk (M.JK) in each navigation task and subsequently calculate the average and peak of these metrics over multiple tasks. For instance, for M.V, the value outside the bracket signifies the average of the maximum velocities recorded in multiple tasks, whereas the value inside the bracket illustrates the absolute peak value. "Exe.T" is the average flight duration per navigation task. "Suc.R" refers to the average success rate. Constraint violations are marked in red, while the highest success rates are highlighted in blue.

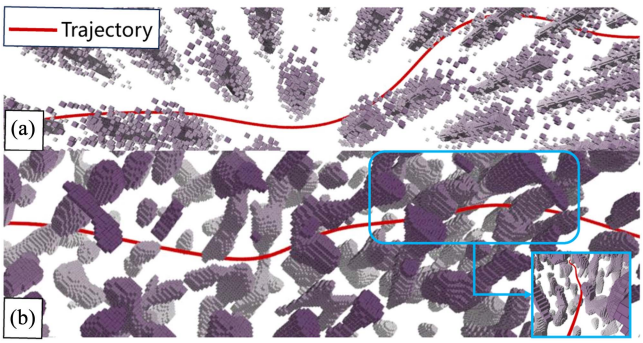


Fig. 4. Environment visualization: Obstacle colors transition from light to dark, representing heights ranging from low to high. Figure a shows a noise-random forest, while Figure b depicts a cave environment.

TABLE II
STATISTICS IN REAL-WORLD EXPERIMENTS

Statistic	Speed (m/s)	Acceleration (m/s^2)	Roll (deg)	Pitch (deg)	Yaw (deg)
Max	**5.0**	**5.6**	6.9	25	9.3
Mean	3.7	1.6	0.068	5.7	-0.49
STD	1.2	1.2	2.6	10	3.9

Speed and attitude data of the drone are derived from visual-inertial odometry. Acceleration is the norm of the acceleration vector in the world frame minus gravity. We use the attitude of the drone to transfer the data from the accelerometer in the IMU to the world frame.

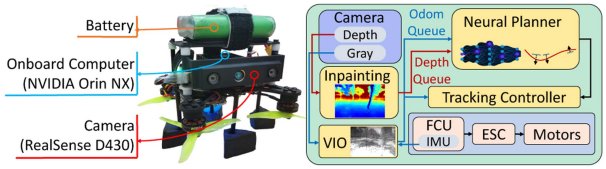
Although the learning-based Iplanner [14] demonstrates the highest time efficiency, our method is only 2 ms slower, which also greatly satisfies the real-time requirements of the system. However, Iplanner's lack of consideration for multimodal problem characteristics makes it prone to unsafe local optima. Notably, although Iplanner incorporates penalties for dynamic constraints during offline training to guide the network towards constraint compliance, it is less comprehensive compared to our explicit spatial-temporal optimization applied directly to the trajectory online. For instance, In highly aggressive scenarios, the peak acceleration exceeds constraints by more than four times, which is clearly outside the robot's expected operational mode and risks severe crashes.

B. Generalization Discussion

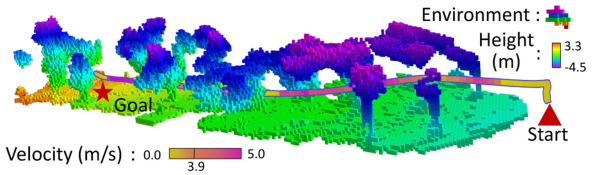
To assess generalization, we create two new environments: (1) a noisy random forest (Fig. 4(a)), where Gaussian-distributed noise points are randomly scattered around the original cylindrical obstacles, and (2) a cave-like environment (Fig. 4(b)), characterized by irregularly shaped obstacles of varying morphologies derived from Poisson noise. We generate 100 random start-end pairs to evaluate flight success rates over 25–35 meters. Two models are tested: one trained only in a random forest with cylindrical obstacles and another retrained in new environments. The retrained model achieves 98% success in the noisy random forest and 90% in the cave-like setting, versus 90% and 79% for the untrained model. Although these rates reflect a decline from the training environment's performance, they remain high. Indeed, we acknowledge that our approach shares a common limitation inherent to learning-based methods, wherein the model exhibits reduced performance in new environments. In the future, we will explore strategies to improve the planner's generalization across diverse environments through smarter network designs, without relying on additional data.

C. Real-World Experiments

To validate the effectiveness of our framework in the real world, we conduct a total of ten flight experiments in different sections of a forested area, where training data are exclusively extracted from simulations and do not involve real-world depth images. One of the forested areas is approximately 28,461.37 square meters in size, with a density of approximately 0.05 trees per square meter, with tree radii approximately ranging from 1.0 m to 2.0 m. Besides, in the simulation, the depth images are obtained by ray casting in parallel using GPU. In the real wooded area, we randomly select goals more than 50 m away from the robot, one of which is illustrated in the bottom half of Fig. 5. In this case, the maximum speed and acceleration of the drone are limited to 5 m/s and 6 m/s^2 respectively.



(a) Hardware and software configurations.



(b) A Case from real-world experiments.

Fig. 5. Real-world Experiments. The grid map representing the environment is obtained by recording rosbag and projecting the depth data to the world frame by aligning the depth images and odometry after the experiment.

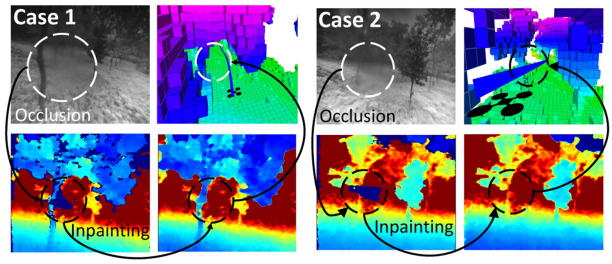


Fig. 6. Image inpainting makes the system more stable. The drone can still fly agilely when the camera lens is blurred.

Table II shows statistics in the real-world experiments corresponding to the case shown in Fig. 5. As we can see, the robot maintains a relatively high speed throughout to reach the target state without violating the maximum speed and acceleration constraints we set. More demonstrations can be found in the attached multimedia and project page.

In the real world, drones can get dust, water droplets, and other foreign objects, which can contaminate the lens and thus create holes in the depth image. These untreated pixels may be perceived as obstacles by the robot and cause it to fail to pass through the region or behave conservatively. To reduce the sim2real gap, we patch the depth images from the D430 by solving an equation similar to the Navier-Stokes equations numerically [26], which makes the system more stable, as shown in Fig. 6.

VII. Conclusion

In this paper, we propose an end-to-end visual navigation system that learns optimal trajectories from depth images and integrates a spatial-temporal trajectory optimizer to ensure adherence to kinematic constraints, enhancing interpretability and scalability. Benchmarks and real-world experiments demonstrate the advanced nature and practicality of our method. In the future, we plan to expand our method by incorporating task-specific objective functions into the optimization process to better adapt to specific task scenarios.

References

[1] L. Han, F. Gao, B. Zhou, and S. Shen, "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots," in *Proc. 2019 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 4423–4430.
[2] M. Jacquet and K. Alexis, "N-MPC for deep neural network-based collision avoidance exploiting depth images," in *Proc. 2024 IEEE Int. Conf. Robot. Autom.*, 2024, pp. 13536–13542.
[3] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Sci. Robot.*, vol. 6, no. 59, 2021, Art. no. eabg5810.
[4] J. Tordesillas and J. P. How, "Deep-PANTHER: Learning-based perception-aware trajectory planner in dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 8, no. 3, pp. 1399–1406, Mar. 2023.
[5] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Trans. Robot.*, vol. 38, no. 5, pp. 3259–3278, Oct. 2022.
[6] A. L. Dontchev and R. T. Rockafellar, *Implicit Functions and Solution Mappings*, vol. 543. Berlin, Germany: Springer, 2009.
[7] G. Zhao, T. Wu, Y. Chen, and F. Gao, "Learning speed adaptation for flight in clutter," *IEEE Robot. Automat. Lett.*, vol. 9, no. 8, pp. 7222–7229, Aug. 2024.
[8] J. Hou et al., "Primitive-planner: An ultra lightweight quadrotor planner with time-optimal primitives," *IEEE Trans. Robot.*, vol. 41, pp. 3629–3648, 2025.
[9] F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," in *Proc. 2016 IEEE Int. Symp. Saf., Secur., Rescue Robot.*, 2016, pp. 139–146.
[10] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3529–3536, Oct. 2019.
[11] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An ESDF-free gradient-based local planner for quadrotors," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 478–485, Apr. 2021.
[12] J. Lu, X. Zhang, H. Shen, L. Xu, and B. Tian, "You only plan once: A learning-based one-stage planner with guidance learning," *IEEE Robot. Automat. Lett.*, vol. 9, no. 7, pp. 6083–6090, Jul. 2024.
[13] Y. Wu, X. Sun, I. Spasojevic, and V. Kumar, "Deep learning for optimization of trajectories for quadrotors," *IEEE Robot. Automat. Lett.*, vol. 9, no. 3, pp. 2479–2486, Mar. 2024.
[14] F. Yang, C. Wang, C. Cadena, and M. Hutter, "iPlanner: Imperative path planning," in *Proc. Robot.: Sci. Syst.*, 2023.
[15] W. Xiao et al., "BarrierNet: Differentiable control barrier functions for learning of safe robot control," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 2289–2307, Jun. 2023.
[16] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.
[17] Z. Han et al., "An efficient spatial-temporal trajectory planner for autonomous vehicles in unstructured environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 25, no. 2, pp. 1797–1814, Feb. 2024.
[18] J. Li, J. Sun, T. Long, and Z. Zhou, "Differential flatness-based fast trajectory planning for fixed-wing unmanned aerial vehicles," 2024, *arXiv:2412.01468*.
[19] Z. Han, Z. Wang, N. Pan, Y. Lin, C. Xu, and F. Gao, "Fast-racing: An open-source strong baseline for SE(3) planning in autonomous drone racing," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 8631–8638, Oct. 2021.
[20] X. Zhou et al., "Swarm of micro flying robots in the wild," *Sci. Robot.*, vol. 7, no. 66, 2022, Art. no. eabm5954.
[21] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probability*, L. M. L. Cam and J. Neyman, Eds., Univ. California Press, 1967, vol. 1, pp. 281–297.
[22] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, no. 1, pp. 503–528, 1989.
[23] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.
[24] L. Pineda et al., "Theseus: A library for differentiable nonlinear optimization," in *Proc. Adv. Neural Inf Process. Syst.*, 2022, vol. 35, pp. 3801–3818.
[25] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 136–145.
[26] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," in *Proc. 2001 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2001, vol. 1, pp. I–I.