You Only Plan Once: A Learning-Based One-Stage Planner With Guidance Learning

Junjie Lu, Xuewei Zhang, Hongming Shen, Liwen Xu, and Bailing Tian

***Abstract*—In this work, we propose a learning-based one-stage planner for trajectory generation of quadrotor in obstacle-cluttered environment without relying on explicit map. We integrate perception and mapping, front-end path searching, and back-end optimization into a single network. We frame the motion planning problem as a regression of spatially separated polynomial trajectories and associated scores. Specifically, our approach adopts a set of motion primitives to cover the searching space, and predicts the offsets and scores of primitives for local optimization in a single forward propagation. A novel unsupervised learning strategy, termed guidance learning, is developed to provide numerical gradients as the guidance for training. We train the network policy with privileged information about the surroundings while only the noisy depth observations are available during inference. Finally, a series of experiments are conducted to demonstrate the effectiveness and time-efficiency of the proposed method in both simulation and real-world.**

***Index Terms*—Integrated planning and learning, collision avoidance, aerial systems, perception and autonomy.**

I. Introduction

AUTONOMOUS navigation of quadrotors in obstacle-cluttered environment has been extensively investigated. However, the limited computational resources and low-precision airborne sensors make this task still challenging.

Traditionally, the autonomous navigation problem is usually separated into (i) perception and mapping, (ii) front-end path searching, and (iii) back-end trajectory optimization. Firstly, an efficient perception and mapping module [1], [2] is essential for autonomous aerial system to navigate in unknown environments. However, compared to LiDAR sensors with more precise and long-range measurements, stereo cameras only provide noisy depth images with about 0.6 - 6 m sensing range. It significantly affects the performance of the planning algorithms, especially for the generation of flight corridors in hard-constrained methods. Besides, the navigation problem is in general multi-modal because many equally valid solutions may exist. As a result, the planner can be trapped in different local minima of a small fraction of solution space around different initial paths. To avoid suboptimal solutions, the front-end such as topological path searching algorithm [3], [4] is adopted to find multiple initial paths. Finally, the path is further improved by the back-end optimization module such as [5], [6] to make it smooth, safe, and dynamically feasible. The divide-and-conquer pipeline makes the overall system more interpretable but introduces additional latency, which is fatal for high-speed flights.

In recent years, the learning-based planners that process raw sensory inputs and perform complex behaviors directly have shown great potential. Some policies are trained to imitate a privileged expert, and others adopt reinforcement learning to explore the optimal policy by trial-and-error. The supervised learning (imitation learning or behavior cloning) based methods [7], [8] achieve impressive performance by training a lightweight network to approximate the solution of a computationally expensive algorithm. Specifically, the network policy is trained by minimizing the distance between predictions and labels demonstrated by the expert. However, the evaluation metrics should take the smoothness and safety into account, instead of the distance to the optimal demonstrations. For example, it is equally feasible to avoid a spherical obstacle from all around, but the limited expert labels, even with the multi-hypothesis winner-takes-all loss, cannot represent the realistic distribution of cost. On the other hand, the upper bound of the network policy is the expert policy, making it highly dependent on the performance of the expert. On the contrary, the network policy in reinforcement learning [9], [10] is trained by maximizing the rewards from environment, which realistically reflects the performance of the action. Furthermore, it explores the optimal policy by trial-and-error rather than imitation, thus having the potential to outperform the classical expert. However, compared to the direct association between inputs and labels in supervised learning, the reward signal of reinforcement learning is frequently sparse, noisy, and delayed, making it harder to converge. Additionally, some other methods [11], [12] train a network for collision prediction of pre-defined motion primitives, which reflects the feasibility of trajectories accurately and considers the multi-modality of planning problem. However, the finite pre-defined primitives cannot represent complex trajectories and cover feasible solutions comprehensively.

In contrast to these approaches, we propose a learning-based one-stage planner without relying on explicit map. As visualized in Fig. 1, we integrate perception and mapping, front-end path searching, and back-end trajectory optimization into a single network. Specifically, our approach takes the noisy depth images as

Manuscript received 16 December 2023; accepted 3 May 2024. Date of publication 10 May 2024; date of current version 20 May 2024. This letter was recommended for publication by Associate Editor K. Alexis and Editor G. Loianno upon evaluation of the reviewers' comments. This work was supported in part by the National Key R&D Program of China under Grant 2023YFB4704900 and in part by the National Natural Science Foundation of China under Grant 62273249, Grant 62203415, Grant 62022060, and Grant 62073234. *(Corresponding author: Bailing Tian.)*

The authors are with the School of Electrical and Information Egineering, Tianjin University, Tianjin 300072, China (e-mail: lqzx1998@tju.edu.cn; zhangxuewei@tju.edu.cn; shenhm@tju.edu.cn; xlw_2000@tju.edu.cn; bailing_tian@tju.edu.cn).

For supplementary video see: https://youtu.be/m7u1MYIuIn4. The code will be released at https://github.com/TJU-Aerial-Robotics/YOPO.

Digital Object Identifier 10.1109/LRA.2024.3399589

2377-3766 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

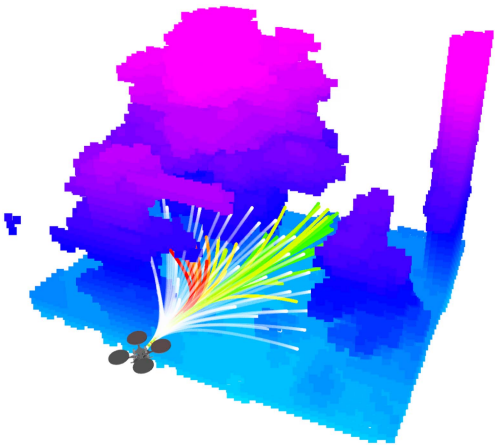


Fig. 1. We utilize the motion primitives (the white curves) as anchors for comprehensive exploration of the solution space, and predict the offsets and scores for further improvement (the curves color-coded by scores). To clarify, the map is unavailable and only the optimal trajectory is solved in practice.

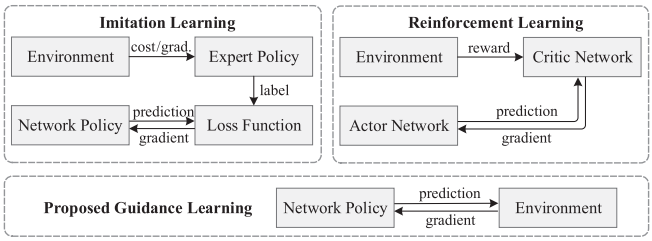


Fig. 2. Comparison of different learning paradigms.

sensory inputs and adopts a lightweight backbone for perception and feature extraction. Benefiting from the data-driven nature and privileged learning strategy, the proposed planner exhibits competitive robustness to sensory noise without mapping and filtering. To search the solution space thoroughly and yield better replanning, we adopt a set of motion primitives (represented by the boundary state) uniformly sampled within the FOV of depth camera as the front end. Inspired by the one-stage object detector YOLO (You Only Look Once) [13] predicting the offsets and confidences of pre-defined anchor boxes, we utilize the primitives as anchors and parallelly predict the offsets and scores of them for further improvement in a single forward propagation. We select the optimal primitive according to the scores and apply the offsets to the state of primitive as the back-end optimization. Finally, the optimal trajectory is represented as a high-order polynomial by solving the boundary value problem.

Besides, to address the above problems in present learning paradigms, we propose a novel unsupervised learning strategy termed guidance learning. As visualized in Fig. 2, the network policy in supervised learning is trained by the gradient of an analytic loss function between predictions and labels, whereas the actor-critic based reinforcement learning utilizes a critic network to model the environment and serve as the differentiable function. On the contrary, we consider that the essence of both classical and learning-based methods is gradient optimization. In classical gradient-based trajectory optimization, the numerical gradient of the cost (obtained by querying ESDF map, etc.) is applied to the parametric representation of trajectory (e.g., waypoints of polynomials or control points of B-spline). We further back-propagate this numerical gradient to the weights of neural network via the chain rule in the training process. Therefore, the feedback is realistic, accurate, and timely. Furthermore, the expert-free strategy allows us to initialize various states and goals for a depth sample in training for data augmentation, while not requiring re-annotation or re-interaction with the simulator. There is no need to assign labels to the predictions, making it practical to predict more alternative trajectories simultaneously. Compared to giving expert demonstrations for imitation in supervised learning or exploring by trial-and-error in reinforcement learning, we provide numerical gradient as guidance to lead the learning process. Thus, we term it "guidance learning". For the multi-modal nature of navigation, we explicitly divide the solution space and predict trajectories from the corresponding region of image using a fully convolutional header with shared weights. Compared with predicting few trajectories by different neurons and mapping labels by the winner-takes-all principle, our predictions are unambiguous and clear-corresponding, and therefore, avoid the mode collapse problem (all predictions of the network are close to the same label).

The main contributions of this work are as follows:

1) An efficient learning-based one-stage planner is proposed, which integrates perception, front-end path search, and back-end optimization in a single neural network and generates multiple alternative trajectories for comprehensive exploration of the solution space.
2) To train the proposed network, a realistic, accurate, and data-efficient unsupervised learning strategy is proposed, which directly back-propagates the numerical gradients from the environment to the weights of the neural network.
3) The proposed method is trained by privileged learning and integrated into a fully autonomous quadrotor system. A series of simulated and real-world experiments are conducted to validate the presented method.

II. Method

A. One-Stage Planner

1) Trajectory Representation: In this work, the trajectory is presented as three independent one-segment time-parameterized polynomials in the body frame with dimensions $\mu \in \{x, y, z\}$:

$$f_\mu(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + \cdots + a_n t^n . \quad (1)$$

The vector of coefficients is written as

$$\boldsymbol{A} = [a_0, a_1, a_2, a_3, \ldots, a_n] . \quad (2)$$

The overall system is illustrated in Fig. 3. Firstly, the navigation problem is in general multi-modal, and therefore, the planner can be trapped in different local minima around different initial paths. The front-end is utilized to search the solution space more thoroughly and provide topologically distinct initial paths for local optimization. Similarly, we adopt a set of motion primitives $\mathcal{P} = \{p_0, p_1, p_2, \ldots, p_{M-1}\}$ as the anchor to cover the searching space, each of which guides an independent optimization in training. Consistent with our previous approach [11], the primitive library is defined in the state lattice space rather than in polynomial coefficients. This expression is spatially separated,

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

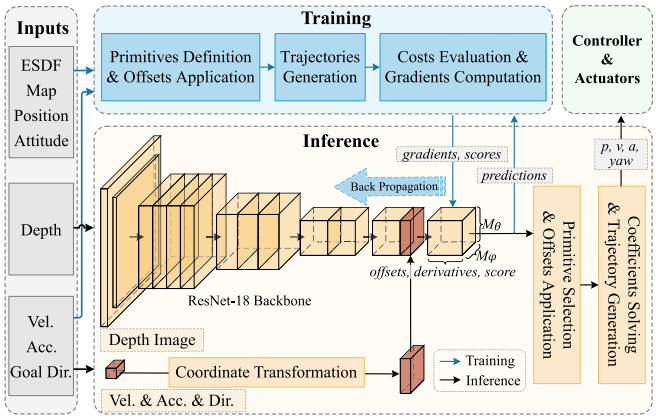


Fig. 3. System Overview. Our method takes the depth image, current states, and goal direction as input, and predicts the offsets, end-derivatives, and score for each primitive. The ground truth of ESDF map is utilized for numerical gradient computation in training, while is unavailable for the network policy.

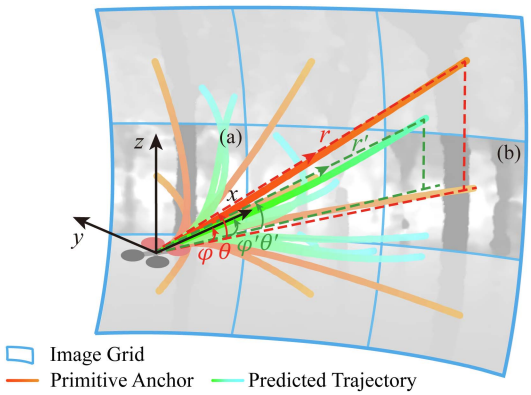


Fig. 4. Visualization of the primitives and predicted trajectories. We divide the depth image into $M_\varphi \times M_\theta$ grids, each of them corresponds to a primitive. We predict the offsets and end-derivatives to improve the performance of primitives within a local solution space.

facilitating the integration between network predictions and primitive anchors. Specifically, we uniformly sample M_θ polar angles and M_φ azimuth angles in the visible range of depth sensor (denoted by θ and φ, respectively). As visualized in Fig. 4, the motion primitive $\boldsymbol{p}_{ij}$ is defined as (3) in the body frame:

$$\boldsymbol{p}_{ij} = (r\cos\theta_j\cos\varphi_i,\ r\cos\theta_j\sin\varphi_i,\ r\sin\theta_j)\,. \tag{3}$$

Where $i \in [1, M_\varphi]$, $j \in [1, M_\theta]$ represent the index of primitive in horizon and vertical directions respectively, and r is the radius of planning horizon. Instead of solving the coefficients of the primitive trajectory in [11], we treat (3) as independent initial value to guide the optimization process in training.

In classical pipelines, the safety and smoothness of initial path are further improved by non-linear optimization, which incorporates the gradient information from the ESDF map and dynamic constraints. Differently, we utilize an end-to-end network to achieve this. For each primitive, we predict the offsets $\Delta\theta$, $\Delta\varphi$, and Δr, as well as the end-derivatives (velocity and acceleration for 5-order polynomial) and corresponding score. The refined end-state of the ij-th trajectory can be expressed by:

$$\boldsymbol{p}'_{ij} = (r'_{ij}\cos\theta'_{ij}\cos\varphi'_{ij},\ r'_{ij}\cos\theta'_{ij}\sin\varphi'_{ij},\ r'_{ij}\sin\theta'_{ij}). \tag{4}$$

Where $r'_{ij} = r + \Delta r_{ij}$, $\varphi'_{ij} = \varphi_i + \Delta\varphi_{ij}$, and $\theta'_{ij} = \theta_j + \Delta\theta_{ij}$. For brevity, we omit the subscript ij if not cause ambiguity in the following. In this work, we fix the execution time and generate receding-horizon trajectories with varying radius r'. Finally, we incorporate the current state (given by the state estimator) and end state (the position $\boldsymbol{p}'_{ij}$ and its derivatives predicted by network) into a vector $\boldsymbol{d}$, and introduce a constant matrix $\boldsymbol{M}$ to map it to the coefficients $\boldsymbol{A}$:

$$\boldsymbol{A} = \boldsymbol{M}^{-1}\boldsymbol{d}\,. \tag{5}$$

Note that the primitive anchors are pre-defined and do not require additional computation during inference. Moreover, we predict multiple feasible trajectories with different topologies simultaneously in a single forward propagation and only the optimal one is solved according to the predicted scores.

2) Network Architecture: In this work, we unify the separate components of motion planning into a single neural network. As shown in Fig. 3, it takes the depth image, current state (velocity and acceleration), and goal direction (a unit vector) as inputs, and predicts the offsets, end-derivatives, and score of every primitive. For dimensionality reduction, the inputs and outputs are represented in the body frame, thereby avoiding the need for position and attitude. Firstly, we divide the depth image into $M_\varphi \times M_\theta$ grids, each of them corresponds to a primitive in its frustum. Each grid cell is responsible for predicting the offsets $\Delta\theta$, $\Delta\varphi$, and Δr, as well as the end-derivatives and score of the corresponding primitive. We modify the ResNet-18 as the backbone for depth feature extraction. For example, we use the downsampling factor of 32 when $M_\varphi \times M_\theta = 3 \times 3$ and the image resolution is 96×96. Instead of predicting all primitives simultaneously using a fully connected layer in previous work [11], we predict independent trajectory at every location in the feature map by 1×1 convolutional layer with shared weights. However, different from object detection tasks, the motion planning problem is not translation invariant. That is, we want different predictions at different grids even with the same features. Taking the grids in Fig. 4 as example, we want the rightward offsets at the left grid (a), while the leftward offsets at the right grid (b). To avoid contradictions, we introduce the primitive frame and define the rotation matrix from body to primitive frame as:

$$\boldsymbol{R}_{ij} = \boldsymbol{R}_z(\varphi_i)\boldsymbol{R}_y(-\theta_j)\,. \tag{6}$$

Where $\boldsymbol{R}_y(\cdot)$ and $\boldsymbol{R}_z(\cdot)$ represents the rotation in axis y and z, respectively. Subsequently, the inputs are transformed to the primitive frames by (7) and concatenated to the corresponding locations of feature map:

$$\boldsymbol{x}'_{ij} = \boldsymbol{R}_{ij}^{-1}\,\boldsymbol{x}\,. \tag{7}$$

Where $\boldsymbol{x}$ represents the vector of current state and goal direction in the body frame. In addition, the end-derivatives predicted by the network are also defined in the primitive frame. Through this transformation, the inputs and predictions are decoupled with the location of grids.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

Subsequently, the network outputs a tensor y with the shape of $M_\varphi \times M_\theta \times M_d$, where $M_\varphi \times M_\theta$ corresponds to each primitive, and M_d is the raw representation of offsets, end-derivatives, and scores. In our experiments, $M_d = 10$ for 3 offsets, 3 velocities, 3 accelerations, and 1 score for 5-order polynomial. We use a hyperbolic tangent activation function to constrain the outputs $(y_\theta, y_\varphi, y_r)$ and bound the offsets to a fraction of solution space around the initial primitive. If the local solution space prior is $(-d_\theta, +d_\theta)$, $(-d_\varphi, +d_\varphi)$, and $(-d_r, +d_r)$, then the predicted offsets correspond to:

$$\begin{aligned} \Delta\theta &= \tanh(y_\theta)\, d_\theta \\ \Delta\varphi &= \tanh(y_\varphi)\, d_\varphi \\ \Delta r &= \tanh(y_r)\, d_r\,. \end{aligned} \tag{8}$$

To explicitly capture the multimodality of planning problem and thoroughly cover the visible space, the local space prior is slightly larger than the FOV of the grid. Besides, the outputs of end-derivatives $\boldsymbol{y}_v$ and $\boldsymbol{y}_a$ are also constrained by $tanh(\cdot)$ for dynamical feasibility and transformed to the body frame by:

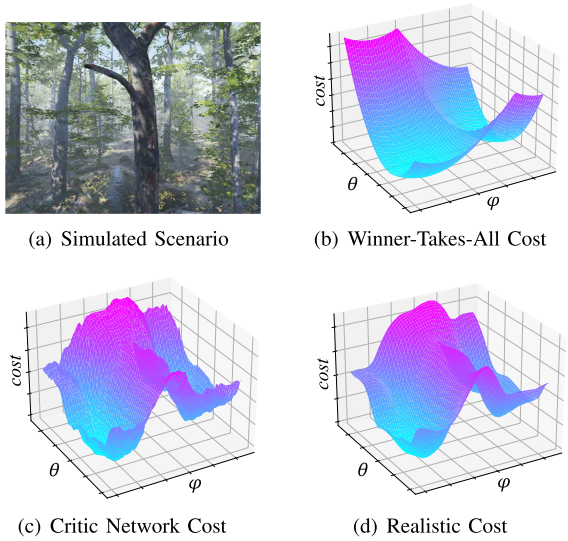
$$\begin{aligned} \boldsymbol{v} &= \boldsymbol{R}_{ij} \tanh(\boldsymbol{y}_v)\, v_{\max} \\ \boldsymbol{a} &= \boldsymbol{R}_{ij} \tanh(\boldsymbol{y}_a)\, a_{\max}\,. \end{aligned} \tag{9}$$

After that, the predictions with optimal score are substituted to (1)–(5) to solve the receding-horizon trajectory. Specifically, we predict trajectories with 2 s time window at 15 Hz. That is, only 1/30 of the trajectory is executed to ensure timely reaction to obstacles. In addition, the heading angle is defined as the bisector of the current velocity and goal directions, and the trajectory is finally discretized into reference states at 50 Hz for tracking by a geometric controller [14].

B. Training Strategy

1) Guidance Learning: The core of training is gradient optimization of weights by a differentiable loss function, yet the motion planning problem is typically complex and non-analytic. In imitation learning, the loss function is simplified into a quadratic form by the distance to the optimal trajectories as shown in Fig. 5(b). However, an extra expert policy is required and its performance determines the upper bound of the network policy. On the other hand, a label assignment method is essential for multi-modal learning, but the distance to the assigned label still cannot reflect the realistic cost. On the contrary, the actor-critic based reinforcement learning method trains a differentiable critic-network to model the environment for action evaluation. However, as depicted in Fig. 5(c), training by trial-and-error requires a large amount of data and may not converge accurately, which is particularly restrictive for complex and slow-to-simulate systems.

To solve above problems, a novel training method is proposed in this work to guide the learning process. It provides realistic trajectory evaluation as shown in Fig. 5(d), and is experimentally demonstrated to outperform the classical gradient-based method (i.e., the expert) with the same cost function. Traditionally, the numerical gradient of the cost (by querying ESDF map, etc.) is applied to the parametric representation of trajectory (waypoints of polynomial or control points of B-spline) for non-linear optimization. We further back-propagate this numerical gradient



(a) Simulated Scenario (b) Winner-Takes-All Cost

(c) Critic Network Cost (d) Realistic Cost

Fig. 5. Illustrative cost function of different training paradigms in scenario (a). For visualization, we plot the distributions with φ and θ as free variables, while keeping the current state, planning horizon, and end-derivatives fixed.

to the parameters of neural network via the chain rule as the guidance for training.

Firstly, the formulation of the cost function is written as:

$$J = \lambda_s J_s + \lambda_o J_o + \lambda_g J_g\,. \tag{10}$$

Where J_s is the smoothness cost to constrain the integral of squared derivatives, J_o is the safety cost to penalize the distance to obstacles, J_g is the goal cost to guide the quadrotor flying towards the goal, and λ_i are the weight parameters for trade-off. Inspired by [15], we rearrange the state vector $\boldsymbol{d}$ in (5) into a fixed block $\boldsymbol{d}_F$ (i.e., the current state) and free block $\boldsymbol{d}_P$ (i.e., the end state) using a selection matrix $\boldsymbol{C}$:

$$\boldsymbol{A} = \boldsymbol{M}^{-1}\boldsymbol{C}\begin{bmatrix}\boldsymbol{d}_F \\ \boldsymbol{d}_P\end{bmatrix}\,. \tag{11}$$

As addressed in [15], the cost of the smoothness term can be expressed by:

$$J_s = \begin{bmatrix}\boldsymbol{d}_F \\ \boldsymbol{d}_P\end{bmatrix}^T \boldsymbol{C}^T\boldsymbol{M}^{-T}\boldsymbol{Q}\boldsymbol{M}^{-1}\boldsymbol{C}\begin{bmatrix}\boldsymbol{d}_F \\ \boldsymbol{d}_P\end{bmatrix}\,. \tag{12}$$

Denote $\boldsymbol{C}^T\boldsymbol{M}^{-T}\boldsymbol{Q}\boldsymbol{M}^{-1}\boldsymbol{C}$ as matrix $\boldsymbol{B}$, then the cost can be written in a partitioned form as:

$$J_s = \begin{bmatrix}\boldsymbol{d}_F \\ \boldsymbol{d}_P\end{bmatrix}^T \begin{bmatrix}\boldsymbol{B}_{FF} & \boldsymbol{B}_{FP} \\ \boldsymbol{B}_{PF} & \boldsymbol{B}_{PP}\end{bmatrix}\begin{bmatrix}\boldsymbol{d}_F \\ \boldsymbol{d}_P\end{bmatrix}\,. \tag{13}$$

The Jacobian of J_s with respect to $\boldsymbol{d}_P$ can be computed as:

$$\frac{\partial J_s}{\partial \boldsymbol{d}_P} = 2\boldsymbol{d}_F^T\boldsymbol{B}_{FP} + 2\boldsymbol{d}_P^T\boldsymbol{B}_{PP}\,. \tag{14}$$

Different from [6] optimizing a piecewise polynomial with fixed end-state, we optimize the free end-state of a single polynomial with fixed time T. Then, defining safety cost as the line integral of the potential function over the trajectory leads the quadrotor minimizing the cost by reducing the planning length. Therefore, we reformulate the safety cost as the time integral of

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

the potential function $c(p(t))$:

$$J_o = \int_0^T c(p(t))dt = \sum_{\kappa=0}^{T/\delta t} c(p(\mathcal{T}_\kappa))\delta t\,. \tag{15}$$

Where $\mathcal{T}_\kappa = \kappa \cdot \delta t$. Then, we follow [6] to define the potential function as:

$$c(d_t) = \exp(-(d_t - d_0)/k)\,. \tag{16}$$

Where d_t is the distance to the nearest obstacle at $p(t)$, d_0 and k are pre-defined scaling factors. Then, the Jacobian of J_o in discrete form is:

$$\frac{\partial J_o}{\partial \boldsymbol{d}_{P\mu}} = \sum_{\kappa=0}^{T/\delta t} \nabla_\mu c(p(\mathcal{T}_\kappa))\boldsymbol{T}\boldsymbol{L}_P\,\delta t\,. \tag{17}$$

Where $\boldsymbol{T} = [\mathcal{T}_\kappa^0, \mathcal{T}_\kappa^1, \ldots, \mathcal{T}_\kappa^n]^T$, and $\boldsymbol{L}_P$ is the right block of matrix $\boldsymbol{M}^{-1}\boldsymbol{C}$ which corresponds to the free derivatives.

Additionally, we define the goal cost as the distance to a temporary goal for numerical stability:

$$J_g = (\boldsymbol{d}_{Pp} - g)^2\,. \tag{18}$$

Where $\boldsymbol{d}_{Pp}$ is the position component of $\boldsymbol{d}_P$, and g is the projection of the final goal onto the fixed-radius sphere around the quadrotor. Then, the Jacobian of J_g can be simply calculated as:

$$\frac{\partial J_g}{\partial \boldsymbol{d}_{Pp}} = 2(\boldsymbol{d}_{Pp} - g)\,. \tag{19}$$

The safety constraint is central to the planner. However, the distance d_t and gradient $\nabla_\mu c(\cdot)$ are queried from the ESDF map, which is non-analytic. As described above, different approaches are employed to approximate this non-analytic cost function for training in imitation and reinforcement learning. Instead, we discretize the integral on different time stamps in (15) for numerical calculation, and back-propagate the numerical gradients to the parameters of network directly via the chain rule. With total Jacobian $\partial J/\partial \boldsymbol{d}_P = \lambda_s \partial J_s/\partial \boldsymbol{d}_P + \lambda_o \partial J_o/\partial \boldsymbol{d}_P + \lambda_g \partial J_g/\partial \boldsymbol{d}_P$, the gradients of the network's predictions can be expressed by:

$$\frac{\partial J}{\partial y_\epsilon} = \frac{\partial J}{\partial \boldsymbol{d}_P}\frac{\partial \boldsymbol{d}_P}{\partial \Delta\epsilon}\frac{\partial \Delta\epsilon}{\partial y_\epsilon}$$
$$\frac{\partial J}{\partial y_\varepsilon} = \frac{\partial J}{\partial \boldsymbol{d}_P}\frac{\partial \boldsymbol{d}_P}{\partial y_\varepsilon}\,. \tag{20}$$

Where $\epsilon \in \{\theta, \varphi, r\}$ and $\varepsilon \in \{v, a\}$. They can be easily calculated by substituting (4), (8), and (9). Besides, the trajectory score is defined as $-J$ and supervised by the smooth L1 loss:

$$\mathcal{L} = SmoothL_1(y_s, -J)\,. \tag{21}$$

Where y_s is the score predicted by the network policy. It is analytic and can be automatically derived by the deep learning frameworks. Finally, we train the network by the Adam optimizer with the manually calculated gradients as guidance. To avoid the influence of negative trajectories, we optimize the predicted scores of all primitives, but only the end-states whose scores are greater than the threshold are trained.

2) Privileged Learning: Since the data-driven nature of deep learning, the network policy can achieve competitive performance to the privileged expert while relying only on limited sensory observations. The ground truth of the environment (point cloud and ESDF map) and complete state of the quadrotor are accessible for gradient computation in training, while only the noisy sensory observations are available for the network. As illustrated in Fig. 6, the classical gradient-based planners can be trapped in obstacles due to the insufficient or wrong observations. Alternatively, the gradient-free method MPPI (Model Predictive Path Integration) solve the planning problem using forward sampling of stochastic diffusion process, but may also fail to escape the sudden obstacles in the traveling direction. In comparison, we train the network with the perfect knowledge about the environment, thereby avoiding falling into incorrect local minima. In addition, the data-driven approach leverages the regularities in the training data, which makes it more robust to sensor noise.

To prevent crashes in the early training stage, a dataset including positions, orientations, and depth images is collected, where the positions and orientations are only utilized for gradient computation. We randomly reset the state of quadrotor in Flightmare [16] simulator to collect 100 K samples, and preprocess the invalid depths by nearest-neighbor interpolation to avoid confusion with near obstacles. Then, we iterate 50 epochs on the pre-collected dataset with the learning rate of 1.5×10^{-4} and batch size of 16 to train the proposed network. The threshold for trajectory selection in training is set to 0.04 in our experiment. A too-small threshold will lead the network overfitting to the safety conditions and perform unstably on entire dataset. On the contrary, it cannot converge to satisfactory results due to the influence of negative samples when the threshold is too large. To ensure sufficient coverage of the state space, we also use the dataset aggregation strategy (DAgger) for further fine-tuning. Besides, the expert-free strategy allows us to initialize various states and goals randomly for each depth sample for data augmentation, while not requiring re-annotation or re-interaction with the simulator.

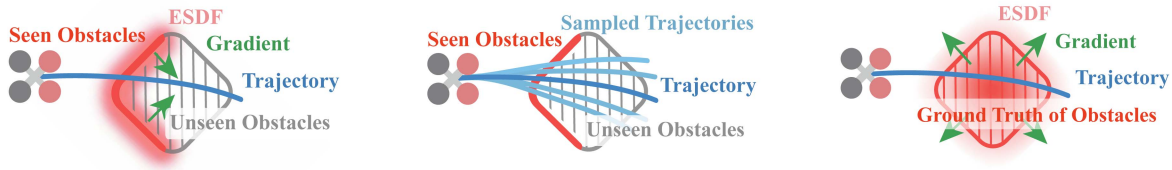
III. Experiment

In this section, a series of experiments are performed to verify the proposed learning-based planner and training strategy in simulation and real-world. In our physical platform, the RealSense D455 is used to provide depth images with the aspect ratio of 16 : 9. Therefore, we scale the raw images to 160×96 resolution and predict 5×3 trajectories with the downsampling factor of 32. The simulated comparisons are conducted in the open-source simulator Flightmare [16] on i7-9700 CPU and RTX 3060 GPU. Furthermore, the network policy is deployed with NVIDIA TensorRT for inference acceleration and implemented on a physical quadrotor with the computational unit of NVIDIA Xavier NX for real-world experiment.

A. Training Strategy Validation

In this section, we verify the proposed guidance learning strategy against the classical gradient-based method [6], which can be utilized as the expert to provide demonstrations for the network policy in imitation learning. Consistent with our

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

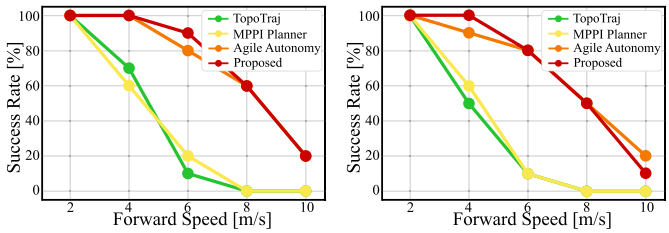


(a) Gradient-based Optimization (b) Sampling-based Optimization (c) Privileged Optimization

Fig. 6. The classical gradient-based (a) and sampling-based (b) methods may be trapped in obstacles with insufficient observations or unfavorable initialization, while the proposed learning-based method (3) can avoid falling into infeasible local minima by privileged learning.

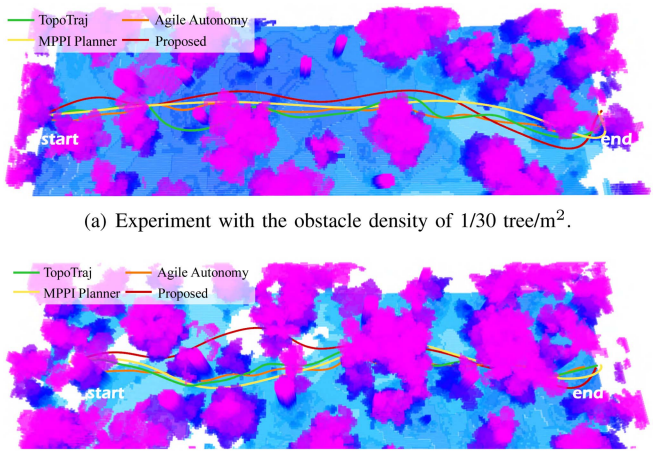
TABLE I
COMPARISON WITH THE EXPERT

Method	Metric	Cost		Latency
Expert	Average Optimal	0.078 0.036		30 ms
Network	Average Optimal	0.105 (initial) 0.056 (initial)	0.050 (final) 0.036 (final)	1.6 ms



(a) sparse scenario (b) dense scenario

Fig. 7. Success rate with the forward speed varying from 2 m/s to 10 m/s in obstacle densities of (a) 1/30 tree/m^2 and (b) 1/20 tree/m^2.



(a) Experiment with the obstacle density of 1/30 tree/m^2.

(b) Experiment with the obstacle density of 1/20 tree/m^2.

Fig. 8. Trajectory comparisons in simulated forest with massive trees and low bushes.

strategy, we adopt the same cost function defined in Section II-B1 and optimize the end-states of the trajectories with fixed execution time. We replace the front end of [6] by the primitives as initial values to guide independent optimization and the privileged information of the map is accessible in this validation. For a fair comparison, we train the network for 50 epochs by the proposed strategy and perform gradient descent 50 steps for each sample by the expert.

We compare the average and the optimal cost of 5×3 trajectories between the proposed method and expert. As shown in Table I, the network policy outperforms the expert in average cost and exhibits similar performance with the expert in optimal cost. The result validates the proposed training strategy and indicates that the network policy has the potential to outperform the corresponding expert (which is the upper bound of imitation learning). On the one hand, the planning problem is usually complex and non-convex, while the gradient-based expert is sensitive to the initial conditions and is easily stuck in suboptimal solutions with unfavorable initialization. On the contrary, the network policy has larger receptive field and context, and is able to discover patterns from the whole dataset and leverage the correlations for improvement. Besides, it is worth noting that most trajectories are feasible although only the high-scoring ones are trained. It illustrates that the weight-shared head trained with safe conditions is also workable for unfavorable initialization.

Additionally, we compared the latency of the network and expert policy. As visualized in Table I, the network policy runs over 10 times faster than the gradient-based optimization. It is understandable because the network predicts all trajectories in parallel, whereas the computation of classical methods increases linearly with the number of trajectories.

B. Simulated Comparison

In this section, we compare the proposed approach with three state-of-the-art vision-based methods: Fast Planner with Topological Paths [3] (denoted as TopoTraj), MPPI Planner with Hybrid A* [17], and the learning-based planner Agile Autonomy [8] as baselines. The comparisons are conducted in a simulated forest [16] with massive trees and low bushes, while only the depth images and states are accessible from the simulator. To ensure a fair comparison, we fine-tune the baselines for better real-time and safety performance trade-offs in the evaluation environment.

1) Obstacle Density: We evaluate the performance of above methods with respect to obstacle density under the average speed of 4 m/s. Trees are randomly placed with the diameters of 0.3 - 0.6 m and densities of about 1/20 and 1/30 tree/m^2, respectively. The average performance statistics and the qualitative results are illustrated in Table II and Fig. 8.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

TABLE II
SIMULATED COMPARISON

Method	Density (tree/m^2)	Latency (ms)				Safety (m)		Smoothness (m^2/s^5)	Traj. Length (m)
		Mapping	Front	Back	Total	Avg	Min		
TopoTraj	1 / 30	48.5	6.1	7.3	61.9	1.23	0.18	352.72	53.54
	1 / 20					1.00	0.08	810.17	54.28
MPPI	1 / 30	6.5	5.0	8.2	19.7	1.23	0.08	**14.92**	**51.58**
	1 / 20					0.94	0.06	**16.83**	52.88
Agile Autonomy	1 / 30	/	/	/	5.4	1.13	0.12	2324.47	51.65
	1 / 20					1.01	0.10	2954.46	**52.42**
Proposed	1 / 30	/	/	/	**1.6**	**1.63**	**0.24**	37.87	52.24
	1 / 20					**1.28**	**0.16**	73.52	52.78

* The best planning metrics under different densities are highlighted in blue and red, and the optimal real-time performance is marked in bold.

Firstly, we compare the computational cost of our approach against the baselines. With total computation time of 61.9 ms per frame, TopoTraj incurs the highest processing latency, most of which is spent on mapping and ESDF computation. Although MPPI Planner significantly reduces the mapping delay by implementing parallel on GPU, it performs back-end optimization with a Monte Carlo approximation using forward sampling, which is computationally complex. Besides, the back end of TopoTraj is also time-consuming because it performs independent optimization from multiple topologically distinct initial paths. By contrast, our approach achieves significantly lower processing latency, which only takes about 1.6 ms for inference. It can be attributed to the unified framework integrating perception, front-end path search, and back-end optimization. Compared with Agile Autonomy using independent heads for multi-trajectories prediction, we incorporate the solution space coverage and trajectory inference into a one-stage fully convolutional network, making our pipeline more concise.

Subsequently, we evaluate the safety metric (distance to the nearest obstacle) and smoothness metric (integral of the squared jerk) of our method and the baselines. As illustrated, our approach demonstrates the best safety and competitive smoothness performance in comparisons of different densities. It can be explained that we simultaneously predict multiple feasible trajectories with distinct topologies for comprehensive exploration of the solution space in significantly lower latency. Similarly, the safety performance of TopoTraj with multiple topological guiding paths is superior to MPPI with a single Hybrid A* as the front end. Furthermore, Agile Autonomy also achieves impressive performance under dense scenario, indicating that the learning-based method with lower latency is safer for high-speed flight with abrupt obstacles. In the smoothness metric, the MPPI Planner exhibits best by formulating the planning problem as a stochastic optimal control problem, while the Agile Autonomy results in higher costs without considering the smoothness penalty during training.

2) Flight Speed: We validate the success rate of above methods with the forward speed varying from 2 m/s to 10 m/s across different obstacle densities. We repeat the experiments 10 times randomly for each condition and the results are depicted in Fig. 7. As visualized, the learning-based methods exhibit significantly higher success rate than the classical vision-based method in both scenarios. Essentially, the decrease in the performance of the classical baselines can be attributed to the limited sensor range, noisy depth observations, and insufficient map. On the one hand, the mapping and ESDF updating are time-consuming as shown in Table II. On the other hand, the probabilistic updating process is necessary to remove sensing noise but makes the reconstruction of obstacles in the map slower. Comparatively, the proposed method and [8] can react to the observations in an extremely short time, making it safer for high-speed flight with unpredictable obstacles. Due to the data-driven nature and privileged learning strategy, the network policy demonstrates competitive robustness to the sensory noise without the temporal filtering operations. Additionally, our approach outperforms the Agile Autonomy below 6 m/s, since our method is able to predict more alternative trajectories with different primitives to explore the solution space more thoroughly. However, we take the smoothness of trajectories into account during training, which sacrifices the safety performance to a certain extent. Therefore, Agile Autonomy is more competitive at higher speeds in dense scenarios.

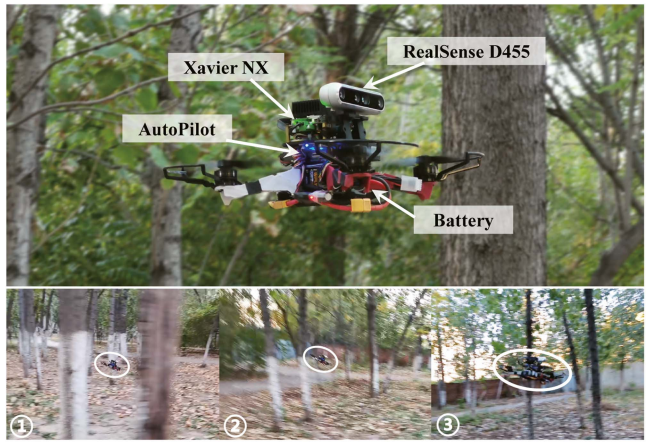


Fig. 9. The physical platform and real-world experimental scenario. ①–③ are the snapshots of the corresponding positions in Fig. 10.

C. Real-World Experiment

In this section, we validate the proposed learning-based one-stage planner in real-world. The network policy is implemented on a small quadrotor with 250 mm diameter and 1.13 kg mass, as illustrated in Fig. 9. The main computational unit is NVIDIA Xavier NX with 6-core ARM CPU and 384 CUDA cores GPU,

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.

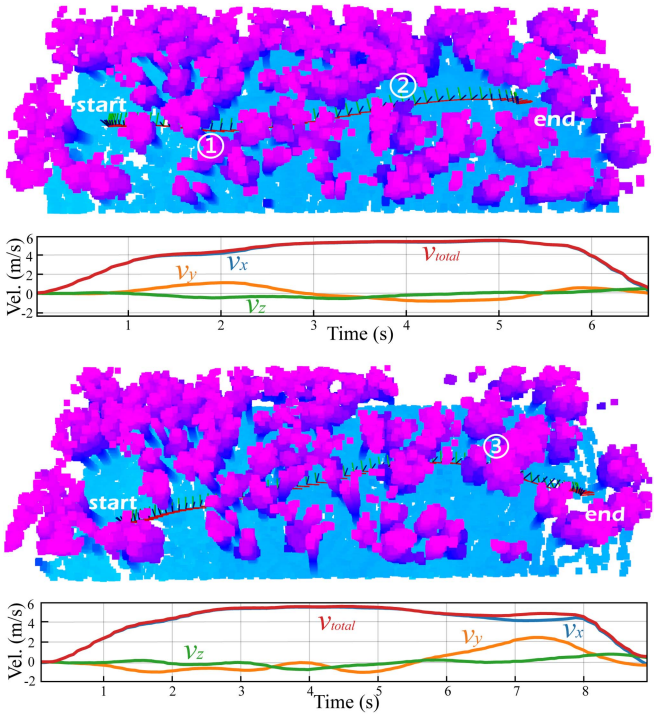


Fig. 10. Trajectories visualization of real-world flight experiments in dense forest with velocity profiles. Note that the map is only constructed for demonstration and ① - ③ are the positions of the corresponding snapshots in Fig. 9.

which only takes about 16 ms for inference after being deployed on TensorRT. The RealSense D455 is adopted to provide depth images with $87^\circ \times 58^\circ$ FOV and around 6 m sensor range. Consistent with training, we perform nearest-neighbor interpolation on invalid depths for computational efficiency. Besides, the state estimation is performed by the visual-inertial odometry VINS-Fusion and the reference trajectory is tracked by a geometric controller. All modules including localization, planning, and control are integrated into an airborne computational unit Xavier NX.

As shown in Fig. 9, the real-world experiments are conducted in a dense forest with the density of 1/10 tree/m^2 and diameter of the tree about 0.25 m. It is challenging for aggressive autonomous flight with noisy depth observations, limited sensing range, and airborne computational resources. The trajectories should be regenerated within extremely short time to address the abrupt and unexpected obstacles. The trajectories and velocity profiles are depicted in Fig. 10. As visualized, the quadrotor executes aggressive trajectories through the forest and achieves a maximum speed of 5.52 m/s. Note that the map is only constructed for visualization and the experimental environment is never observed at training time. We refer readers to the video for more information.

IV. Conclusion

In this letter, we propose a learning-based one-stage planner which integrates perception and mapping, front-end path searching, and back-end optimization into a single network. Our approach adopts a set of motion primitives to cover the searching space, and predicts the offsets and scores of all primitives in a single forward propagation. By comparisons, the proposed method exhibits significantly lower latency and demonstrates competitive performance to the SOTA methods. Moreover, an unsupervised learning method is proposed to train the network policy with the guidance of numerical gradient from the privileged ESDF map. Compared with imitation learning, the feedback of the proposed training strategy is realistic and the network policy has the potential to outperform the corresponding expert. Finally, autonomous flight experiments are conducted in a dense forest to validate the efficiency of our method.

References

[1] Y. Chen, S. Lai, J. Cui, B. Wang, and B. M. Chen, "GPU-accelerated incremental Euclidean distance transform for online motion planning of mobile robots," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 6894–6901, Jul. 2022.

[2] Y. Ren, Y. Cai, F. Zhu, S. Liang, and F. Zhang, "ROG-map: An efficient robocentric occupancy grid map for large-scene and high-resolution LiDAR-based motion planning," 2023, *arXiv:2302.14819*.

[3] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust real-time UAV replanning using guided gradient-based optimization and topological paths," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 1208–1214. [Online]. Available: https://api.semanticscholar.org/CorpusID:209515709

[4] H. Ye, X. Zhou, Z. Wang, C. Xu, J. Chu, and F. Gao, "TGK-planner: An efficient topology guided kinodynamic planner for autonomous quadrotors," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 494–501, Apr. 2021.

[5] 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.

[6] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3681–3688.

[7] 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.

[8] 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.

[9] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Proc. Robot.: Sci. Syst.*, 2017, pp. 1–10.

[10] R. Penicka, Y. Song, E. Kaufmann, and D. Scaramuzza, "Learning minimum-time flight in cluttered environments," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 7209–7216, Jul. 2022.

[11] J. Lu, B. Tian, H. Shen, X. Zhang, and Y. Hui, "LPNet: A reaction-based local planner for autonomous collision avoidance using imitation learning," *IEEE Robot. Automat. Lett.*, vol. 8, no. 11, pp. 7058–7065, Nov. 2023.

[12] H. Nguyen, S. H. Fyhn, P. D. Petris, and K. Alexis, "Motion primitives-based navigation planning using deep collision prediction," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 9660–9667.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[14] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proc. IEEE 49th Conf. Decis. Control*, 2010, pp. 5420–5425.

[15] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Robot. Research: 16th Int. Symp.*, 2016, pp. 649–666.

[16] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Proc. Conf. Robot Learn.*, 2021, pp. 1147–1157.

[17] H. Lu, Q. Zong, S. Lai, B. Tian, and L. Xie, "Flight with limited field of view: A parallel and gradient-free strategy for micro aerial vehicle," *IEEE Trans. Ind. Electron.*, vol. 69, no. 9, pp. 9258–9267, Sep. 2022.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on August 03,2025 at 08:49:26 UTC from IEEE Xplore. Restrictions apply.