2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

A Self-Supervised Learning Approach with Differentiable Optimization for UAV Trajectory Planning

Yufei Jiang*[1], Yuanzhu Zhan*[1], Harsh Vardhan Gupta[2], Chinmay Borde[2], Junyi Geng[1]

***Abstract*— While Unmanned Aerial Vehicles (UAVs) have gained significant traction across various fields, path planning in 3D environments remains a critical challenge, particularly under size, weight, and power (SWAP) constraints. Traditional modular planning systems often introduce latency and suboptimal performance due to limited information sharing and local minima issues. End-to-end learning approaches streamline the pipeline by mapping sensory observations directly to actions but require large-scale datasets, face significant sim-to-real gaps, or lack dynamical feasibility. In this paper, we propose a self-supervised UAV trajectory planning pipeline that integrates a learning-based depth perception with differentiable trajectory optimization. A 3D cost map guides UAV behavior without expert demonstrations or human labels. Additionally, we incorporate a neural network-based time allocation strategy to improve the efficiency and optimality. The system thus combines robust learning-based perception with reliable physics-based optimization for improved generalizability and interpretability. Both simulation and real-world experiments validate our approach across various environments, demonstrating its effectiveness and robustness. Our method achieves a 30.90% reduction in control effort while maintaining competitive tracking performance compared with state-of-the-art.**

I. Introduction

Over the past decade, interest in Unmanned Aerial Vehicles (UAVs) has grown rapidly across a wide range of fields, including applications such as 3D mapping [1], exploration [2], physical interaction [3], [4] and package delivery [5]. One of the critical UAV tasks is efficient path planning. In the environment without a pre-established map, the UAVs must re-plan quickly and efficiently to avoid collisions and perform safe navigation even under size, weight, and power (SWAP) constraints. Path planning for UAVs poses unique challenges comparing to the planning for ground mobile robot. First, UAVs operate in three-dimensional (3D) space, significantly increasing the search area and requiring consideration of obstacles at varying altitudes. Second, UAVs typically move at higher speeds, imposing stricter safety requirements. Third, planned paths must be dynamically feasible to account for 3D maneuvers, avoiding unplanned collisions resulting from discrepancies between planned and executed paths.

Traditional planning pipelines are typically modular, with perception, mapping, and path-searching modules designed separately and then integrated [6], [7], [8]. This approach, however, has several drawbacks. Limited information sharing between modules introduces latency, slowing the system's response. Independent module design also leads to suboptimal performance, as conservative assumptions in one component may trap the system in local minima. Moreover, the modular separation requires iterative updates during testing, where parameters of one module are fine-tuned while others are frozon, making the process tedious and time-consuming. End-to-end learning, which map sensory observations directly to actions or trajectories, have gained attention for their streamlined pipelines and robust perception capabilities. However, they often require large datasets, suffer from low sample efficiency, and face significant sim-to-real gaps. These methods also lack interpretability, making it difficult to incorporate physical knowledge. For instance, some supervised learning approaches that imitate expert-generated trajectories perform well in controlled settings but struggle to generalize [9]. Reinforcement learning (RL) and its variants aim to improve efficiency but often encounter issues like slow convergence, low sample efficiency, and overfitting [10], [11]. Additionally, designing effective reward functions is time-consuming, and even with well-crafted rewards, sim-to-real transfer remains a major challenge.

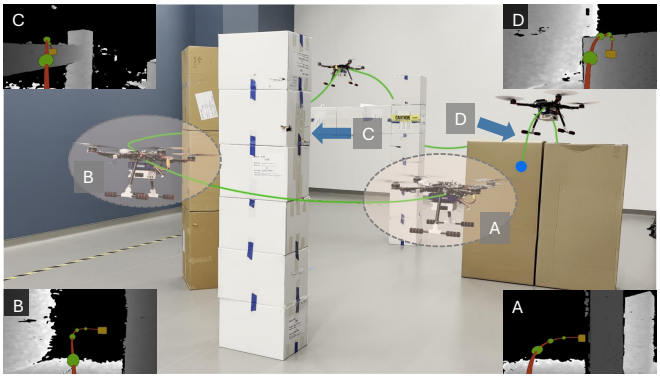


Fig. 1. 3D UAV trajectory planning in complex environment. Different desired waypoints are given by user, the UAV relies solely on depth images as input. The green curve showcases the UAV's trajectory. Starting from point (A) avoids vertical pillar (B), then performs maneuver to position (C) avoids the horizontal beam, (D) performs another vertical avoidance maneuver at different height and reach the blue target point.

Recent research has explored hybrid approaches that combine learning-based and model-based methods by integrating differentiable traditional components with learning modules [12]. For example, [13] introduced a self-supervised method combining a learning-based depth perception module with trajectory optimization (TO), using a pre-built traversability cost map for local planning. However, this approach

* Equal contribution in alphabetical order.

[1] Department of Aerospace Engineering, Pennsylvania State University, University Park, PA, 16802, USA. {yufei_jiang, yvz6008, jgeng}@psu.edu

[2] Department of Engineering and Applied Science, University at Buffalo, NY, 14260, USA. {hgupta4, cborde}@buffalo.edu

is limited to ground robots due to its reliance on a 2D cost map projection. Moreover, it employs a closed-form solution for cubic spline in TO without gradient backpropagation as claimed Bi-level optimization (BLO), reducing its generalizability to more complex or constrained scenarios. While [14] extends [13] with differentiable model predictive control (MPC) to enforce kinematic constraints, it guarantees only kinematic (not dynamic) feasibility due to its bicycle-model state choice. It is also limited to a 2D setting and handles only equality constraints. [15] embeds safe corridors and differentiable TO in the network for generating UAV dynamically feasible trajectories. However, it requires an offline motion-primitive library and still relies on supervised training.

To address these limitation, we propose a self-supervised pipeline for UAV path planning that combines a learning-based depth perception with differentiable, metric-based TO, forming a bi-level optimization (BLO). Our approach includes a 3D cost map to account for UAVs' 3D operations, providing collision costs to guide behavior without requiring expert demonstrations or human labels. We also develop a differentiable minimum snap TO module to ensure dynamically feasible trajectories with both equality and inequality constraints and enable gradient backpropagation on iterative optimization for end-to-end training. Additionally, a time allocation network predicts segment durations, enhancing efficiency and optimality. During deployment, the policy predicts collision conditions directly from first-person view (FPV) depth observations and plans paths accordingly. The end-to-end design optimizes observation features for planning objectives, combining the robustness of learning-based perception with the reliability, generalizability, and interpretability of physics-based methods.

In summary, the main contributions of this work are:

- We create a self-supervised pipeline for 3D UAV path planning that combines a learning-based depth perception module with differentiable, metric-based TO.
- We train the planner using geometry-derived collision signals from a 3D cost map for self-supervision without the need for expert demonstrations or human labels.
- We develop a differentiable minimum snap TO module for dynamically feasible trajectories while allowing end-to-end training. A time allocation network is designed to enhance efficiency and optimality.
- We present both simulation and real-world experiments to evaluate the proposed system in various environments.

II. Related Works

A. UAV Path Planning

Many classic planning frameworks have demonstrated effective for autonomous UAV navigation in challenging environments. For example, the modular approach [8] demonstrated in the DARPA Subterranean Challenge introduces latency and lacks robustness to noise and real-world errors. Gradient-based planning approaches optimize trajectories with manually designed safety constraints but can get stuck in local minima [16], [17], [18]. [19] is a UAV motion planning framework that leverages Euclidean Distance Fields (EDF) for gradient-based B-spline optimization. Similarly, corridor-based methods rely on low-dimensional search algorithms, often neglecting higher-order kinematics and producing dynamically infeasible trajectories [18], [20]. End-to-end learning methods have become emerging by eliminating explicit mapping and reducing latency. For instance, [9] trained a deep neural network (DNN) using human flight data, but it requires large datasets and struggles to generalize to diverse environments, with performance limited by the suboptimality of expert trajectories. Reinforcement learning has also been explored for UAV path planning [10], using real-world data to overcome the sim-to-real gap [11]. However, these approaches rely on neural networks and lack guarantees for kinematic feasibility and trajectory optimality.

Recently, hybrid methods combining neural networks with numerical optimization are emerging. While many work still treated the two separately [21], there are some research incorporates differentiable optimization for end-to-end training [15], [22], [23]. For example, [23] integrates perception and trajectory optimization by predicting the offsets and scores of a set of motion primitives for local optimization. However, the sampling-based method and predicted end-derivative cannot guarantee optimality. [22] integrate DNNs within the motion planning framework to predict the time allocation of a given set of waypoints. However, it formulates path searching as a separate module and rely on supervised learning for training. In contrast, our method avoids handcrafted primitives and uses self-supervision to jointly optimize perception and planning for 3D UAV scenario, reducing labeling and integration overhead.

B. Differentiable Optimization

Differentiable optimization refers to optimization algorithms where the output is differentiable with respect to the input parameters, enabling the integration of traditional model-based optimization with learning-based approaches. This paradigm combines the strengths of both methods—infusing interpretable domain knowledge into deep learning pipelines and reducing the need for tedious parameter tuning in traditional optimization. It has become a powerful paradigm in robotics for perception [24], planning [13], control [25], and physics-based simulation [26]. A pioneering work, OptNet [27], embeds optimization as a differentiable layer within neural networks, enabling gradient backpropagation for argmin problems. This approach has later been extended to model predictive control (MPC) using convex quadratic approximations for non-convex MPC problems [28], and to actor-critic MPC by integrating RL [29]. Recently, BPQP reformulated the backward pass as a quadratic programming (QP) problem [30], improving computational efficiency. Tools like Theseus [31] and PyPose [32] have also emerged, offering differentiable optimization capabilities for robotics. Our approach formulates the problem as a BLO where gradient backpropagation from the upper to lower level is critical, and employs a differentiable QP solver to incorporate both equality and

inequality physical constraints.

III. Methodology

A. Problem Definition

At each time stamp, given the observation of the depth image $\mathcal{D}$ and a target location ${}_{\mathcal{B}}\mathbf{G} \in \mathbb{R}^3$ in the UAV body frame $\mathcal{B}$, the planning problem is to find a trajectory $\boldsymbol{\tau}$ to guide the UAV from the current position $\mathbf{p}^R \in \mathbb{R}^3$ to the target while avoiding obstacles $\mathcal{O} \subset \mathcal{M} \subset \mathbb{R}^3$, where $\mathcal{M}$ is the whole workspace, $\mathcal{O}$ is the obstacle space the UAV cannot fly through.

B. Planning with Hybrid Learning and Optimization

The proposed hybrid learning and optimization process for UAV planning is shown in Figure 2. First, front-end network f: a convolutional neural network (CNN) front-end takes the depth input and encode into observation embedding. This embedding, combined with the target, is fed into the second planning network to predict a n-key-point path $\boldsymbol{\xi} \in \mathbb{R}^{n\times 3}$. In the back end, a Time Allocation Net g takes into the key-point path and generates the timestamp $\mathcal{T} \in \mathbb{R}^n$ for each trajectory segment. Next, a differentiable minimum snap trajectory optimizer (MSTO) takes into both the key-point path, generated time allocation and outputs a dynamic feasible trajectory $\boldsymbol{\tau}^*$. Then, a well-designed collision costs $\mathcal{U}$ based on a 3D cost map is evaluated and backpropagated through both the differentiable MSTO and the networks to update the network parameters of both the front-end network $\boldsymbol{\theta}_f$ and the Time Allocation Net $\boldsymbol{\theta}_T$. This process results in a nested bi-level optimization (BLO) problem, where the networks and differentiable MSTO can be jointly optimized.

$$\begin{aligned} \min_{\boldsymbol{\theta}_f,\boldsymbol{\theta}_T} \ & \mathcal{U}(f(\boldsymbol{\theta}_f), g(\boldsymbol{\theta}_T), \boldsymbol{\tau}^*) \\ \text{s.t. } & \boldsymbol{\tau}^* = \arg\min_{\tau} \mathcal{L}(f(\boldsymbol{\theta}_f), g(\boldsymbol{\theta}_T)) \\ & \text{s.t. } h(f(\boldsymbol{\theta}_f), g(\boldsymbol{\theta}_T)) \le 0 \end{aligned} \tag{1}$$

where $\mathcal{L}$ represents the cost of differentiable MSTO, h represents the constraints of the sub-optimization problem.

C. Neural Networks

1) Perception Network: Here we use a similar network structure as in [13]. At each time step, upon receiving a depth measurement $\mathcal{D}$, the perception network encodes this observation into a high-dimensional embedding $\mathbf{o}$ using ResNet-18 , a widely adopted and efficient CNN architecture to extract features while preserving spatial and geometric information for planning [33].

2) Planning Network: The planning network takes into the perception embedding $\mathbf{o}$ and the goal position $\mathbf{G}$ to compute a collision-free key-point path $\boldsymbol{\xi}$. The goal position is first mapped to a higher-dimensional feature embedding using a linear layer and then concatenated with the perception embedding to serve as the input for the planning network. The planning network consists of a CNN and MLP layers with ReLU activation to predict a key-point path $\boldsymbol{\xi}$ and the collision probability η. This path is then optimized by the following differentiable MSTO.

D. Differentiable Minimum Snap Trajectory Optimization (MSTO)

To enforce the dynamic feasibility and ensure the safety of the local path generated by the planner, we develop a MSTO to optimize the predicted key-point path.

1) UAV model: The UAV dynamics can be modeled as the well-known Netwon-Euler method:

$$\begin{aligned} m\ddot{\mathbf{r}} &= -mg\mathbf{z}_W + F\mathbf{z}_B \\ \mathbf{J}\dot{\boldsymbol{\omega}} &= -\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{m} \end{aligned} \tag{2}$$

where m is the UAV mass, $\mathbf{r} = [x, y, z]^\top$ is the UAV position vector in world frame $\mathcal{W}$, g is the gravity, F is the body thrust, $\mathbf{J}$ is the moment of inertia along the body principle axes. $\boldsymbol{\omega}$ is the robot angular velocity, $\mathbf{m} = [m_x, m_y, m_z]$ is the body moment. We use 3-2-1 Euler angles $\boldsymbol{\Omega} = [\phi, \theta, \psi]$ to define the orientation. The system state is defined as: $\mathbf{x} = [\mathbf{r}, \boldsymbol{\Omega}, \dot{\mathbf{r}}, \boldsymbol{\omega}]^\top$, and control input $\mathbf{u} = [F; \mathbf{m}]^\top$.

2) Trajectory Generation: It has been proved that the quadrotor dynamics is *differential flat* with the *flat outputs* as $\boldsymbol{\sigma} = [\mathbf{r}^\top, \psi]^\top \in \mathbb{R}^4$ [34]. In other words, the states and the inputs can be written as algebraic functions of the UAV position and heading and their derivatives. Therefore, the trajectory planning can be simplified as planning for $\boldsymbol{\sigma}$ rather than the full dynamic states, which also ensures the dynamic feasibility. Computation efficiency can thus be improved compared to the traditional nonlinear MPC. In fact, the inputs m_x and m_y appear as functions of the 4^{th} derivatives of the positions (snap), m_z appears in the 2^{nd} derivative of the yaw angle, the TO problem is formulated as minimizing the total snap and control effort along a time horizon t_n given key-point time stamp as $\mathcal{T} = [t_1, ..., t_n]^\top \in \mathbb{R}^n$:

$$\min \int_0^{t_n} \mu_r \left\| \frac{d^4\mathbf{r}}{dt^4} \right\|^2 + \mu_\psi \left(\frac{d^2\psi}{dt^2} \right)^2 dt \tag{3a}$$

$$\text{s.t. } \mathbf{r}^{[3]}(0) = \bar{\mathbf{r}}_0^{[3]}, \mathbf{r}^{[3]}(t_n) = \bar{\mathbf{r}}_T^{[3]},$$

$$\psi^{[1]}(0) = \bar{\psi}_0^{[1]}, \psi^{[1]}(t_n) = \bar{\psi}_T^{[1]}, \tag{3b}$$

$$\boldsymbol{\sigma}_i(t_{i+1}) = \boldsymbol{\xi}_{i+1} \tag{3c}$$

$$\mathbf{r}_i^{[3]}(t_{i+1}) = \mathbf{r}_{i+1}^{[3]}(t_{i+1}), \psi_i^{[1]}(t_{i+1}) = \psi_{i+1}^{[1]}(t_{i+1}) \tag{3d}$$

where μ_r and μ_ψ are constant weight factors to make the integral nondimensional. $\boldsymbol{\sigma}(t)$ are piecewise polynomials of order m over n (Here $m = 7$) time intervals for each dimension of the flat output $\mathbf{r}(t)$ and $\psi(t)$ parameterized by time t. $\boldsymbol{\sigma}_i(t) = [\mathbf{r}_i(t)^\top, \psi_i(t)]^\top$ denotes the polynomials of the i^{th} segment between time interval t_i and t_{i+1}. $(\cdot)^{[k]}$ represents the quantity up to k^{th} order derivative, e.g $\mathbf{r}^{[k]}(t) = [\mathbf{r}(t)^\top, \mathbf{r}'(t)^\top, ..., \mathbf{r}^{(k)}(t)^\top]^\top$. $(\cdot)_0, (\cdot)_T$ are initial and final boundary conditions. Specifically, (3b) constrains the waypoint and its higher-order derivatives at the start and end points, the intermediate waypoints is constrained by (3c), while (3d) ensures the continuity of the position and its higher-order derivatives at the intermediate waypoints. Notice that the hard constraints can be relaxed to inequality constraints to incorporating flight corridor, actuator limit, etc.

In fact, we can represent $\boldsymbol{\sigma}_i = \mathbf{c}_i^\top \boldsymbol{\Omega}(t)$, where $\boldsymbol{\Omega}(t) = [1, t, ..., t^m]^\top$. With the coefficients of all the piecewise

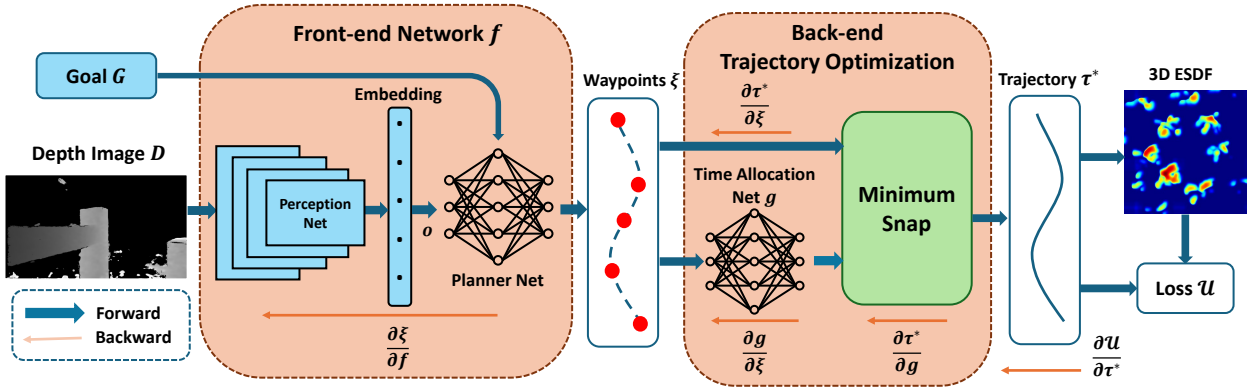


Fig. 2. Overview of the planning pipeline. It consists of two parts, forming a bi-level optimization. The perception and planning network encodes the depth measurements and goal position to predict a key-point path with an collision probability. Then, the low-level trajectory optimization refines the path under specific constraints and cost. A well designed upper-level loss including trajectory cost and time allocation loss updates the network via gradients backpropagated through the trajectory optimizer.

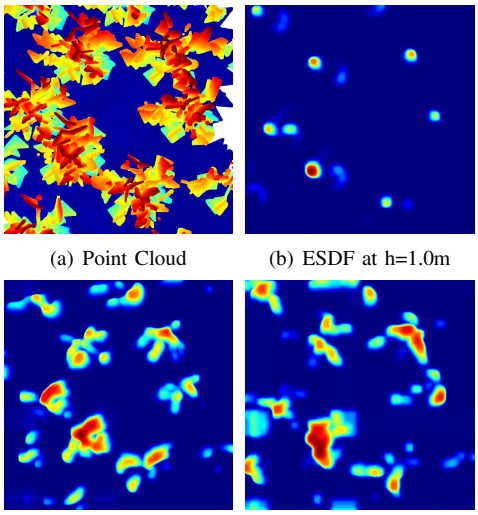
polynomials as the decision variables $\mathbf{c} = [\mathbf{c}_1^\top, ..., \mathbf{c}_n^\top]^\top$, Problem (3) can thus be formulated as a QP problem:

$$\min_{\mathbf{c}} \ \mathbf{c}^\top \mathbf{Q}\mathbf{c} \qquad (4)$$
$$\text{s.t.} \quad \mathbf{A}\mathbf{c} = \mathbf{b}, \quad \mathbf{G}\mathbf{c} \leq \mathbf{h}$$

where $\mathbf{A}$, $\mathbf{b}$ encapsulate the equality constraints, and $\mathbf{G}$, $\mathbf{h}$ describe the inequality constraints. Notice that the key point path $\boldsymbol{\xi}$ is predicted by the front-end network planner $f(\boldsymbol{\theta}_f)$. Problem (4) is the inner/lower level optimization of Problem (1). With optimized $\mathbf{c}^*$, we then evaluate $\boldsymbol{\sigma}(t)$ at control frequency to obtain the planned trajectory $\boldsymbol{\tau}^*$.

3) Time Allocation: In Problem (3), the arrival times at different key points are critical but cannot be predetermined, especially in dynamic environments. A common approach in previous work is to solve a separate iterative optimization problem using gradient descent with backtracking line search [34]. However, this will lead to a tri-level optimization in our setup, significantly degrading real-time performance. To address this, we develop a Time Allocation Net (TAN) $g(\boldsymbol{\theta}_T)$ that predicts the time allocation for each segment of a key-point trajectory. It takes the key-point path $\boldsymbol{\xi}$ as input and output the allocated times $\mathcal{T}$. We use the traditional gradient-descent-based method as in [34] to compute optimal time allocations for training. Then, during inference, our network efficiently predicts the time allocations in real time. In practice, TAN utilizes an MLP followed by a softmax layer, predicting the time allocation percentage for each trajectory segment relative to the total time. Notice that TAN was jointly trained with the front-end network with the supervised reference generated by the optimization method [34], rather than expert demonstration or human annotations.

4) Differentiable Optimization: Since $\boldsymbol{\tau}^*$ is a function of $f(\boldsymbol{\theta}_f)$ and $g(\boldsymbol{\theta}_T)$, and will be passed to the upper level optimization for the final end-to-end training, the key part is to calculate the implicit gradient $\frac{\partial \boldsymbol{\tau}^*}{\partial \xi}$. The traditional unrolling approach maintain the computational graph throughout the entire iteration process, which poses significant computation burden. It may also run into divergent or gradient vanishing. In this work, we employ the implicit function differentiation theorem and leveraging the KKT condition of the Problem (4) at the optimal point to analytically compute the gradients of the parameters. Thus, there is no need for explicit unrolling of the entire iteration process. In practice, we leverage a fast differentiable QP solver for PyTorch as the optimization layer [35], [27].



(a) Point Cloud (b) ESDF at h=1.0m (c) ESDF at h=2.5m (d) ESDF at h=3.5m

Fig. 3. 3D ESDF cost map in the forest environment. (a) Reconstructed point cloud from depth images. (b–d) ESDF slices at different altitudes, color-coded by cost.

E. 3D Cost Map and Training Loss

1) 3D Cost Map: We develop a 3D cost map that provides collision costs to guide UAV behavior by first reconstructing the environment offline from depth images, UAV poses, and camera poses. A 3D Euclidean Signed Distance Field (ESDF) like Figure 3 is then built based on the reconstructed environment. Unlike common approaches that assign distances only in obstacle regions (leaving free space as a constant), we also label the distance in free space to its nearest obstacle boundary. This ensures existence of valid gradient for network learning, as large internal volume of free space in the 3D space would otherwise cause gradient vanishing. Finally, we smooth the ESDF with a Gaussian filter to improve local differentiability. This 3D cost map

enables self-supervision for our BLO framework.

2) Training Loss: Based on the 3D cost map, the upper-level training loss is defined as the weighted summation of obstacle cost $\mathcal{U}^{\mathcal{O}}$, target cost $\mathcal{U}^{\mathcal{G}}$, smoothness cost $\mathcal{U}^{\mathcal{S}}$, and escape cost $\mathcal{U}^{\mathcal{E}}$ in a similar way as [13]. We also add the time allocation cost $\mathcal{U}^{\mathcal{T}}$, e.g.:

$$\begin{aligned}\mathcal{U}(\boldsymbol{\tau}, \boldsymbol{\theta}_T) =& \gamma_1 \mathcal{U}^{\mathcal{O}}(\boldsymbol{\tau}) + \gamma_2 \mathcal{U}^{\mathcal{G}}(\boldsymbol{\tau}) + \gamma_3 \mathcal{U}^{\mathcal{S}}(\boldsymbol{\tau}, \mathbf{p}^R, \mathbf{G}) \\ &+ \gamma_4 \mathcal{U}^{\mathcal{E}}(\boldsymbol{\tau}) + \gamma_5 \mathcal{U}^{\mathcal{T}}(g(\boldsymbol{\theta}_T)) \end{aligned} \tag{5}$$

where $\gamma_k \in \mathbb{R}^+, k = 1, 2, ..., 5$ are weight factors for different terms. Obstacle cost is defined as

$$\mathcal{U}^{\mathcal{O}}(\boldsymbol{\tau}) = \sum_i \text{ESDF}(\mathbf{p}_i), \quad \mathbf{p}_i \in \boldsymbol{\tau} \tag{6}$$

where each point $\mathbf{p}_i$ on trajectory $\boldsymbol{\tau}$ will be projected on our 3D ESDF map to get the cost value $\text{ESDF}(\mathbf{p}_i)$.

Target cost is the Euclidean distance from the final point $\mathbf{p}_f$ on $\boldsymbol{\tau}$ to the target $\mathbf{G}$, e.g.

$$\mathcal{U}^{\mathcal{G}}(\boldsymbol{\tau}) = \|\mathbf{p}_f - \mathbf{G}\|_2 \tag{7}$$

Smoothness cost is defined as the deviation between the segment-wise distances of the generated trajectory and those of a direct linear connection between the start and goal positions:

$$\mathcal{U}^{\mathcal{S}}(\boldsymbol{\tau}) = \sum_{i=1}^{N-1} |\|\mathbf{p}_{i+1} - \mathbf{p}_i\|_2 - \|\hat{\mathbf{p}}_{i+1} - \hat{\mathbf{p}}_i\|_2| \tag{8}$$

where N is the number of waypoints, $\mathbf{p}_i$ is i^{th} waypoint of predicted trajectory, $\hat{\mathbf{p}}_i$ denotes the i^{th} waypoint of the straight line connecting start point and goal point. It ensures evenly distributed length of each trajectory segments between consecutive waypoints. By aligning the segment lengths more closely with those of a straight-line path, the optimization process reduces unnecessary oscillations and ensures a more smooth motion.

Escape loss is to provide the planner with the flexibility to escape from the local minima. Rather than setting large obstacle costs which could cause over-conservative policy, we define the escape loss as

$$\mathcal{U}^{\mathcal{E}}(\boldsymbol{\tau}) = \begin{cases} \text{BCELoss}(\eta, 1.0) & \boldsymbol{\tau} \bigcap \mathcal{O} \neq \varnothing \\ \text{BCELoss}(\eta, 0.0) & \text{otherwise.} \end{cases} \tag{9}$$

where η is a network predicted collision probability for each trajectory. During deployment, the planner will execute the trajectory with $\eta < 0.5$. BCE is the binary cross entropy.

Time allocation cost guides the update of Time Allocation Net during training. Specifically, it is trained to minimize the discrepancy between its predicted time allocation and the optimal time allocation $\mathcal{T}^*$ obtained via iterative optimization with backtracking line search.

$$\mathcal{U}^{\mathcal{T}}(g(\boldsymbol{\theta}_T)) = \frac{1}{n} \sum_{i=1}^{n} \|g(\boldsymbol{\theta}_T)_i - \mathcal{T}_i^*\|^2 \tag{10}$$

F. Bi-level Optimization

The whole pipeline forms a BLO problem, where the differentiable MSTO is the lower-level optimization taking the output of the planner network and generate the trajectory $\boldsymbol{\tau}^*(f(\boldsymbol{\theta}_f), g(\boldsymbol{\theta}_T))$ to optimize the control effort $\mathcal{L}$. Then, the upper-level optimization finds the network parameter $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_T$ to optimize the overall training loss $\mathcal{U}$. Thus, the gradient of the training loss $\mathcal{U}$ must be propagated back through the lower-level optimization, and then the network parameters can be updated using gradient descent:

$$\nabla_{\theta_f} \mathcal{U} = \frac{\partial \mathcal{U}}{\partial \boldsymbol{\tau}^*} \frac{\partial \boldsymbol{\tau}^*}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{\partial \boldsymbol{\theta}_f} + \frac{\partial \mathcal{U}}{\partial \boldsymbol{\tau}^*} \frac{\partial \boldsymbol{\tau}^*}{\partial f} \frac{\partial f}{\partial \boldsymbol{\theta}_f} \tag{11}$$

$$\nabla_{\theta_T} \mathcal{U} = \frac{\partial \mathcal{U}}{\partial \boldsymbol{\tau}^*} \frac{\partial \boldsymbol{\tau}^*}{\partial g} \frac{\partial g}{\partial \boldsymbol{\theta}_T} + \frac{\partial \mathcal{U}}{\partial g} \frac{\partial g}{\partial \boldsymbol{\theta}_T} \tag{12}$$

$$\boldsymbol{\theta}_{f,t+1} = \boldsymbol{\theta}_{f,t} - \alpha \cdot \nabla_{\theta_f} \mathcal{U} \tag{13}$$

$$\boldsymbol{\theta}_{T,t+1} = \boldsymbol{\theta}_{T,t} - \alpha \cdot \nabla_{\theta_T} \mathcal{U} \tag{14}$$

where α is the learning rate during training. Usually, $\frac{\partial \mathcal{U}}{\partial \boldsymbol{\tau}^*}$ and $\frac{\partial \mathcal{U}}{\partial g}$ can be explicitly computed with the expression of $\mathcal{U}$. $\frac{\partial f}{\partial \boldsymbol{\theta}_f}$ and $\frac{\partial g}{\partial \boldsymbol{\theta}_T}$ can also be easily obtained from the computation graph of the network forward pass. The challenging part is to compute $\frac{\partial \boldsymbol{\tau}^*}{\partial \boldsymbol{\theta}_f}$ and $\frac{\partial \boldsymbol{\tau}^*}{\partial \boldsymbol{\theta}_T}$ due to the argmin operation. Thanks to our differentiable optimization developed in Sec. III-D, both of them can be computed without the need of unrolling the entire iteration process of the optimization.

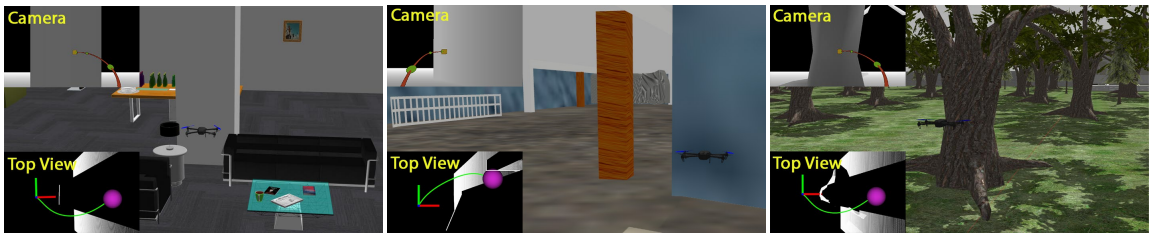
IV. Experiments

We conduct experiments in both simulated and real-world environments to assess the effectiveness of our method. We evaluate navigation tasks in various settings and compare our approach to several baseline methods.

1) Experimental Settings: For simulation, we use the open-source Autonomous Exploration Development Environment [8] and our customized UAV Gazebo simulator, running on a 2.4GHz i9 laptop with an NVIDIA RTX 4060 GPU. Real-world experiments are conducted on our custom-built quadrotor UAV with an NVIDIA Jetson Orin onboard computer running the planning pipeline and a Pixracer autopilot running our customized PX4 firmware. At runtime, trajectory optimizer outputs a time-parameterized sequence (position, velocity, acceleration) as the onboard outer-loop control reference. A front facing Intel RealSense D435 camera provides depth perception at 30 Hz. A motion capture system is used for indoor localization, which can be easily switched to visual inertial odometry [36], [37] for outdoor navigation.

2) Training Data: We collected training data from both simulated and real-world environments. In simulation, we used a joystick to manually fly the UAV in Gazebo simulator, collecting 3 datasets from the environments in Figure 4 (office, garage, and forest), which contains approximately 2000 depth images and corresponding robot poses. We also collected 7 datasets in real-world to account for perception noise and varying lighting conditions. Notice that our method does not require labeled ground-truth data or human expert supervision; its training is fully self-supervised, relying solely on a well-designed loss and the 3D cost map.

3) Training Details: The network follows an encoder–decoder architecture. A ResNet-18 backbone encodes visual observations, while the planning module fuses the



(a) Office (b) Garage (c) Forest

Fig. 4. Illustration of different simulation environments. The purple spheres represent goal points, while the green curve indicates the planned trajectories.

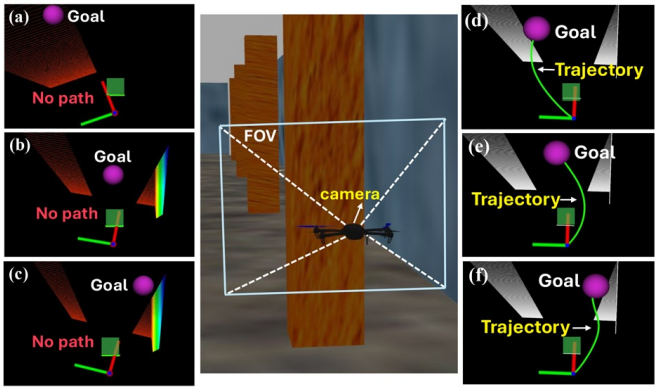
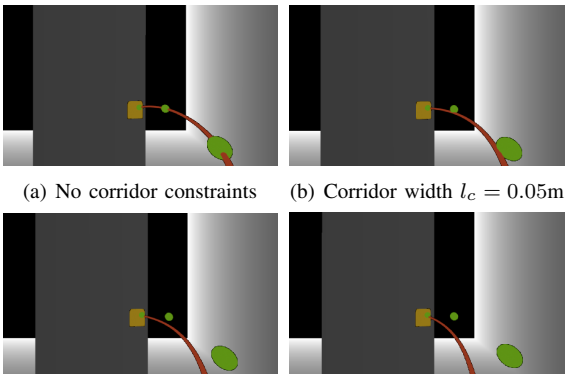


Fig. 5. Navigation in narrow space. (a)(b)(c) MP approach gets stuck in local minima, failing to generate feasible trajectories. (d)(e)(f) Our method is more robust to viewpoint variations, successfully planning trajectories regardless of whether the goal is on left, center, or right behind the obstacle.



(a) No corridor constraints (b) Corridor width $l_c = 0.05$m

(c) Corridor width $l_c = 0.10$m (d) Corridor width $l_c = 0.20$m

Fig. 6. Planning performance when incorporating corridor constraints. The optimized trajectories deviate from the planned key-point path to satisfy the corridor (inequality) constraints.

encoded features with a linearly projected goal embedding. The fused representation is processed by two convolutional layers and a three-layer MLP to generate k 3D waypoints, with a parallel MLP branch predicting a confidence score. The networks are trained with batch size of 16 using the Adam optimizer with a learning rate 0.0001. We used the pre-trained model from iPlanner [13] and trained it for 50 epochs on an NVIDIA 4060 GPU. The entire training process took approximately 70 hours.

4) Baselines: For the non-learning method, we choose two traditional approaches as baselines. **MP** [38]: Generate trajectories by searching over a set of pre-defined dynamically feasible motion segments. **EGO-Planner** [17]: A gradient-based planner without the ESDF construction. For the learning-based method, we adopt **iPlanner** [13] as the baseline, which has already compared with other learning-based methods, including RL. Since iPlanner only plans for 2D motion, we re-train it on our 3D cost map. All methods use a front-facing stereo vision depth camera.

A. Simulation Experiments

1) Overall Performance: We perform simulation experiments in Gazebo as shown in Figure 4. To evaluate the overall planning performance, we first measure the success rate, defined as the UAV reaching its goal from its starting point without any collision along the flight path. Specifically, we sample 60 start-goal pairs per environment. As shown in Table I, our method outperforms iPlanner in all cases because our differentiable MSTO module accounts for UAV dynamics in 3D, producing dynamically feasible trajectories. Although the traditional MP performs slightly better in some scenarios, it can get stuck in local minima from which it cannot escape. For example, as shown in Figure 5, when the vehicle moves directly behind a pillar, its field of view is heavily restricted. In this scenario, target points located within a large area behind the obstacle often lead the MP planner to local minima. In contrast, our method demonstrates more robust, successfully generating collision-free and feasible trajectories. We notice that EGO-Planner's success rate is close to 100% under all environments, our method has more benefits on control effort (will discuss more in next section).

TABLE I

SUCCESS RATE (%) (↑)

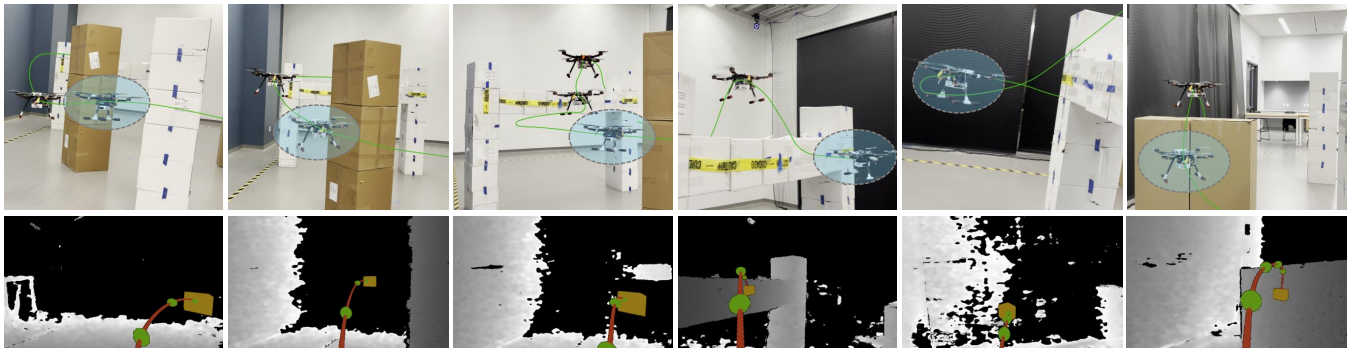
Method	Office	Garage	Forest	Overall
MP	86.7	**96.7**	48.3	77.2
iPlanner	75.0	78.3	63.3	72.2
Ours	**96.7**	91.7	**76.7**	**88.3**

2) Control Effort: We then evaluate and compare the control effort of different methods through the low-level optimization cost $\mathcal{L}$ (integral of the squared Snap) in our differentiable MSTO module, see Table II. Thanks to the deliberate minimization of the snap of 3D trajectory, our method achieves the lowest total snap, demonstrating its superiority in UAV planning over other baseline approaches.

TABLE II

CONTROL EFFORTS AND PLANNING LATENCY

Method	Control Effort (m^2/s^7) (↓)		Latency (ms) (↓)	
	Mean	Std	Mean	Std
MP	97.65	32.52	29.13	4.20
Ego	32.97	7.83	20.08	3.15
iPlanner	58.24	11.95	**7.51**	**0.97**
Ours	**21.16**	**4.86**	13.16	2.28



(a) (b) (c) (d) (e) (f)

Fig. 7. A sequence of snapshots of real-world flight experiments in complex environment. The depth images are captured from the onboard camera of the UAV in blue circles. The green curve showcases the UAV's trajectory. (a-c) The UAV avoids multiple vertical pillars; then (d-e) turns around and avoids a horizontal beam; finally (f) passes another obstacle with different height.

3) Computational Efficiency: Moreover, we compare the computational efficiency of different methods, as shown in Table II. The MP planner incurs significant higher planning latency due to its complex, modularized pipeline. Although iPlanner achieves the lowest latency by relying on a closed-form solution for lower-level trajectory optimization without iterative steps, it thereby loses the ability to enforce physical constraints. In contrast, our method uses iterative optimization for the differentiable MSTO with proper gradient backpropagation, yet still achieves competitive low latency.

4) Inequality Constraints: Thanks to the iterative optimization in our MSTO, we can explicitly incorporate inequality constraints, such as flight corridor or actuator limit. Figure 6 shows a scenario where we incorporate corridor constraints. Our method successfully optimizes the planned path within different required corridors. This capability is crucial for agile flight in constrained environments.

5) Time Allocation: We finally evaluate our TAN by comparing different time allocation strategies: uniform, acceleration–deceleration, and gradient descent with line search [34], see Table III. We can see that the uniform strategy is simple but ignores dynamic feasibility and environmental constraints, performing poorly in complex settings. The Acceleration-Deceleration strategy, usually built on a 5th-order polynomial, ensures smooth acceleration and deceleration with multiple initial and terminal conditions but may fail to optimize time allocation in obstacle-laden or dynamically constrained environments. Gradient Descent with line search cannot satisfy the real-time efficiency and is highly sensitive to the initial guess. In contrast, our approach incorporates iterative line search as the supervision to address these challenges, achieving optimality and computational efficiency for real-time execution.

TABLE III

PERFORMANCE OF DIFFERENT TIME ALLOCATION STRATEGIES

Method	Control Effort (m^2/s^7) (↓)		Latency (ms) (↓)	
	Mean	Std	Mean	Std
Uniform	601.50	131.69	13.13	2.24
5th Order Poly	22.93	3.03	**11.78**	**1.82**
Gradient Descent	26.99	**2.59**	118.19	24.58
Ours	**21.16**	4.86	13.16	2.28

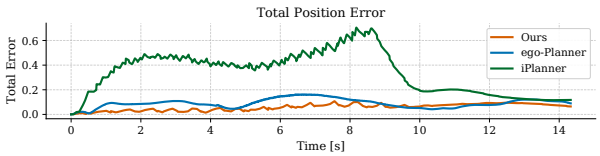


Fig. 8. Trajectory tracking error in real-world experiment

B. Real-World Experiment

To validate the real-time effectiveness of our method, we conducted real-world flight experiments in a different physical room from where the training data was collected, with the depth camera as the only perception. The environment involves multiple vertical pillars, walls, horizontal beams and stacked boxes of varying height. Several narrow corridors are naturally generated between these obstacles. The depth measurements are also noisy during flight. Despite these challenges, our UAV successfully performed continuous obstacle avoidance, see Figure 7. The UAV exhibits smooth left-right and up-down evasive actions to avoid obstacles while maintaining stable trajectories. These results highlight the flexibility and robustness of our approach for obstacle avoidance in constrained 3D environments.

We also evaluate the overall performance of our approach when avoiding obstacles. Table IV shows the quantitative evaluation. Because all methods were deployed with the same low-level controller, tracking error directly reflects planning quality. Thanks to the MSTO, our approach achieves the lowest control effort with 27.93 m^2/s^7 while maintaining competitive tracking accuracy with the mean of 0.0607 m and maximum of 0.1268 m. Figure 8 illustrates a representative tracking run across all three methods.

TABLE IV

TRACKING ERROR AND CONTROL EFFORTS (↓)

Method	Control Effort (m^2/s^7) (↓)		Tracking Error (m) (↓)		
	Mean	Std	Mean	Std	Max
Ego	40.42	18.24	0.0910	0.0358	0.1625
iPlanner	55.21	13.45	0.3422	0.1699	0.7068
Ours	**27.93**	**8.67**	**0.0564**	**0.0269**	**0.1078**

V. CONCLUSION

This paper develops a self-supervised UAV path planning pipeline that integrates a learning-based depth perception

with differentiable trajectory optimization. A 3D cost map was introduced to self-supervise UAV behavior. Additionally, we designed a differentiable minimum snap trajectory optimization module to ensure dynamically feasible paths. A time allocation network improves the real-time efficiency and optimality. Our approach thus improves generalizability and interpretability. Both simulation and real-world experiments demonstrate that our method can enable UAV navigate effectively by avoiding obstacles and execute dynamics feasible trajectory across various environments. Our method achieves 30.90% reduction in control effort compared to the state-of-the-art. Future work includes further testing under diverse operating conditions including dynamic obstacles and degraded lighting conditions.

REFERENCES

[1] S. Zhao, H. Zhang, P. Wang, L. Nogueira, and S. Scherer, "Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8729–8736.

[2] Y. Hu, J. Geng, C. Wang, J. Keller, and S. Scherer, "Off-policy evaluation with online adaptation for robot exploration in challenging environments," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 6, pp. 3780–3787, 2023.

[3] G. He, Y. Janjir, J. Geng, M. Mousaei, D. Bai, and S. Scherer, "Image-based visual servo control for aerial manipulation using a fully-actuated uav," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[4] X. Guo, G. He, J. Xu, M. Mousaei, J. Geng, S. Scherer, and G. Shi, "Flying calligrapher: Contact-aware motion and force planning and control for aerial manipulation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 9, no. 12, pp. 11 194–11 201, 2024.

[5] J. Geng and J. W. Langelaan, "Cooperative transport of a slung load using load-leading control," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 7, pp. 1313–1331, 2020.

[6] C. Cao, H. Zhu, H. Choset, and J. Zhang, "Tare: A hierarchical framework for efficiently exploring complex 3d environments." in *Robotics: Science and Systems*, vol. 5, 2021, p. 2.

[7] G. Best, R. Garg, J. Keller, G. A. Hollinger, and S. Scherer, "Resilient multi-sensor exploration of multifarious environments with a team of aerial robots," in *Robotics: Science and Systems (RSS)*, 2022.

[8] C. Cao, H. Zhu, F. Yang, Y. Xia, H. Choset, J. Oh, and J. Zhang, "Autonomous exploration development environment and the planning algorithms," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8921–8928.

[9] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[10] M. Kulkarni and K. Alexis, "Reinforcement learning for collision-free flight exploiting deep collision encoding," *arXiv preprint arXiv:2402.03947*, 2024.

[11] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[12] C. Wang, K. Ji, J. Geng, Z. Ren, T. Fu, F. Yang, Y. Guo, H. He, X. Chen, Z. Zhan *et al.*, "Imperative learning: A self-supervised neural-symbolic learning framework for robot autonomy," *arXiv preprint arXiv:2406.16087*, 2024.

[13] F. Yang, C. Wang, C. Cadena, and M. Hutter, "iPlanner: Imperative Path Planning," in *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

[14] Q. Li, Z. Chen, H. Zheng, H. He, S. Su, J. Geng, and C. Wang, "ikap: Kinematics-aware planning with imperative learning," *arXiv preprint arXiv:2412.09496*, 2024.

[15] Z. Han, L. Xu, L. Pei, and F. Gao, "Dynamically feasible trajectory generation with optimization-embedded networks for autonomous flight," *IEEE Robotics and Automation Letters*, vol. 10, no. 10, pp. 9995–10 002, 2025.

[16] B. Zhou, J. Pan, F. Gao, and S. Shen, "Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1992–2009, 2021.

[17] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.

[18] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1526–1545, 2020.

[19] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[20] J. Tordesillas and J. P. How, "Mader: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, 2021.

[21] M. Jacquet and K. Alexis, "N-mpc for deep neural network-based collision avoidance exploiting depth images," *arXiv preprint arXiv:2402.13038*, 2024.

[22] Y. Wu, X. Sun, I. Spasojevic, and V. Kumar, "Deep learning for optimization of trajectories for quadrotors," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2479–2486, 2024.

[23] 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 Robotics and Automation Letters*, vol. 9, no. 7, pp. 6083–6090, 2024.

[24] Z. Teed and J. Deng, "Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras," *Advances in neural information processing systems*, vol. 34, pp. 16 558–16 569, 2021.

[25] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Pontryagin differentiable programming: An end-to-end learning and control framework," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7979–7992, 2020.

[26] Y. Zhang, Y. Hu, Y. Song, D. Zou, and W. Lin, "Back to newton's laws: Learning vision-based agile flight via differentiable physics," *arXiv preprint arXiv:2407.10648*, 2024.

[27] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 136–145.

[28] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *Advances in neural information processing systems*, vol. 31, 2018.

[29] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 777–14 784.

[30] J. Pan, Z. Ye, X. Yang, X. Yang, W. Liu, L. Wang, and J. Bian, "Bpqp: A differentiable convex optimization framework for efficient end-to-end learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 77 468–77 493, 2025.

[31] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson *et al.*, "Theseus: A library for differentiable nonlinear optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 3801–3818, 2022.

[32] C. Wang, D. Gao, K. Xu, J. Geng, Y. Hu, Y. Qiu, B. Li, F. Yang, B. Moon, A. Pandey *et al.*, "Pypose: A library for robot learning with physics-based optimization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 22 024–22 034.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

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

[35] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, 2019.

[36] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[37] T. Qin and S. Shen, "Online temporal calibration for monocular visual-inertial systems," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3662–3669.

[38] J. Zhang, C. Hu, R. G. Chadha, and S. Singh, "Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1300–1313, 2020.