

Learning-Initialized Trajectory Planning in Unknown Environments

Yicheng Chen¹, Jinjie Li¹, Wenyuan Qin², Yongzhao Hua^{2*}, Xiwang Dong^{1,2}, and Qingdong Li¹

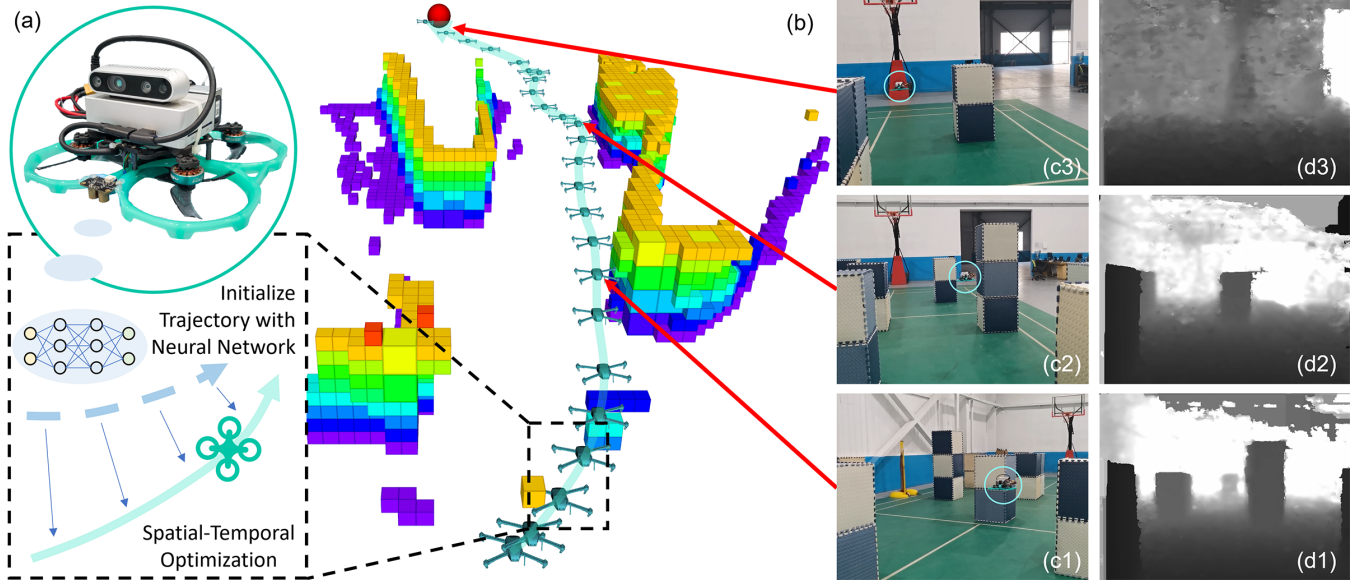


Fig. 1. A real-world demonstration of fully autonomous flight using the proposed Learning-Initialized Trajectory Planner (LIT-Planner). The drone has no prior knowledge of the environment, and the entire software stack runs onboard in real-time. (a) Our quadrotor platform. (b) The map built during the flight. (c1)-(c3) Three snapshots during the flight. (d1)-(d3) The corresponding depth images.

Abstract—Autonomous flight in unknown environments requires precise planning for both the spatial and temporal profiles of trajectories, which generally involves nonconvex optimization, leading to high time costs and susceptibility to local optima. To address these limitations, we introduce the Learning-Initialized Trajectory Planner (LIT-Planner), a novel approach that guides optimization using a Neural Network (NN) Planner to provide initial values. We first leverage the spatial-temporal optimization with batch sampling to generate training cases, aiming to capture multimodality in trajectories. Based on these data, the NN-Planner maps visual and inertial observations to trajectory parameters for handling unknown environments. The network outputs are then optimized to enhance both reliability and explainability, ensuring robust performance. Furthermore, we propose a framework that supports robust online replanning with tolerance to planning latency. Comprehensive simulations validate the LIT-Planner’s time efficiency without compromising trajectory quality compared

to optimization-based methods. Real-world experiments further demonstrate its practical suitability for autonomous drone navigation.

Video: <https://youtu.be/V15ZPjLziQI>

I. INTRODUCTION

Spatial-temporal motion planning aims to generate collision-free trajectories with refinement in both energy and time. This has been a challenging problem for autonomous drones in unknown environments because it is required to precisely handle the complexity from both the environment and the drone dynamics, while ensuring a real-time performance for high-frequency replanning.

Optimization-based approaches [1]–[3] are considered as one of the mainstream solutions, where the planning task is formulated as an optimization problem that incorporates different constraints and costs using one objective function. However, the nonconvex nature of the optimization problem, owing to its high-dimensional variables and intricate constraints, often results in convergence to local optima. Furthermore, computational costs are highly sensitive to the initial values used to initialize optimization [4], [5]. To provide a proper initial guess, a straightforward approach is to employ path planning methods such as A* [6] or Rapidly-exploring Random Tree (RRT) [7] to generate sparse waypoints for trajectory optimization. However, these path planning methods do not consider drone dynamics,

This work was supported by the Science and Technology Innovation 2030-Key Project of “New Generation Artificial Intelligence” under Grant 2020AAA0108200, the National Natural Science Foundation of China under Grants U2241217, 62103023, 61973013, 62103016, the Beijing Natural Science Foundation under Grant 4232046, and the Young Elite Scientists Sponsorship Program by CAST under Grant 2021QNRC001.

¹ Y. Chen, J. Li, X. Dong, and Q. Li are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. {yicheng, lijinnie, xwdong, liqingdong}@buaa.edu.cn

² W. Qin, Y. Hua (*Corresponding author), and X. Dong are with the Institute of Artificial Intelligence, Beihang University, Beijing 100191, China. {wyqin, yongzhaohua, -}@buaa.edu.cn

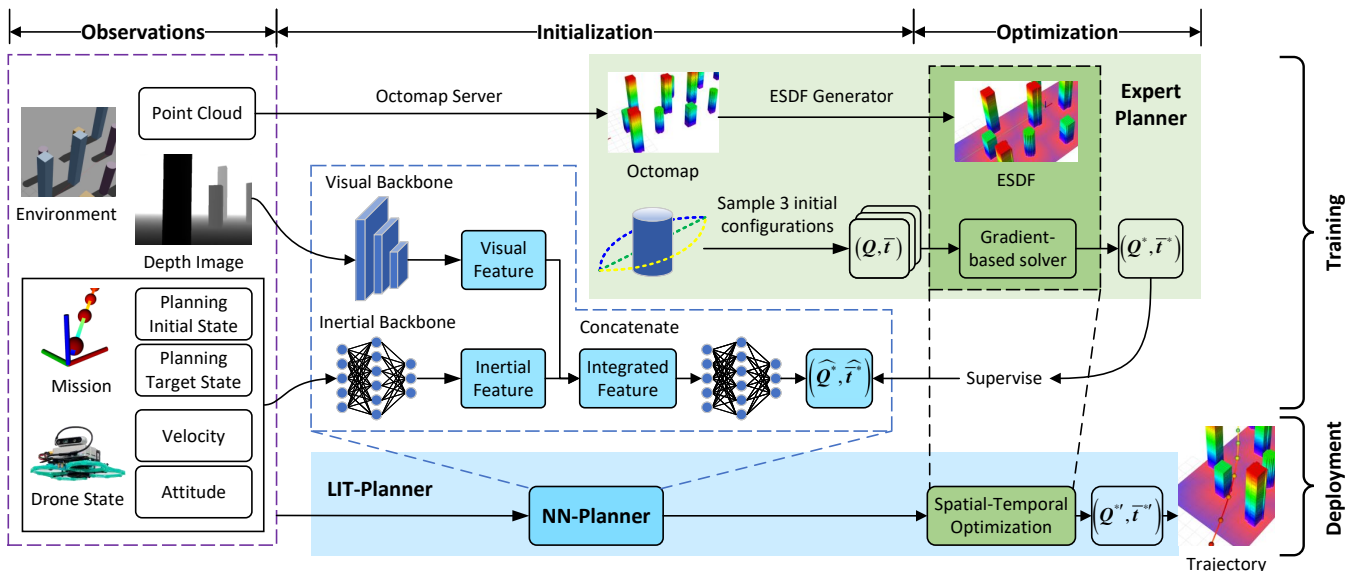


Fig. 2. **System Overview.** The LIT-Planner leverages a neural network to generate high-quality initial trajectories from onboard observations and subsequently conducts spatial-temporal optimization on the neural network’s output. The NN-Planner is trained using supervised learning, with training cases provided by an expert planner based on a standard mapping-planning-control stack.

potentially resulting in unreasonable initial values for online replanning. Additionally, the time required for these methods makes them unsuitable for high-frequency replanning [8]. Another method involves the sampling of multiple initial configurations, performing separate optimizations for each, and retaining the best solution [9]. Nevertheless, this approach inevitably escalates planning costs linearly with the number of initial configurations sampled, which is impractical for micro aerial robots with limited onboard computing resources. Therefore, we need an approach to provide reasonable trajectory initialization with minimal time cost.

On the other hand, learning-based approaches have gained widespread attention in recent years due to the neural network’s ability to model nonlinear mappings [10] and perform fast inference [11]. Researchers have employed supervised learning and reinforcement learning for various tasks, including obstacle avoidance [12]–[14], flying in the wilderness [15], formation flight [16], and autonomous racing [17], [18]. For instance, Tordesillas et al. [12] present a learning-based method for perception-aware trajectory planning. However, the proposed method assumes that the drone has perfect knowledge of the obstacle’s trajectory, which leaves a gap for real-world applications. Song et al. [13] perform hardware-in-the-loop simulations on their learned policy, but the neural network may experience failures, which requires an abrupt takeover with a state-based controller. Loquercio et al. [15] use a neural network to enable high-speed flight. However, the method exhibits limited success rates in some cases since the difficulties to ensure the trajectory’s temporal consistency over a long time horizon. The challenge is then how to generate reliable and explainable spatial-temporal trajectories.

To address these limitations, we propose a learning-based approach to initialize trajectories and further refine them using an optimization approach. Several existing works have

employed neural networks to initialize trajectory optimization for manipulator arms [4], [5] or mobile robots [19]. However, our research distinguishes itself from these studies by the additional challenge of understanding unknown environments in real time. In comparison to pure optimization-based methods, our approach significantly reduces planning time while maintaining trajectory quality. As opposed to end-to-end learning-based methods, our approach benefits from explainability due to the integration of optimization techniques into the solution process. We outline the contributions of this paper as follows:

- A Learning-Initialized Trajectory Planner (LIT-Planner) that incorporates a Neural Network Planner into an optimization-based approach to reduce planning cost and provide explainable high-quality trajectories.
- A robust online trajectory planning framework that enables autonomous flight in unknown environments with tolerance to planning latency.
- A set of simulations and real-world experiments validating the efficiency of our method compared to existing approaches and demonstrating its feasibility for real-world applications.

II. METHODOLOGY

A. System Overview

Fig. 2 shows an overview of the system. We first parameterize the trajectories using MINCO representation (Section II-B), where a polynomial trajectory is completely defined by the variables (Q, \bar{t}) . The LIT-Planner performs spatial (Q)-temporal (\bar{t}) optimization (Section II-C) on the trajectories with initial values generated by a NN-Planner (Section II-E). The NN-Planner is trained using supervised learning. We leverage a powerful yet computational-expensive expert planner (Section II-D) to provide the training data. In addition, we introduce an online replanning framework in Section II-F.

B. Trajectory Parameterization

We denote scalars in regular $x \in \mathbb{R}$ or $X \in \mathbb{R}$, vectors in bold lowercase $\mathbf{x} \in \mathbb{R}^n$, and matrices in bold uppercase $\mathbf{X} \in \mathbb{R}^{n \times m}$. In addition, we denote the time variable in t , time points in t_i , time intervals in $\bar{t}_i := t_i - t_{i-1}$, and then $t_i = \sum_{j=1}^i \bar{t}_j$. The coordinate system contains the world frame $\{\mathcal{W}\}$ (ENU: X East, Y North, Z Up) and the body frame $\{\mathcal{B}\}$ (FLU: X Forward, Y Left, Z Up). The geometric variables are in the world frame if not specifically annotated.

We represent a D -dimensional M -piece t -indexed polynomial trajectory $\mathbf{p}(t)$ through MINCO [3], a polynomial trajectory class to perform a spatial-temporal deformation of the flat output trajectory [20]:

$$\begin{aligned} \mathfrak{T}_{\text{MINCO}} = \{ & \mathbf{p}(t) : [0, t_M] \rightarrow \mathbb{R}^D \mid \mathbf{C} = \mathcal{C}(\mathbf{Q}, \bar{\mathbf{t}}), \\ & \mathbf{Q} \in \mathbb{R}^{D \times (M-1)}, \bar{\mathbf{t}} \in \mathbb{R}_{>0}^M \}, \end{aligned} \quad (1)$$

where \mathbf{C} are trajectory coefficients, $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_{M-1}]$ represent the intermediate waypoints, $\bar{\mathbf{t}} = [\bar{t}_1, \dots, \bar{t}_M]^T$ are the time allocated for each piece, and t_M is the total time. Given a set of $(\mathbf{Q}, \bar{\mathbf{t}})$, we can obtain a unique trajectory of minimum control effort in polynomial form through the mapping $\mathcal{C}(\cdot)$. This mapping is achieved by solving a boundary-intermediate value problem described in [3], which returns the coefficients $\mathbf{C} = [\mathbf{C}_1^T, \dots, \mathbf{C}_M^T]^T$ with linear time and space complexity. Based on $(\mathbf{C}, \bar{\mathbf{t}})$, for a system of S order integrator chain [3], a polynomial trajectory $\mathbf{p}(t)$ of $N = 2S - 1$ degree can be defined as

$$\mathbf{p}(t) = \begin{cases} \mathbf{p}_1(t - t_0), & \text{if } t \in [t_0, t_1) \\ \dots \\ \mathbf{p}_i(t - t_{i-1}), & \text{if } t \in [t_{i-1}, t_i) \\ \dots \\ \mathbf{p}_M(t - t_{M-1}), & \text{if } t \in [t_{M-1}, t_M) \end{cases}, \quad (2)$$

$$\mathbf{p}_i(t) = \mathbf{C}_i^T \cdot \boldsymbol{\beta}(t), \quad \forall t \in [0, \bar{t}_i], \quad (3)$$

where $\mathbf{C}_i = [\mathbf{c}_{i,1}, \dots, \mathbf{c}_{i,D}] \in \mathbb{R}^{(N+1) \times D}$ is the coefficient matrix of the i^{th} piece, $\boldsymbol{\beta}(t) := [1, t, \dots, t^N]^T \in \mathbb{R}^{N+1}$ is the natural basis.

Based on the above parameterization, MINCO's objective $\mathcal{H}(\mathbf{Q}, \bar{\mathbf{t}})$ can be computed as

$$\mathcal{H}(\mathbf{Q}, \bar{\mathbf{t}}) := K = \mathcal{K}(\mathcal{C}(\mathbf{Q}, \bar{\mathbf{t}}), \bar{\mathbf{t}}). \quad (4)$$

For any second-order continuous cost function $\mathcal{K}(\mathbf{C}, \bar{\mathbf{t}})$, we can compute $\partial \mathcal{H} / \partial \mathbf{Q}$ and $\partial \mathcal{H} / \partial \bar{\mathbf{t}}$ from $\partial \mathcal{K} / \partial \mathbf{C}$ and $\partial \mathcal{K} / \partial \bar{\mathbf{t}}$ [3], and use gradient descent to optimize the objective.

C. Spatial-Temporal Optimization

We construct the trajectory problem in the form of unconstrained optimization:

$$\min_{\mathbf{Q}, \bar{\mathbf{t}}} \mathcal{K}(\mathcal{C}(\mathbf{Q}, \bar{\mathbf{t}}), \bar{\mathbf{t}}) := \sum_x \omega_x K_x, \quad (5)$$

where subscripts $x = \{e, t, o, d\}$ stands for *Control Effort* (e), *Trajectory Time* (t), *Obstacle Avoidance* (o), and *Dynamical Feasibility* (d). ω_x are the weights for different costs.

The cost function K_x and its gradients are composed of the cost and gradients of each trajectory piece:

$$K_x = \sum_{i=1}^M K_x^i, \quad (6)$$

$$\frac{\partial K_x}{\partial \mathbf{C}} = \left[\frac{\partial K_x^1}{\partial \mathbf{C}_1}, \frac{\partial K_x^2}{\partial \mathbf{C}_2}, \dots, \frac{\partial K_x^M}{\partial \mathbf{C}_M} \right]^T, \quad (7)$$

$$\frac{\partial K_x}{\partial \bar{\mathbf{t}}} = \left[\frac{\partial K_x^1}{\partial \bar{t}_1}, \frac{\partial K_x^2}{\partial \bar{t}_2}, \dots, \frac{\partial K_x^M}{\partial \bar{t}_M} \right]^T. \quad (8)$$

1) *Control Effort* K_e : The control effort and its gradient of the i^{th} piece trajectory can be written as

$$K_e^i = \int_0^{\bar{t}_i} \left\| \mathbf{p}_i^{(S)}(t) \right\|^2 dt, \quad (9)$$

$$\frac{\partial K_e^i}{\partial \mathbf{C}_i} = 2 \left(\int_0^{\bar{t}_i} \boldsymbol{\beta}^{(S)}(t) \cdot \boldsymbol{\beta}^{(S)}(t)^T dt \right) \mathbf{C}_i, \quad (10)$$

$$\frac{\partial K_e^i}{\partial \bar{t}_i} = \sum_{j=1}^D \mathbf{c}_{i,j}^T \cdot \boldsymbol{\beta}^{(S)}(\bar{t}_i). \quad (11)$$

2) *Trajectory Time* K_t : We minimize the total time of the trajectory $K_t = \sum_{i=1}^M \bar{t}_i$. The gradients are given by $\partial K_t / \partial \mathbf{C} = \mathbf{0}$ and $\partial K_t / \partial \bar{\mathbf{t}} = \mathbf{1}_{M,1}$.

3) *Obstacle Avoidance* K_o and *Dynamical Feasibility* K_d : The constraints for obstacle avoidance and dynamical feasibility are time-integral constraints, which require sampling on the trajectory to derive the associated penalties. We first give the general form of time-integral constraints, then give the specific forms of K_o and K_d . For the i^{th} piece trajectory, the penalty for violation of time-integral constraints can be calculated as

$$\begin{aligned} K_x^i &= \mathcal{K}_x^i(\mathbf{C}_i, \bar{t}_i, k_i) \\ &= \frac{\bar{t}_i}{k_i} \sum_{j=0}^{k_i} \gamma_j \cdot \max \left(J_x \left(\mathbf{C}_i, \bar{t}_i, \frac{j}{k_i} \right), \mathbf{0} \right)^3, \end{aligned} \quad (12)$$

$$J_x \left(\mathbf{C}_i, \bar{t}_i, \frac{j}{k_i} \right) = J_x(\mathbf{p}(t')), \quad t' = \frac{j}{k_i} \cdot \bar{t}_i, \quad (13)$$

where $x = \{o, d\}$, k_i is the number of points sampled on the i^{th} piece trajectory, $(\gamma_0, \gamma_1, \dots, \gamma_{k_i-1}, \gamma_{k_i}) = (1/2, 1, \dots, 1, 1/2)$ are coefficients following the trapezoidal rule [21]. The specific form of $J_x(\cdot)$ is related to the type of constraints, which will be discussed below. The gradients can be calculated by

$$\frac{\partial K_x^i}{\partial \mathbf{C}_i} = \frac{\partial K_x^i}{\partial J_x} \frac{\partial J_x}{\partial \mathbf{C}_i}, \quad (14)$$

$$\frac{\partial K_x^i}{\partial \bar{t}_i} = \frac{K_x^i}{\bar{t}_i} + \frac{\partial K_x^i}{\partial J_x} \frac{\partial J_x}{\partial t'} \frac{\partial t'}{\partial \bar{t}_i}, \quad (15)$$

$$\frac{\partial K_x^i}{\partial J_x} = 3 \frac{\bar{t}_i}{k_i} \sum_{j=0}^{k_i} \gamma_j \max \left(J_x \left(\mathbf{C}_i, \bar{t}_i, \frac{j}{k_i} \right), \mathbf{0} \right)^2, \quad (16)$$

$$\frac{\partial t'}{\partial \bar{t}_i} = \frac{j}{k_i}. \quad (17)$$

To obtain K_x^i and its gradients for different types of time integral constraints, we just need to construct $J_x(\cdot)$ for each sampling point on the trajectory and calculate $\partial J_x / \partial \mathbf{C}_i$ and $\partial J_x / \partial t'$. Obstacle avoidance constraints and dynamical constraints are analyzed respectively below.

3.1) *Obstacle Avoidance J_o* : We formulate the penalty and its gradients using the Euclidean Signed Distance Fields (ESDF). For every point on the map, the ESDF provides the distance from this point to its nearest obstacle $d(\mathbf{p}(t'))$ and the gradient $\nabla d(\mathbf{p}(t'))$. Thus, $J_o(\mathbf{p}(t'))$ can be defined as

$$J_o(\mathbf{p}(t')) = \begin{cases} d_{thr} - d(\mathbf{p}(t')), & \text{if } d(\mathbf{p}(t')) < d_{thr} \\ 0, & \text{if } d(\mathbf{p}(t')) \geq d_{thr} \end{cases}. \quad (18)$$

For $J_o(\mathbf{p}(t')) > 0$, the gradients can be calculated as

$$\frac{\partial J_o}{\partial \mathbf{C}_i} = -\beta(t') \cdot \nabla d(\mathbf{p}(t'))^T, \quad (19)$$

$$\frac{\partial J_o}{\partial t'} = -\nabla d(\mathbf{p}(t'))^T \cdot \dot{\mathbf{p}}(t'). \quad (20)$$

3.2) *Dynamical Feasibility J_d* : Here we imply constraints on the maximum velocity on the trajectory as an example, the principle of acceleration and higher-order dynamical constraints is the same. Assuming that the maximum allowed velocity is v_m , we define

$$J_d = \dot{\mathbf{p}}(t')^2 - v_m^2, \quad (21)$$

$$\frac{\partial J_d}{\partial \mathbf{C}_i} = 2\dot{\beta}(t') \cdot \dot{\mathbf{p}}(t')^T, \quad (22)$$

$$\frac{\partial J_d}{\partial t'} = 2\dot{\beta}(t')^T \cdot \mathbf{C}_i \cdot \dot{\mathbf{p}}(t')^T. \quad (23)$$

So far, for the trajectory optimization problem (5), we have presented detailed approaches to calculate the costs and gradients. We use L-BFGS to solve the problem. To avoid nonpositive values of \bar{t}_i , we introduce a proxy variable τ and modify \bar{t}_i as

$$\bar{t}_i = \frac{\bar{t}_{\max} - \bar{t}_{\min}}{1 + e^{-\tau_i}} + \bar{t}_{\min}, \quad i = 1, 2, \dots, M. \quad (24)$$

This establishes the mapping $\tau_i \in (-\infty, +\infty) \rightarrow \bar{t}_i \in (\bar{t}_{\min}, \bar{t}_{\max})$, where \bar{t}_{\max} and \bar{t}_{\min} are the upper and lower bounds of the duration of the i^{th} piece trajectory that can be set according to the users' demand.

D. Expert Planner and Multimodality

To provide training data for the NN-Planner, we implement an expert planner based on the optimization method [22] presented in Section II-C. In each replanning of a flight, it takes in the local initial state $\mathbf{S}_{\text{init}} = [\mathbf{p}_{\text{init}}, \mathbf{v}_{\text{init}}]^T$ and the target state $\mathbf{S}_{\text{target}} = [\mathbf{p}_{\text{target}}, \mathbf{v}_{\text{target}}]^T$, samples three different initial trajectories (one straight line and two curves that deform towards both sides based on the straight one), performs optimization from each configuration, and outputs the optimized result $(\mathbf{Q}^*, \bar{\mathbf{t}}^*)$ with the lowest objective.

The objective function of the optimization problem is not in an analytical form, which introduces a nonconvex nature to this problem and significantly influences both the solution process and the quality of the resultant solution.

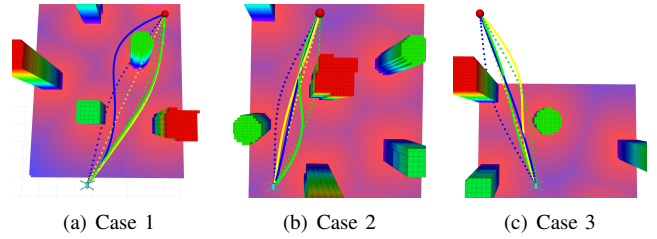


Fig. 3. **Illustration of the multimodality in three cases of local planning.** In each case, the optimization of expert planner starts from three different initial trajectories and converges toward different results.

TABLE I: TRAJECTORY COST AND COMPUTATION TIME OF OPTIMIZATION STARTING FROM DIFFERENT INITIAL VALUES

Case	Metric	Trajectory Color		
		Yellow	Green	Blue
Case 1	Trajectory Cost	12.91	12.93	14.75
	Computation Time (s)	0.44	0.27	0.49
Case 2	Trajectory Cost	10.87	26.62	10.83
	Computation Time (s)	0.23	0.26	0.32
Case 3	Trajectory Cost	12.76	12.59	11.90
	Computation Time (s)	0.25	0.40	0.36

This sensitivity to the initial optimization value is illustrated in Fig. 3, and its influence on both the quality of results and the computational cost of the solving process is quantified in Table I. These results intuitively demonstrate the necessity of introducing a suitable initialization method.

E. Initialization: Neural Network Planner

To mitigate the influence of the aforementioned nonconvexity on trajectory optimization, the NN-Planner's primary purpose is to capture the potential high-quality trajectories from raw sensory observations.

1) *Structure*: The NN-Planner takes in an observation

$$\mathbf{O} = (\mathbf{I}, {}^B \mathbf{v}_{\text{drone}}, {}^W \mathbf{R}, {}^B \mathbf{S}_{\text{init}}, {}^B \mathbf{S}_{\text{target}}), \quad (25)$$

where $\mathbf{I} \in \mathbb{R}^{640 \times 480}$ is the depth image, ${}^B \mathbf{v}_{\text{drone}} \in \mathbb{R}^3$ is the drone's velocity in body frame, ${}^W \mathbf{R} \in \mathbb{R}^{3 \times 3}$ is the drone's attitude (rotation from body frame to world frame), ${}^B \mathbf{S}_{\text{init}} \in \mathbb{R}^{2 \times 3}$ and ${}^B \mathbf{S}_{\text{target}} \in \mathbb{R}^{2 \times 3}$ are the local initial state and target state in body frame, respectively.

We have developed a neural network tailored for processing the given observation and producing the output $({}^B \hat{\mathbf{Q}}, {}^B \hat{\bar{\mathbf{t}}})$. This observation encompasses both visual and inertial information and is processed through two distinct branches within the neural network, as illustrated in Fig. 2. For the visual information \mathbf{I} , we utilize a pretrained ResNet-18 [23] attached with a fully-connected layer to generate the visual feature in \mathbb{R}^{24} . In the case of the inertial information $({}^B \mathbf{v}_{\text{drone}}, {}^W \mathbf{R}, {}^B \mathbf{S}_{\text{init}}, {}^B \mathbf{S}_{\text{target}})$, we first flatten each of these elements and concatenate them to form a \mathbb{R}^{24} vector. This vector is then processed by a four-layer perceptron with [48, 24, 24] hidden nodes to extract the inertial feature. Subsequently, the visual feature and inertial feature are concatenated and passed through another four-layer-perceptron with [48, 96, 96] hidden nodes to generate the

output vector. All multi-layer perceptrons employ the Leaky ReLU activation function. Finally, the estimated $({}^B\hat{Q}, {}^B\hat{t})$ are derived from the output vector and transformed into the world frame.

2) *Data Acquisition and Training:* We train the NN-Planner using supervised learning. We use the expert planner to collect training data in a self-built simulation environment, where each planning operation yields a training data entry comprising \mathcal{O} and the corresponding reference output $({}^B\hat{Q}^*, {}^B\hat{t}^*)$ from the expert planner. We train the NN-Planner using Mean Squared Error (MSE) as the loss function and Adam [24] as the optimizer.

F. Online Replanning Framework

Autonomous flight in unknown environments requires online replanning because of the limited perception horizon. We design a robust online replanning framework that enables tolerance to planning latency. Within this framework, the planner maintains an evolving trajectory for the tracker to follow, as outlined in Fig. 4.

In a replanning at time t_x , the planner first selects the local target state $\mathcal{S}_{\text{target}}$ and the local initial state $\mathcal{S}_{\text{init}}$. $\mathcal{S}_{\text{target}}$ comprises a collision-free point p_{init} at a specific distance ahead and a desired velocity v_{init} . For the local initial state, the planner retrieves the state at $t_x + \Delta T_f$ along the existing trajectory as $\mathcal{S}_{\text{init}}$, where ΔT_f is the foreseeing horizon. The planner then generates a trajectory connecting them. This newly generated trajectory supersedes the portion of the existing trajectory beyond the time $t_x + \Delta T_f$. After a interval ΔT_r comes the next round of replanning, and the existing trajectory beyond the time $t_x + \Delta T_f + \Delta T_r$ is updated. The replanning interval ΔT_r can be a constant value, typically set as the upper bound of the estimated planning time, or a variable value based on real-time measurements recorded during each planning iteration.

This planning framework has two major advantages: 1) **Tolerance to planning latency:** The tracker continually accesses the real-time state of the trajectory, while the point at which the trajectory is updated becomes accessible to the tracker after the foreseeing horizon ΔT_f . Consequently, as long as the planning process can be completed within the

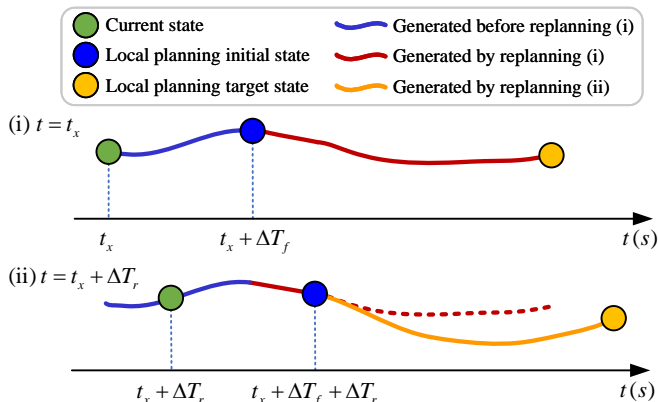


Fig. 4. Illustration of two consecutive replanning within the online replanning framework.

ΔT_f timeframe, the tracking operation proceeds seamlessly.

2) **Decoupling between planning and control frequencies:** The tracking controller possesses the capability to publish desired control commands at a very high frequency, independent of the frequency at which the planner updates the trajectory.

The feasibility of this framework is based on the following assumption. In each planning, the portion of the newly generated trajectory within the time window of ΔT_f is dependable. This assumption remains reasonable, provided that ΔT_f is not excessively extended.

III. EXPERIMENTS

In this section, we present the results of both simulations and real-world experiments to evaluate the performance of our algorithm.

A. Simulations

We perform simulations in Gazebo with PX4 software-in-the-loop (SITL). We build three different scenes (poles, forest, and bricks) to perform simulations. In each scene, the drone flies using onboard observations without prior information about the environment. The network in LIT-Planner is deployed with ONNX-Runtime. The trajectory tracking controller accesses the planned trajectory to obtain the desired commands of position, velocity, and acceleration. The desired yaw angle, as an additional degree of freedom, is set tangential to the trajectory for perception-awareness to the environments. These desired commands are published to the PX4 controller at a frequency of 60 Hz. The parameters used for the algorithms are summarized in Table II.

1) *The Effectiveness of Neural Network:* To verify the effectiveness of neural network for initialization, we conduct comparisons between the expert planner and the LIT-Planner, and evaluate the performance using the following two metrics: 1) **Average replanning time:** In each run, the drone performs dozens of replanning. The average replanning time is defined as the arithmetic mean of all replanning time within one run. This metric reflects the computational cost of the planner. 2) **Weighted cost:** In each run, we record the state of drone every 0.5 seconds. The weighted cost consists trajectory length, penalties for obstacle collision (from (12), (13), (18)), and violations of dynamical feasibility (from (12), (13), (21)). The weights between them are set to $[1, 1, 1]$. This metric reflects the quality of the trajectory.

We conduct ten repeated experiments for both the expert planner and the LIT-Planner in each scene. Fig. 6 presents the comparison results, demonstrating that the LIT-Planner achieves similar trajectory quality while significantly reducing replanning time in comparison to the expert planner.

TABLE II: MAIN PARAMETERS OF THE PLANNERS

Parameter	Value	Parameter	Value
M	3	\bar{t}_{\min}	0.5 s
D	2	\bar{t}_{\max}	5.0 s
S	3	ΔT_r	1.0 s
v_{\max}	1.0 m/s	ΔT_f	1.0 s
d_{thr}	0.7 m	Weights between costs	$[1, 1, 10000, 1]$

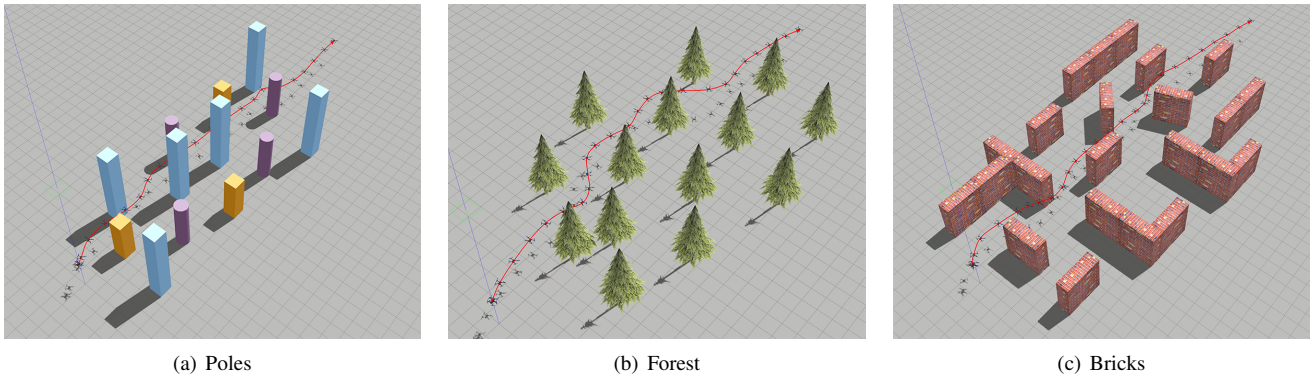


Fig. 5. **Three scenes for simulation comparisons.** In ‘Poles’ and ‘Forest’, the target point is located 30 meters ahead of the starting point, in ‘Bricks’, the target point is located 35 meters ahead of the starting point. The red curves show trajectories flown using LIT-Planner.

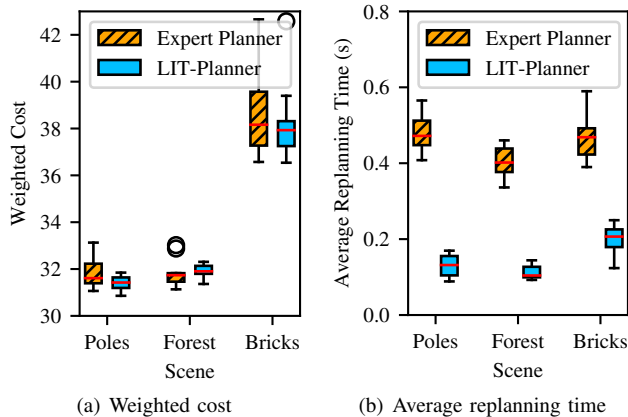


Fig. 6. **Comparison of the weighted cost and average replanning time.** LIT-Planner is able to generate trajectories of similar quality compared with the expert planner, but with much less replanning time.

This underscores the effectiveness of the neural network in providing reasonable initial trajectories.

2) *Tolerance to Planning Latency:* The proposed online replanning framework (Section II-F) is designed to tolerate planning latency owing to the foreseeing horizon ΔT_f . To validate its effectiveness, we intentionally introduce an additional planning latency of 0.8 seconds and conduct ten repeated comparisons with and without ($\Delta T_f = 0$ s) the foreseeing horizon in the three scenes, respectively. In each run, we record the desired position and velocity commands, as well as the actual position and velocity of the drone every 0.1 seconds, and calculate the root-mean-square tracking error from the recorded data. We present the average RMSE of the ten experiments in Table III, which proves the incorporation of ΔT_f to be effective. In the absence of the foreseeing horizon, the drone’s tracking performance falters, as the current desired position may undergo abrupt changes, making trajectory tracking more difficult, which is reflected by a larger RMSE.

B. Real-World Experiments

For real-world experiments, we deploy the LIT-Planner on a drone equipped with a NVIDIA Jetson Orin NX as computational unit, an Intel RealSense D435 depth camera, and a Holybro Kakute H7 mini flight controller. The drone runs VINS-Fusion [25] for state estimation. The neural network

TABLE III: TRACKING ERROR COMPARISONS WITH AND WITHOUT THE FORESEEING HORIZON

Scene	Metric	Average RMSE	
		$\Delta T_f = 0$ s	$\Delta T_f = 1$ s
Poles	Position Error (m)	0.23	0.13
	Velocity Error (m/s)	0.21	0.08
Forest	Position Error (m)	0.28	0.16
	Velocity Error (m/s)	0.26	0.12
Bricks	Position Error (m)	0.21	0.10
	Velocity Error (m/s)	0.19	0.07

TABLE IV: REAL-WORLD FLIGHT RESULTS IN THREE TRIALS

Scene	Traj. Length (m)	Travel Time (s)	Avg. Vel. (m/s)
1	12.51	12.00	1.04
2	16.23	14.90	1.09
3	10.70	10.42	1.03

trained in simulations is deployed directly using ONNX-Runtime for fast inference. We carry out the experiments in a cluttered area of badminton court size, and we build three different scenes to test our method. In each scene, the drone takes off from one end of the area, traverses the obstacles autonomously, and arrives at the other end. The process of one trial is shown in Fig. 1, and the recorded data from the three trials is presented in Table IV. The results demonstrate our method’s ability to navigate the drone through complex environments while satisfying dynamical constraints.

IV. CONCLUSIONS

This paper presented LIT-Planner, a motion planner that initializes a trajectory with a neural network and further improves it with spatial temporal optimization. The neural network’s primary role is to provide initial values in both spatial and temporal profiles. Compared to conventional optimization techniques, the incorporation of the neural network has demonstrated substantial reductions in computational time while maintaining a comparable level of trajectory quality. Furthermore, we presented a robust online planning framework that exhibits tolerance towards planning latency and decouples the planning and control frequencies. These methods have been validated through real-world experiments, affirming their practical utility and effectiveness.

REFERENCES

- [1] 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, Feb. 2022.
- [2] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, and F. Gao, "Swarm of Micro Flying Robots in the Wild," *Science Robotics*, vol. 7, no. 66, pp. 1–17, May 2022.
- [3] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically Constrained Trajectory Optimization for Multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, Oct. 2022.
- [4] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep Learning can Accelerate Grasp-Optimized Motion Planning," *Science Robotics*, vol. 5, no. 48, pp. 1–12, Nov. 2020.
- [5] M. Yoon, M. Kang, D. Park, and S.-E. Yoon, "Learning-based Initialization of Trajectory Optimization for Path-following Problems of Redundant Manipulators," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2023, pp. 9686–9692.
- [6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.
- [7] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.
- [8] J. Gao, F. He, W. Zhang, and Y. Yao, "Obstacle-Aware Topological Planning over Polyhedral Representation for Quadrotors," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2023, pp. 10 097–10 103.
- [9] J. Tordesillas and J. P. How, "PANTHER: Perception-Aware Trajectory Planner in Dynamic Environments," *IEEE Access*, vol. 10, pp. 22 662–22 677, Feb. 2022.
- [10] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–37, Apr. 2023.
- [11] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411–444, May 2022.
- [12] J. Tordesillas and J. P. How, "Deep-PANTHER: Learning-Based Perception-Aware Trajectory Planner in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1399–1406, Mar. 2023.
- [13] Y. Song, K. Shi, R. Penicka, and D. Scaramuzza, "Learning Perception-Aware Agile Flight in Cluttered Environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2023, pp. 1989–1995.
- [14] N. J. Sanket, C. D. Singh, C. Fermüller, and Y. Aloimonos, "Ajna: Generalized Deep Uncertainty for Minimal Perception on Parsimonious Robots," *Science Robotics*, vol. 8, no. 81, pp. 1–17, Aug. 2023.
- [15] 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, pp. 1–16, Oct. 2021.
- [16] G. Shi, W. Honig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-Swarm2: Planning and Control of Heterogeneous Multirotor Swarms Using Learned Interactions," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1063–1079, Apr. 2022.
- [17] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza, "Reaching the Limit in Autonomous Racing: Optimal Control Versus Reinforcement Learning," *Science Robotics*, vol. 8, no. 82, pp. 1–14, 2023.
- [18] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-Level Drone Racing Using Deep Reinforcement Learning," *Nature*, vol. 620, no. 7976, pp. 982–987, Aug. 2023.
- [19] S. Banerjee, T. Lew, R. Bonalli, A. Alfaadhel, I. A. Alomar, H. M. Shageer, and M. Pavone, "Learning-based Warm-Starting for Fast Sequential Convex Programming and Trajectory Optimization," in *2020 IEEE Aerospace Conference*. IEEE, Mar. 2020, pp. 1–8.
- [20] D. Mellinger and V. Kumar, "Minimum Snap Trajectory Generation and Control for Quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011, pp. 2520–2525.
- [21] W. H. Press, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge university press, Sept. 2007.
- [22] L. Quan, L. Yin, C. Xu, and F. Gao, "Distributed Swarm Trajectory Optimization for Formation Flight in Dense Environments," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2022, pp. 4979–4985.
- [23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016, pp. 770–778.
- [24] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *2015 International Conference for Learning Representations (ICLR)*. ICLR, May 2015, pp. 1–11.
- [25] 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, Aug. 2018.