AERO-MPPI: Anchor-Guided Ensemble Trajectory Optimization for Agile Mapless Drone Navigation

Xin Chen, Longbin Tang, Rui Huang, Lin Zhao

Abstract— Agile mapless navigation in cluttered 3D environments poses significant challenges for autonomous drones. Conventional mapping–planning–control pipelines incur high computational cost and propagate estimation errors. We present AERO-MPPI, a fully GPU-accelerated framework that unifies perception and planning through an anchor-guided ensemble of Model Predictive Path Integral (MPPI) optimizers. Specifically, we design a multi-resolution LiDAR point-cloud representation that rapidly extracts spatially distributed "anchors" as look-ahead intermediate endpoints, from which we construct polynomial trajectory guides to explore distinct homotopy path classes. At each planning step, we run multiple MPPI instances in parallel and evaluate them with a two-stage multi-objective cost that balances collision avoidance and goal reaching. Implemented entirely with NVIDIA Warp GPU kernels, AERO-MPPI achieves real-time onboard operation and mitigates the local-minima failures of single-MPPI approaches. Extensive simulations in forests, verticals, and inclines demonstrate sustained reliable flight above 7 m/s, with success rates above 80% and smoother trajectories compared to state-of-the-art baselines. Real-world experiments on a LiDAR-equipped quadrotor with NVIDIA Jetson Orin NX 16G confirm that AERO-MPPI runs in real time onboard and consistently achieves safe, agile, and robust flight in complex cluttered environments. The code will be open-sourced upon acceptance of the paper.

I. INTRODUCTION

Agile flight in cluttered, unknown environments remains a significant challenge for autonomous aerial robots. Conventional autonomous flight stacks [1]–[4] typically employ a sequence of modules, such as mapping, safe-flight corridor generation, planning, control, and so on. These pipelines perform well for low-speed flight in sparse obstacle fields. However, such designs typically demand heavy onboard computation and introduce cumulative latency that limits flight speed. Moreover, the sequential processing propagates estimation and planning errors, making the system prone to crashes in densely cluttered environments.

Mapless navigation generates and executes collision-free motions using only local, onboard perception (e.g., LiDAR point clouds). This avoids the error accumulation inherent in maintaining a global map and thereby enhances flight agility. Among sampling-based trajectory optimizers, Model Predictive Path Integral (MPPI) control [5] has emerged as a strong candidate because it flexibly accommodates diverse objective functions, system dynamics, and constraints. More importantly, its sampling-based optimization is well suited to parallelization.

Xin Chen, Longbin Tang, Rui Huang and Lin Zhao are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583 (Email: e1499417@u.nus.edu, longbin@u.nus.edu, ruihuang@u.nus.edu, elezhli@nus.edu.sg).
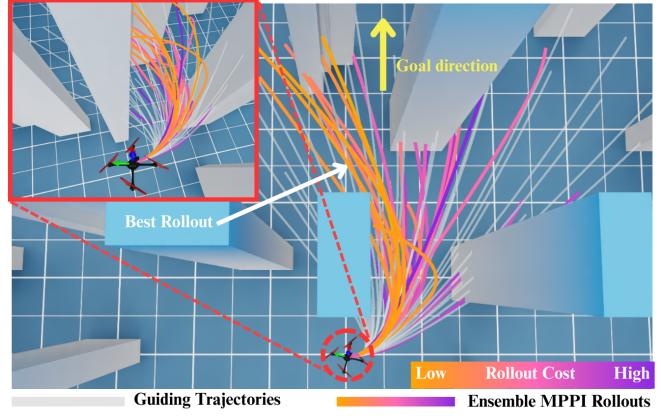
Fig. 1. AERO-MPPI navigating a challenging obstacle environment. The algorithm samples spread anchor points and generates corresponding guiding trajectories (gray lines) on the LiDAR point cloud. These trajectories are then optimized by parallel MPPI instances to produce multiple trajectory proposals (orange–purple curves). The best is the one with the lowest cost.

However, MPPI-based planning directly on raw point clouds presents two major challenges. First, the repeated evaluation of collision costs against dense point sets is computationally intensive and can easily break real-time operation. Second, a single-start MPPI is prone to poor exploration in cluttered environments: with only one goal point guidance, MPPI rollouts often remain confined to a single homotopy class, making the planner sensitive to cost shaping and initial seeds and thus prone to local traps. Increasing the sampling covariance can partly alleviate this issue, but it typically slows the optimization process and can even lead to divergence or instability.

To address these challenges, we propose AERO-MPPI, a parallel ensemble of MPPI optimizers guided by diversified look-ahead references we call *anchors* (see the overall diagram in Fig. 2). An anchor is a per-cycle, goal-directed look-ahead point from which we generate a guiding trajectory (grey curves in Fig. 1); Anchors are sampled along the goal direction with a configurable look-ahead distance and angular spread, ensuring that the resulting generated guiding trajectories span distinct corridors around obstacles and enable more robust navigation. In addition, we build a multi-resolution LiDAR representation that supports fast generation of these anchor-guided trajectories. They serve as high-quality initial guesses for subsequent MPPI optimizations. At each planning step, AERO-MPPI samples anchors, generates the corresponding guiding trajectories, runs the MPPI instances concurrently, and selects the best proposal under a multi-objective cost that balances collision avoidance and goal progress. The entire planning pipeline is implemented with
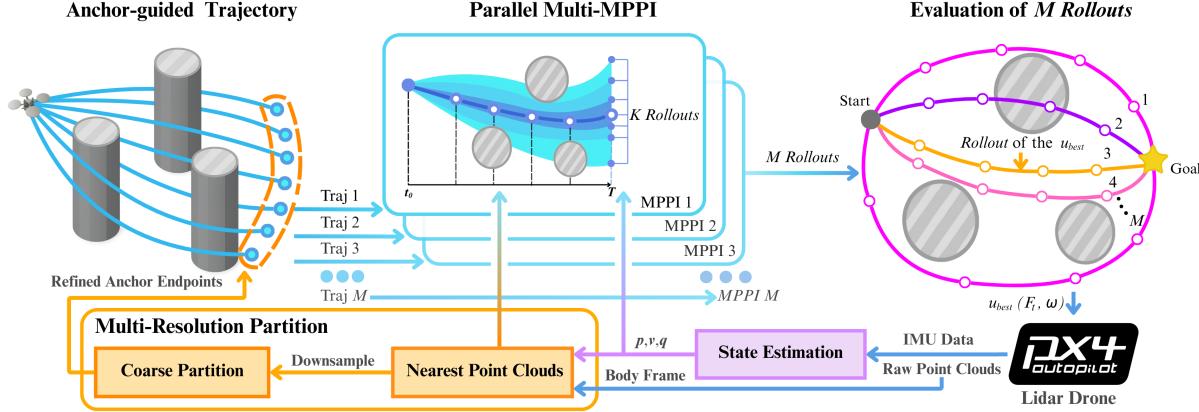
Fig. 2. **AERO-MPPI System Overview.** We perform state estimation by fusing IMU data with raw LiDAR point clouds to recover the system state $(\mathbf{p}, \mathbf{v}, \mathbf{q})$. To enhance efficiency, we design a multi-resolution partitioning scheme that downsamples the nearest point clouds onto coarsely defined partitions to obtain refined anchor endpoints. For free-space exploration, the drone leverages the refined endpoints together with the current state to compute guiding trajectories, which serve as initial guesses for parallel multi-MPPI optimization. After obtaining the candidate control sequences from multi-MPPI, we evaluate them through explicit rollouts and select the best control input.

NVIDIA Warp [6] GPU kernels, fully leveraging large-scale parallelism for onboard, real-time performance.

Our main contributions are summarized as follows:

- We propose AERO-MPPI, an anchor-guided ensemble of MPPI optimizers for parallel trajectory optimization. This approach mitigates the local-minima issues common in traditional MPPI and significantly improves navigation robustness during high-speed flight.
- We implement the full perception-to-planning pipeline with GPU kernels (NVIDIA Warp), enabling real-time, high-speed flight with onboard computation.
- We conducted extensive validation in simulation across three challenging scenarios. Results show that our method achieves improved agility, success rate, and smoothness compared with strong baseline planners [1]–[4], while maintaining real-time performance, safety, and deployability.

II. RELATED WORK

Popular drone navigation algorithms mostly relies on various forms of local maps for collision avoidance trajectory planning. For example, Fast-Planner [1] performs unconstrained trajectory optimization on an online-updated Euclidean Signed Distance Field (ESDF) for collision avoidance. Following Fast-Planner, Topo-Planner [2] generates and selects among multiple homotopy-aware trajectories on the ESDF, improving global optimality in cluttered environments. Furthermore, Ego-Planner [3] performs local greedy trajectory optimization directly on the occupancy map without explicit ESDF construction. In addition, Ego-Planner v2 further improves trajectory quality and generation efficiency by adopting MINCO [7] as the trajectory parameterization. Moreover, IPC [4] constructs a local map and a Safe Flight Corridor (SFC) [8], and enforces SFC as hard constraints in a linear model predictive control (MPC) framework for integrated planning and control. These methods still require frequent ESDF updates or SFC construction, which require significant computation and induce large latency, thus limiting the flight speed in cluttered scenarios.

The aforementioned works rely on gradient information from explicit maps. In contrast, MPPI control [5] is a sampling-based, model-predictive method that repeatedly draws random control sequences, simulates (rolls out) the system's nonlinear dynamics under those controls, and evaluates a cost for each trajectory. The controls are then updated—via a path-integral weighted average of the samples, favoring lower cost trajectories and yielding an optimized control action at each planning step. This formulation naturally handles nonlinear dynamics and non-smooth obstacle costs, while GPU parallelization enables thousands of rollouts in real time.

Recent works have implemented GPU-accelerated MPPI for onboard quadrotor computation, enabling real-time collision avoidance [9]. To improve sampling efficiency and trajectory smoothness, spline-interpolated MPPI with Stein variational inference has been proposed [10]. Other variants, such as Tube-MPPI [11], Robust-MPPI (RMPPI) [12], and Risk-Aware MPPI (RA-MPPI) [13], enhance robustness to disturbances and model uncertainties. To provide stronger safety guarantees, Control Barrier Functions (CBFs) [14] have been incorporated into MPPI in methods like Shield-MPPI [15] and SCBF-MPPI [16] to filter unsafe trajectory samples.

However, all these approaches rely on a single MPPI instance, which often leads to local minima and conservative behavior. They also depend on explicit mapping modules to evaluate risk and safety, which are expensive to construct in unknown or dynamic environments. In contrast, our AERO-MPPI fosters global exploration and can plan more diverse homotopy trajectories, enabling more effective obstacle-avoidance navigation.

III. PRELIMINARIES

A. Quadrotor Model

The following quadrotor model is employed:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \ \dot{\boldsymbol{q}} = \tfrac{1}{2}\, \boldsymbol{q} \odot \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \ \dot{\boldsymbol{v}} = \tfrac{1}{m}\, \boldsymbol{R}(\boldsymbol{q}) \begin{bmatrix} 0 \\ 0 \\ F_t \end{bmatrix} + \boldsymbol{g}, \quad (1)$$

where $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3$ denote the position and velocity of the body frame B with respect to the world frame W, respectively. The body's orientation relative to an inertial/world frame is represented by the unit quaternion $\boldsymbol{q}$ with $|\boldsymbol{q}| = 1$, and $\boldsymbol{\omega}$ is the angular velocity of the body expressed in the body frame, $\boldsymbol{R}(\boldsymbol{q})$ is the rotation matrix corresponding to $\boldsymbol{q}$, $\odot$ denotes quaternion multiplication, m is the drone mass, and $\boldsymbol{g}$ is the gravitational acceleration vector. We adopt CTBR (collective thrust and body rate) control with input $\boldsymbol{u} := (F_t, \boldsymbol{\omega}) \in \mathbb{R}^4$. For brevity, we denote $\boldsymbol{x} := [\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}]$.

B. Model Predictive Path Integral Control

MPPI is an optimal control framework that solves nonlinear and nonconvex problems through Monte Carlo sampling rather than gradient-based optimization. This sampling-based formulation allows direct incorporation of complex dynamics, high-dimensional constraints, and cost functions with sparse or nonexistent gradients. With receding-horizon execution, MPPI continuously replans using updated perception information, making it well suited for motion planning in cluttered environments.

At each control step, MPPI predicts K candidate trajectories (rollouts) over N discrete time steps (prediction horizon). The current state is $\boldsymbol{x_0}$, and the nominal control sequence is $\boldsymbol{u}^{\mathrm{nom}} = (\boldsymbol{u}_0^{\mathrm{nom}}, \ldots, \boldsymbol{u}_{N-1}^{\mathrm{nom}})$. Gaussian disturbances $\boldsymbol{\delta}_j^k \sim \mathcal{N}(\boldsymbol{0}, \Sigma)$ are added to the nominal controls to generate perturbed controls $\boldsymbol{u}_j^k = \boldsymbol{u}_j^{\mathrm{nom}} + \boldsymbol{\delta}_j^k$ for rollout k, where $\Sigma \in \mathbb{R}^{n_u \times n_u}$ is the covariance matrix.

Each rollout k is simulated forward using the continuous-time dynamics (1), which are discretized with a time step Δt using the fourth-order Runge–Kutta (RK4) integration method:

$$\boldsymbol{x}_{j+1}^k = \boldsymbol{x}_j^k + f_{\mathrm{RK4}}(\boldsymbol{x}_j^k, \boldsymbol{u}_j^k, \Delta t). \tag{2}$$

where $j \in \{0, \ldots, N-1\}$ denotes the time step along the prediction horizon.

For each trajectory, a task-specific cost S_k is computed, and the K costs are normalized to generate a distribution $(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_K)$:

$$\rho = \min\{S_1, \cdots, S_K\},$$
$$\eta = \sum_{k=1}^{K} \exp\left(-\tfrac{1}{\lambda}(S_k - \rho)\right), \tag{3}$$
$$\varepsilon_k = \frac{\exp\left(-\tfrac{1}{\lambda}(S_k - \rho)\right)}{\eta},$$

where $\lambda > 0$ is the so-called temperature parameter that controls the spread of the distribution. It balances exploration and exploitation: as $\lambda \to 0$, the weights collapse to the minimum-cost trajectory, while larger λ values distribute weights more evenly across rollouts.

The nominal controls are updated by the weighted sum of the sampled disturbances $\boldsymbol{\delta u}^k = (\boldsymbol{\delta}_0^k, \ldots, \boldsymbol{\delta}_{N-1}^k)$:

$$\boldsymbol{u}^{\mathrm{nom}} \leftarrow \boldsymbol{u}^{\mathrm{nom}} + \sum_{k=1}^{K} \varepsilon_k \, \boldsymbol{\delta u}^k. \tag{4}$$

Cell (i,j) ☐ ☐ **Cell** (I,J) M_h ★ **Goal** r_{max} M_v β α ℓ $p_{m,n}^{\mathrm{init}}$ p_0

■ Cell Cones   ● Raw Points   ○ Nearest Points   ○ Initial Endpoints   ○ Refined Endpoints   ▬ Guiding Trajectory
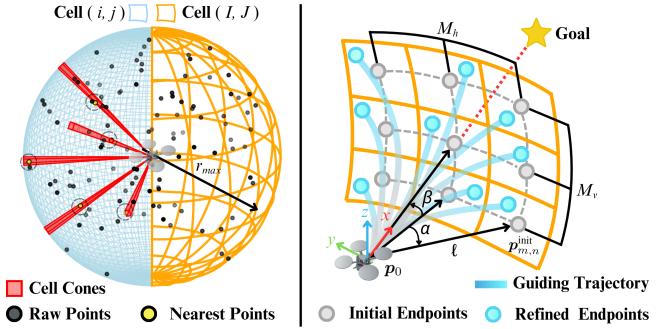
Fig. 3. **Multi-resolution partition for generating guiding trajectories.** Black points denote raw point clouds within the sensing sphere. The left hemisphere illustrates fine-grained cells, where selected cells are highlighted by the corresponding red spherical cones extending to the nearest LiDAR points (yellow). The right hemisphere depicts a coarse-grained partition used to adjust the initial anchor endpoints (gray) to refined endpoints (blue). Facilitated by this multi-resolution partition, guiding trajectories are generated efficiently from the current drone's pose to the refined endpoints.

This update shifts $\boldsymbol{u}^{\mathrm{nom}}$ toward the controls of lower-cost rollouts, iteratively improving the planned trajectory until convergence.

IV. AERO-MPPI

AERO-MPPI consists of three main components: 1) Multi-resolution partitioning of the environment; 2) Anchor-guided trajectory generation; 3) Parallel MPPI ensemble trajectory optimization. An overview of the pipeline is shown in Fig. 2.

A. Multi-Resolution Partition

We build both a high-resolution partition and a coarse partition to facilitate anchor points generation and collision evaluation. The high-resolution partition decomposes the local search space into spherical cones, and the distance of the nearest obstacle in each cone is computed. The coarse partition is mainly used to generate anchor points to probe distinct corridors around obstacles. An illustration of the partition is illustrated in the hemispherical view of Fig. 3.

1) High-resolution partition (3°): We adopt the lightweight Livox Mid-360 LiDAR, which features a non-repetitive scanning pattern with a $10\,\mathrm{Hz}$ scan rate. Compounded with the drone's fast motion, typically a single frame is insufficient to provide reliable coverage of the environment. Therefore, we accumulate the most recent 10 consecutive frames (200,000 points) and project them into the current body frame using the transforms from state estimation, yielding the accumulated point-cloud set $\mathcal{P}_t$. For onboard computation, we replan at 50 Hz using the latest drone pose provided by the PX4 EKF, which fuses IMU measurements with Fast-LIO localization [17]. The perception range for processing the point-cloud is limited to $r_{\max} = 10\,\mathrm{m}$. Centered at the drone body frame, we discretize the spherical Field of View (FOV) into an angular grid at a $3°$ resolution, resulting in 120×60 cells. Let (α_i, β_j) denote the azimuth and elevation of Cell (i,j) with spherical cone $C_{i,j}$, $i \in \{1, \cdots, 120\}$, $j \in \{1, \cdots, 60\}$. For each cone, we compute the nearest distance to the obstacle

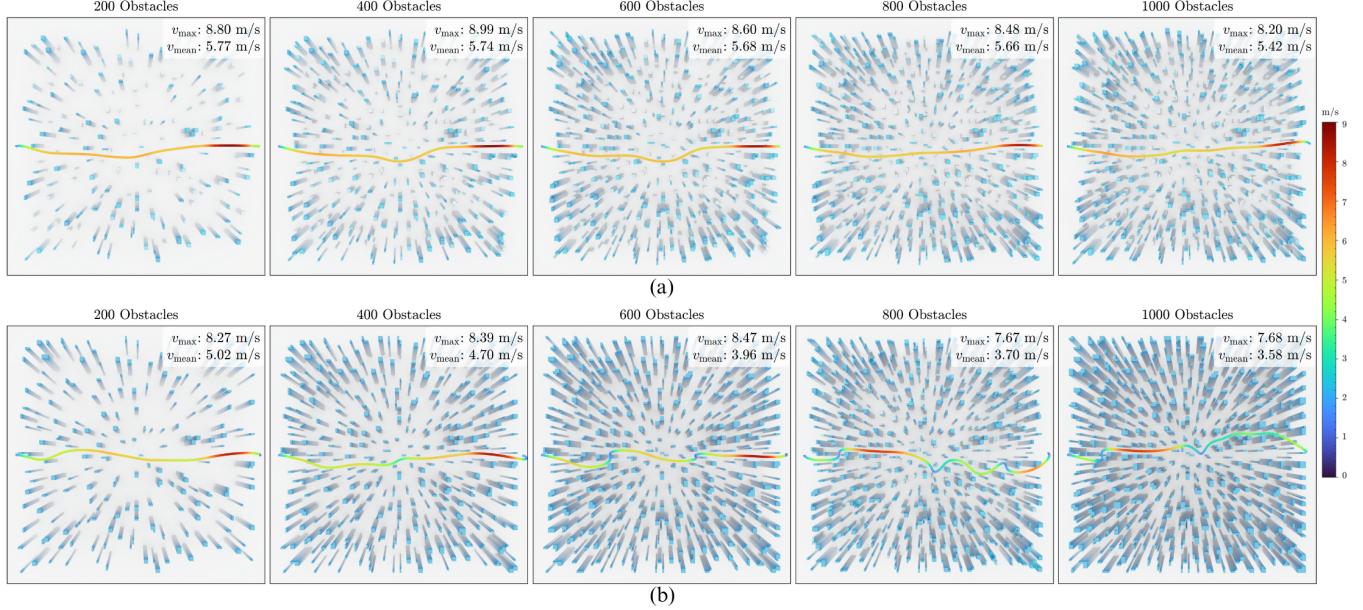$$r_{i,j} := \min_{\mathbf{p}_c \in \mathcal{P}_t \cap C_{i,j}} \|\mathbf{p}_c\|, \tag{5}$$

Fig. 4. **Comparison across obstacle numbers (200–1000) and height ranges (1–6 m) in a** $40\,\text{m} \times 40\,\text{m}$ **map.** (a) Randomized heights between 1–6 m: the five subfigures correspond to obstacle counts from 200 to 1000. With 1000 obstacles (each with a width ranging from 0.4 m to 1.1 m), the maximum velocity reaches 8.20 m/s and the mean velocity is 5.42 m/s, demonstrating the agility of the proposed AERO-MPPI. (b) Fixed height of 6 m: the uniform obstacle height prevents the drone from exploiting z-axis variation for avoidance. This configuration is more challenging, with narrow gaps that often cause local minima, reducing the maximum velocity to 7.68 m/s.

where $\mathbf{p}_c$ denotes the coordinate of a point in the cloud expressed in the drone body frame.

The unit direction $\boldsymbol{d}_{i,j} \in \mathbb{R}^3$ of Cell (i,j) is

$$\boldsymbol{d}_{i,j} \; := \; \begin{bmatrix} \cos\beta_j \cos\alpha_i \\ \cos\beta_j \sin\alpha_i \\ \sin\beta_j \end{bmatrix}. \qquad (6)$$

2) Coarse partition (18°): We aggregate the 3°-resolution cells using $6{\times}6$ pooling to form an 18°-resolution grid with a total of $20{\times}10$ cells. Let $\Omega_{I,J}$ denote the set of high-resolution cell indices pooled into the coarse cell (I,J).

For each Cell (I,J), we can compute the *coarse safe point* $\boldsymbol{v}_{I,J}$ as

$$(i^\star, j^\star) = \arg \max_{(i,j)\in\Omega_{I,J}} r_{i,j},$$
$$\boldsymbol{v}_{I,J} = r_{i^\star,j^\star} \, \boldsymbol{d}_{i^\star,j^\star}, \qquad (7)$$

where $(i^\star, j^\star)$ is the index with the maximum safe range in $\Omega_{I,J}$ and $\boldsymbol{d}_{i,j}$ is the unit direction vector of Cell (i,j). Slightly abusing the notation, we denote the coarse safe direction of Cell(I,J) by $\boldsymbol{d}_{I,J} := \boldsymbol{d}_{i^\star,j^\star}$.

B. Anchor-Guided Trajectory Generation

To broaden the search for distinct corridors around obstacles and reduce the risk of local minima, we first generate look-ahead anchor points. These anchors serve as endpoints for constructing polynomial trajectories. In Section IV-C, an ensemble of MPPI instances is then executed in parallel to track these trajectories.

1) Sampling of the initial endpoints: As illustrated in Fig. 3, we set a look-ahead distance of $\ell = 5\,\text{m}$. At this distance, a grid of $M_h \times M_v$ cells is first picked centered on the drone's current heading direction from the coarse partition. We then rotate the entire grid so that its central

axis aligns with the goal direction, defined by the azimuth angle α and elevation angle β in the figure. Each gray dot in the diagram marks the center of a coarse cell and is used as an *initial endpoint* for trajectory generation. We further refine these initial endpoints (highlighted by the cyan circles) according to the maximum safe distance and construct the *guiding trajectories* (shown in blue) in the next subsection.

2) Refining the initial endpoints: The initial endpoint is refined by re-aligning it to the direction of the maximum safe distance within its Cell (I,J). Specifically, with the drone's position denoted by $\mathbf{p}_0$, $\forall I \in \{1,\ldots,M_h\}, J \in \{1,\ldots,M_v\}$, we have the refined endpoints calculated by

$$\boldsymbol{p}_{I,J}^{\text{ref}} \; = \; \boldsymbol{p}_0 + \ell \cdot \boldsymbol{v}_{I,J}, \qquad (8)$$

where $\boldsymbol{v}_{I,J}$ is computed as in (7). This enforces that anchors strictly follow the locally safest heading provided by each cell in the coarse partition.

3) Building Polynomial Guiding Trajectory: Each guiding trajectory uses three fifth-order polynomials $f_\mu(t)$, $\mu \in \{x,y,z\}$, over horizon $T = N\Delta t$:

$$f_\mu(t) \; = \; \sum_{i=0}^{5} a_{i,\mu} t^i, \quad \forall t \in [0,T] \qquad (9)$$

We note that the coefficients $\{a_{i,\mu}\}_{i=0}^{5}$ in (9) are uniquely determined by the boundary conditions on position, velocity, and acceleration at the start and end of the horizon. This results in a closed-form polynomial solution that generates smooth, dynamically feasible guiding trajectories aligned with the coarse safe directions and distributed around the goal direction.

C. Parallel MPPI Ensemble Optimization

Unlike previous works [9], [14], [15] that rely on a single MPPI and often fall into local minima, our approach employs
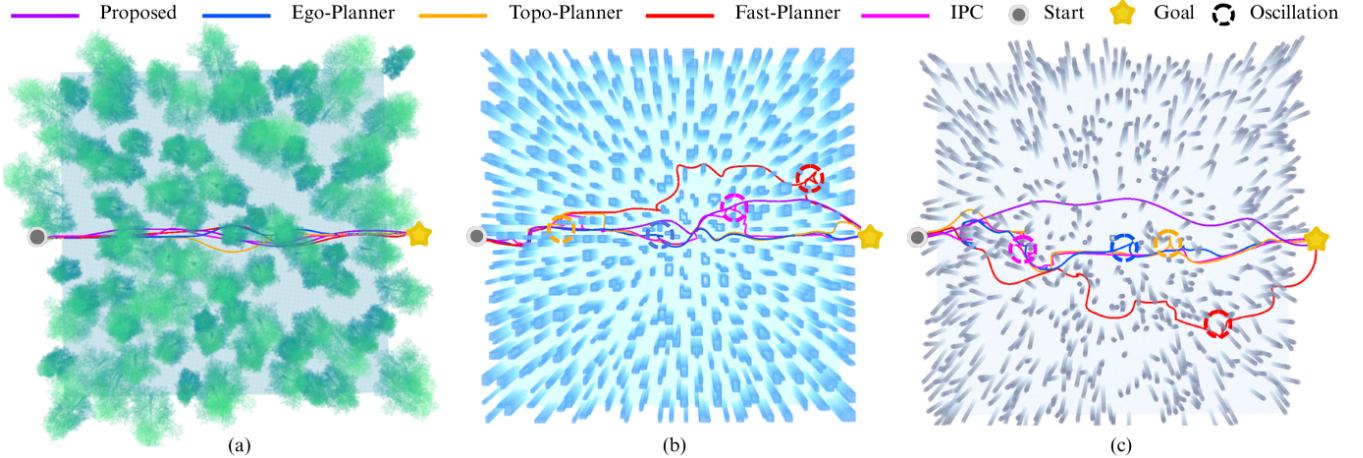
Fig. 5. **Benchmark results with previous works [1]–[4] under different scenarios in a** $40\,\mathrm{m} \times 40\,\mathrm{m}$ **map.** (a) Forest: All methods generate smooth trajectories of approximately the same length. (b) Verticals: Mapping-based methods exhibit frequent replanning, leading to oscillatory and non-smooth trajectories, as tall vertical obstacles often cause severe occlusions. In contrast, our method directly leverages local perception for optimization, ensuring continuity and smoothness. (c) Inclines: Similar to the Verticals case, frequent replanning occurs, and some trajectories even become trapped in local geometric pitfalls (e.g., small holes or narrow gaps), causing stagnation at local optima. Our method employs explicit parallel optimization and robust sampling to effectively avoid such traps.

multiple MPPI instances running in parallel to enhance the exploration. Specifically, we leverage the guiding trajectories as the tracking references to initialize $M := M_h \times M_v$ MPPIs in parallel.

Furthermore, prior methods are typically amenable to only a single-objective cost, which can make the planner either overly conservative or overly aggressive and ultimately lead to suboptimal performance. Our MPPI ensemble addresses this with a two-stage optimization: In Stage I, each MPPI instance optimizes its guiding trajectory via sampling under a composite cost $S^{(1)}$, obtaining the optimized control sequence along the prediction horizon. In Stage II, we roll out the M optimized control sequences via RK4 integration of the drone dynamics, re-evaluate them with a simplified cost $S^{(2)}$, and select the optimal rollout control sequence for execution. This dual-stage design preserves directional diversity while enforcing an explicit safety filter. The two stage costs are defined respectively as follows:

$$
\begin{aligned}
S^{(1)} &= J_{\text{track}} + J_{\text{vnorm}} + J_{\text{ctrl}} + S^{(2)}, \\
S^{(2)} &= J_{\text{goal}} + J_{\text{col}}.
\end{aligned}
\tag{10}
$$

Each cost term in (10) is defined below. Matrices of the form $\boldsymbol{Q}_*$ denote the weighting matrices, with subscripts indicating the corresponding cost components.

1) Guiding-Trajectory Tracking Cost: To keep rollouts close to the chosen guide, we penalize the position deviation from the guiding trajectory:

$$
J_{\text{track}} = \sum_{t=0}^{N-1} \boldsymbol{Q}_{track} \lVert \boldsymbol{p}_t - \boldsymbol{p}_{\text{traj}}(t) \rVert,
\tag{11}
$$

where $\mathbf{p}_t$ and $\mathbf{p}_{\text{traj}}(t) = [f_x(t), f_y(t), f_z(t)]$ denote, respectively, the MPPI-predicted (rollout) position and the guiding-trajectory position at time step t.

2) Velocity Norm Cost: To discourage excessively large velocities and ensure safe, we penalize the squared norm of

the velocity along the rollouts:

$$
J_{\text{vnorm}} = \sum_{t=0}^{N-1} \boldsymbol{Q}_{vnorm} \lVert \boldsymbol{v}_t \rVert^2,
\tag{12}
$$

where $\boldsymbol{v}_t$ is the MPPI predicted (rollout) velocity at time step t.

3) Control Cost: To limit effort and smooth inputs, we penalize control magnitude and its rate:

$$
J_{\text{ctrl}} = \sum_{t=0}^{N-2} \boldsymbol{Q}_c \lVert \boldsymbol{u}_t \rVert^2 + \sum_{t=1}^{N-2} \boldsymbol{Q}_{c\Delta} \lVert \Delta \boldsymbol{u}_t \rVert^2,
\tag{13}
$$

where $\Delta \boldsymbol{u}_t \triangleq \boldsymbol{u}_t - \boldsymbol{u}_{t-1}$, with $\boldsymbol{u}_{t-1}$ denoting the previously applied control.

4) Goal Cost: To bias the rollouts toward the goal, we penalize both running and terminal errors in position, velocity, and attitude:

$$
\begin{aligned}
J_{\text{goal}} = \sum_{t=0}^{N-1} \Big(&\boldsymbol{Q}_p \lVert \boldsymbol{p}_t - \boldsymbol{p}_{\text{goal}} \rVert + \boldsymbol{Q}_v \lVert \boldsymbol{v}_t - \boldsymbol{v}_{\text{goal}} \rVert \\
&+ \boldsymbol{Q}_q \lVert \boldsymbol{R}(\boldsymbol{q}_t)\,\boldsymbol{R}(\boldsymbol{q}_{\text{goal}})^\top - \boldsymbol{I} \rVert_F \Big),
\end{aligned}
\tag{14}
$$

where $\boldsymbol{p}_{\text{goal}}, \boldsymbol{v}_{\text{goal}}, \boldsymbol{q}_{\text{goal}}$ denote the goal position, velocity, and quaternion, respectively; $\boldsymbol{q}_t$ is the quaternion at time step t; $\boldsymbol{I}$ is the identity matrix; and $\lVert \cdot \rVert_F$ denotes the Frobenius norm.

5) Collision Cost: We penalize insufficient clearance by measuring the Euclidean distance from the rollout position $\boldsymbol{p}_t$ at time step t to the nearest observed obstacle point $\boldsymbol{p}_{\text{obs}}$ in the high-resolution ($3°$) partition. We define the nearest distance as $d_t = \min_{\boldsymbol{p}_{\text{obs}} \in \mathcal{P}_f} \lVert \boldsymbol{p}_t - \boldsymbol{p}_{\text{obs}} \rVert$, where $\mathcal{P}_f$ is the filtered point cloud containing only the closest LiDAR point in each Cell (i, j). The instantaneous collision penalty is then

computed as:

$$
J_{\text{col},t} = \begin{cases} C, & d_t < d_{\text{obs}}^{\min}, \\ C \exp\big(-a(d_t - d_{\text{obs}}^{\min})\big), & d_{\text{obs}}^{\min} \leq d_t < d_{\text{obs}}^{\max}, \\ 0, & d_t \geq d_{\text{obs}}^{\max}, \end{cases}
$$

$$
J_{\text{col}} = \sum_{t=0}^{N-1} J_{\text{col},t},
$$

(15)

where $d_{\text{obs}}^{\min}$ and $d_{\text{obs}}^{\max}$ denote the minimum and maximum safe distances, satisfying $d_{\text{obs}}^{\min} < d_{\text{obs}}^{\max}$; $C > 0$ is a scaling factor, and $a > 0$ is the slope parameter.

V. Experiment

A. Implementation Choice

We validate AERO-MPPI extensively in both simulation and real-world environments. For practical deployment, the controller must run at high frequency with efficient rollout computation. However, CPU execution of MPPI is prohibitive due to the overhead of point-cloud nearest-neighbor costs. A pure CUDA implementation [18] can deliver high efficiency but is difficult to prototype and maintain because it demands low-level GPU programming, manual memory management, and is tied to NVIDIA hardware. In contrast, JAX [19] provides automatic differentiation and a more productive Python environment but lacks low-level GPU control. We therefore adopt NVIDIA Warp [6], which combines near-CUDA efficiency with concise Python APIs, PyTorch integration, and explicit kernel control. For implementation, all modules of AERO-MPPI are written as Warp kernels to fully exploit GPU parallelism and ensure real-time performance. LiDAR point clouds are accumulated over the most recent 10 consecutive frames, and $M = 15$ $\big(M_h = 5,\ M_v = 3,$ corresponding to searching a field of view $90° \times 54°\big)$ anchor points are sampled, seeding 15 parallel MPPI instances. The other parameters are summarized in Table I.

TABLE I
MPPI AND AERO-MPPI PARAMETERS

MPPI Parameters		AERO-MPPI Parameters	
m	1.0 kg	Q_p	3.0
$F_{t,min}$	0.3 N	Q_v	0.25
$F_{t,max}$	16.35 N	Q_q	1.0
$\omega_{xy,max}$	3.0 rad/s	Q_{vnorm}	0.15
$\omega_{z,max}$	2.0 rad/s	Q_{track}	15.0
Σ_{F_t}	1.0	Q_c	0.5
$\Sigma_{\omega_{xy}}$	1.0	$Q_{c\Delta}$	0.5
Σ_{ω_z}	0.5	ℓ	5.0 m
Δt	50 ms	C	10^6
K	128	a	5.0
N	25	d_{obs}^{min}	0.4 m
λ	0.1	d_{obs}^{max}	1.0 m

B. Simulation Experiments

For a fair comparison, all simulation environments initialize the drone at $(0, 0, 2)$ with a target at $(45, 0, 2)$. The LiDAR sensing parameters of all baselines are set identical to
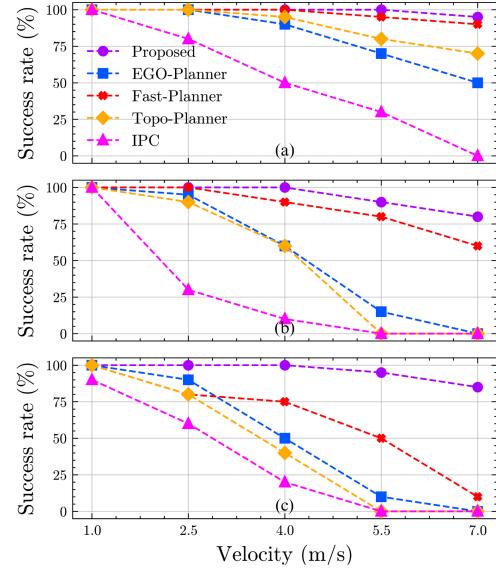

Fig. 6. **Benchmark results with baseline SOTA works [1]–[4] under different velocity limits (e.g., 1.0–7.0 m/s) in a** $40\,\text{m} \times 40\,\text{m}$ **map.** Figures (a)–(c) illustrate the comparison results of success rate in scenarios Forest, Verticals, and Inclines, respectively.

those of our framework. In mapping-based methods, the map inflation size is matched to AERO-MPPI's minimum safety distance $d_{\text{obs}}^{\min}$ for fair comparison, and all are set equal to the UAV radius.

1) Performance across Varying Obstacle Densities: We first evaluate the proposed AERO-MPPI in the IsaacLab simulator [20] under environments with varying obstacle densities and heights to examine its agility and effectiveness in mitigating local minima issues. The drone navigates through environments with progressively increasing levels of clutter. The results are presented in Fig. 4, which report the maximum and average flight velocities under different clutter levels. Notably, even in the densest and most challenging scenarios with 1,000 obstacles, AERO-MPPI consistently maintains agile forward motion and avoids being trapped in local minima or stalling in place. Beyond navigation performance, our GPU-accelerated implementation achieves real-time execution at **500 Hz** on a GeForce RTX 4080 SUPER GPU, with moderate GPU memory usage (less than 600 MB when $N{=}25$ and $K{=}256$), scaling linearly with the number of trajectories and horizon length.

2) Comparisons with SOTA Planners under Different Cluttered Environments: To further demonstrate AERO-MPPI's superior performance, we benchmark it against four SOTA obstacle-avoidance systems, namely EGO-Planner [3], Topo-Planner [2], Fast-Planner [1], and IPC [4], in the LiDAR-based drone simulator MARSIM [21] across a variety of cluttered environments, as illustrated in Fig. 5.

Success rates are averaged over 50 trials (accounting for randomness from LiDAR measurement noise, algorithm initializations, etc.) for each velocity limit, as shown in Fig. 6. The evaluation map size is $40\,\text{m} \times 40\,\text{m}$, with three cluttered scenarios: **Forest**: 100 randomly scaled and tilted trees; **Verticals**: 1000 vertical cylinders with a height of 6 m and radii ranging from 0.4 m to 1.1 m; **Inclines**: 800 tilted

TABLE II
PERFORMANCE COMPARISON IN THREE SCENARIOS

Method	Scene	Vel. (m/s) ↑		Smooth (m²/s⁵) ↓	Length (m) ↓	Clearance (m)*	
		Avg	Max			Avg	Min
Ego-planner [3]	Forest	1.82 ± 0.11	6.37 ± 0.17	278.08 ± 44.99	52.95 ± 2.33	1.66 ± 0.09	0.43 ± 0.14
	Verticals	1.33 ± 0.07	3.24 ± 0.05	39.02 ± 32.33	52.60 ± 2.13	1.26 ± 0.08	0.35 ± 0.10
	Inclines	1.04 ± 0.06	2.31 ± 0.72	70.02 ± 73.54	57.88 ± 3.42	1.09 ± 0.07	0.31 ± 0.05
Topo-planner [2]	Forest	3.27 ± 0.24	6.64 ± 1.08	87.26 ± 25.60	46.46 ± 0.34	1.96 ± 0.07	0.32 ± 0.15
	Verticals	1.42 ± 0.08	3.02 ± 0.24	49.96 ± 24.54	57.90 ± 4.37	1.18 ± 0.05	0.48 ± 0.02
	Inclines	1.02 ± 0.08	2.70 ± 0.84	28.51 ± 9.49	59.24 ± 4.75	1.24 ± 0.04	0.49 ± 0.04
Fast-planner [1]	Forest	2.53 ± 0.25	7.26 ± 0.88	67.26 ± 23.59	47.05 ± 0.93	1.76 ± 0.03	0.65 ± 0.17
	Verticals	1.21 ± 0.09	3.13 ± 0.57	105.72 ± 10.17	59.33 ± 3.74	1.02 ± 0.05	0.37 ± 0.10
	Inclines	1.16 ± 0.10	2.32 ± 0.15	$\mathbf{18.51 \pm 8.02}$	68.75 ± 11.13	1.07 ± 0.07	0.36 ± 0.13
IPC [4]	Forest	1.84 ± 0.14	3.16 ± 0.02	79.38 ± 27.06	45.72 ± 0.64	1.76 ± 0.07	0.65 ± 0.07
	Verticals	1.29 ± 0.06	3.23 ± 1.31	281.08 ± 122.35	62.86 ± 6.77	0.93 ± 0.06	0.33 ± 0.08
	Inclines	1.33 ± 0.03	3.60 ± 0.37	303.30 ± 86.76	66.09 ± 5.71	0.97 ± 0.08	0.32 ± 0.02
Proposed	Forest	$\mathbf{5.60 \pm 0.11}$	$\mathbf{7.74 \pm 0.22}$	$\mathbf{59.38 \pm 27.06}$	$\mathbf{45.92 \pm 0.55}$	1.27 ± 0.05	0.51 ± 0.06
	Verticals	$\mathbf{3.52 \pm 0.21}$	$\mathbf{6.33 \pm 0.57}$	$\mathbf{38.20 \pm 18.55}$	$\mathbf{51.69 \pm 2.33}$	0.89 ± 0.07	0.39 ± 0.02
	Inclines	$\mathbf{5.08 \pm 0.19}$	$\mathbf{8.09 \pm 0.66}$	65.30 ± 30.76	$\mathbf{48.66 \pm 1.03}$	0.96 ± 0.09	0.33 ± 0.02

*Clearance (m) denotes the distance to the nearest obstacle. Extremely small values indicate collision risk, while excessively large values suggest conservative detours. Moderate values typically reflect a balance between safety and efficiency.

cylinders with a height of 10 m, tilt angles ranging from 0°–30°, and radii ranging 0.06 m-0.3 m.

At low speeds, all methods achieve high success rates. However, performance degrades as velocity increases. In relatively open settings such as Forest, most baselines maintain above 50% success at 7 m/s. An exception is IPC [4]: its use of safe-flight corridor as hard MPC constraints makes the solver brittle when corridor boundaries vary rapidly, causing failures and collisions. In denser Verticals and Inclines, the limited perceptual horizon yields partial local maps. Shortest-path plans derived from these myopic maps are suboptimal; once occlusions clear, frequent replanning is triggered. This adds tracking burden and induces oscillatory flight (see Fig. 5). AERO-MPPI searches broader spatial area and maintains high success rates above 80% across all scenarios, even at high speeds.

Table II summarizes the performance from 50 successful runs of all methods. Note that the baseline methods had to reduce their maximum speed limits to achieve 50 successful runs. Specifically, AERO-MPPI attains the highest average and peak velocities, sustaining > 5 m/s on average and exceeding 8 m/s in Inclines, while maintaining safe clearance (typically ≥ 0.3 m). Among baselines, EGO-Planner [3] shows limited agility (average speed < 2 m/s) and notably poor smoothness in Forest (> 270 m²/s⁵). Topo-Planner [2] achieves the largest obstacle margins (average up to ~ 1.9 m) but at the cost of lower speed (1–3 m/s) and longer paths. Fast-Planner [1] reaches competitive peak speed in Forest (> 7 m/s) yet suffers from poor smoothness and unstable trajectories. IPC [4] can yield high minimum clearance (≈ 0.65 m), but is constrained to low maximum speeds (< 3.6 m/s) and long detours.

Overall, the results demonstrate that AERO-MPPI achieves a balanced trade-off, sustaining higher speed, sufficient clearance, and smoother, shorter trajectories, thereby ensuring both safety and dynamic feasibility while delivering superior agility in cluttered environments.

C. Real-world Experiment

1) Hardware details: We deployed our method on a 380 mm quadrotor platform weighing 1.65 kg, equipped with a Livox Mid-360 LiDAR, a PX4-based embedded autopilot, and a Jetson Orin NX 16 GB onboard computer (see Fig. 7). The PX4 Extended Kalman Filter (EKF) runs at 100 Hz, fusing IMU measurements with Fast-LIO2 localization [17] to provide high-frequency pose estimates. Using these estimates, AERO-MPPI runs on the Jetson's GPU for real-time inference and replans at 50 Hz to generate control commands that the PX4 autopilot executes.

2) Experiment: To further validate the real-world performance of our proposed framework, we construct four representative obstacle scenarios, and the flight results are captured in Fig. 7. The quadrotor is initially commanded to fly at a target height of 0.8 m toward a goal located 3.75 m ahead. In the (a) **Pillar** scenario, AERO-MPPI successfully generates collision-free trajectories that steer the quadrotor around the obstacle. In the (b) **Wall** scenario, the framework leverages vertical trajectory samples to enable the quadrotor to climb over the wall and reach the target. In the (c) **Double-Wall Trap** scenario, where the obstacle layout restricts the flight altitude below the barrier and easily induces local minima, AERO-MPPI still identifies a feasible trajectory and reaches the goal. In the (d) **Random Clutter** scenario, with obstacles placed arbitrarily and varying inter-obstacle distances, AERO-MPPI achieves safe navigation by avoiding collisions and reaching the target reliably and smoothly. Additional hardware experiments are presented in the accompanying video.

VI. CONCLUSIONS

We introduced AERO-MPPI, an anchor-guided MPPI ensemble trajectory optimization framework that departs from conventional cascaded planning pipelines by unifying perception and planning into a single GPU-accelerated system. Unlike prior MPPI variants that rely on explicit maps or single-sample exploration, AERO-MPPI leverages multi-
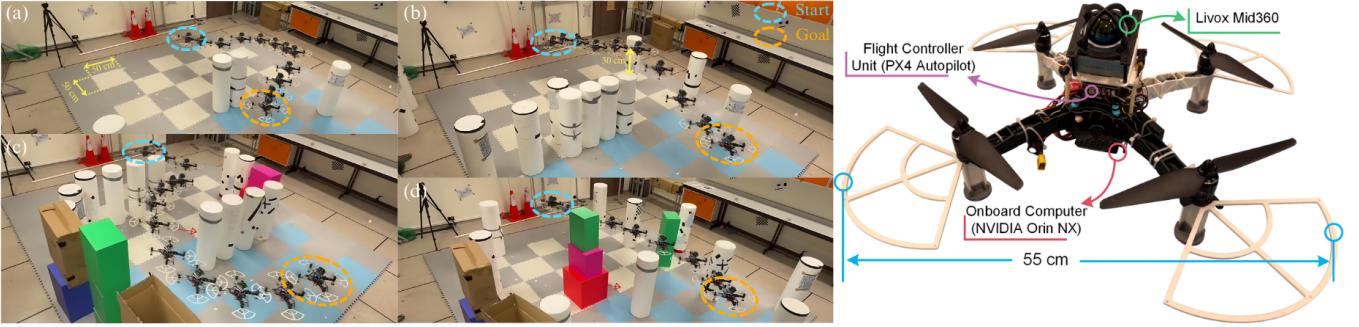
Fig. 7. **Indoor flight experiments in four representative scenarios.** Left: four representative scenarios — (a) Pillars: navigating around multiple vertical obstacles; (b) Wall: climbing over a barrier; (c) Double-Wall Trap: overcoming trap configurations; (d) Random Clutter: safe navigation through arbitrary obstacles. Right: the hardware platform used in the experiments. All experiments is initialized at 0.8 m height and tasked to reach a goal 3.75 m ahead.

resolution spherical partitioning and anchor-guided trajectory generation to diversify exploration and mitigate local minima. This design enables the planner to discover homotopically distinct solutions that are difficult for conventional methods to capture. Through extensive simulation studies, we showed that AERO-MPPI sustains agile flight above 7 m/s with success rates exceeding 80%, consistently maintaining larger smoother trajectories compared to SOTA baselines. Real-world experiments on a LiDAR-equipped quadrotor confirmed that the entire pipeline can run fully onboard on an NVIDIA Jetson Orin NX 16 GB, achieving reliable obstacle avoidance and robust navigation even in highly cluttered and unstructured environments.

Overall, these results demonstrate that AERO-MPPI provides a practical and scalable solution for mapless flight in complex 3D environments. Moving forward, we plan to extend the framework toward multi-robot coordination, outdoor large-scale deployments, and integration with learning-based perception and adaptive dynamics models, aiming to further enhance robustness and autonomy in dynamic real-world scenarios.

REFERENCES

[1] 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, Oct. 2019.

[2] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust Real-time UAV Replanning Using Guided Gradient-based Optimization and Topological Paths," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France: IEEE, May 2020, pp. 1208–1214.

[3] 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, Apr. 2021.

[4] W. Liu, Y. Ren, and F. Zhang, "Integrated Planning and Control for Quadrotor Navigation in Presence of Suddenly Appearing Objects and Disturbances," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 899–906, Jan. 2024.

[5] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[6] M. Macklin, "Warp: A high-performance python framework for gpu simulation and graphics," https://github.com/nvidia/warp, March 2022, nVIDIA GPU Technology Conference (GTC).

[7] 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, 2022.

[8] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[9] M. Minařík, R. Pěnička, V. Vonásek, and M. Saska, "Model Predictive Path Integral Control for Agile Unmanned Aerial Vehicles," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Abu Dhabi, United Arab Emirates: IEEE, Oct. 2024, pp. 13 144–13 151.

[10] T. Miura, N. Akai, K. Honda, and S. Hara, "Spline-interpolated model predictive path integral control with stein variational inference for reactive navigation," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 13 171–13 177.

[11] I. M. Balci, E. Bakolas, B. Vlahov, and E. A. Theodorou, "Constrained Covariance Steering Based Tube-MPPI," in *2022 American Control Conference (ACC)*, 2022, pp. 4197–4202.

[12] J. Yin, Z. Zhang, and P. Tsiotras, "Risk-aware model predictive path integral control using conditional value-at-risk," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 7937–7943.

[13] X. Cai, M. Everett, L. Sharma, P. R. Osteen, and J. P. How, "Probabilistic traversability model for risk-aware motion planning in off-road environments," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 11 297–11 304.

[14] C. Tao, H.-J. Yoon, H. Kim, N. Hovakimyan, and P. Voulgaris, "Path integral methods with stochastic control barrier functions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 1654–1659.

[15] J. Yin, C. Dawson, C. Fan, and P. Tsiotras, "Shield Model Predictive Path Integral: A Computationally Efficient Robust MPC Method Using Control Barrier Functions," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7106–7113, 2023.

[16] T. Jin, J. Di, X. Wang, and H. Ji, "Safety barrier certificates for path integral control: Safety-critical control of quadrotors," *IEEE Robotics and Automation Letters*, vol. 8, no. 9, pp. 6006–6012, 2023.

[17] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast Direct LiDAR-Inertial Odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.

[18] B. Vlahov, J. Gibson, M. Gandhi, and E. A. Theodorou, "MPPI-Generic: A CUDA Library for Stochastic Trajectory Optimization," 2024. [Online]. Available: https://arxiv.org/abs/2409.07563

[19] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[20] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.

[21] F. Kong, X. Liu, B. Tang, J. Lin, Y. Ren, Y. Cai, F. Zhu, N. Chen, and F. Zhang, "MARSIM: A Light-Weight Point-Realistic Simulator for LiDAR-Based UAVs," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2954–2961, 2023.