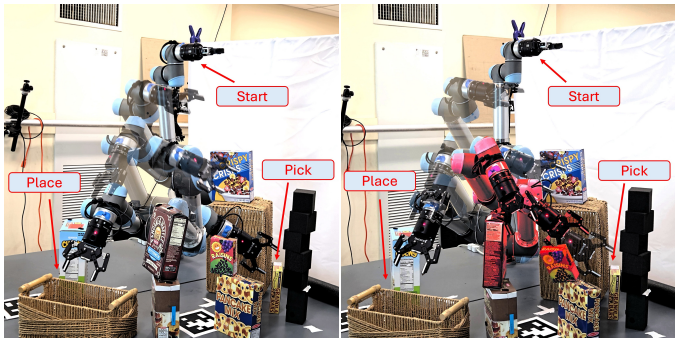
Ultrafast Sampling-based Kinodynamic Planning via Differential Flatness

Thai Duong, Clayton W. Ramsey, Zachary Kingston, Wil Thomason, and Lydia E. Kavraki

***Abstract*—Motion planning under dynamics constraints, *i.e.*, kinodynamic planning, enables safe robot operation by generating dynamically feasible trajectories that the robot can accurately track. For high-DoF robots such as manipulators, sampling-based motion planners are commonly used, especially for complex tasks in cluttered environments. However, enforcing constraints on robot dynamics in such planners requires solving either challenging two-point boundary value problems (BVPs) or propagating robot dynamics, both of which cause computational bottlenecks that drastically increase planning times. Meanwhile, recent efforts have shown that sampling-based motion planners can generate plans in microseconds using parallelization, but are limited to geometric paths. This paper develops `FLASK`, a fast parallelized sampling-based kinodynamic motion planning framework for a broad class of differentially flat robot systems, including manipulators, ground and aerial vehicles, and more. Differential flatness allows us to transform the motion planning problem from the original state space to a flat output space, where an analytical time-parameterized solution of the BVP problem can be obtained. A trajectory in the flat output space is then converted back to a closed-form dynamically feasible trajectory in the original state space, enabling fast validation via "single instruction, multiple data" parallelism. Our framework is fast, exact, and compatible with any sampling-based motion planner, while offering theoretical guarantees on probabilistic exhaustivity and asymptotic optimality based on the closed-form BVP solutions. We extensively verify the effectiveness of our approach in both simulated benchmarks and real experiments with cluttered and dynamic environments, requiring mere microseconds to milliseconds of planning time.**

***Index Terms*—Sampling-based Kinodynamic Planning, Differential Flatness, Hardware Acceleration**



(a) Tracking our trajectory (b) Tracking a geometric path

Fig. 1: Motion planning for a "*pick and place*" task in a cluttered environment: a dynamically feasible trajectory (a) generated from our `FLASK` framework can be accurately tracked by a UR5 robot. Meanwhile, tracking a geometric path (b) leads to collisions (shown in red) that topple the nearby boxes. Multiple intermediate states are overlaid to illustrate the robot's motion. Our planning framework is real-time and generates trajectories in $\sim 90\mu s$ by leveraging differential flatness and "single instruction, multiple data" (SIMD) parallelism.

I. Introduction

Motion planning is critical for safe and accurate robot operation in many applications, *e.g.*, transportation [1], environment monitoring [2, 3], warehouses [4], healthcare [5], and home assistance [6]. In such applications, the robot has to react quickly to changes in its environment, promptly (re)plan a collision-free trajectory, and safely track the trajectory to reach a goal state, using a controller. This task requires fast motion planning, subject to both collision avoidance and robot dynamics constraints, to generate a dynamically feasible trajectory from a start to a goal that the robot is able to follow. While recent advances have significantly reduced motion planning times for geometric planning via parallelization techniques [7, 8], sampling-based motion planning under dynamics constraints, *i.e.*, kinodynamic planning, remains a challenge for real-time applications especially for high degree-of-freedom (DoF) robots such as manipulators. In this paper, we address this problem by leveraging the *differential flatness* property of many common robot systems, such as ground and aerial vehicles, manipulators, and more, to enable ultrafast sampling-based kinodynamic motion planning via parallelization techniques.

Kinodynamic planning [9–11] generates dynamically feasible trajectories by directly enforcing dynamics constraints in the planner. Without such constraints, controllers often struggle to accurately track motion plans, especially with high-speed maneuvers, potentially leading to unsafe behavior. Optimization-based approaches, *e.g.*, [12–17], formulate kinodynamic planning as a nonlinear optimization problem and solve for the solution by minimizing a trajectory cost subject to constraints such as collision avoidance, robot dynamics, and joint angle and velocity limits. While these approaches can work well in high-dimensional state spaces, they are often susceptible to local minima and are sensitive to initial solutions. Meanwhile, search-based approaches, *e.g.*, [18–22], build a graph in the state space on a grid or lattice and search for a sequence of motion primitives that connects the start and the goal. Search-based methods provide optimality guarantees but suffer from the curse of dimensionality, thus requiring complex heuristics or domain knowledge to guide the search. On the other hand, sampling-based approaches randomly sample to

Thai Duong, Clayton W. Ramsey are with the Department of Computer Science, Rice University, Houston, TX, 77005, USA. Lydia E. Kavraki is with the Department of Computer Science, Rice University, Houston, TX, 77005 USA, and the Ken Kennedy Institute, Houston, TX, 77005 USA. Work by Zachary Kingston and Wil Thomason was done on this article while they were at Rice University. Correspondence: thaiduong@rice.edu, kavraki@rice.edu.

Videos and software will be made available at https://kavrakilab.org/flask.

Thai Duong and Lydia E. Kavraki were supported in part by NSF 2336612, NSF 2411219 and ERDC W912HZ2320003.

expand from the start to the goal, constructing a tree [9, 23–27] or sometimes a graph for simple systems [28].

A key challenge in enforcing dynamics constraints in sampling-based planning is to solve challenging two-point boundary value problems (BVPs) for dynamically feasible *local paths* between robot states. Many planners [9, 10, 26] avoid this by instead sampling an often sub-optimal control input and then propagating the dynamics forward using numerical integration [29]. Dynamic propagation with randomly sampled control often causes the planner to "wander" to irrelevant parts of the state space and might lead to longer paths and longer planning times. Other works solve the BVP problem for simple systems [24] and linearized robot dynamics [23, 30], or approximate the solution using a neural network [31–33]. Instead, we obtain an *exact analytical solution* of the BVP, thanks to the *differential flatness* of many robot platforms such as mobile robots and manipulators [34, 35].

Differential flatness [34] is a powerful system property that, similar to feedback linearization [36], simplifies motion planning and control designs by converting the nonlinear robot dynamics to an equivalent linear system. For differentially flat systems, the robot states and control inputs can be described by a set of carefully chosen flat outputs and their derivatives (see Def. 1 for more details). A trajectory in the flat output space can be converted to a dynamically feasible trajectory in the robot's original state space, allowing us to simplify motion planning problems via a change of variables. This technique has been used mainly to plan trajectories for mobile robots, most notably quadrotors [21, 35, 37]. However, the planning time is still hindered by computationally expensive subroutines such as forward kinematics and collision checking, which are particularly worse with high-DoF robots such as manipulators.

Recently, advances in parallelization [7, 38] have improved planning time to the range of microseconds and milliseconds. However, these works focus on either geometric [7] or kinematic [38] planning problems. One family of these techniques uses "fine-grained" parallelism via "single instruction, multiple data" (SIMD) instructions, available on consumer CPUs, to perform collision checking on multiple samples at the same time [7, 8]. However, directly enforcing dynamics constraints in parallel is non-trivial, as it requires sampling a trajectory at different time steps in advance while obeying robot dynamics. We address this problem by transforming the kinodynamic planning problem from the original state space to a flat output space, where closed-form *local paths* are time-parameterized polynomials and hence, amenable to SIMD parallelization for fast collision checking. The resulting flat output trajectory is converted to a dynamically feasible collision-free trajectory in the original state space for the robot to execute. Our approach is fast, exact and general for most sampling-based planners.

In summary, we develop **FLASK**[1], a "Differential **Fl**atness-based **A**ccelerated **S**ampling-based **K**inodynamic Planning" framework that:

- generates dynamically feasible trajectories in the range of milliseconds for nonlinear differentially-flat robot systems, such as manipulators, and mobile robots,
- constructs a planning tree or graph by solving the boundary value problem or dynamics propagation for continuous closed-form trajectories in the flat output space,
- performs fast collision checking of the closed-form trajectories using fine-grained SIMD parallelism.

We provide a theoretical analysis on our approach's probabilistic exhaustivity, a concept stronger than completeness introduced in [39, 40], and optimality guarantees. We extensively verify our approach with challenging motion planning problems on low- and high-DoF, fully- and under-actuated robot systems in both simulated and real experiments. **FLASK** is shown to achieve planning times of merely a few milliseconds, even on high-DoF robot platforms in cluttered environments, while respecting dynamics constraints by design. Our framework is general and can be integrated in the context of many common sampling-based motion planners, such as RRT-Connect [41], SST* [26] and others, effective turning geometric planners into kinodynamic planners with theoretical guarantees.

[1]The code will be made publicly available at https://kavrakilab.org/flask.

II. Related Work

A. Motion Planning with Dynamics Constraints

Kinodynamic motion planning [42], *i.e.*, motion planning under dynamics constraints, is a challenging task where the robot plans a dynamically feasible trajectory from a start, *e.g.*, a robot state, containing its joint angles and derivatives, to a goal region while satisfying the robot dynamics and avoiding collision with obstacles in the environment. There are three main approaches: optimization-based, search-based, and sampling-based kinodynamic planning.

Optimization-based kinodynamic planning, such as TrajOpt [13], GuSTO [14], and Crocoddyl [16], generates robot trajectories by formulating and solving an optimization problem, subject to dynamics and collision-avoidance constraints, with an objective function measuring the cost of the trajectory. This optimization problem is often nonlinear and can be solved via sequential convex programming [12–14, 43], iterative linear quadratic regulators [15], differential dynamic programming [16, 44], augmented Lagrangian methods [45], or general solvers [46, 47]. In general, optimization-based trajectory planners provide smooth trajectories, but often get stuck in a local minimum and require good initialization. While trajectory optimization with a graph of convex sets [48–50] can avoid local minima, it requires expensive precomputation of the graph in the state space.

Search-based kinodynamic planning [18–22, 51] instead constructs a graph, often on a predefined grid or lattice, where each edge is chosen from a precomputed, discrete set of motion primitives, generated by propagating the robot dynamics for a short period of time under a set of control inputs. A motion primitive is valid if it does not collide with an obstacle. A search algorithm, such as A^* [52], can be used to find the shortest path on the graph, providing a trajectory as a sequence of motion primitives connecting the start with the goal. A major challenge of search-based approaches is the need to precompute dynamics propagation, where a numerical approximation with fine lattice resolution is required for high accuracy. They also

suffer from the curse of dimensionality and require a good heuristic to guide the search.

Sampling-based kinodynamic planning [9, 10, 23–27, 53] uses sampling to discretize the high-dimensional state space and build a tree (or, less often, a graph for simple systems such as car-like robots [28]), growing from the start towards the goal region. To find a feasible trajectory connecting two samples, a difficult *two-point boundary value problem* (BVP) has to be solved [11], posing a major challenge for sampling-based kinodynamic planning. A common approach to avoid solving a BVP problem for tree expansion is to sample the control input space, and propagate the robot dynamics, *e.g.*, using a numerical integrator [9, 10, 26] or physics-based models [54], for a short period of time, with asymptotic optimality guarantees analyzed in [26]. Other approaches only solve the BVP problem for simple robot dynamics with low-dimensional state space [23, 24], or linearized dynamics [23, 30]. Meanwhile, learning-based kinodynamic motion planning uses neural networks to approximate the control input and the steering cost of expanding the tree towards a new node [31–33, 55, 56].

BVPs and dynamics propagation cause major computational bottlenecks for kinodynamic planning. While BVPs can be analytically solved for linear or simple systems [23, 30], this is not true in general for most nonlinear systems and it is computationally expensive to obtain an approximate solution. Kinodynamic RRT* [23, 30] linearizes the dynamics around an operating point and demonstrates that BVPs can be solved in closed form for certain robots, *e.g.*, quadrotors around a hovering position. However, the approximated solution only works well around the operating point, limiting aggressive maneuvers. To avoid solving difficult BVPs, most existing kinodynamic planners [26] resort to dynamics propagation, where a constant value of the control is sampled instead. A numerical integrator such as Euler's or Runge-Kutta methods [29] is then used to sequentially calculate the next robot state over multiple small time steps to maintain high accuracy. Typically, the sampled control is suboptimal, and therefore, causes the planner to wander in the state space before reaching the goal. Existing planners often mitigate this "wandering" effect via best-state selection and tree pruning [26], but still require long planning times to find a dynamically feasible trajectory.

Optimization-based, search-based and sampling-based approaches can be combined to improve trajectory generation [17, 57–65]. For example, a path or trajectory from sampling-based or search-based planners can be used as an initial solution for optimization-based ones [17, 57]. INSAT planners [58–60] interleave between search-based planning on a low-dimensional subspace and optimization-based planning on the full-dimensional space to improve planning times and success rates. Meanwhile, a library of precomputed motion primitives from search-based planning can be sampled to expand the planning tree in sampling-based motion planning [61, 62]. Furthermore, optimized local paths can be used to generate collision-free edges and bias the sampling regions in a sampling-based planner [63, 64]. Another approach is to fit a geometric path with a time-parameterized trajectory using trajectory optimization such as TOPP-RA [66] or Ruckig [67], typically without considering collision avoidance constraints.

Our method performs fast sampling-based kinodynamic planning by leveraging SIMD parallelism (Sec. II-B) and the differential flatness of the dynamics of common robot platforms [21, 68–70] to directly tackle the BVP problems. A system is called *differentially flat* if there exist variables, called flat outputs, whose values and derivatives dictate the robot state and control inputs (see Def. 1 for details). This approach has been applied to sampling-based [69, 71–73], search-based [21], and optimization-based motion planning [35, 70, 74–76], but primarily for low-dimensional robot platforms (often with simple geometry shapes such as spheres or boxes for collision checking), *e.g.*, quadrotors [35], unicycles and 2-link arms [76], or for a specific system, *e.g.*, gantry cranes [77]. Instead, we integrate differential flatness with sampling-based kinodynamic planning for generic high-DoF robots, where forward kinematics and collision checking are complex and time-consuming besides enforcing dynamics constraints. While most existing methods approximate a solution to the BVP problems, differential flatness allows us to obtain a closed-form fixed- or minimum-time polynomial BVP solution. Such closed-form trajectories are amenable to parallelized collision checking using *fine-grained* parallelization based on SIMD instructions (Sec. II-B), enabling trajectory generation in microseconds to milliseconds. It is also important to note that existing flatness-based sampling-based planners either do not prove completeness and optimality [71–73] or loosely mention the existing guarantees of geometric planners, *e.g.*, those of RRT* [69, 77]. Achieving such theoretical guarantees with closed-form BVP solutions is nontrivial as it requires careful consideration of nonlinear local paths rather than linear edges on the graph. We address this by offering a theoretical analysis of probabilistic exhaustivity, which is stronger than completeness, and asymptotic optimality of our approach in Sec. VI.

B. Hardware-accelerated Motion Planning

Motion planning can be time-consuming as it relies on multiple computationally expensive subroutines, such as forward kinematics (FK), collision checking (CC) and nearest neighbor (NN) search. With recent advances in parallel computing, much progress has been made to improve these subroutines and enhance planning performances via both *coarse-grained* and *fine-grained* parallelization.

Coarse-grained parallelization techniques typically run multiple subroutines or even multiple instances of the planners at the thread or process levels. Early work focuses on improving motion plans by merging and averaging out the paths from different instances of the planners [78], or by adapting existing planners to run their subroutines in parallel [79, 80]. Parallelized motion planning can also be achieved by partitioning the configuration space [81, 82] or the planning tree construction [77, 83, 84]. Closely related to our work, [77] also employs differential flatness to generate dynamically feasible trajectories, however, by coarsely growing multiple planning subtrees in parallel for a specific gantry crane system. Recently, the prevalence of GPUs enables impressive improvements in motion planning [38, 85–88] but suffers from costly GPU resources and communication

overhead between CPUs and GPUs. GPU-based planning methods often grow a planning tree in parallel by propagating the robot dynamics, *e.g.*, Kino-PAX [84], or via approximate dynamic programming recursion, *e.g.*, GMT* [89], and are shown to generate a robot trajectory in milliseconds for low-DoF systems with simple forward kinematics. For high-DoF robots, cuRobo [38] generates geometric paths as seeds for a parallelized trajectory optimization solver under kinematics constraints such as velocity, acceleration and jerk limits.

Fine-grained parallelization techniques focus on parallelizing primitive operations in a motion planning algorithm, *e.g.*, via "single instruction/multiple data" (SIMD) instructions on consumer-grade CPUs. This has been shown to provide extremely fast geometric motion planning subject to collision avoidance constraints, with planning times ranging from microseconds to milliseconds [7, 8, 90]. To check an edge (a line segment) for collision, VAMP [7] uses SIMD instructions to efficiently perform parallelized forward kinematics and collision checking on multiple configurations, generated via linear interpolation. While this approach is promising, it is challenging to enforce additional requirements via parallelized primitive operations such as dynamics and non-holonomic constraints. Particularly, enforcing nonlinear dynamics constraints with fine-grained parallelism is non-trivial due to the intractability of BVP problems, as mentioned in Sec. II-A. We instead leverage the differential flatness property to obtain time-parameterized solutions that can be discretized at arbitrary times and hence, amenable to SIMD-based primitive operations. Our deliberate fusion of differential flatness and SIMD parallelism effectively brings the benefits of "fine-grained" parallelized forward kinematics and collision checking to kinodynamic planning. Thanks to the closed-form BVP solution in the flat output space, we will later show that our approach achieves planning times of merely a few milliseconds.

III. Problem Formulation

Consider a robot with state $\mathbf{x} \in \mathcal{X}$ and control $\mathbf{u} \in \mathcal{U}$. For example, the state of a manipulator can include the joint configuration $\mathbf{q}$ and possibly its derivatives, while the control input can be the torques being applied on the robot joints. Let $\mathcal{X}_{free}$ and $\mathcal{X}_{obs} = \mathcal{X} \setminus \mathcal{X}_{free}$ be the free and occupied spaces, respectively, which can be generated from robot constraints such as collision avoidance or joints limits. The robot motion is governed by a nonlinear dynamics function $\mathbf{f}$ of the state $\mathbf{x}$ and the control $\mathbf{u}$ as follows:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{1}$$

A time-parameterized trajectory $\boldsymbol{\sigma} : [0,1] \to \mathcal{X}$ for time $t \in [0,1]$ is called dynamically feasible if there exists a time-parameterized control input $\mathbf{u} : [0,1] \to \mathcal{U}$ such that the robot dynamics is satisfied by the trajectory:

$$\dot{\boldsymbol{\sigma}}(t) = \mathbf{f}(\boldsymbol{\sigma}(t), \mathbf{u}(t)), \quad \forall t \in [0,1]. \tag{2}$$

Given an initial robot state $\mathbf{x}_s$ and a goal region $\mathcal{G} \subset \mathcal{X}$, as a subset of $\mathcal{X}$, the kinodynamic motion planning problem aims to find a dynamically feasible trajectory $\boldsymbol{\sigma}(t)$ with control input $\mathbf{u}(t)$, connecting the initial state $\mathbf{x}_s$ to the goal region $\mathcal{G}$ in the free space $\mathcal{X}_{free}$ as described in Problem 1. The goal region $\mathcal{G}$ commonly represents a desired state or a region that a goal state can be sampled from.

Problem 1. Given an initial robot state $\mathbf{x}_s$ and a goal region $\mathcal{G} \subset \mathcal{X}$, find a control input $\mathbf{u}(t)$ that generates a dynamically feasible trajectory $\boldsymbol{\sigma}(t)$ such that:

$$\begin{aligned} &\dot{\boldsymbol{\sigma}}(t) = \mathbf{f}(\boldsymbol{\sigma}(t), \mathbf{u}(t)), \\ &\boldsymbol{\sigma}(0) = \mathbf{x}_s, \boldsymbol{\sigma}(1) \in \mathcal{G}, \\ &\boldsymbol{\sigma}(t) \in \mathcal{X}_{free}, \mathbf{u}(t) \in \mathcal{U} \quad \forall t \in [0,1]. \end{aligned} \tag{3}$$

In the remainder of the paper, we solve Problem 1 by developing **FLASK**, a parallelized kinodynamic motion planning framework for differential flat robot systems, including common platforms such as ground and aerial vehicles, and manipulators. Our approach offers ultrafast planning time and exact dynamically feasible trajectory solution without the need to approximate the robot dynamics. Occasionally, we will drop the notation of time dependence for readability.

IV. Preliminaries

In this section, we provide a brief review and necessary background on sampling-based kinodynamic planning, differential flatness and fine-grained parallelization that will be useful for the derivation of our approach in Sec. V.

A. Sampling-based Kinodynamic Motion Planning

Most sampling-based geometric planners, such as RRT-connect [41] and PRM [91] consider the robot configuration $\mathbf{q}$ as the state $\mathbf{x}$ and approximate the robot's configuration space with a tree or graph $\mathbb{G}$ with a set of nodes $\mathbb{V}$ and a set of edges $\mathbb{E}$. Though individual sampling-based planners differ wildly in their exact approach, they all have roughly the same structure for their main search loop. At each iteration, such planners attempt to add a new sample $\mathbf{x}_f$ to $\mathbb{V}$, then connect $\mathbf{x}_f$ to some existing $\mathbf{x}_0 \in \mathbb{V}$, and if successful, add the resulting edge to $\mathbb{E}$. This is called a **CONNECT** or **EXTEND subroutine**. To construct edges, all geometric sampling-based planners require a *local planner* to produce a local path between configurations.

However, in kinodynamic motion planning, where robots are subjected to dynamics constraints, finding **a local path** $\mathbf{x}_{loc}(t)$ to reach $\mathbf{x}_f$ with control input $\mathbf{u}_{loc}(t)$ in duration T, requires solving a *boundary value problem* (BVP), subject to the robot dynamics with initial state $\mathbf{x}_0$ and terminal state $\mathbf{x}_f$ as follows:

$$\dot{\mathbf{x}}_{loc} = \mathbf{f}(\mathbf{x}_{loc}, \mathbf{u}_{loc}), \quad \mathbf{x}_{loc}(0) = \mathbf{x}_0, \mathbf{x}_{loc}(T) = \mathbf{x}_f. \tag{4}$$

Solving the BVP in practice is often computationally intractable, so most kinodynamic planners (*e.g.*, [9, 10, 26]) instead take a propagation-based approach: they integrate a sampled control input $\mathbf{u}_0$ from a reachable state for a short time T to generate a new state $\mathbf{x}_f$ rather than connecting sampled states. In this case, the **local path** $\mathbf{x}_{loc}(t)$ and the new state $\mathbf{x}_f$ are defined as:

$$\mathbf{x}_{loc}(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}_0) d\tau, \quad \mathbf{x}_f = \mathbf{x}_{loc}(T). \tag{5}$$

B. Differential Flatness

Definition 1 (Differential Flatness)**.** A dynamical system (1) is called "*differentially flat*" [34] if there exists an n-dimensional output:

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \dots, \mathbf{u}^{(k)}) \in \mathbb{R}^n, \tag{6}$$

such that the robot state $\mathbf{x}$ and control input $\mathbf{u}$ can be described in terms of the output $\mathbf{y}$ and its derivatives $\mathbf{y}^{(\cdot)}$:

$$\begin{aligned} \mathbf{x} &= \boldsymbol{\alpha}(\mathbf{y}, \dot{\mathbf{y}}, \dots, \mathbf{y}^{(l)}), \\ \mathbf{u} &= \boldsymbol{\beta}(\mathbf{y}, \dot{\mathbf{y}}, \dots, \mathbf{y}^{(m)}), \end{aligned} \tag{7}$$

for non-negative derivative orders k, l and m. The output $\mathbf{y}$ must be differentially independent, *i.e.*, there does not exist any differential relationship among the components of $\mathbf{y}$. The exact form of the functions $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ depends on the robot system, several of which can be found in [34, 35].

We provide three common examples of differentially flat fully-actuated and under-actuated robot platforms, as follows.

Example 1. Consider a *fully-actuated manipulator* with joint angles $\mathbf{q}$, and control input $\mathbf{u}$, *e.g.*, the joint torques. This is typically the case for common manipulators such as Franka or KUKA platforms. The robot dynamics is described by the Euler-Lagrange equation of motions:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{B}(\mathbf{q})\mathbf{u}, \tag{8}$$

where the control gain matrix $\mathbf{B}(\mathbf{q})$ is typically invertible, *i.e.*, the system is fully actuated. The robot dynamics can be expressed in the form of Eq. (1) with the robot state $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$. For this system, the flat output is the same as the configuration: $\mathbf{y} = \mathbf{q}$. The state $\mathbf{x}$ and the control input $\mathbf{u}$ can be derived from the flat output $\mathbf{y}$ as follows,

$$\begin{aligned} \mathbf{x} &= \boldsymbol{\alpha}(\mathbf{y}, \dot{\mathbf{y}}) = (\mathbf{q}, \dot{\mathbf{q}}) \\ \mathbf{u} &= \boldsymbol{\beta}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}) = \mathbf{B}^{-1}(\mathbf{q}) \left(\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\right). \end{aligned} \tag{9}$$

As the control gain $\mathbf{B}(\mathbf{q})$ is invertible, the control input $\mathbf{u}$ in (9) is guaranteed to exist.

Example 2. Consider a *unicyle* robot whose state $\mathbf{x}$ is defined as $\mathbf{x} = (x, y, \theta)$ where (x, y) is the position and θ is the heading angle of the vehicle. The control input $\mathbf{u} = (v, \omega)$ includes the speed v and angular velocity ω. The flat output $\mathbf{y}$ is defined as the position: $\mathbf{y} = (x, y)$. The yaw angle θ and the control input $\mathbf{u}$ can be determined from $\mathbf{y}$ as follows:

$$\begin{aligned} \mathbf{x} &= \boldsymbol{\alpha}(\mathbf{y}, \dot{\mathbf{y}}) = (x, y, \arctan2(\dot{y}, \dot{x}) + \kappa\pi), \\ \mathbf{u} &= \boldsymbol{\beta}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}) = \left((-1)^\kappa \sqrt{\dot{x}^2 + \dot{y}^2}, \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2}\right), \end{aligned} \tag{10}$$

where $\kappa \in \{0, 1\}$ depends on whether the vehicle is moving forward or backward, respectively.

Example 3. Consider an *under-actuated quadrotor* whose state $\mathbf{x}$ is defined as $\mathbf{x} = (\mathbf{p}, \mathbf{R}, \mathbf{v}, \boldsymbol{\omega})$, where $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ is the position of the center of mass, $\mathbf{R} \in SO(3)$ is the rotation matrix, $\mathbf{v}$ is the linear velocity, and $\boldsymbol{\omega}$ is the angular velocity. The control input $\mathbf{u} = (f, \boldsymbol{\tau})$ consists of a thrust $f \in \mathbb{R}_{\geq 0}$ and a torque $\boldsymbol{\tau} \in \mathbb{R}^3$, generated from the motors. The flat output for quadrotor systems is $\mathbf{y} = (\mathbf{p}, \psi) \in \mathbb{R}^4$, where ψ is the yaw angle of the robot [35].

The state $\mathbf{x} = \boldsymbol{\alpha}(\mathbf{p}, \dot{\mathbf{p}}, \ddot{\mathbf{p}}, \mathbf{p}^{(3)}, \psi, \dot{\psi})$ can be expressed in terms of the flat outputs and their derivatives as follows. Clearly, the position $\mathbf{p}$ is already part of the flat output $\mathbf{y}$, and hence the linear velocity is calculated as $\mathbf{v} = \dot{\mathbf{p}}$. Let m and $\mathbf{J}$ be the mass and inertia matrix of the quadrotor, respectively. Let $\mathbf{t} = m(\ddot{\mathbf{p}} + g\mathbf{e}_z)$ be the thrust vector applied on the quadrotor's center of mass, which coincides with the $z-$axis of the body frame where g is the gravitational acceleration, and $\mathbf{e}_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$ is the $z-$axis unit vector in the world frame. The rotation matrix $\mathbf{R} = \begin{bmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{r}_z \end{bmatrix}$ can be calculated as:

$$\mathbf{r}_z = \frac{\mathbf{t}}{\|\mathbf{t}\|}, \quad \mathbf{r}_x = \frac{\mathbf{r}_\psi \times \mathbf{r}_z}{\|\mathbf{r}_\psi \times \mathbf{r}_z\|}, \quad \mathbf{r}_y = \mathbf{r}_z \times \mathbf{r}_x, \tag{11}$$

where $\mathbf{r}_\psi = [-\sin\psi, \cos\psi, 0]$. The derivative of the rotation matrix is $\dot{\mathbf{R}} = \begin{bmatrix} \dot{\mathbf{r}}_x & \dot{\mathbf{r}}_y & \dot{\mathbf{r}}_z \end{bmatrix}$ with:

$$\begin{aligned} \dot{\mathbf{r}}_x &= \mathbf{r}_x \times \frac{\dot{\mathbf{r}}_\psi \times \mathbf{r}_z + \mathbf{r}_\psi \times \dot{\mathbf{r}}_z}{\|\mathbf{r}_\psi \times \mathbf{r}_z\|} \times \mathbf{r}_x, \\ \dot{\mathbf{r}}_y &= \dot{\mathbf{r}}_z \times \mathbf{r}_x + \mathbf{r}_z \times \dot{\mathbf{r}}_x, \quad \dot{\mathbf{r}}_z = \mathbf{r}_z \times \frac{\dot{\mathbf{t}}}{\|\mathbf{t}\|} \times \mathbf{r}_z. \end{aligned} \tag{12}$$

The angular velocity $\boldsymbol{\omega}$ is calculated as: $\boldsymbol{\omega} = (\mathbf{R}^\top \dot{\mathbf{R}})^\vee$, where the $(\cdot)^\vee$ operator maps a skew-symmetric vector $\hat{\boldsymbol{\omega}} \in \mathfrak{so}(3)$ to a vector $\boldsymbol{\omega} \in \mathbb{R}^3$. Meanwhile, the control input $\mathbf{u} = \boldsymbol{\beta}(\mathbf{p}, \dot{\mathbf{p}}, \ddot{\mathbf{p}}, \mathbf{p}^{(3)}, \mathbf{p}^{(4)}, \psi, \dot{\psi}, \ddot{\psi})$ is described as:

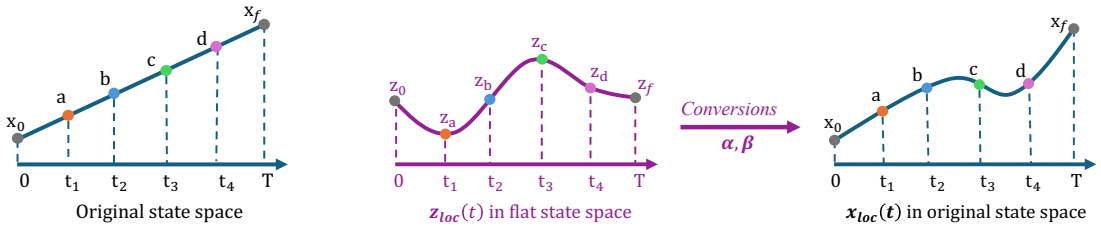
$$\mathbf{u} = (f, \boldsymbol{\tau}) = \left(\|\mathbf{t}\|, \mathbf{J}(\dot{\mathbf{R}}^\top \dot{\mathbf{R}} + \mathbf{R}^\top \ddot{\mathbf{R}})^\vee + \mathbf{R}^\top \dot{\mathbf{R}} \mathbf{J} \boldsymbol{\omega}\right). \tag{13}$$

We refer the readers to [35, 92, 93] for the detailed derivation.

C. CPU fine-grained parallelism

Fine-grained parallelization techniques such as "single instruction, multiple data" (SIMD) are ubiquitous on consumer CPUs and have recently shown significant improvement in sampling-based geometric motion planning by performing forward kinematics and collision checking on multiple configurations in parallel. Vectorization via SIMD allows simultaneously applying the same primitive operations on multiple variables. VAMP [7], a SIMD-accelerated planning method, uses this technique to perform parallelized collision checking of multiple robot configurations, evenly sampled along a line segment (Fig. 2a) connecting two nodes on a planning tree or graph. Given the set of configurations, the robot's geometric shape, described by a set of SIMD-compatible shapes such as capsules or spheres in the workspace, is calculated via branchless forward kinematics and checked for collision against the obstacle geometry.

SIMD-based collision checking requires access to all the states along a local path, typically through an analytical form of the motion. However, under dynamics constraints, the nonlinear local path $\mathbf{x}_{loc}(t)$ that solves the BVP problem (Sec. IV-A) is computationally intractable, prohibiting configuration sampling at multiple arbitrary times in advance. In the next section, we address this problem by deriving a closed-form time-parameterized local path $\mathbf{x}_{loc}(t)$ based on the differential flatness property and sampling multiple states at different times, as illustrated in Fig. 2b.



(a) Samples from a linear path [7].

(b) Samples from a nonlinear local path $\mathbf{x}_{loc}(t)$.

Fig. 2: Configuration samples $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{d}$, discretized from a linear path (a), as in VAMP [7], and from our closed-form time-parameterized motions (b), can be efficiently checked for collision using SIMD parallelism.

V. Technical Approach

In this section, we present our kinodynamic planning framework, **FLASK**, by showing that the flat output evolves as a linear system (Sec. V-A), and hence, allows us to convert Problem 1 from the original state space $\mathcal{X}$ to a flat state space (Problem 2 in Sec. V-B). In the flat state space, we develop primitive subroutines, FLASKEXTEND for planning graph construction in Sec. V-C and FLASKCC for parallelized forward kinematics and collision checking in Sec. V-D, that work with any sampling-based planning method. These subroutines are the core of our framework in Alg. 1, which effectively turns any geometric motion planner into a kinodynamic version. Instead of being tied to a specific planner, our general framework gives rise to a new class of ultrafast kinodynamic planners for a broad class of differentially flat robot systems. Finally, we discuss trajectory postprocessing in our approach in Sec. V-E.

A. Flat Output Dynamics as a Linear System

Consider the n-dimensional flat output $\mathbf{y}$ in Def. 1, that can be used to recover the state $\mathbf{x}$ and control input $\mathbf{u}$ via Eq. (7). Let $\mathbf{w} \in \mathcal{W} \subset \mathbb{R}^n$ be a pseudo-control input, defined as the rth derivative of the output $\mathbf{y}$:

$$\mathbf{w} = \mathbf{y}^{(r)}, \quad r \geq \max(l, m), \tag{14}$$

where the derivative orders l, m are defined in Eq. (7). Let $\mathbf{z} \in \mathcal{Z} \subset \mathbb{R}^{rn}$ be the flat state, defined as:

$$\mathbf{z} = (\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(r-1)}). \tag{15}$$

The state $\mathbf{z}$ satisfies linear dynamics with the pseudo-control input $\mathbf{w}$ as follows:

$$\dot{\mathbf{z}} = \mathbf{A}\mathbf{z} + \mathbf{B}\mathbf{w} \in \mathbb{R}^{rn}, \tag{16}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_n & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I}_n \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{I}_n \end{bmatrix},$$

with $\mathbf{A} \in \mathbb{R}^{rn \times rn}$, $\mathbf{B} \in \mathbb{R}^{rn \times n}$, and the identity matrix $\mathbf{I}_n \in \mathbb{R}^{n \times n}$. Instead of enforcing the nonlinear dynamics constraint (1) in a motion planning problem, we can implicitly enforce a much simpler linear dynamics (16) in the flat state space via the conversion (7). Note that the matrix $\mathbf{A}$ is nilpotent with index r, *i.e.*, $\mathbf{A}^r = 0$.

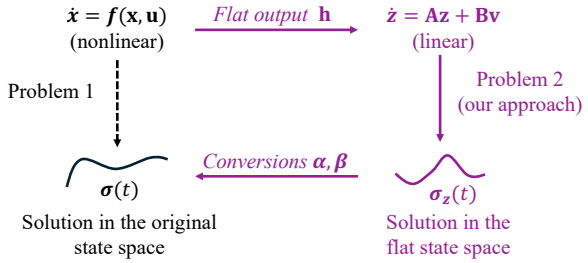


Fig. 3: Formulating the kinodynamic planning problem in the flat state space with linear dynamics.

B. Sampling-based Kinodynamic Planning in Flat State Space

Due to the non-linearity of the robot dynamics (1), it is challenging to plan robot motions in the original state space $\mathcal{X}$, as most sampling-based motion planning algorithms require a challenging EXTEND subroutine that either solves a boundary value problem to connect two states $\mathbf{x}_0$ and $\mathbf{x}_f$ or propagates the dynamics to predict the next state $\mathbf{x}_f$ given a constant control input $\mathbf{u}_0$. As shown in Sec. V-A, the flat state $\mathbf{z}$ satisfies a linear dynamics in (16) with a nilpotent matrix $\mathbf{A}$, potentially leading to much simpler BVP problem and dynamics propagation. This motivates us to perform motion planning in the flat state space $\mathcal{Z}$ instead of the original state space $\mathcal{X}$, thanks to the conversions (6) and (7). Our approach is illustrated in Fig. 3.

Given an initial state $\mathbf{x}_s$ with an initial control input $\mathbf{u}_s$, and a goal region $\mathcal{G}$ with a desired control $\mathbf{u}_\mathcal{G}$, the initial flat output and goal region are calculated as $\mathbf{y}_s = \mathbf{h}(\mathbf{x}_s, \mathbf{u}_s, \dot{\mathbf{u}}_s, \ldots, \mathbf{u}_s^{(k)})$ and $\mathcal{G}_\mathbf{y} = \{\mathbf{h}(\mathbf{x}, \mathbf{u}_\mathcal{G}, \dot{\mathbf{u}}_\mathcal{G}, \ldots, \mathbf{u}_\mathcal{G}^{(k)}) \mid \mathbf{x} \in \mathcal{G}\}$, respectively. Note that in many common robot systems such as Examples 1 to 3, the flat output $\mathbf{y}$ does not depend on $\mathbf{u}$ and therefore, the values of $\mathbf{u}_s$ and $\mathbf{u}_\mathcal{G}$ need not be specified. The original kinodynamic motion planning (Problem 1) becomes finding a dynamically feasible trajectory $\boldsymbol{\sigma}_\mathbf{z}(t)$ in the flat state space $\mathcal{Z}$ that connects the start $\mathbf{z}_s$ to the goal region $\mathcal{G}_\mathbf{z}$, and satisfies the linear dynamics (16), as summarized in Problem 2.

Problem 2. Given an initial flat state $\mathbf{z}_s$ and a goal region $\mathcal{G}_\mathbf{z}$, calculated from $\mathbf{y}_s$ and $\mathcal{G}_\mathbf{y}$ via (15), the original kinodynamic motion planning (Problem 1) is equivalent to finding a control input function $\mathbf{w}(t)$ that generates a dynamically feasible

trajectory $\boldsymbol{\sigma}_{\mathbf{z}}(t)$:

$$\begin{aligned}
&\dot{\boldsymbol{\sigma}}_{\mathbf{z}} = \mathbf{A}\boldsymbol{\sigma}_{\mathbf{z}} + \mathbf{B}\mathbf{w},\\
&\boldsymbol{\sigma}_{\mathbf{z}}(0) = \mathbf{z}_s, \boldsymbol{\sigma}_{\mathbf{z}}(1) \in \mathcal{G}_{\mathbf{z}},\\
&\mathbf{u}(t) = \boldsymbol{\beta}\left(\boldsymbol{\sigma}_{\mathbf{z}}(t), \mathbf{w}(t)\right) \in \mathcal{U},\\
&\mathbf{x}(t) = \boldsymbol{\alpha}(\boldsymbol{\sigma}_{\mathbf{z}}(t), \mathbf{w}(t)) \in \mathcal{X}_{free}, \forall t \in [0,1],
\end{aligned} \tag{17}$$

where the matrices $\mathbf{A}$ and $\mathbf{B}$ are defined in (16).

To solve Problem 2, we develop primitive subroutines in the flat state space $\mathcal{Z}$ for a sampling-based kinodynamic planning framework proposed in Alg. 1, which is consistent with most existing sampling-based motion planners [23, 24, 26]. Leveraging the linear dynamics (16), we develop an efficient FLASKEXTEND subroutine (see Sec. V-C) to construct a graph or tree $\mathbb{G}_{\mathbf{z}} = (\mathbb{V}_{\mathbf{z}}, \mathbb{E}_{\mathbf{z}})$ in the flat state space $\mathcal{Z}$, where $\mathbb{V}_{\mathbf{z}}$ and $\mathbb{E}_{\mathbf{z}}$ denote the sets of nodes and edges, respectively, by solving the BVP and dynamics propagation problems for a polynomial **local "flat" path** $\mathbf{z}_{loc}(t)$. The resulting trajectory $\boldsymbol{\sigma}_{\mathbf{z}}(t)$ is a continuous piecewise-polynominal, consisting of M local "flat" paths found on $\mathbb{G}_{\mathbf{z}}$:

$$\begin{aligned}
\boldsymbol{\sigma}_{\mathbf{z}}(t) &= \{(\mathbf{z}_i(t), t_i)\}_{i=1}^{M},\\
\mathbf{w}_{\mathbf{z}}(t) &= \{\mathbf{w}_i(t), t_i)\}_{i=1}^{M},
\end{aligned} \tag{18}$$

with $0 = t_0 < t_1 < \ldots < t_M$ where the pseudo-control $\mathbf{w}_i(t)$ is applied in the time interval $[t_{i-1}, t_i]$, for $i = 1, \ldots, M$. The trajectory $\boldsymbol{\sigma}_{\mathbf{z}}(t)$ is converted back to a trajectory $\boldsymbol{\sigma}(t)$ with control $\mathbf{u}(t)$ as follows:

$$\begin{aligned}
\boldsymbol{\sigma}(t) &= \boldsymbol{\alpha}(\boldsymbol{\sigma}_{\mathbf{z}}(t), \mathbf{w}(t)),\\
\mathbf{u}(t) &= \boldsymbol{\beta}(\boldsymbol{\sigma}_{\mathbf{z}}(t), \mathbf{w}(t)),
\end{aligned} \tag{19}$$

which is possible because the pseudo-control input $\mathbf{w}$ is the r-th order derivative of $\mathbf{y}$ with $r \geq \max(l, m)$ in Eq. (14).

C. FLASKEXTEND *Subroutine on Flat Output Space*

The goal of the FLASKEXTEND subroutine, described in Alg. 2, is to find a collision-free dynamically feasible **local "flat" path** $\mathbf{z}_{loc}(t), t \in [0, T]$ with a time duration T, that connects an existing node $\mathbf{z}_0 \in \mathbb{V}$ to a new node $\mathbf{z}_f$. A motion $\mathbf{z}_{loc}(t)$ can be generated by either sampling $\mathbf{z}_f$ and solving a BVP problem (Sec. V-C1) or sampling a pseudo-control input $\mathbf{w}_0$ and integrating the flat state dynamics (Sec. V-C2). For either case, we show that a closed-form expression of $\mathbf{z}_{loc}(t)$ can be obtained for parallelized collision checking in FLASKCC subroutine in Sec. V-D. If $\mathbf{z}_{loc}(t)$ is valid, the node $\mathbf{z}_f$ is added to $\mathbb{V}_{\mathbf{z}}$ while the edge $(\mathbf{x}_0, \mathbf{x}_f)$ associated with the local "flat" path $\mathbf{z}_{loc}(t)$ is added to $\mathbb{E}_{\mathbf{z}}$.

1) Solving the BVP Problem in Closed Forms: Given an existing node $\mathbf{z}_0$ and a sampled $\mathbf{z}_f$, we find a local "flat" path $\mathbf{z}_{loc}(t), t \in [0, T]$ that connects $\mathbf{z}_0$ and $\mathbf{z}_f$ and satisfies the flat state dynamics (16). Consider a cost function of a motion $\mathbf{z}(t), t \in [0, T]$ with pseudo-control $\mathbf{w}(t)$ that accounts for the total control effort and the time duration T as follows:

$$\mathcal{C}(\mathbf{z}(t), \mathbf{w}(t), T) = \int_0^T \mathbf{w}^\top \mathbf{R}\mathbf{w} dt + \rho T, \tag{20}$$

where $\mathbf{R} \succ 0$ is a user-defined symmetric positive definite weight matrix, and ρ controls the trade-off between the control effort and the time it takes to finish the trajectory. We formulate a Linear Quadratic Minimum Time (LQMT) problem [94] to solve for our local "flat" path $\mathbf{z}_{loc}(t)$ and control $\mathbf{w}_{loc}(t)$:

$$\begin{aligned}
&\min_{\mathbf{w}(t), T} \mathcal{C}(\mathbf{z}(t), \mathbf{w}(t), T)\\
&\text{s.t.} \quad \dot{\mathbf{z}} = \mathbf{A}\mathbf{z} + \mathbf{B}\mathbf{w},\\
&\qquad \mathbf{z}(0) = \mathbf{z}_0, \mathbf{z}(T) = \mathbf{z}_f,
\end{aligned} \tag{21}$$

Algorithm 1: **FLASK**: SAMPLING-BASED KINODYNAMIC MOTION PLANNING VIA DIFFERENTIAL FLATNESS

Input: Initial state $\mathbf{x}_s$, initial control $\mathbf{u}_s$, goal region $\mathcal{G}$ with control input $\mathbf{u}_{\mathcal{G}}$, maximum iterations I
Output: A collision-free dynamically feasible trajectory $\boldsymbol{\sigma}(t)$ with control input $\mathbf{u}(t)$.

/* Convert to the flat state space */
$\mathbf{y}_0 \leftarrow \mathbf{h}(\mathbf{x}_0, \mathbf{u}_0, \dot{\mathbf{u}}_0, \ldots, \mathbf{u}_0^{(k)})$
$\mathcal{G}_{\mathbf{y}} \leftarrow \{\mathbf{h}(\mathbf{x}, \mathbf{u}_{\mathcal{G}}, \dot{\mathbf{u}}_{\mathcal{G}}, \ldots, \mathbf{u}_{\mathcal{G}}^{(k)}) : \mathbf{x} \in \mathcal{G}\}$
$\mathbf{z}_0 \leftarrow (\mathbf{y}_0, \dot{\mathbf{y}}_0, \ldots, \mathbf{y}_0^{(r-1)})$
$\mathcal{G}_z = \{(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(r-1)}) : \mathbf{y} \in \mathcal{G}_{\mathbf{y}}\}$
Create a planning graph or tree $\mathbb{G}_{\mathbf{z}} = (\mathbb{V}_{\mathbf{z}}, \mathbb{E}_{\mathbf{z}})$ with set of vertices $\mathbb{V}_{\mathbf{z}}$ and set of edges $\mathbb{E}_{\mathbf{z}}$.
$\mathbb{V}_{\mathbf{z}} \leftarrow \{\mathbf{z}_0\}$
while *Iteration* $i < I$ **do**
  /* Grow $\mathbb{G}_{\mathbf{z}}$ on the flat state space */
  $\mathbb{G}_{\mathbf{z}} \leftarrow$ FLASKEXTEND($\mathbb{G}_{\mathbf{z}}$)
  /* Return $\boldsymbol{\sigma}(t)$ if reaching $\mathcal{G}_{\mathbf{z}}$ */
  if *Goal region* $\mathcal{G}_{\mathbf{z}}$ *is reached* **then**
    Find the trajectory $\boldsymbol{\sigma}_{\mathbf{z}}(t)$ with pseudo-input $\mathbf{w}(t)$ from $\mathbb{G}_{\mathbf{z}}$.
    /* Convert the trajectory $\boldsymbol{\sigma}_z(t)$ to the original state space */
    $\boldsymbol{\sigma}(t) \leftarrow \boldsymbol{\alpha}(\boldsymbol{\sigma}_{\mathbf{z}}(t), \mathbf{w}(t))$
    $\mathbf{u}(t) \leftarrow \boldsymbol{\beta}(\boldsymbol{\sigma}_{\mathbf{z}}(t), \mathbf{w}(t))$
    return Trajectory $\boldsymbol{\sigma}(t)$ with control $\mathbf{u}(t)$.
return Trajectory not found.

Algorithm 2: FLASKEXTEND

Input: The planning graph/tree $\mathbb{G}_{\mathbf{z}} = (\mathbb{V}_{\mathbf{z}}, \mathbb{E}_{\mathbf{z}})$
Output: Updated $\mathbb{G}_{\mathbf{z}}$

if *Sample* $\mathbf{z}_f \in \mathcal{Z}$ **then**
  Pick an existing node $\mathbf{Z}_0 \in \mathbb{V}_{\mathbf{z}}$
  /* Solve BVP in closed form */
  $\mathbf{z}_{loc}(t) \leftarrow$ Eq. (25) with a sampled T or an optimal $T = T^*$ from (26).
  $\mathbf{w}_{loc}(t) \leftarrow$ Eq. (23).
if *Sample* $(\mathbf{z}_0, \mathbf{w}_0, T) \in \mathbb{V}_{\mathbf{z}} \times \mathbb{R}^n \times \mathbb{R}_+$ **then**
  /* Analytically propagate dynamics */
  $\mathbf{z}_{loc}(t) \leftarrow$ Eq. (29).
  $\mathbf{w}_{loc}(t) \leftarrow \mathbf{w}_0$.
if not FLASKCC($\mathbf{z}_{loc}(t), \mathbf{w}_{loc}(t)$) **then**
  $\mathbb{V}_{\mathbf{z}} \leftarrow \mathbb{V}_{\mathbf{z}} \cup \{\mathbf{z}_f\}$
  $\mathbb{E}_{\mathbf{z}} \leftarrow \mathbb{E}_{\mathbf{z}} \cup \{(\mathbf{z}_0, \mathbf{z}_f, (\mathbf{z}_{loc}(t), \mathbf{w}_{loc}(t), T))\}$
return $\mathbb{G}$

where the matrices $\mathbf{A}$ and $\mathbf{B}$ are defined in (16). Let us define $\mathbf{d}_T = \mathbf{z}_f - e^{\mathbf{A}T}\mathbf{z}_0$ and the Gramian matrix

$$\mathbf{G}_T = \int_0^T e^{\mathbf{A}t}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T e^{\mathbf{A}^\top t} dt. \tag{22}$$

The duration T can be either a) set to a fixed or sampled value or b) optimized for a minimum-time trajectory, as shown next. Given a sample $\mathbf{z}_f$, the cost $\mathcal{C}_{loc}$ of the local path $\mathbf{z}_{loc}(t)$ is commonly used to choose an existing node $\mathbf{z}_0$ on the graph $\mathbb{G}_{\mathbf{z}}$, *e.g.*, those in the neighborhood of $\mathbf{z}_f$ with $\mathcal{C}_{loc}$ smaller than a threshold ζ (see Sec. VI for how to choose the value of ζ as the number of nodes $N = |\mathbb{V}_{\mathbf{z}}|$ increases).

a) Fixed-time Optimal Local Paths: For a fixed time duration T, the LQMT problem (21) has a closed-form solution, by following [94], for the optimal pseudo-control input:

$$\mathbf{w}_{loc}(t) = \mathbf{R}^{-1}\mathbf{B}^\top e^{\mathbf{A}^\top(T-t)}\mathbf{G}_T^{-1}\mathbf{d}_T. \tag{23}$$

with the optimal cost:

$$\mathcal{C}_{loc}(T) = \mathbf{d}_T^\top \mathbf{G}_T^{-1}\mathbf{d}_T + \rho T, \tag{24}$$

and the optimal local "flat" path:

$$\mathbf{z}_{loc}(t) = e^{\mathbf{A}t}\mathbf{z}_0 + \mathbf{G}_t e^{A^\top(T-t)}\mathbf{G}_T^{-1}\mathbf{d}_T. \tag{25}$$

Since the matrix $\mathbf{A}$ is nilpotent, *i.e.,* $\mathbf{A}^r = 0$, we have $e^{\mathbf{A}t} = \sum_{j=0}^{r-1}\frac{\mathbf{A}^j t^j}{j!}$. Therefore, the Grammian $\mathbf{G}_T$ and $\mathbf{d}_T$ become polynomials of the time duration T. The optimal control input $\mathbf{w}_{loc}(t)$ becomes a $(r-1)$th order polynomial while the optimal flat output trajectory $\mathbf{y}_{loc}(t)$ is a $(2r-1)$th order polynomial, which is compatible with SIMD-based collision checking in Sec. V-D.

b) Minimum-time Optimal Local Paths: If we have the freedom to choose the duration T, we can solve the following equation for a minimum time $T = T^*$ [94]:

$$\frac{d}{dT}\mathcal{C}_{loc}(T) = 0, \qquad T > 0. \tag{26}$$

For an arbitrary value of the pseudo-control order r, the minimum-time condition (26) can be solved for a positive T^* using a numerical root-finding solver [95], that is amenable to SIMD parallelization such as L-BFGS [96]. More notably, many common systems such as manipulators (Example 1) or unicycles (Example 2) have $r = 2$ while systems with higher r, such as quadrotors (Example 3), can reduce the pseudo-control order to $r = 2$ to simplify motion planning in practice, as shown in [21]. For such cases, Example 4 below shows the optimal local "flat" path, pseudo-control input, and optimal duration, where the condition (26) is equivalent to solving a 4th-order polynomial in (28) for a positive root, which can also be done in closed forms. As a result, solving for an optimal time $T = T^*$ is suitable for SIMD parallelization.

Example 4 (Fixed-time and minimum-time local paths). We consider a *special case with* $r = 2$ and $\mathbf{R} = \mathbf{I}_n$, *e.g.*, for a fully-actuated manipulator or a unicycle. Given a flat output $\mathbf{y}$, the flat state is $\mathbf{z} = (\mathbf{y}, \dot{\mathbf{y}})$ with the pseudo-control input $\mathbf{w}$ defined as the acceleration: $\mathbf{w} = \ddot{\mathbf{y}}$. Similar to [21] (for $n = 3$), the flat state $\mathbf{z}$ follows the linear dynamics (16) with:

$$\mathbf{A} = \begin{bmatrix}\mathbf{0} & \mathbf{I}_n \\ \mathbf{0} & \mathbf{0}\end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix}\mathbf{0} \\ \mathbf{I}_n\end{bmatrix}. \tag{27}$$

Since $\mathbf{A}^j = 0$ for all $j \geq 2$, we have $e^{\mathbf{A}t} = \mathbf{I}_{2n} + \mathbf{A}t$ and $e^{\mathbf{A}^\top t} = \mathbf{I}_{2n} + \mathbf{A}^\top t$, leading to:

$$\mathbf{d}_T = \mathbf{z}_f - (\mathbf{I}_{2n} + \mathbf{A}T)\mathbf{z}_0 = \mathbf{z}_f - \mathbf{z}_0 - T\mathbf{A}\mathbf{z}_0,$$
$$\mathbf{G}_T = \int_0^T e^{\mathbf{A}t}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T e^{\mathbf{A}^\top t} dt = \begin{bmatrix}\frac{T^3}{3}\mathbf{I}_n & \frac{T^2}{2}\mathbf{I}_n \\ \frac{T^2}{2}\mathbf{I}_n & T\mathbf{I}_n\end{bmatrix}.$$

The matrix inverse of $\mathbf{G}_T$ can be derived using the Schur complement [97]: $\mathbf{G}_T^{-1} = \begin{bmatrix}\frac{12}{T^3}\mathbf{I}_n & -\frac{6}{T^2}\mathbf{I}_n \\ -\frac{6}{T^2}\mathbf{I}_n & \frac{4}{T}\mathbf{I}_n\end{bmatrix}$. The optimal control input (23) becomes:

$$\begin{aligned}\mathbf{w}_{loc}(t) &= \mathbf{R}^{-1}\mathbf{B}^\top e^{\mathbf{A}^\top(T-t)}\mathbf{G}_T^{-1}\mathbf{d}_T, \\ &= \begin{bmatrix}-\frac{12\mathbf{I}_n}{T^3} & \frac{6\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t + \begin{bmatrix}\frac{6\mathbf{I}_n}{T^2} & -\frac{2\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T.\end{aligned}$$

This leads to a minimum-acceleration local "flat" path $\mathbf{z}_{loc}(t)$ as follows:

$$\begin{aligned}\mathbf{z}_{loc}(t) &= \mathbf{G}_t e^{A^\top(T-t)}\mathbf{G}_T^{-1}\mathbf{d}_T + e^{\mathbf{A}t}\mathbf{z}_0, \\ &= \begin{bmatrix}\begin{bmatrix}-\frac{2\mathbf{I}_n}{T^3} & \frac{\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t^3 + \begin{bmatrix}\frac{3\mathbf{I}_n}{T^2} & -\frac{\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T t^2 + \dot{\mathbf{y}}_0 t + \mathbf{y}_0 \\ \begin{bmatrix}-\frac{6\mathbf{I}_n}{T^3} & \frac{3\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t^2 + \begin{bmatrix}\frac{6\mathbf{I}_n}{T^2} & -\frac{2\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T t + \dot{\mathbf{y}}_0\end{bmatrix}.\end{aligned}$$

In other words, the optimal flat output is:

$$\mathbf{y}_{loc}(t) = \begin{bmatrix}-\frac{2\mathbf{I}_n}{T^3} & \frac{\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t^3 + \begin{bmatrix}\frac{3\mathbf{I}_n}{T^2} & -\frac{\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T t^2 + \dot{\mathbf{y}}_0 t + \mathbf{y}_0.$$

The optimal cost function (24) is calculated as:

$$\begin{aligned}\mathcal{C}_{loc}(T) &= \mathbf{d}_T^\top \mathbf{G}_T^{-1}\mathbf{d}_T + \rho T, \\ &= \frac{12\|\mathbf{y}_f - \mathbf{y}_0\|^2}{T^3} - \frac{12(\dot{\mathbf{y}}_f + \dot{\mathbf{y}}_0)^\top(\mathbf{y}_f - \mathbf{y}_0)}{T^2} \\ &\quad + \frac{4(\|\dot{\mathbf{y}}_0\|^2 + \dot{\mathbf{y}}_0^\top\dot{\mathbf{y}}_f + \|\dot{\mathbf{y}}_f\|^2)}{T} + \rho T.\end{aligned}$$

To find a minimum time T, we solve $d\mathcal{C}_{loc}/dT$ for a positive real root T^*:

$$\begin{aligned}\frac{d\mathcal{C}_{loc}(T)}{dT} &= -\frac{36\|\mathbf{y}_f - \mathbf{y}_0\|^2}{T^4} + \frac{24(\dot{\mathbf{y}}_f + \dot{\mathbf{y}}_0)^\top(\mathbf{y}_f - \mathbf{y}_0)}{T^3} \\ &\quad - \frac{4(\|\dot{\mathbf{y}}_0\|^2 + \dot{\mathbf{y}}_0^\top\dot{\mathbf{y}}_f + \|\dot{\mathbf{y}}_f\|^2)}{T^2} + \rho,\end{aligned}$$

which is equivalent to solving a 4th-order polynomial with closed-form solutions:

$$\begin{aligned}&-36\|\mathbf{y}_f - \mathbf{y}_0\|^2 + 24(\dot{\mathbf{y}}_f + \dot{\mathbf{y}}_0)^\top(\mathbf{y}_f - \mathbf{y}_0)T \\ &\quad - 4(\|\dot{\mathbf{y}}_0\|^2 + \dot{\mathbf{y}}_0^\top\dot{\mathbf{y}}_f + \|\dot{\mathbf{y}}_f\|^2)T^2 + \rho T^4 = 0.\end{aligned} \tag{28}$$

The detailed derivation can be found in Sec. IX-A.

c) Remarks: We note that the closed-form polynomial solution (25) of the LQMT problem (21) is derived without considering additional constraints such as collision avoidance, velocity and acceleration limits. In our approach, such constraints are described by an occupied regions in the flat state space (see Def. 4 in Sec. VI) and enforced via collision checking in Sec. V-D. If the trajectory $\mathbf{z}_{loc}(t)$ in (25) from $\mathbf{z}_0$ to $\mathbf{z}_f$ violates a constraint, it will be considered invalid. However, it does not necessarily mean that $\mathbf{z}_f$ is unreachable from $\mathbf{z}_0$. If there exists a constrained optimal trajectory $\mathbf{z}^*_{loc}(t)$, our Theorem 1 in Sec. VI shows that as the number of samples $N = |\mathbb{V}_{\mathbf{z}}|$ goes to infinity, the probability of having a piecewise-polynomial trajectory that is close to $\mathbf{z}^*_{loc}(t)$ (see Def. 2) tends to 1, *i.e.*, $\mathbf{z}_f$ can still be reached via a sequence of nodes.

Interestingly, recent results from optimal control [76] find an optimal piecewise-polynomial local path under constraints for differentially flat systems by carefully switching between modes, each of which corresponds to a polynomial motion primitive. This approach solves an optimality condition on the cost function and the constraints to find an optimal mode schedule and is shown to generate an optimal trajectory for low-DOF systems with simple closed-form constraints in a few milliseconds. However, it is challenging to design such a mode-switching mechanism for high-DOF robots with complex geometry due to complicated forward kinematics, especially when the local path generation time is often restricted to the nanosecond range. Intriguingly, from a sampling-based perspective, our Theorem 1 (see Sec. VI) shows that such an optimal piecewise-polynomial local path can be found on our planning graph with high probability, *i.e.*, a sequence of nodes on our graph is equivalent to the mode-switching schedule from [76]. Nevertheless, this is an exciting direction for generating a constrained local path $\mathbf{z}_{loc}(t)$ in our framework that we leave for future work.

2) Dynamics Propagation in Closed-Forms: As BVP problems are challenging to solve, most existing work [26] resorts to dynamics propagation, where the planning tree is extended by sampling a constant control input and integrating the robot dynamics over small time steps using numerical methods. While our *BVP closed-form solution* in Sec. V-C1 *is the main focus of our paper*, this section shows that transforming the kinodynamic planning problem to the flat state space (Problem 2) also leads to closed-form dynamics propagation, and hence gets rid of numerical approximations with potentially high accumulated errors. Instead of sampling the original control input, we sample the pseudo-control input $\mathbf{w}_{loc}(t) = \mathbf{w}_0$ and the duration T, leading to a closed-form local path:

$$\mathbf{z}_{loc}(t) = \left(\mathbf{y}_{loc}, \dot{\mathbf{y}}_{loc}, \dots, \mathbf{y}_{loc}^{(r-1)}\right), \tag{29}$$

where $\mathbf{y}_{loc}(t) = \frac{\mathbf{w}_0}{r!}t^r + \sum_{i=0}^{r-1} \frac{\mathbf{y}_0^{(i)}}{i!}t^i$. These local "flat" paths can be used in tree-based kinodynamic planners such as Stable Sparse RRT [26] and are also suitable for parallelized forward kinematics and collision checking in Sec. V-D. Propagation-based planners tend to "wander" in the state space due to the suboptimality of random control with small time steps T. Interestingly, our BVP solutions can help by finding shortcuts to the goal rather than keep expanding the tree, and therefore reduce the planning time significantly as illustrated in Sec. VII-A.

D. Vectorized Collision Checking

To perform collision checking on a local "flat" path $\mathbf{z}_{loc}(t)$ with pseudo-control $\mathbf{w}_{loc}(t)$ (Alg. 3), we convert it back to the original state space $\mathcal{X}$ and obtain the corresponding closed-form local path and control:

$$\begin{aligned} \mathbf{x}_{loc}(t) &= \boldsymbol{\alpha}(\mathbf{z}_{loc}(t), \mathbf{w}_{loc}(t)) \\ \mathbf{u}_{loc}(t) &= \boldsymbol{\beta}(\mathbf{z}_{loc}(t), \mathbf{w}_{loc}(t)). \end{aligned} \tag{30}$$

We next discretize $\mathbf{x}_{loc}(t)$ and $\mathbf{u}_{loc}(t)$ at N times: $t_1, t_2, \dots, t_N \in [0, T]$, grouped into several spatially distributed batches of size K:

$$\mathbf{b}_i = \left\{t_i, t_{\lceil \frac{N}{K} \rceil + i}, t_{2\lceil \frac{N}{K} \rceil + i}, \dots, t_{(K-1)\lceil \frac{N}{K} \rceil + i}\right\},$$

Algorithm 3: FLASKCC

Input: Flat state path $\mathbf{z}_{loc}(t)$ with pseudo-control $\mathbf{w}(t)$
Output: Whether $\mathbf{z}_{loc}(t)$ collides with obstacles

```
/* Convert to the original state space */
x_loc(t) ← α(z_loc(t), w_loc(t)),
u_loc(t) ← β(z_loc(t), w_loc(t))
for i ∈ {0, ..., ⌈N/K⌉} do
    /* Parallel checking via SIMD [7] */
    if ∃j ∈ b_i : x_loc(t_j) ∈ X_obs then
        return true
    /* checking other constraints such as
       state and control limits */
    if ∃j ∈ b_i: x_loc(t_j), u_loc(t_j) violate other
     constraints then
        return true
return false
```

for $i = 0, \dots, \lceil \frac{N}{K} \rceil$. For each batch $\mathbf{b}_i$, we obtain a set of samples $\{\mathbf{x}_{loc}(t_j)\}_{t_j \in \mathbf{b}_i}$ and perform fast parallelized collision checking via SIMD instructions, as illustrated in Fig. 2b. The batch size K is the number of floats that a SIMD register can store, *e.g.*, $K = 8$ for the commonly used AVX2 instruction set. The robot's geometry is represented by a set of SIMD-compatible primitive shapes such as spheres or capsules, whose poses are calculated via forward kinematics for multiple states and checked for collisions with obstacles using SIMD parallelism. If any state in batch $\mathbf{b}_i$ leads to collisions with an obstacle, we terminate the collision checking subroutine early and move on to other local paths.

Similarly, other constraints such as state and control limits can be checked in parallel for each batch $\mathbf{b}_i$ to validate $\mathbf{z}_{loc}(t)$. In general, the conversions (6) and (7) are nonlinear and might complicate the exact conversions of state and control limits on $(\mathbf{x}, \mathbf{u})$ to corresponding flat state and pseudo-control limits on $(\mathbf{z}, \mathbf{w})$. For many common robot systems, an upper bound on the conversions (7) can be obtained to estimate the limits in the flat state space as the flat output $\mathbf{y}$ is often part of the original state $\mathbf{x}$ (see Examples 1, 2 and 3). For example, any limits on the joint angles and velocities of a manipulator in Example 1 can be directly translated to the flat state or any limits on the speed v of a unicyle in Example 2 can be used to bound the first derivative of the flat out put $(\dot{x}, \dot{y})$. Furthermore, since the flat output describes certain physical properties of the robot, *e.g.*, the position and yaw angle of a quadrotor in Example 3, users can also directly specify the flat state and pseudo-control constraints, *e.g.*, maximum linear velocity and yaw rates. For general nonlinear systems, this issue can be addressed by using large limits in the flat state space, and checking the local path $(\mathbf{x}_{loc}(t), \mathbf{u}_{loc}(t))$ in (30) against the original state and control limits on $(\mathbf{x}, \mathbf{u})$ via sampling. However, there is a trade-off between the size of the flat state space and the risk of missing valid paths, which can be tuned based on the planner's performance, *e.g.*, the larger flat state space may lead to longer planning time but reduce the risk.

Algorithm 4: TRAJECTORY POSTPROCESSING

Input: A collision-free piecewise-polynomial trajectory $\boldsymbol{\sigma}_{\mathbf{z}}(t) = \{(\mathbf{z}_i(t), t_i)\}_{i=1}^{M}$, with control inputs $\mathbf{w}_{\mathbf{z}}(t) = \{\mathbf{w}_i(t), t_i)\}_{i=1}^{M}$,

1 **for** $i \in \{1, \ldots, M\}$ **do**
2 **for** $j \in \{M, \ldots, 1\}$ **do**
3 Calculate $\mathbf{z}_{ij}(t)$ from Eq. (25) with $\mathbf{z}_0 = \mathbf{z}_i(t_{i-1})$, $\mathbf{z}_f = \mathbf{z}_j(t_j)$, and a time $T_{ij} = t_j - t_{i-1}$ or an optimal T_{ij} from (26).
4 Calculate $\mathbf{w}_{ij}(t)$ from Eq. (23).
 /* Bypass unnecessary motions if the trajectory $\mathbf{z}_{ij}(t)$ is valid */
5 **if not** FLASKCC($\mathbf{z}_{ij}(t), \mathbf{w}_{ij}(t)$) **then**
6 Replace $\{(\mathbf{z}_i(t), t_i)\}_{k=i}^{j}$ by $(\mathbf{z}_{ij}(t), t_{i-1} + T_{ij})$.
7 Replace $\{(\mathbf{w}_i(t), t_i)\}_{k=i}^{j}$ by $(\mathbf{w}_{ij}(t), t_{i-1} + T_{ij})$.

E. Trajectory Postprocessing

Our kinodynamic planning approach in Sec. V-B returns a piecewise-polynomial trajectory: $\boldsymbol{\sigma}_{\mathbf{z}}(t) = \{(\mathbf{z}_i(t), t_i)\}_{i=1}^{M}$, for $0 = t_0 < t_1 < \ldots < t_M$ where the polynomial $\mathbf{z}_i(t), t \in [t_{i-1}, t_i]$ corresponds to the pseudo-control $\mathbf{w}_i(t)$ and time duration $T_i = t_i - t_{i-1}$. Due to the sampling-based nature of our approach, the trajectory might contain unnecessary local paths and often requires further postprocessing, *e.g.*, by seeking shortcuts between nodes. For completeness, a simple trajectory postprocessing scheme is provided in Alg. 4 such that if there exists a collision-free trajectory $\mathbf{z}_{ij}(t)$ that connects two nodes $\mathbf{z}_i(t_{i-1})$ and $\mathbf{z}_j(t_j)$, the local paths $\mathbf{z}_k$, $i \leq k \leq j$ in between can simply be bypassed by $\mathbf{z}_{ij}(t)$. We note that our approach is general and therefore, compatible with other complex trajectory shortening schemes such as [98, 99].

VI. PROBABILISTIC EXHAUSTIVITY AND OPTIMALITY ANALYSIS

In this section, we will examine the *probabilistic exhaustivity* of our **FLASK** framework (Sec. VI-A), which is a key concept for proof of optimality common among asymptotically optimal planners [100–102]. As our approach is general and compatible with most sampling-based motion planners, we provide an outline of the *optimality analysis* in Sec. VI-B based on the proven probabilistic exhaustivity property.

A. Probabilistic Exhaustivity

The probabilistic exhaustivity property, introduced in [39, 40], is stronger than probabilistic completeness [11]. While probabilistic completeness only requires *one* solution to be found, *probabilistic exhaustivity* requires that *any* trajectory $\boldsymbol{\pi}$, with a δ-clearance to the occupied regions of the state space (see Def. 2), can be approximated arbitrarily well with high probability by a piecewise trajectory $\boldsymbol{\sigma}$, composed of BVP solutions $\boldsymbol{\sigma}(t) = \{(\mathbf{x}_i(t), t_i)\}_{i=1}^{M}$ connecting $M+1$ nodes $\{\bar{\mathbf{x}}_i\}_{i=0}^{M}$ on the graph, as the number of nodes tends to infinity. Our analysis relies on the following assumptions and definitions.

Assumption 1. The pseudo-control input $\mathbf{w} = \mathbf{y}^{(r)}$ in Eq. (14) has r strictly larger than l (defined in Eq. (7)), *i.e.*, the original state $\mathbf{x}$ can be recovered using Eq. (7) solely from the flat state $\mathbf{z}$: $\mathbf{x} = \boldsymbol{\alpha}(\mathbf{z})$. This is the common case as seen in Examples 1, 2, and 3.

Definition 2 (δ-clearance trajectory [39, 100])**.** A trajectory $\boldsymbol{\pi}$ with duration $T_{\boldsymbol{\pi}}$ in the state space $\mathcal{X}$ is called "δ-clearance" if the state $\mathbf{x}(t)$ stays in the δ-interior of the free space $\mathcal{X}_{free}$:

$$\mathcal{X}_{free}^{\delta} = \{\mathbf{x} \in \mathcal{X}_{free} : \min_{\mathbf{y} \in \mathcal{X}_{obs}} \|\mathbf{x} - \mathbf{y}\| \geq \delta\} \text{ for a } \delta > 0.$$

Definition 3 ($(\varepsilon, \zeta, \eta)$-close piecewise trace [103])**.** Given a trajectory $\boldsymbol{\pi}(t)$ with control $\mathbf{u}_{\boldsymbol{\pi}}(t)$ and duration $T_{\boldsymbol{\pi}}$, a piecewise trajectory $\boldsymbol{\sigma}(t) = \{(\mathbf{x}_i(t), t_i)\}_{i=1}^{M}$ with control $\mathbf{u}_{\boldsymbol{\sigma}}(t) = \{(\mathbf{u}_i(t), t_i)\}_{i=1}^{M}$ and duration $T_{\boldsymbol{\sigma}}$, connecting $M+1$ points $\{\bar{\mathbf{x}}_i\}_{i=0}^{M}$, is called an "$(\varepsilon, \zeta, \eta)$-close piecewise trace" of $\boldsymbol{\pi}$ if:

(i) The cost of $\boldsymbol{\sigma}$ is ε-close to that of $\boldsymbol{\pi}$:

$$\mathcal{C}(\boldsymbol{\sigma}, \mathbf{u}_{\boldsymbol{\sigma}}, T_{\boldsymbol{\sigma}}) \leq (1 + \varepsilon)\mathcal{C}(\boldsymbol{\pi}, \mathbf{u}_{\boldsymbol{\pi}}, T_{\boldsymbol{\pi}}).$$

(ii) The cost of the i-th segment is bounded by ζ:

$$\mathcal{C}(\mathbf{x}_i(t), \mathbf{u}_i(t), T_i) \leq \zeta, \quad \forall i \in \{1, \ldots, M\}$$

with $T_i = t_i - t_{i-1}$.

(iii) The maximum distance from a point on $\boldsymbol{\sigma}$ to $\boldsymbol{\pi}(t)$ is bounded by η:

$$\max_{t^{\boldsymbol{\sigma}} \in [0, T_{\boldsymbol{\sigma}}]} \min_{t^{\boldsymbol{\pi}} \in [0, T_{\boldsymbol{\pi}}]} \|\boldsymbol{\sigma}(t^{\boldsymbol{\sigma}}) - \boldsymbol{\pi}(t^{\boldsymbol{\pi}})\| \leq \eta. \tag{31}$$

Definition 4 (Occupied region $\mathcal{Z}_{obs}$)**.** Similar to the occupied region $\mathcal{X}_{obs}$ in the original state space, we define an occupied region $\mathcal{Z}_{obs}$ in the flat state space. If there are only state constraints in the original state space, i.e., collision avoidance described by $\mathcal{X}_{obs}$, we can define $\mathcal{Z}_{obs}$ as: $\mathcal{Z}_{obs} = \{\mathbf{z} \in \mathcal{Z} : \boldsymbol{\alpha}(\mathbf{z}) \in \mathcal{X}_{obs}\}$.

In many cases, there are additional constraints such as limits on the original control input $\mathbf{u}$ or on a higher-order derivative $\mathbf{x}^{(p)}$, $p \geq 1$. Such constraints, often described as $\boldsymbol{\gamma}(\mathbf{u}) \leq \mathbf{0}$ or $\boldsymbol{\gamma}(\mathbf{x}^{(p)}) \leq \mathbf{0}$ for some function $\boldsymbol{\gamma}$, can also be represented by $\mathcal{Z}_{obs}$ as follows. Due to Eq. (7), we have $\mathbf{u} = \boldsymbol{\beta}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(m)})$ and $\mathbf{x} = \boldsymbol{\alpha}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(l)})$, i.e., the pth-order derivative $\mathbf{x}^{(p)}$ is a function of $\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(l+p)}$. Therefore, additional constraints on $\mathbf{u}$ or $\mathbf{x}^{(p)}$ can be described in terms of $\{\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{\max\{m,(l+p)\}}\}$ as:

$$\boldsymbol{\gamma}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{\max\{m,(l+p)\}}) \leq \mathbf{0}. \tag{32}$$

Since we have the freedom to choose the order r of the pseudo-control input $\mathbf{w} = \mathbf{y}^{(r)}$ in Eq. (14), we can choose $r \geq (1 + \max\{m, (l+p)\})$ so that the constraints in Eq. (32) only depend on $\mathbf{z}$: $\boldsymbol{\gamma}(\mathbf{z}) \leq \mathbf{0}$. Consequently, this allows us to capture the additional constraints completely by $\mathcal{Z}_{obs}$:

$$\mathcal{Z}_{obs} = \{\mathbf{z} \in \mathcal{Z} : \boldsymbol{\alpha}(\mathbf{z}) \in \mathcal{X}_{obs}, \boldsymbol{\gamma}(\mathbf{z}) \leq \mathbf{0}\},$$

which is implemented as part of the collision checking routine (line 6) in Alg. 3. A collision-free trajectory $\boldsymbol{\sigma}_{\mathbf{z}}$ in the flat state space leads to a trajectory $\boldsymbol{\sigma}$ that satisfies the original state

and control constraints. Meanwhile, the pseudo-control input $\mathbf{w} = \mathbf{y}^{(r)}$ remains unconstrained in our theoretical analysis.

Theorem 1 (Probabilistic Exhaustivity)**.** *Consider our sampling-based kinodynamic planning framework,* ***FLASK****, in Alg. 1 where the* FLASKEXTEND *subroutine in Alg. 2 uses the BVP solution with optimal time T^* (lines 1-4) and the cost function in Eq.* (20)*. Under Assumption 1, let $\mathcal{Z}_{free} = \mathcal{Z} \setminus \mathcal{Z}_{obs}$ denote the free flat state space, where the invalid region $\mathcal{Z}_{obs}$ is defined in Def. 4. Let $\boldsymbol{\pi}_\mathbf{z}$ with pseudo-control $\mathbf{w}_{\boldsymbol{\pi}}$ and duration $T_{\boldsymbol{\pi}}$ be a δ-clearance trajectory for a $\delta > 0$ with respect to $\mathcal{Z}_{free}$. Let $N = |\mathbb{V}_\mathbf{z}|$ denote the number of nodes/samples in our planning graph $\mathbb{G}_\mathbf{z} = (\mathbb{V}_\mathbf{z}, \mathbb{E}_\mathbf{z})$. Given a constant C_μ, define $C_{\mathcal{Z}_{free}} = C_\mu^{-1} D^{-1} 6^{rn+r^2n/2} 2^{rn/2} \mu(\mathcal{Z}_{free})$ with the volume of the free space $\mu(\mathcal{Z}_{free})$, a constant $D = (rn + r^2 n)/2$ and a user-defined parameter $\kappa \geq 0$.*

(i) In the flat state space, let $\mathcal{A}_N$ define the event that there exists an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace of $\boldsymbol{\pi}_\mathbf{z}$, denoted as $\boldsymbol{\sigma}_\mathbf{z}(t) = \{(\mathbf{z}_i(t), t_i)\}_{i=1}^M$ with:

$$\zeta_N = (1+\kappa)^{(1/D)} C_{\mathcal{Z}_{free}} \left(\frac{\log N}{N}\right)^{1/D}, \quad \eta_N = C_p \zeta_N, \tag{33}$$

for some constant M and C_p. There exists a constant C_μ such that as $N \to \infty$, the probability that $\mathcal{A}_N$ does not occur is bounded as:

$$1 - P(\mathcal{A}_N) = O\left(N^{-\kappa/D} \log^{-1/D} N\right). \tag{34}$$

(ii) Under Assumption 1, let $\boldsymbol{\pi} = \boldsymbol{\alpha}(\boldsymbol{\pi}_\mathbf{z})$ be the corresponding trajectory of $\boldsymbol{\pi}_\mathbf{z}$ in the original state space $\mathcal{X}$. If the conversion $\boldsymbol{\alpha} : \mathcal{Z} \to \mathcal{X}$ is Lipschitz-continuous on $\mathcal{Z}$ with a Lipschitz constant $L_{\boldsymbol{\alpha}}$, we define an event $\mathcal{B}_N$ that there exists a $(\varepsilon, \zeta_N, L_{\boldsymbol{\alpha}}\eta_N)$-close piecewise trace of $\boldsymbol{\pi}$ where the bounds ζ_N η_N are calculated in (33)*. Then, the probability that $\mathcal{B}_N$ does not occur is bounded as:*

$$1 - P(\mathcal{B}_N) = O\left(N^{-\kappa/D} \log^{-1/D} N\right). \tag{35}$$

Proof. Our proof follows a common intuition in *geometric* motion planning [40, 100], that as the number of samples increases, there exists a sequence of spheres along the trajectory $\boldsymbol{\pi}_\mathbf{z}$ and contains a piecewise-linear path arbitrarily ε-close to $\boldsymbol{\pi}_\mathbf{z}$ with high probability. However, this is not trivial in our case due to the dynamics constraints. Unlike a linear edge that can be bounded easily by a sphere, the nonlinear local path $\mathbf{z}_{loc}$ is bounded by an ellipsoid in our proof instead, as shown below.

(i) **Probabilistic exhaustivity in the flat state space**: Consider our linear system (16) in the flat state space $\mathcal{Z} \subset \mathbb{R}^{rn}$ with matrices $\mathbf{A} \in \mathbb{R}^{rn \times rn}$, $\mathbf{B} \in \mathbb{R}^{rn \times n}$, and unconstrained pseudo-control input $\mathbf{w}$. Consider the controllability matrix:

$$\begin{aligned}\mathbf{C}(\mathbf{A}, \mathbf{B}) &= \begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{rn}\mathbf{B},\end{bmatrix}, \\ &= \begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_n & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_n & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{I}_n & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{I}_n & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}.\end{aligned} \tag{36}$$

Clearly, we have $\text{rank}(\mathbf{C}(\mathbf{A}, \mathbf{B})) = rn$ with rn linearly independent column vectors: $\left\{\{\mathbf{b}_j, \mathbf{A}\mathbf{b}_j, \ldots, \mathbf{A}^{r-1}\mathbf{b}_j\}_{j=1}^n\right\}$,

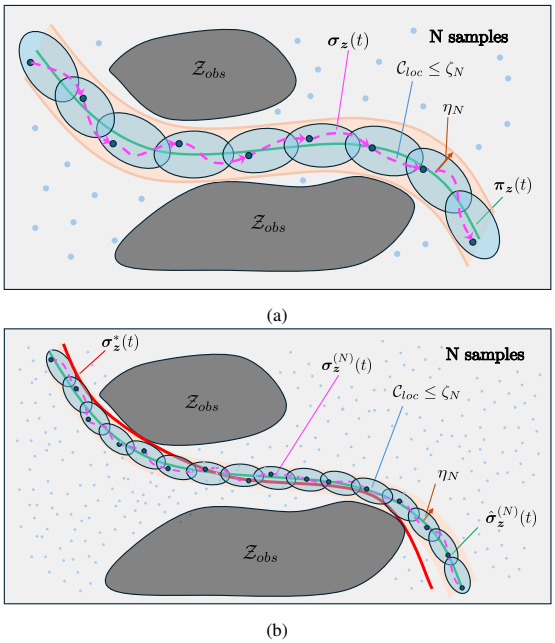


Fig. 4: Our theoretical analysis: (a) Probabilistic exhaustivity: as $N \to \infty$, the probability that there exists an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}_\mathbf{z}(t)$ (magenta), on our planning graph $\mathbb{G}$, of any trajectory $\boldsymbol{\pi}_\mathbf{z}$ (green) with δ-clearance approaches 1; (b) Optimality: our approach will find an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}_\mathbf{z}^{(N)}(t)$ (magenta) of a δ_N-clearance trajectory $\hat{\boldsymbol{\sigma}}_\mathbf{z}^{(N)}(t)$ (green). As $\delta_N \to 0$, the trajectory $\hat{\boldsymbol{\sigma}}_\mathbf{z}^{(N)}(t)$ converges to the optimal trajectory $\boldsymbol{\sigma}_\mathbf{z}^*(t)$ (red). Since $\zeta_N \to 0$, $\eta_N \to 0$ and $\delta_N \to 0$ as $N \to \infty$, our piecewise trajectory $\boldsymbol{\sigma}_\mathbf{z}^{(N)}(t)$ converges to $\boldsymbol{\sigma}_\mathbf{z}^*(t)$ in probability.

where $\mathbf{b}_j$ denotes the j-th column of the matrix $\mathbf{B}$ in (16). Therefore, our linear system (16) is controllable with the following controllability indices:

$$\nu_1 = \nu_2 = \ldots = \nu_n = r. \tag{37}$$

The controllability indices, as shown later, allow us to bound the volume of an ellipsoid containing our local path $\mathbf{z}_{loc}(t)$.

As the number of samples N goes to infinity, the duration T of a local path $\mathbf{z}_{loc}$ approaches 0 with high probability due to the higher sample density. Therefore, we are interested in how the cost $\mathcal{C}_{loc}(T)$ in Eq. (24) behaves near $T = 0$. As the Gramian matrix $\mathbf{G}_T \succ 0$, we consider an ellipsoid $\mathcal{E}_\psi(\mathbf{z}, T)$ around a point $\mathbf{z}$, defined as

$$\mathcal{E}_\psi(\mathbf{z}, T) = \{\mathbf{z} : (\mathbf{z}' - \mathbf{z})^\top \mathbf{G}_T^{-1} (\mathbf{z}' - \mathbf{z}) \leq \psi^2\},$$

which can be used to bound the quadratic component $\mathbf{d}_T^\top \mathbf{G}_T^{-1} \mathbf{d}_T$ of $\mathcal{C}_{loc}(T)$. The volume of this ellipsoid is $\mu(\mathcal{E}_\psi(\mathbf{z}, T)) = \psi^{rn} \mu(\mathcal{S}_{rn}) \sqrt{\det(\mathbf{G}_T)}$, where $\mu(\mathcal{S}_{rn})$ is the volume of a unit ball $\mathcal{S}_{rn}$ in $\mathbb{R}^{rn}$. As $T \to 0$, the value of $\det(\mathbf{G}_T)$ determines how fast the volume of $\mathcal{E}_\psi(\mathbf{z}, T)$ goes to 0. Due to the controllability indices in Eq. (37), the determinant of the Gramian matrix $\mathbf{G}_T$ (22) is $\Theta(T^{\sum_{j=1}^n \nu_j^2}) = \Theta(T^{r^2 n})$ as $T \to 0$ according to Lemma III.4 in [103], *i.e.*, there exist constants $T_0, C_1, C_2 > 0$ such that

$$C_1 T^{r^2 n} \leq \det(\mathbf{G}_T) \leq C_2 T^{r^2 n}, \quad \forall T < T_0.$$

As a result, we have:

$$\mu(\mathcal{E}_\psi(\mathbf{z}, T)) \geq C_1 \mu(\mathcal{S}_{rn}) \psi^{rn} T^{r^2 n/2}, \quad \forall T < T_0. \tag{38}$$

Denote $C_\mu = \sqrt{C_1}\mu(\mathcal{S}_{rn})$. Eq. (38) allows us to lower bound the rate of the quadratic cost component of the optimal cost in Eq. (24) as $T \to 0$. The lower bound (38) specifies the minimum coverage of the state space by the ellipsoid $\mathcal{E}_\psi(\mathbf{z}, T)$.

As our linear system (16) is controllable with an unconstrained pseudo-control input $\mathbf{w}$ (see Def. 4) and the cost weight matrix $\mathbf{R}$ in Eq. (20) is symmetric positive definite, we now can apply Theorem IV.6 in [103] with the constant C_μ calculated above to construct a sequence of ellipsoids that generates an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}_\mathbf{z}$. While we refer the readers to [103] for a detailed proof, we provide a sketch of the derivation for completeness as follows.

Consider a sequence of fixed points on the trajectory $\boldsymbol{\pi}_\mathbf{z}$: $\{\mathbf{z}_i^{\boldsymbol{\pi}}\}_{i=0}^M$ with $\mathbf{z}_i^{\boldsymbol{\pi}} = \boldsymbol{\pi}_\mathbf{z}(t_i^{\boldsymbol{\pi}})$, such that the cost of each segment $i \in \{1, \ldots, M\}$, from $\mathbf{z}_{i-1}^{\boldsymbol{\pi}}$ to $\mathbf{z}_i^{\boldsymbol{\pi}}$, is exactly $\zeta_N/2$. The number of segments M is finite and bounded as $M \leq \lceil 2\mathcal{C}(\boldsymbol{\pi}_\mathbf{z}, \mathbf{w}_{\boldsymbol{\pi}}, T_{\boldsymbol{\pi}})/\zeta_N \rceil$. Following [103], we construct two sequences of ellipsoids centered around $\{\mathbf{z}_i^{\boldsymbol{\pi}}\}_{i=0}^M$: $\mathcal{T} = \{\mathcal{T}_i\}_{i=0}^M$ with $\mathcal{T}_i = \mathcal{E}_{\frac{1}{6}\sqrt{\zeta_N/2}}(\mathbf{z}_i^{\boldsymbol{\pi}}, \zeta_N/6)$ and $\mathcal{S} = \{\mathcal{S}_i\}_{i=0}^M$ with $\mathcal{S}_i = \mathcal{E}_{\frac{1}{12}\varepsilon\sqrt{\zeta_N/2}}(\mathbf{z}_i^{\boldsymbol{\pi}}, \zeta_N/6)$. Let $\mathcal{A}'_N$ be the event that every ellipsoid in $\mathcal{T}$ and at least $(1 - \frac{\varepsilon}{4})$ fraction of the ellipsoids in $\mathcal{S}$ contains a sample in $\mathbb{V}_\mathbf{z}$. Using random geometric graph theory [104], Schmerling et al. [103] shows that $\mathcal{A}'_N$ occurs with high probability as $N \to \infty$:

$$1 - P(\mathcal{A}'_N) = O\left(N^{-\kappa/D} \log^{-1/D} N\right). \tag{39}$$

Given that $\mathcal{A}'_N$ occurs, we can choose a sequence of points $\{\bar{\mathbf{z}}_i\}_{i=0}^M$ from $\mathbb{V}_\mathbf{z}$ by picking $\bar{\mathbf{z}}_i$ as a sample in the ellipsoid $\mathcal{S}_i$ if possible, and in the ellipsoid $\mathcal{T}_i$ otherwise. In other words, we compose a new sequence of ellipsoids, whose $(1-\frac{\varepsilon}{4})$ fraction is from $\mathcal{S}$ and $\frac{\varepsilon}{4}$ fraction is from $\mathcal{T}$. By connecting $\{\bar{\mathbf{z}}_i\}_{i=0}^M$ via the BVP solutions in Eq. (25), we obtain a trajectory $\boldsymbol{\sigma}_\mathbf{z}(t) = \{(\mathbf{z}_i(t), t_i)\}_{i=1}^M$. Intuitively, the ellipsoids $\mathcal{S}_i$'s have a smaller $O(\varepsilon)$ extent by definition (*i.e.*, closer to $\boldsymbol{\pi}_\mathbf{z}$ with small enough ε) and cover $1 - O(\varepsilon)$ fraction of the segments $\mathbf{z}_i(t)$'s. Meanwhile, the ellipsoids $\mathcal{T}_i$'s have a larger $O(1)$ extent in terms of ε (*i.e.*, allowing farther distance from $\boldsymbol{\pi}_\mathbf{z}$) but only include an $O(\varepsilon)$ fraction of the segments $\mathbf{z}_i(t)$'s. This is the main intuition to show that the constructed $\boldsymbol{\sigma}_\mathbf{z}(t)$ is an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace of $\boldsymbol{\pi}_\mathbf{z}$ by Theorem IV.6 in [103]. Therefore, the event $\mathcal{A}'_N$ implies the event $\mathcal{A}_N$ and as $N \to \infty$, the probability that $\mathcal{A}_N$ *does not* occur is bounded by:

$$P(\bar{\mathcal{A}}_N) = 1 - P(\mathcal{A}_N) = O\left(N^{-\kappa/D} \log^{-1/D} N\right).$$

(ii) **Probabilistic exhaustivity in the original state space**: As the conversion $\mathbf{x} = \boldsymbol{\alpha}(\mathbf{z})$ is Lipschitz-continuous on $\mathcal{Z}$ with a Lipschitz constant $L_{\boldsymbol{\alpha}}$, we have:

$$\|\mathbf{x}_1 - \mathbf{x}_2\| = \|\boldsymbol{\alpha}(\mathbf{z}_1) - \boldsymbol{\alpha}(\mathbf{z}_2)\| \leq L_{\boldsymbol{\alpha}} \|\mathbf{z}_1 - \mathbf{z}_2\|, \forall \mathbf{z}_1, \mathbf{z}_2 \in \mathcal{Z}$$

Let $\boldsymbol{\sigma} = \boldsymbol{\alpha}(\boldsymbol{\sigma}_\mathbf{z})$ be the corresponding trajectory in the original state space of the $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}_\mathbf{z}$. Given a time $t_{\boldsymbol{\sigma}} \in [0, T_{\boldsymbol{\sigma}}]$, define $t^*_{\boldsymbol{\pi}} = \operatorname{argmin}_{t_{\boldsymbol{\pi}} \in [0, T_{\boldsymbol{\pi}}]} \|\boldsymbol{\sigma}_\mathbf{z}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}_\mathbf{z}(t_{\boldsymbol{\pi}})\|$. Then, we have:

$$\|\boldsymbol{\sigma}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}(t^*_{\boldsymbol{\pi}})\| \leq L_{\boldsymbol{\alpha}} \|\boldsymbol{\sigma}_\mathbf{z}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}_\mathbf{z}(t^*_{\boldsymbol{\pi}})\| \leq L_{\boldsymbol{\alpha}} \eta_N,$$

since $\boldsymbol{\sigma}_\mathbf{z}$ is a $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace. Let $\tau^*_{\boldsymbol{\pi}} = \operatorname{argmin}_{\tau_{\boldsymbol{\pi}} \in [0, T_{\boldsymbol{\pi}}]} \|\boldsymbol{\sigma}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}(\tau_{\boldsymbol{\pi}})\|$. Clearly, we have the following bound that holds *for all* $t_{\boldsymbol{\sigma}} \in [0, T_{\boldsymbol{\sigma}}]$:

$$\|\boldsymbol{\sigma}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}(\tau^*_{\boldsymbol{\pi}})\| \leq \|\boldsymbol{\sigma}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}(t^*_{\boldsymbol{\pi}})\| \leq L_{\boldsymbol{\alpha}} \eta_N.$$

By maximizing over $t_{\boldsymbol{\sigma}} \in [0, T_{\boldsymbol{\sigma}}]$, the maximum distance between a point on $\boldsymbol{\sigma}$ to $\boldsymbol{\pi}$ in the state space $\mathcal{X}$ satisfies:

$$\max_{t_{\boldsymbol{\sigma}} \in [0, T_{\boldsymbol{\sigma}}]} \min_{t_{\boldsymbol{\pi}} \in [0, T_{\boldsymbol{\pi}}]} \|\boldsymbol{\sigma}(t_{\boldsymbol{\sigma}}) - \boldsymbol{\pi}(t_{\boldsymbol{\pi}})\| \leq L_{\boldsymbol{\alpha}} \eta_N.$$

As the cost function (20) is defined in the flat state space, the trajectory $\boldsymbol{\sigma}$ has the same cost as $\boldsymbol{\sigma}_\mathbf{z}$. Therefore, an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}_\mathbf{z}$ implies an $(\varepsilon, \zeta_N, L_{\boldsymbol{\alpha}}\eta_N)$-close piecewise trace $\boldsymbol{\sigma}$ or in other words, the event $\mathcal{A}_N$ implies the event $\mathcal{B}_N$, *i.e.*, $P(\mathcal{A}_N) \leq P(\mathcal{B}_N)$. Therefore, we have:

$$1 - P(\mathcal{B}_N) \leq 1 - P(\mathcal{A}_N) = O\left(N^{-\kappa/D} \log^{-1/D} N\right). \quad \square$$

Theorem 1 shows that given any trajectory $\boldsymbol{\pi}$ with δ clearance, as the number of samples $N \to \infty$, there exists, with high probability, an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}$ of the trajectory $\boldsymbol{\pi}$ (illustrated in Fig. 4a). More importantly, the bounds ζ_N and η_N are $O\left(\left(\frac{\log N}{N}\right)^{1/D}\right)$, which will be shown later to be critical for our asymptotic optimality analysis.

Theorem 1 assumes that we can completely check if a trajectory is valid or has δ-clearance, e.g., via continuous-time collision checking. However, the standard practice in sampling-based motion planning, *e.g.*, [11, 41], is to perform this check approximately via sampling for efficiency, as shown in Alg. 3, with the accuracy up to the sampling resolution. While integrating continuous-time collision checking can avoid caveats of sampling-based collision checking, *e.g.*, tunneling effects with thin obstacles, we leave this direction for future work.

B. Optimality Analysis

As our **FLASK** framework in Sec. V (Alg. 1) is compatible with any sampling-based motion planners [53], the optimality guarantees of our trajectory depend on the optimality guarantees of the specific planner that integrates our FLASKEXTEND subroutine. However, we provide an outline of an optimality analysis based on random geometric graphs [104] which many asymptotically optimal sampling-based motion planners follow (see [23, 101, 102] for an example and [100] for an excellent review of this topic). Therefore, our framework not only transforms any sampling-based motion planner into its kinodynamic version but also preserves its optimality guarantees.

We define an optimal trajectory solution of Problem 2 with respect to the cost (20) in Def. 5. We note that while our cost function Eq. (20) is defined in the flat state space, it bears physical meaning in the original space as the flat output $\mathbf{y}$ often describes certain physical properties of the robot, *e.g.*, the position and yaw angle of a quadrotor in Example 3. For example, if the pseudo-control input $\mathbf{w} = \mathbf{y}^{(r)}$ in (14) has $r = 2, 3, 4, \ldots$, the cost (20) means that we would like to find a minimum acceleration-time, jerk-time, or snap-time trajectory, respectively (*e.g.*, in [21, 35, 93]).

Definition 5 (Optimal Trajectory)**.** Denote $\mathcal{C}_{\boldsymbol{\sigma}_\mathbf{z}} = \mathcal{C}(\boldsymbol{\sigma}_\mathbf{z}(t), \mathbf{w}(t), T)$. Assume that there exists a minimum cost $\mathcal{C}^* = \min_{\boldsymbol{\sigma}_\mathbf{z}} \mathcal{C}_{\boldsymbol{\sigma}_\mathbf{z}}$, defined in Eq. (20), over all trajectories $\boldsymbol{\sigma}_\mathbf{z}(t)$ that solve Problem 2. A trajectory $\boldsymbol{\sigma}^*_\mathbf{z}(t)$ with control $\mathbf{w}^*(t)$ and its corresponding trajectory $\boldsymbol{\sigma}^*(t) = \boldsymbol{\alpha}(\boldsymbol{\sigma}^*_\mathbf{z}(t), \mathbf{w}^*(t))$ with control $\mathbf{u}^*(t) = \boldsymbol{\beta}(\boldsymbol{\sigma}^*_\mathbf{z}(t), \mathbf{w}^*(t))$ in the original state space are called optimal if they achieve the minimum cost $\mathcal{C}^*$.

In the flat space, the optimal trajectory $\boldsymbol{\sigma}^*_\mathbf{z}(t)$, defined in Def. 5, might have 0 clearance as illustrated in Fig. 4b, *i.e.*, it might touch the boundary of $\mathcal{Z}_{obs}$. However, we assume that there exists a sequence of δ_N-clearance trajectories $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$ with control $\hat{\mathbf{w}}^{(N)}$ and duration $T^{(N)}$ such that its cost $\mathcal{C}_{\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}} = \mathcal{C}(\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t), \hat{\mathbf{w}}^{(N)}(t), T^{(N)})$ satisfies: $\lim_{N\to\infty} \mathcal{C}_{\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}} = \mathcal{C}^*$, where $N = |\mathbb{V}_\mathbf{z}|$ denotes the number of nodes/samples in our planning graph $\mathbb{G}_\mathbf{z} = (\mathbb{V}_\mathbf{z}, \mathbb{E}_\mathbf{z})$. This is a common assumption in sampling-based motion planning as the δ_N clearance allows a region around $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$ with nonzero measure for sampling.

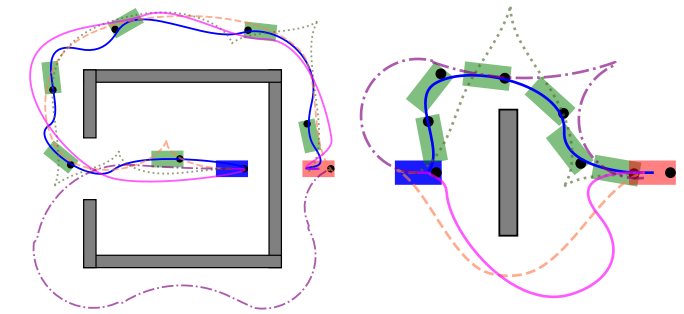
For each δ_N-clearance trajectory $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$, Theorem 1(i) shows that as $N \to \infty$, the probability that there is an $(\varepsilon, \zeta_N, \eta_N)$-close piecewise trace $\boldsymbol{\sigma}^{(N)}_\mathbf{z}(t)$ of $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$ goes to 1. This allows us to cover $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$ by a sequence of ellipsoids of extent ζ_N so that the piecewise trace $\boldsymbol{\sigma}^{(N)}_\mathbf{z}(t)$ will stay inside, as illustrated in Fig. 4b. Meanwhile, to maintain connectivity around $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$, the bound ζ_N can be used as thresholds for expanding (*e.g.*, for picking $\mathbf{z}_0$ in FLASKEXTEND in Alg. 2) and rewiring the planning graph $\mathbb{G}_\mathbf{z}$ to find a better cost for each neighboring node, similar to RRT* [101] or for recursive cost updates via dynamic programming as in FMT* [102, 103].

More importantly, as $N \to \infty$, the upper bounds ζ_N, η_N in (33), proportional to $\left(\frac{\log N}{N}\right)^{1/D}$, go to 0. Therefore, the probability that the piecewise trace $\boldsymbol{\sigma}^{(N)}_\mathbf{z}(t)$ converges to $\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}(t)$ with cost $\mathcal{C}_{\boldsymbol{\sigma}^{(N)}_\mathbf{z}} \leq (1+\varepsilon)\mathcal{C}_{\hat{\boldsymbol{\sigma}}^{(N)}_\mathbf{z}}$ also approaches 1. Furthermore, by choosing a decreasing clearance δ_N so that $\delta_N \to 0$ as $N \to \infty$, *e.g.*, $\delta_N = O(\eta_N)$, we have $P\left(\mathcal{C}_{\boldsymbol{\sigma}^{(N)}_\mathbf{z}} \leq (1+\varepsilon)\mathcal{C}^*\right) \to 1$, for an arbitrarily small $\varepsilon > 0$, *i.e.*, our **FLASK** framework maintains asymptotic optimality. This is illustrated in Fig. 4b with denser samples and smaller clearance δ_N than those of Fig. 4a, making our piecewise trajectory $\boldsymbol{\sigma}^{(N)}_\mathbf{z}$ closer to the optimal one as N increases.

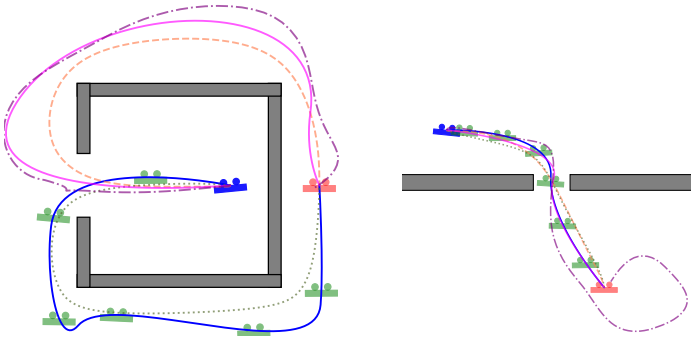
Finally, the asymptotic optimality guarantees can be transformed to the original state space following similar steps due to Theorem 1(ii). This analysis importantly affirms that our **FLASK** framework can be transform any existing sampling-based motion planners into kinodynamic planners without breaking their asymptotic optimality guarantees.

VII. EVALUATION

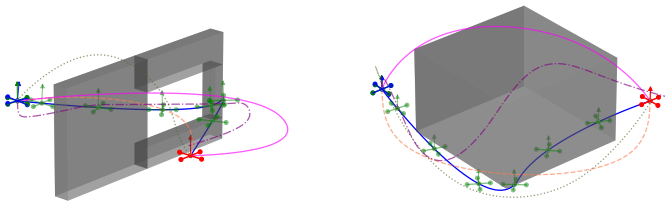
In this section, we will evaluate the effectiveness of our **FLASK** framework, applied on different planners with different robot platforms, from low to high dimensional state spaces in both simulation and real experiments. As our method focuses on fast kinodynamic planning on CPUs, the results from our planners and the baselines are reported on an Intel i7-6900K CPU for a fair comparison. As a reference, we also provide the results from the GPU-based Kino-PAX [84] planner, which is run on a GeForce RTX 4070 GPU.



(a) Unicycle: *Bugtrap* (left) & *Wall* (right).



(b) 2D quadrotor: *Bugtrap* (left) & *Hole* (right).



(c) 3D quadrotor: *Window* (left) & *Obstacle* (right).

Fig. 5: Visualization of our trajectories with DynoBench [17] benchmarking problems: the trajectories from our kinodynamic `FLASK-RRTConnect` and our BVP-augmented `FLASK-SST*` are plotted as blue and magenta solid curves, respectively, while those from the iDb-A* and SST* baselines are shown as orange dashed and green dotted curves, respectively, and the Kino-PAX trajectories are plotted as purple dash-dotted curves.

A. Kinodynamic planning for low-DoF systems

In this section, we compare our approach with other kinodynamic planning algorithms provided by the *DynoBench* benchmark [17], which contains multiple low-DoF robots and benchmarking environments. We consider 3 robot platforms: a *unicycle* (Example 2), a 2D *quadrotor*, and a 3D *quadrotor* (Example 3). The *unicycle* robot has the following constraints: maximum linear velocity $v \leq v_{max} = 1.0\,\mathrm{m/s}$, maximum angular velocity $\omega \leq 1.5\,\mathrm{rad/s}$. The $3D$ *quadrotor* has the following parameters: mass $m = 1\,\mathrm{kg}$, inertia matrix $\mathbf{J} = \mathrm{diag}([0.1, 0.1, 0.2])\mathrm{kg\,m^2}$, maximum linear velocity $\|\mathbf{v}\| \leq 4\,\mathrm{m/s}$, maximum angular velocity $\|\boldsymbol{\omega}\| \leq 8\,\mathrm{rad/s}$, maximum thrust $f = 1.5mg\,\mathrm{N}$, and maximum torque $\boldsymbol{\tau} \leq 2\,\mathrm{N\,m}$, where $g \approx 9.81\,\mathrm{m/s^2}$ is the gravitational acceleration. The 2D *quadrotor*'s dynamics is a special case of the 3D *quadrotor*'s with position $\mathbf{p} = [0, y, z]$, yaw angle $\psi = 0$, and the rotation matrix $\mathbf{R} = \mathbf{R}_\phi$ of a pitch angle ϕ, leading to a simplified flat output $\mathbf{y} = [y, z] \in \mathbb{R}^2$. The 2D *quadrotor* is modeled after a Crazyflie [105] with mass $m = 0.034\,\mathrm{kg}$, inertia

TABLE I: Planning performance with DynoBench benchmarks [17]. †For the anytime planners iDb-A* [17], SST* [26], the *first solution time* and *final trajectory length* within a 180-second limit are reported.

Robot	Problem	FLASK-RRTConnect		FLASK-SST* (BVP-aug.)		FLASK-SST* (SIMD-only)		FLASK-RRT (DP-based)		iDb-A*† (DynoBench)		Orig. SST*† (DynoBench)		Kino-PAX (GPU-based)	
		Traj. Len. (m)	Plan. Time (ms)	Traj. Len. (m)	Plan. Time (ms)	Traj. Len. (m)	Plan. Time (ms)	Traj. Len. (m)	Plan. Time (ms)	Traj. Len. (m)	Plan. Time (ms)	Traj. Len. (m)	Plan. Time (ms)	Traj. Len. (m)	Plan. Time (ms)
Uni.	Bugtrap	12.39	**6.0**	12.51	48.6	12.27	407	12.25	565	**11.51**	375	12.38	126	13.77	5.8
	Wall	4.82	**0.6**	5.24	11.0	5.27	30	5.61	25	**3.89**	77	4.97	67	5.28	2.6
Quad. (2D)	Bugtrap	11.46	**2.7**	11.67	45.0	11.46	388	11.50	909	9.69	10746	**9.64**	28667	18.59	19.7
	Hole	5.16	**0.4**	4.93	270.0	5.14	1813	5.24	2133	4.62	353	**4.59**	17602	17.82	124.9
Quad. (3D)	Block	8.08	**0.9**	8.61	1.4	9.56	4752	8.75	7602	**7.94**	4573	9.32	46289	10.36	21.4
	Window	5.49	**0.5**	6.28	22.6	6.96	629	7.98	2631	**5.12**	2667	6.72	45798	16.41	178.6

$J = 1 \times 10^{-4}\,\mathrm{kg\,m^2}$, and arm length $\ell = 0.1\,\mathrm{m}$. The thrusts f_1 and f_2, generated from the two onboard motors, satisfy the following constraint: $0 \leq f_1, f_2 \leq 0.65mg$ N. The thrust and torque are calculated as: $f = f_1 + f_2$ and $\tau = \ell(f_1 - f_2)$.

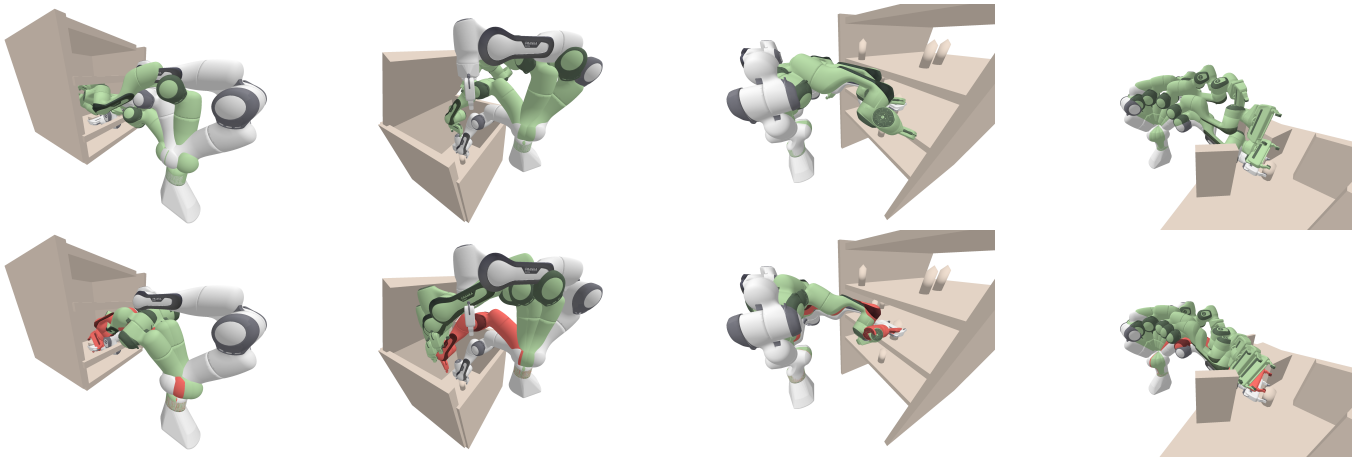
For each platform, we consider 2 DynoBench environments, shown in Fig. 5: *Bugtrap* and *Wall* for the unicycle, *Bugtrap* and *Hole* for the 2D quadrotor, and *Window* and *Obstacle* for the 3D quadrotor. To be compatible with SIMD-only collision checking, *e.g.*, [7], the obstacles in each environment are represented by a set of spheres, while the unicycle and quadrotor's geometries are both modeled as single spheres.

We apply our **FLASK** framework to obtain two planners: **FLASK-RRTConnect** with BVP solutions in Sec. V-C1 and three variants of **FLASK-SST*** with dynamics propagation in Sec. V-C2. Our kinodynamic **FLASK-RRTConnect** replaces a line segment connecting two nodes in an RRT tree by our closed-form BVP solution (local "flat" path) in the FLASKEXTEND subroutine (Alg. 2), using Eq. (25) with an optimal T^* from Eq. (26) and $\rho = 1$. For **FLASK-SST***, our first variant, **BVP-augmented FLASK-SST***, uses Eq. (29) to propagate robot dynamics in closed forms and takes advantage of our local "flat" path in (25) to check whether there is a direct connection from an existing node to the goal. Meanwhile, our second variant is a **SIMD-only FLASK-SST*** that keeps expanding the tree using vectorized collision checking via SIMD, *without* leveraging the BVP solutions. To highlight the "wandering" effect in propagation-based planners, our third variant disables best-state selection and tree pruning in SST*, effectively leading to a **DP-based FLASK-RRT**, where dynamics propagation (DP) with a sampled control is used to extend the RRT tree. The resulting trajectories are simplified using Alg. 4. We compare our approach with three baseline planners: **iDb-A*** [17], **SST*** [26] from OMPL [106], both without parallelized collision checking, and a GPU-based coarse-grained parallelized kinodynamic planner, **Kino-PAX** [84]. We maintain the aforementioned robot state and control constraints in our planners, enforced via our parallelized collision checking algorithm in Alg. 3, and in the baselines for a fair comparison.

As each method has its own definition of trajectory costs, it is challenging to compare the quality of the trajectories. However, most cost functions describe a combination of trajectory duration, length, control efforts, and even velocity, acceleration, or higher-order terms, all of which intuitively and indirectly prioritize shorter trajectories. Therefore, we choose the trajectory length and planning time as common metrics for comparison, shown in Table I. For anytime planners such as iDb-A* [17], SST* [26], we report the time of the first solution and the length of the final solution within a 180-second limit. Qualitatively, Fig. 5 plots the trajectories generated by our FLASK-RRTConnect (blue solid curves), our BVP-augmented FLASK-SST* (magenta solid curves), iDb-A* (orange dashed curves), SST* (green dotted curves), and Kino-PAX (dash-dotted purple curves).

1) Comparison to the baselines: The baselines iDb-A* and SST* generate shorter trajectories but often take several seconds to plan, especially for complicated robot dynamics such as quadrotors. Kino-PAX [84] achieves planning times in the millisecond range by parallelizing the tree expansion *on GPUs* but returns much longer trajectories, possibly due to the "wandering" effect" from dynamics propagation with sampled suboptimal control. While it achieves similar planning times for simple dynamics such as unicycles, it is ∼8–300 times slower than our FLASK-RRTConnect for more complicated systems such as quadrotors. For all 3 robot platforms, our FLASK-RRTConnect achieves significantly faster planning times than the baselines, in just a few milliseconds, offering the capability of real-time kinodynamic planning. Our BVP-augmented FLASK-SST* variant is slightly slower than our FLASK-RRTConnect since it is restricted to a constant control input for each local "flat" path instead of directly calculating a time-parameterized optimal control value. However, it is still significantly faster than the baselines in most cases. Meanwhile, our methods maintain comparable trajectory lengths, within 10–20% of the shortest length across the benchmarking problems. Intuitively, this is because the cost function (20) prioritizes shorter trajectories, which leads to lower total control effort and trajectory time. While our generated trajectories can be further optimized by combining with other optimization-based planners, *e.g.*, as initial solutions, this is out of the scope of our paper and left for future work.

2) The roles of BVP solutions and parallelized collision checking: All of our planners in the experiments use parallelized collision checking but integrate the BVP solution at different levels: *(i)* our FLASK-RRTConnect uses the BVP solution to find all the local paths; *(ii)* our *BVP-augmented* FLASK-SST* only uses the BVP solution to find a shortcut to the goal; *(iii)* our *SIMD-only* FLASK-SST* only leverages the vectorized collision checking without using the BVP solution; and *(iv)* our *DP-based* FLASK-RRT simply expands the planning tree using dynamics propagation without best-state



(a) (b) (c) (d)

Fig. 6: Visualization of trajectories generated by our FLASK−RRTConnect (top) and by the baseline planner, geometric RRTConnect + TOPP-RA (bottom), for a Franka Emika Panda robot in: (a) *cage*, (b) *box*, (c) *bookshelf thin*, and (d) *table pick* environments. For each environment, the bottom image shows a trajectory from the baseline with collided configurations in red, while the top image demonstrates that our trajectory is collision-free.

selection and pruning benefits from SST*.

Parallelized collision checking clearly improves the planning times. Compared to iDb-A* [17] and the original SST* [26], checking the local path for collision in parallel reduces the planning times from tens of seconds to a few seconds (see Table I) for robots with complicated dynamics such as quadrotors. This is the case even when the BVP solution is not used as in our *SIMD-only* FLASK−SST* or there is no best-state selection and tree pruning as in our *DP-based* FLASK−RRT. As propagation-based planners, our *SIMD-only* FLASK−SST* and *DP-based* FLASK−RRT still wander through the state space, a few small time steps at a time, causing longer planning times and trajectory lengths, as shown in Table I. This effect is even worse in *DP-based* FLASK−RRT as the best-state selection and pruning techniques used in SST* are not available.

However, our unique integration of BVP solutions and parallelized collision checking is the key to driving the planning times further down to the (sub-)millisecond range. By employing the BVP solution to find a shortcut from a node to the goal, our *BVP-augmented* FLASK−SST* generates a trajectory in just milliseconds, significantly faster than *SIMD-only* FLASK−SST*. This illustrates the potential of our framework to improve existing propagation-based planners by addressing the common "wandering" effects via our BVP solution. Taking a further step, our FLASK−RRTConnect fully integrates the BVP solution to find all local paths during tree expansion, and therefore, leads to less than a millisecond of planning time in many cases. This illustrates our approach's ability to transform any existing sampling-based geometric planner into an ultrafast kinodynamic planner, requiring only widely available CPUs.

B. Kinodynamic planning for high-DoF manipulators

In this section, we evaluate our algorithms with a simulated 7-DoF Franka Emika Panda robot in PyBullet [108]. We consider 7 realistic and challenging environments, including *bookshelf thin*, *bookshelf tall*, *bookshelf small*, *cage*, *box*, *table under pick* and *table pick*, from the MotionBenchMaker benchmark [107]. For each environment, 100 motion planning problems with 100 different pairs of start and goal configurations are generated for our experiments. The obstacles and the robot's geometries are represented by sets of spheres, which are compatible to SIMD parallelized collision checking as shown in [7]. For each problem, the robot's task is to plan a dynamically feasible trajectory from a start to a goal configuration in the free space.

In our experiments, we use our FLASK−RRTConnect planner, described in Sec. VII-A with trajectory simplification in Alg. 4. Our baseline is a common approach that generates a time-parameterized trajectory, *e.g.*, using TOPP-RA [66], from a collision-free geometric path, under kinematic constraints such as velocity and acceleration limits. The geometric path is provided by a SIMD-parallelized geometric RRTConnect planner from VAMP [7]. For each environment, we use both our planner and the baseline to solve the pre-generated benchmarking problems and report the results in Table II. Our planner's total planning time is calculated as the sum of the RRTConnect time and trajectory simplification time. Meanwhile, the baseline's total planning time includes the RRTConnect time, path simplification time, and TOPP-RA time. As our trajectory and the baseline's are optimized for different cost functions, we use their length as a common metrics, similar to Sec. VII-A. As TOPP-RA [66] only fits a time-parameterized trajectory to a geometric path, it does not consider any collision avoidance constraints. While the geometric path is collision-free, there is no guarantee that the time-parameterized trajectory from TOPP-RA will be valid. Therefore, we sample the resulting trajectories, check for collision and compare the collision risk, measured by the percentage of collided trajectories.

Table II compares our kinodynamic planner and the baseline, in terms of the total planning time, the final trajectory's length, and the collision risk, for each environment and overall. On average, we are able to generate a valid trajectory in around $3.5ms$, and in less than $1ms$ for 75% of the problems, while the baseline's total planning time is consistently around $4.5ms$ for almost all problems. While VAMP can generate a collision-free geometric path fast, the baseline's total planning time

TABLE II: Planning performance with a Franka Emika Panda robot over 100 runs of our FLASK-RRTConnect with different environments in MotionBenchMaker [107]. For a baseline, we use a collision-free geometric path, provided by a SIMD-parallelized geometric RRTConnect [7], followed by time parameterization (TOPP-RA [66]). Overall, our kinodynamic FLASK-RRTConnect generates a dynamically feasible trajectory faster than the baseline in almost all environments. Our trajectory is slightly longer but verified to be collision-free while the baseline's trajectory is colliding in approximately 30% of the cases, due to the time parameterization. Better metrics are shown in bold, while the collision risk is underlined in red if positive.

Env.	Metrics	Our kinodynamic FLASK-RRTConnect						Geometric RRTConnect (VAMP) + TOPP-RA					
		Mean	SD	25%	50%	75%	95%	Mean	SD	25%	50%	75%	95%
Bookshelf (thin)	Total planning time (ms)	**0.61**	1.5	**0.10**	**0.20**	**0.46**	**3.50**	3.96	1.02	3.14	3.70	4.60	5.47
	▷RRTConnect time	0.58	1.25	0.07	0.16	0.42	3.45	**0.16**	0.59	**0.05**	**0.07**	**0.11**	**0.27**
	▷Post./TOPP-RA time	**0.03**	0.02	**0.02**	**0.03**	**0.04**	**0.08**	3.80	0.84	3.04	3.62	4.43	5.28
	Final traj. length (rad)	5.25	1.08	4.53	5.15	5.90	7.63	**4.66**	0.79	**4.15**	**4.68**	**5.05**	**5.92**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.32**	0.47	0	0	**1**	**1**
Bookshelf (tall)	Total planning time (ms)	**1.09**	3.55	**0.07**	**0.15**	**0.36**	**5.2**	3.91	0.99	3.21	3.57	4.62	5.72
	▷RRTConnect time	1.06	3.55	0.05	0.12	0.34	5.19	**0.14**	0.29	**0.04**	**0.06**	**0.11**	**0.47**
	▷Post./TOPP-RA time	**0.03**	0.02	**0.02**	**0.03**	**0.04**	**0.06**	3.77	0.96	3.07	3.38	4.44	5.43
	Final traj. length (rad)	5.30	1.01	4.56	5.19	5.96	7.01	**4.86**	0.69	**4.34**	**4.85**	**5.28**	**6.08**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.34**	0.48	0	0	**1**	**1**
Bookshelf (small)	Total planning time (ms)	7.80	35.63	**0.08**	**0.17**	**0.64**	38.49	**4.11**	1.04	3.19	3.85	4.93	5.74
	▷RRTConnect time	7.77	35.63	0.06	0.14	0.57	38.45	**0.14**	0.27	**0.04**	**0.06**	**0.12**	**0.45**
	▷Post./TOPP-RA time	**0.03**	0.15	**0.02**	**0.03**	**0.04**	**0.06**	3.97	0.95	3.16	3.77	4.66	5.30
	Final traj. length (rad)	5.50	1.31	4.69	5.31	6.17	8.23	**4.83**	0.91	**4.34**	**4.83**	**5.32**	**6.05**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.32**	0.47	0	0	**1**	**1**
Cage	Total planning time (ms)	12.96	27.54	**1.83**	6.35	14.05	41.32	**5.81**	1.80	4.36	**5.32**	**7.21**	**8.92**
	▷RRTConnect time	12.89	27.53	1.78	6.28	13.97	41.13	**0.60**	0.45	**0.27**	**0.50**	**0.88**	**1.66**
	▷Post./TOPP-RA time	**0.07**	**0.04**	**0.05**	**0.06**	**0.09**	**0.14**	5.21	1.62	3.76	4.89	6.53	7.85
	Final traj. length (rad)	10.07	3.61	7.17	9.27	12.30	16.60	**7.09**	1.67	**5.55**	**6.54**	**8.52**	**9.63**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.29**	0.46	0	0	**1**	**1**
Box	Total planning time (ms)	**0.86**	4.25	**0.08**	**0.25**	**0.59**	**1.40**	4.22	0.93	3.35	4.25	4.97	5.64
	▷RRTConnect time	0.81	4.25	0.05	0.20	0.55	1.35	**0.14**	0.14	**0.07**	**0.12**	**0.17**	**0.26**
	▷Post./TOPP-RA time	**0.04**	0.02	**0.03**	**0.04**	**0.05**	**0.07**	4.07	0.91	3.24	4.07	4.84	5.38
	Final traj. length (rad)	5.64	1.20	4.79	5.26	6.14	8.10	**4.68**	0.82	**4.07**	**4.45**	**5.02**	**6.35**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.13**	0.34	0	0	0	**1**
Table under pick	Total planning time (ms)	**0.34**	0.45	**0.17**	**0.25**	**0.35**	**0.63**	5.13	0.99	4.41	5.10	5.89	6.45
	▷RRTConnect time	0.28	0.45	0.11	0.19	0.29	0.52	**0.09**	0.09	**0.06**	**0.07**	**0.10**	**0.15**
	▷Post./TOPP-RA time	**0.06**	0.03	**0.04**	**0.05**	**0.07**	**0.11**	5.04	0.98	4.34	5.03	5.71	6.36
	Final traj. length (rad)	8.46	1.89	7.07	8.48	9.55	11.82	**6.80**	1.39	**5.61**	**6.75**	**7.71**	**9.01**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.40**	0.49	0	0	**1**	**1**
Table pick	Total planning time (ms)	**0.28**	0.91	**0.08**	**0.12**	**0.18**	**0.57**	3.78	0.76	3.08	3.60	4.50	5.00
	▷RRTConnect time	0.26	0.91	0.05	0.09	0.15	0.55	**0.05**	0.03	**0.04**	**0.05**	**0.06**	**0.10**
	▷Post./TOPP-RA time	**0.03**	0.01	**0.02**	**0.03**	**0.03**	**0.05**	3.73	0.75	3.04	3.53	4.44	4.95
	Final traj. length (rad)	5.24	0.91	4.57	5.10	5.69	6.51	**4.76**	0.71	**4.26**	**4.53**	**5.18**	**6.27**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.42**	0.49	0	0	**1**	**1**
Overall	Total planning time (ms)	**3.49**	18.00	**0.11**	**0.24**	**0.76**	14.65	4.43	1.34	3.36	4.19	5.14	**7.16**
	▷ RRTConnect time	3.45	18.00	0.08	0.2	0.70	14.54	**0.18**	0.36	**0.05**	**0.07**	**0.15**	**0.83**
	▷ Post./TOPP-RA time	**0.04**	0.03	**0.02**	**0.04**	**0.05**	**0.09**	4.24	1.22	3.28	4.02	4.99	6.58
	Final traj. length (rad)	6.50	2.56	4.89	5.67	7.37	11.73	**5.26**	1.40	**4.30**	**4.88**	**5.74**	**8.43**
	Collision risk	**0**	0	**0**	**0**	**0**	**0**	**0.33**	0.47	0	0	**1**	**1**

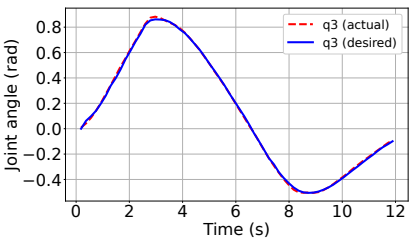
is dominated by the TOPP-RA time. Our approach, instead, spends most of the time in the FLASK-RRTConnect planner, as we directly enforce the dynamics constraints in the RRT tree construction. Across all the problems, the baseline offers a shorter trajectory but, more importantly, poses a collision risk of $\sim$ 30% on average, highlighting the benefit of our approach with guarantees of collision-free trajectories up to the sampling resolution. Fig. 6 visualizes the trajectories from our FLASK-RRTConnect (top) with no collisions and from the baseline (bottom) with collided configurations (red).

For the *table under pick* and *table pick* environments, our total planning time is approximately $\sim$ 15 times faster than that of the baseline, while maintaining $\sim$ 4 times shorter time for the *box*, *bookshelf thin* and *bookshelf tall* environments. For the *bookshelf small* and *cage* environments, our approach is slightly slower on average. Yet, it is still $\sim$ 8 times faster for 75% of the *bookshelf small* problems, suggesting that there are certain challenging cases in those environments that our approach takes slightly longer than the baseline to plan, but still achieves milliseconds of planning time. We emphasize that even though the baseline can be faster in those cases, it still leads to a significant risk of collision, around 30% of the *bookshelf small* and *cage* problems, as shown in Table II.

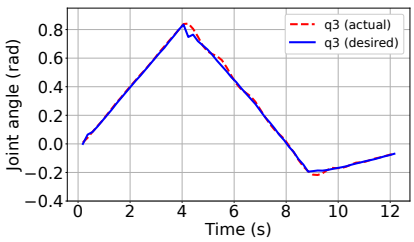
C. Real experiments

Finally, we verify the benefit of our approach for online motion planning on a real Universal Robots (UR5) platform in a cluttered environment. The environment consists of multiple static and dynamic objects, perceived by an Intel Realsense camera with a top-down view. The depth image from the camera is used to generate a set of spheres stored in a collision-affording point tree (CAPT) compatible with SIMD-parallelized motion planning [8]. The spheres represent the obstacles in the environment in our Alg. 3. Our FLASK-RRTConnect planner in Sec. VII-A is used to find a trajectory from the start configuration to the goal.

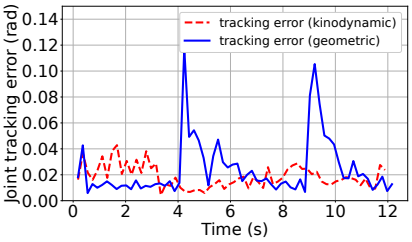
1) Kinodynamic Planning versus Geometric Planning: We first consider a "*pick and place*" scenario in Fig. 1, where the



(a) Tracking our trajectory

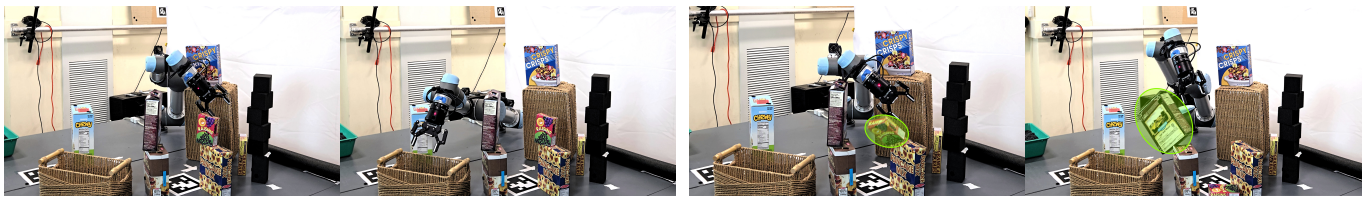


(b) Tracking a geometric path



(c) Tracking error

Fig. 7: Our trajectory is smoother and dynamically feasible, leading to better tracking performance. For example, the third joint angle q_3 in the our UR5's configuration stays close to the desired value from our trajectory (a) while there are overshoots and fluctuates with a geometric path [7] (b), causing collision in Fig. 8b. The tracking error $\|q_{actual} - q_{desired}\|$ also shows two overshooting peaks with the geometric path in Fig. 7c.



(a) (b)

Fig. 8: The "pick and place" task with UR5 robot in a cluttered environment with narrow passages: (a) our kinodynamic planner successfully finishes the task as it generates a smooth and dynamically feasible trajectory that the robot can track accurately; (b) the robot collides with obstacles when trying to track a geometric path [7] due to overshoots caused by the lack of dynamics constraints.

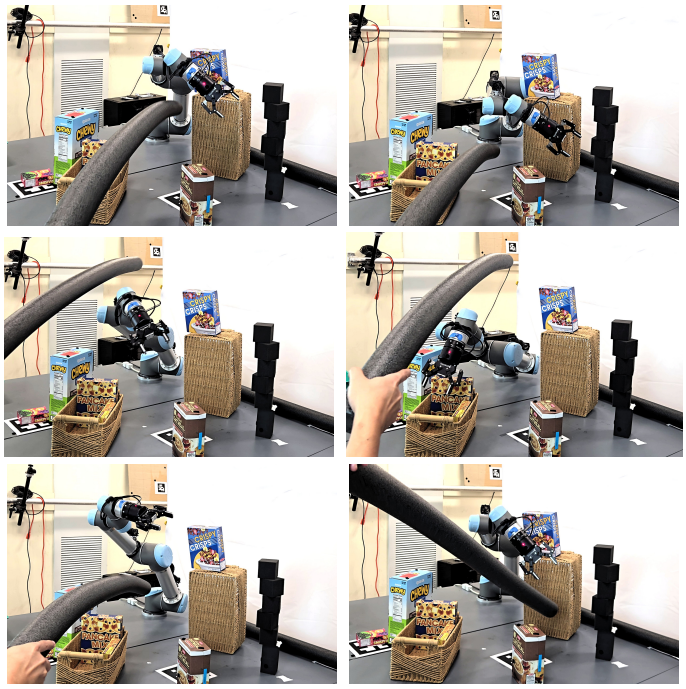


Fig. 9: Reactive planning with moving obstacles: our UR5 robot successfully performs a "*pick, place, and reset*" loop with our `FLASK-RRTConnect` planner without colliding with any of the static or moving obstacles. The obstacles are observed by an overhead Intel Realsense camera.

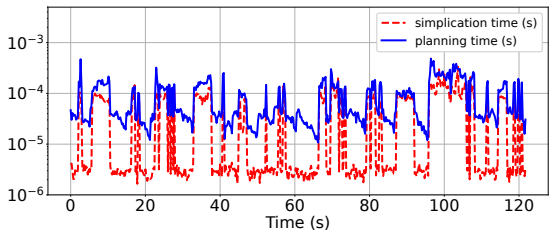


Fig. 10: Planning and simplification time in our "*pick, place, and reset*" loop: our planner takes $\sim 90\mu s$ and $\sim 40\mu s$ for planning and simplification, respectively, illustrating the reactiveness of our approach for real-time applications.

robot generates a motion plan from its start position to a *"pick"* position near a granola box, and then moves to a *"place"* position above a basket. The environment is cluttered with multiple objects, creating narrow passages such that a minor deviation from the planned trajectory can cause collisions. Therefore, accurate tracking is crucial for the safe realization of this "*pick and place*" task. The baseline is a geometric path, generated by a SIMD-parallelized RRTConnect planner from VAMP [7]. The path is sampled with a constant open-loop velocity, chosen such that the task's duration is similar to ours for a fair comparison. A closed-loop velocity control input is computed by Ruckig [67] and sent to the UR5 robot to execute.

Fig. 7 plots the tracking error during the experiments. Our trajectory is dynamically feasible, allowing the robot to smoothly and accurately execute the task without any collisions (Figs. 7a and 8a). Meanwhile, the geometric path has sharp turns, leading to overshoots and fluctuations (Fig. 7b). As a result, the robot collides with two nearby boxes while trying to turn, as seen in Fig. 8b. The overshoots can be observed in Fig. 7c with two spikes in total tracking error $\|\mathbf{q}_{actual} - \mathbf{q}_{desired}\|$, at times $t \approx 4$s and $t \approx 9$s, where $\mathbf{q}_{actual}$ and $\mathbf{q}_{desired}$ denote the actual and desired values of the joint angles from the motion plans. Meanwhile, our tracking error stays low without any spikes, highlighting the benefits of having a dynamically feasible trajectory for accurate task executions.

2) Reactive Kinodynamic Planning: To examine the reactiveness of our kinodynamic planning approach, we consider a "*pick, place, and reset*" loop, where the robot keeps planning

its trajectory to perform a "*pick*" action near a pile of blocks, a "*place*" action above the basket, and then "*reset*" to its original configuration. During the experiment, we move a foam obstacle in the environment and verify that our kinodynamic motion planner is able to quickly react to object changes and safely achieve the task as shown in Fig. 9. On average, our kinodynamic planner takes $\sim 90\mu s$ to plan and $\sim 40\mu s$ to simplify the trajectory (Fig. 10), leading to a total planning time of $0.13ms$. The results illustrate that our approach is fast and suitable for online and reactive planning with dynamic environments, while satisfying the dynamics constraints.

VIII. Conclusions

This paper develops **FLASK**, an ultrafast sampling-based kinodynamic planning framework, by solving the two-point boundary value problem and dynamics propagation in closed forms for a common class of differentially flat robot platforms, and performing extremely fast forward kinematics and collision checking via SIMD parallelism, available on most consumer CPUs. Our approach is exact, general and can be applied to any sampling-based planners while offering theoretical guarantees on probabilistic exhaustivity and asymptotic optimality. It is able to generate a dynamically feasible trajectory in the range of microseconds to milliseconds, which is suitable for online and reactive planning in dynamic environments. Our method outperforms common sampling-based and optimization-based kinodynamic planners as well as time-parameterization of a geometric path in terms of planning times, offering a fast, reliable, and feasible solution for trajectory generation.

Our approach unfolds a promising and exciting research area where it is possible to transform existing sampling-based motion planners with theoretical guarantees into fast kinodynamic versions while only requiring general-purpose and widely available CPUs. As we have closed-form BVP solutions for a large class of nonlinear differentially flat systems, we can largely bypass the propagation-based planners and can even enable roadmap-based kinodynamic planning. This ability potentially can lead to a wide application of our method on different robots, in different settings, and for various tasks. Our method can quickly bootstrap optimization-based planners with quality and dynamically feasible initial solutions, especially when their objective function differs from our cost. Moreover, the ability to specify a duration for our trajectory offers an exciting potential integration task and motion planning with temporal constraints such as task deadlines.

IX. Appendices

A. Derivation of closed-form solutions in Example 4

The optimal control input (23) becomes:

$$\begin{aligned}\mathbf{w}_{loc}(t) &= \begin{bmatrix}\mathbf{0} & \mathbf{I}_n\end{bmatrix}\left(\mathbf{I}_{2n} + \mathbf{A}^T(T-t)\right)\mathbf{G}_T^{-1}\mathbf{d}_T \\ &= \begin{bmatrix}\mathbf{0} & \mathbf{I}_n\end{bmatrix}\begin{bmatrix}\mathbf{I}_n & \mathbf{0}\\ (T-t)\mathbf{I}_n & \mathbf{I}_n\end{bmatrix}\begin{bmatrix}\frac{12}{T^3}\mathbf{I}_n & -\frac{6}{T^2}\mathbf{I}_n\\ -\frac{6}{T^2}\mathbf{I}_n & \frac{4}{T}\mathbf{I}_n\end{bmatrix}\mathbf{d}_T, \\ &= \begin{bmatrix}-\frac{12\mathbf{I}_n}{T^3} & \frac{6\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t + \begin{bmatrix}\frac{6\mathbf{I}_n}{T^2} & -\frac{2\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T.\end{aligned}$$

This leads to a minimum-acceleration optimal motion $\mathbf{z}_{loc}(t)$:

$$\begin{aligned}\mathbf{z}_{loc}(t) &= \begin{bmatrix}\frac{t^3}{3}\mathbf{I}_n & \frac{t^2}{2}\mathbf{I}_n\\ \frac{t^2}{2}\mathbf{I}_n & t\mathbf{I}_n\end{bmatrix}\begin{bmatrix}\mathbf{I}_n & \mathbf{0}\\ (T-t)\mathbf{I}_n & \mathbf{I}_n\end{bmatrix}\begin{bmatrix}\frac{12}{T^3}\mathbf{I}_n & -\frac{6}{T^2}\mathbf{I}_n\\ -\frac{6}{T^2}\mathbf{I}_n & \frac{4}{T}\mathbf{I}_n\end{bmatrix}\mathbf{d}_T \\ &\quad + \begin{bmatrix}\mathbf{I}_n & t\mathbf{I}_n\\ \mathbf{0} & \mathbf{I}_n\end{bmatrix}\begin{bmatrix}\mathbf{y}_0\\ \dot{\mathbf{y}}_0\end{bmatrix}, \\ &= \begin{bmatrix}\begin{bmatrix}-\frac{2\mathbf{I}_n}{T^3} & \frac{\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t^3 + \begin{bmatrix}\frac{3\mathbf{I}_n}{T^2} & -\frac{\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T t^2 + \dot{\mathbf{y}}_0 t + \mathbf{y}_0\\ \begin{bmatrix}-\frac{6\mathbf{I}_n}{T^3} & \frac{3\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t^2 + \begin{bmatrix}\frac{6\mathbf{I}_n}{T^2} & -\frac{2\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T t + \dot{\mathbf{y}}_0\end{bmatrix}.\end{aligned}$$

In other words, the optimal flat output trajectory is:

$$\mathbf{y}_{loc}(t) = \begin{bmatrix}-\frac{2\mathbf{I}_n}{T^3} & \frac{\mathbf{I}_n}{T^2}\end{bmatrix}\mathbf{d}_T t^3 + \begin{bmatrix}\frac{3\mathbf{I}_n}{T^2} & -\frac{\mathbf{I}_n}{T}\end{bmatrix}\mathbf{d}_T t^2 + \dot{\mathbf{y}}_0 t + \mathbf{y}_0.$$

The optimal cost function (24) is expanded as:

$$\begin{aligned}\mathcal{C}_{loc}(T) &= \begin{bmatrix}\mathbf{y}_f - \mathbf{y}_0 - T\dot{\mathbf{y}}_0\\ \dot{\mathbf{y}}_f - \dot{\mathbf{y}}_0\end{bmatrix}^\top\begin{bmatrix}\frac{12}{T^3}\mathbf{I}_n & -\frac{6}{T^2}\mathbf{I}_n\\ -\frac{6}{T^2}\mathbf{I}_n & \frac{4}{T}\mathbf{I}_n\end{bmatrix}\begin{bmatrix}\mathbf{y}_f - \mathbf{y}_0 - T\dot{\mathbf{y}}_0\\ \dot{\mathbf{y}}_f - \dot{\mathbf{y}}_0\end{bmatrix} + \rho T \\ &= 12\|\mathbf{y}_f - \mathbf{y}_0\|^2/T^3 - 12(\dot{\mathbf{y}}_f + \dot{\mathbf{y}}_0)^\top(\mathbf{y}_f - \mathbf{y}_0)/T^2 \\ &\quad + 4(\|\dot{\mathbf{y}}_0\|^2 + \dot{\mathbf{y}}_0^\top\dot{\mathbf{y}}_f + \|\dot{\mathbf{y}}_f\|^2)/T + \rho T.\end{aligned}$$

References

[1] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser. "A Review of Motion Planning for Highway Autonomous Driving". In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2020), pp. 1826–1848.

[2] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian. "Trajectory Planning for Quadrotor Swarms". In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 856–869.

[3] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, et al. "Swarm of micro flying robots in the wild". In: *Science Robotics* 7.66 (2022).

[4] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock. "Lessons from the amazon picking challenge: Four aspects of building robotic systems." In: *Robotics: Science and Systems*. 2016.

[5] L. D. Riek. "Healthcare robotics". In: *Communications of the ACM* 60.11 (2017), pp. 68–78.

[6] R. K. Jenamani, T. Silver, B. Dodson, S. Tong, A. Song, Y. Yang, Z. Liu, B. Howe, A. Whitneck, and T. Bhattacharjee. "FEAST: A Flexible Mealtime-Assistance System Towards In-the-Wild Personalization". In: *Robotics: Science and Systems*. Los Angeles, CA, USA, 2025.

[7] W. Thomason, Z. Kingston, and L. E. Kavraki. "Motions in Microseconds via Vectorized Sampling-Based Planning". In: *IEEE International Conference on Robotics and Automation*. 2024, pp. 8749–8756.

[8] C. W. Ramsey, Z. Kingston, W. Thomason, and L. E. Kavraki. "Collision-Affording Point Trees: SIMD-Amenable Nearest Neighbors for Fast Collision Checking". In: *Robotics: Science and Systems*. 2024.

[9] S. M. LaValle and J. J. K. Jr. "Randomized Kinodynamic Planning". In: *The Int. Journal of Robotics Research* 20.5 (2001), pp. 378–400.

[10] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. "Randomized Kinodynamic Motion Planning with Moving Obstacles". In: *The International Journal of Robotics Research* 21.3 (2002), pp. 233–255.

[11] S. M. LaValle. *Planning algorithms*. Cambridge Univ. Press, 2006.

[12] F. Augugliaro, A. P. Schoellig, and R. D'Andrea. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 1917–1922.

[13] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. "Motion planning with sequential convex optimization and convex collision checking". In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[14] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone. "GuSTO: Guaranteed Sequential Trajectory optimization via Sequential Convex Programming". In: *International Conference on Robotics and Automation*. 2019, pp. 6741–6747.

[15] Y. Tassa, T. Erez, and E. Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 4906–4913.

[16] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard. "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control". In: *IEEE International Conference on Robotics and Automation*. 2020, pp. 2536–2542.

[17] J. Ortiz-Haro, W. Hönig, V. N. Hartmann, and M. Toussaint. "iDb-A*: Iterative Search and Optimization for Optimal Kinodynamic Motion Planning". In: *IEEE Transactions on Robotics* 41 (2025).

[18] M. Pivtoraiko and A. Kelly. "Efficient constrained path planning via search in state lattices". In: *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*. 2005, pp. 1–7.

[19] M. Pivtoraiko and A. Kelly. "Kinodynamic motion planning with state lattice motion primitives". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 2172–2179.

[20] B. J. Cohen, S. Chitta, and M. Likhachev. "Search-based planning for manipulation with motion primitives". In: *IEEE International Conference on Robotics and Automation*. 2010, pp. 2902–2908.

[21] S. Liu, N. Atanasov, K. Mohta, and V. Kumar. "Search-based motion planning for quadrotors using linear quadratic minimum time control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2017, pp. 2872–2879.

[22] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, and M. Horn. "Search-Based Optimal Motion Planning for Automated Driving". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2018, pp. 4523–4530.

[23] D. J. Webb and J. van den Berg. "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics". In: *IEEE Int. Conference on Robotics and Automation*. 2013, pp. 5054–5061.

[24] S. Karaman and E. Frazzoli. "Optimal kinodynamic motion planning using incremental sampling-based methods". In: *IEEE Conference on Decision and Control*. 2010, pp. 7681–7687.

[25] K. Hauser and Y. Zhou. "Asymptotically Optimal Planning by Feasible Kinodynamic Planning in a State–Cost Space". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1431–1443.

[26] Y. Li, Z. Littlefield, and K. E. Bekris. "Asymptotically optimal sampling-based kinodynamic planning". In: *The International Journal of Robotics Research* 35.5 (2016), pp. 528–564.

[27] C. K. Verginis, D. V. Dimarogonas, and L. E. Kavraki. "KDF: Kinodynamic Motion Planning via Geometric Sampling-Based Algorithms and Funnel Control". In: *IEEE Transactions on Robotics* 39.2 (2023), pp. 978–997.

[28] J. van den Berg and M. Overmars. "Kinodynamic motion planning on roadmaps in dynamic environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2007, pp. 4253–4258.

[29] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 2016.

[30] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics". In: *IEEE International Conference on Robotics and Automation*. 2012, pp. 2537–2542.

[31] W. J. Wolfslag, M. Bharatheesha, T. M. Moerland, and M. Wisse. "RRT-CoLearn: towards kinodynamic planning without numerical trajectory optimization". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1655–1662.

[32] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust. "RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4298–4305.

[33] D. Zheng and P. Tsiotras. "Sampling-based kinodynamic motion planning using a neural network controller". In: *AIAA Scitech Forum*. 2021, p. 1754.

[34] R. M. Murray, M. Rathinam, and W. Sluis. "Differential flatness of mechanical control systems: A catalog of prototype systems". In: *ASME International Mechanical Engineering Congress and Exposition*. 1995, pp. 349–357.

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

[36] H. K. Khalil. *Nonlinear systems*. Prentice-Hall, 2002.

[37] D. Mellinger, N. Michael, and V. Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[38] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. "Curobo: Parallelized collision-free robot motion generation". In: *2023 IEEE International Conference on Robotics and Automation*. IEEE. 2023, pp. 8112–8119.

[39] E. Schmerling, L. Janson, and M. Pavone. "Optimal sampling-based motion planning under differential constraints: The driftless case". In: *IEEE International Conference on Robotics and Automation*. 2015, pp. 2368–2375.

[40] L. Kavraki, M. Kolountzakis, and J.-C. Latombe. "Analysis of probabilistic roadmaps for path planning". In: *IEEE Transactions on Robotics and Automation* 14.1 (1998), pp. 166–171.

[41] J. J. Kuffner and S. M. LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *IEEE International Conference on Robotics and Automation*. Vol. 2. 2000, pp. 995–1001.

[42] E. Schmerling and M. Pavone. "Kinodynamic planning". In: *Encyclopedia of Robotics*. Springer, 2019.

[43] Y. Chen, M. Cutler, and J. P. How. "Decoupled multiagent path planning via incremental sequential convex programming". In: *IEEE International Conference on Robotics and Automation*. 2015.

[44] T. A. Howell, B. E. Jackson, and Z. Manchester. "ALTRO: A Fast Solver for Constrained Trajectory Optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2019, pp. 7674–7679.

[45] M. Toussaint. "A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference". In: *Geometric and Numerical Foundations of Movements* (2017), pp. 361–392.

[46] G. L'Erario, G. Nava, G. Romualdi, F. Bergonti, V. Razza, S. Dafarra, and D. Pucci. "Whole-body trajectory optimization for robot multimodal locomotion". In: *IEEE-RAS 21st International Conference on Humanoid Robots*. 2022, pp. 651–658.

[47] S. Beck, C. Nguyen, T. Duong, N. Atanasov, and Q. Nguyen. "High Accuracy Aerial Maneuvers on Legged Robots using Variational Integrator Discretized Trajectory Optimization". In: *IEEE International Conference on Robotics and Automation*. 2025, pp. 10253–10260.

[48] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake. "Motion planning around obstacles with convex optimization". In: *Science Robotics* 8.84 (2023).

[49] D. von Wrangel and R. Tedrake. "Using Graphs of Convex Sets to Guide Nonconvex Trajectory Optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2024, pp. 9863–9870.

[50] B. P. Graesdal, S. Y. C. Chia, T. Marcucci, S. Morozov, A. Amice, P. Parrilo, and R. Tedrake. "Towards Tight Convex Relaxations for Contact-Rich Manipulation". In: *Robotics: Science and Systems*. Delft, Netherlands, 2024.

[51] I. Mishani, Y. Shaoul, R. Natarajan, J. Li, and M. Likhachev. "SRMP: Search-Based Robot Motion Planning Library". In: *arXiv preprint arXiv:2509.25352* (2025).

[52] P. E. Hart, N. J. Nilsson, and B. Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[53] A. Orthey, C. Chamzas, and L. E. Kavraki. "Sampling-Based Motion Planning: A Comparative Review". In: *Annual Review of Control, Robotics, and Autonomous Systems* 7.1 (July 2024), pp. 285–310.

[54] Y. Gao, L. Chen, P. Bhovad, S. Wang, Z. Kingston, and L. H. Blumenschein. "Parallel Simulation of Contact and Actuation for Soft Growing Robots". In: *arXiv preprint arXiv:2509.15180* (2025).

[55] B. Ichter and M. Pavone. "Robot Motion Planning in Learned Latent Spaces". In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2407–2414.

[56] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip. "MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning Under Kinodynamic Constraints". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4496–4503.

[57] J. Ortiz-Haro, W. Hönig, V. N. Hartmann, M. Toussaint, and L. Righetti. "iDb-RRT: Sampling-based Kinodynamic Motion Planning with Motion Primitives and Trajectory Optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2024, pp. 10702–10709.

[58] R. Natarajan, S. Mukherjee, H. Choset, and M. Likhachev. "PINSAT: Parallelized Interleaving of Graph Search and Trajectory Optimization for Kinodynamic Motion Planning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2024.

[59] R. Natarajan, G. L. Johnston, N. Simaan, M. Likhachev, and H. Choset. "Torque-Limited Manipulation Planning through Contact by Interleaving Graph Search and Trajectory Optimization". In: *IEEE International Conference on Robotics and Automation*. 2023.

[60] R. Natarajan, H. Choset, and M. Likhachev. "Interleaving Graph Search and Trajectory Optimization for Aggressive Quadrotor Flight". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5357–5364.

[61] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini. "Sampling-based optimal kinodynamic planning with motion primitives". In: *Autonomous Robots* 43.7 (2019), pp. 1715–1732.

[62] R. Shome and L. E. Kavraki. "Asymptotically optimal kinodynamic planning using bundles of edges". In: *IEEE International Conference on Robotics and Automation*. 2021, pp. 9988–9994.

[63] J. Kamat, J. Ortiz-Haro, M. Toussaint, F. T. Pokorny, and A. Orthey. "BITKOMO: Combining Sampling and Optimization for Fast Convergence in Optimal Motion Planning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2022, pp. 4492–4497.

[64] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer. "Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning". In: *IEEE International Conference on Robotics and Automation*. 2016, pp. 4207–4214.

[65] K. V. Alwala and M. Mukadam. "Joint sampling and trajectory optimization over graphs for online motion planning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021, pp. 4700–4707.

[66] H. Pham and Q.-C. Pham. "A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis". In: *IEEE Transactions on Robotics* 34.3 (2018), pp. 645–659.

[67] L. Berscheid and T. Kröger. "Jerk-limited Real-time Trajectory Generation with Arbitrary Target States". In: *Robotics: Science and Systems* (2021).

[68] R. E. Allen and M. Pavone. "A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance". In: *Robotics and Autonomous Systems* 115 (2019), pp. 174–193.

[69] L. Bascetta, I. M. Arrieta, and M. Prandini. "Flat-RRT*: A sampling-based optimal trajectory planner for differentially flat vehicles with constrained dynamics". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 6965–6970.

[70] J. Welde, J. Paulos, and V. Kumar. "Dynamically Feasible Task Space Planning for Underactuated Aerial Manipulators". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3232–3239.

[71] H. Ye, N. Pan, Q. Wang, C. Xu, and F. Gao. "Efficient Sampling-based Multirotors Kinodynamic Planning with Fast Regional Optimization and Post Refining". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2022, pp. 3356–3363.

[72] Y. Wang, W. Zeng, Y. Peng, Q. Yang, and J. Zhou. "Differential Flatness-Based Trajectory Planning for Small Fixed-Wing UAVs". In: *International Conference on Autonomous Unmanned Systems*. Springer. 2024, pp. 360–369.

[73] M. Seemann and K. Janschek. "Exact and efficient local planning for orbitally flat systems within the RRT* framework". In: *International Conference on Control Automation Robotics & Vision*. IEEE. 2014, pp. 1631–1636.

[74] Z. Han, Y. Wu, T. Li, L. Zhang, L. Pei, L. Xu, C. Li, C. Ma, C. Xu, S. Shen, et al. "An efficient spatial-temporal trajectory planner for autonomous vehicles in unstructured environments". In: *IEEE Transactions on Intelligent Transportation Systems* 25.2 (2023), pp. 1797–1814.

[75] Y. Hao, A. Davari, and A. Manesh. "Differential flatness-based trajectory planning for multiple unmanned aerial vehicles using mixed-integer linear programming". In: *American Control Conference*. 2005, pp. 104–109.

[76] L. E. Beaver and A. A. Malikopoulos. "Optimal control of differentially flat systems is surprisingly easy". In: *Automatica* 159 (2024), p. 111404.

[77] M. N. Vu, M. Schwegel, C. Hartl-Nesic, and A. Kugi. "Sampling-based trajectory (re) planning for differentially flat systems: Application to a 3D gantry crane". In: *IFAC-PapersOnLine* 55.38 (2022), pp. 33–40.

[78] B. Raveh, A. Enosh, and D. Halperin. "A little more, a lot better: Improving path quality by a path-merging algorithm". In: *IEEE Transactions on Robotics* 27.2 (2011), pp. 365–371.

[79] N. M. Amato and L. K. Dale. "Probabilistic roadmap methods are embarrassingly parallel". In: *IEEE International Conference on Robotics and Automation*. Vol. 1. 1999, pp. 688–694.

[80] J. Ichnowski and R. Alterovitz. "Parallel sampling-based motion planning with superlinear speedup". In: *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1206–1212.

[81] S. A. Jacobs, K. Manavi, J. Burgos, J. Denny, S. Thomas, and N. M. Amato. "A scalable method for parallelizing sampling-based motion planning algorithms". In: *IEEE International Conference on Robotics and Automation*. 2012, pp. 2529–2536.

[82] P. Werner, R. Cheng, T. Stewart, R. Tedrake, and D. Rus. "Superfast Configuration-Space Convex Set Computation on GPUs for Online Motion Planning". In: *Robotics: Science and Systems*. Los Angeles, CA, USA, 2025.

[83] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. "Sampling-based roadmap of trees for parallel motion planning". In: *IEEE Transactions on Robotics* 21.4 (2005), pp. 597–608.

[84] N. Perrault, Q. H. Ho, and M. Lahijanian. "Kino-PAX: Highly Parallel Kinodynamic Sampling-based Planner". In: *IEEE Robotics and Automation Letters* (2025).

[85] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots. "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation". In: *Conference on Robot Learning*. PMLR. 2022, pp. 750–759.

[86] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox. "Motion policy networks". In: *Conference on Robot Learning*. PMLR. 2023, pp. 967–977.

[87] A. T. Le, K. Hansel, J. Carvalho, J. Watson, J. Urain, A. Biess, G. Chalvatzaki, and J. Peters. "Global tensor motion planning". In: *IEEE Robotics and Automation Letters* (2025).

[88] A. T. Le, K. Nguyen, M. N. Vu, J. Carvalho, and J. Peters. "Model Tensor Planning". In: *Trans. on Machine Learning Research* (2025).

[89] B. Ichter, E. Schmerling, and M. Pavone. "Group Marching Tree: Sampling-Based Approximately Optimal Motion Planning on GPUs". In: *IEEE International Conference on Robotic Computing*. 2017, pp. 219–226.

[90] T. S. Wilson, W. Thomason, Z. Kingston, L. E. Kavraki, and J. D. Gammell. "Nearest-Neighbourless Asymptotically Optimal Motion Planning with Fully Connected Informed Trees (FCIT*)". In: *IEEE Int. Conference on Robotics and Automation*. 2025, pp. 14140–14146.

[91] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (2002), pp. 566–580.

[92] D. Zhou and M. Schwager. "Vector field following for quadrotors using differential flatness". In: *IEEE International Conference on Robotics and Automation*. 2014, pp. 6567–6572.

[93] S. Liu, K. Mohta, N. Atanasov, and V. Kumar. "Search-Based Motion Planning for Aggressive Flight in SE(3)". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2439–2446.

[94] E. Verriest and F. Lewis. "On the linear quadratic minimum-time problem". In: *IEEE Transactions on Automatic Control* 36.7 (1991), pp. 859–863.

[95] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[96] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Trans. Math. Softw.* 23.4 (1997), pp. 550–560.

[97] K. B. Petersen, M. S. Pedersen, et al. "The matrix cookbook". In: *Technical University of Denmark* 7.15 (2008), p. 510.

[98] R. Geraerts and M. H. Overmars. "Creating high-quality paths for motion planning". In: *The International Journal of Robotics Research* 26.8 (2007), pp. 845–863.

[99] K. Hauser and V. Ng-Thow-Hing. "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts". In: *IEEE Int. Conference on Robotics and Automation*. 2010, pp. 2493–2498.

[100] K. E. Bekris and R. Shome. "Asymptotically Optimal Sampling-Based Planners". In: *Encyclopedia of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–12.

[101] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894.

[102] L. Janson, E. Schmerling, A. Clark, and M. Pavone. "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions". In: *The International Journal of Robotics Research* 34.7 (2015), pp. 883–921.

[103] E. Schmerling, L. Janson, and M. Pavone. "Optimal sampling-based motion planning under differential constraints: The drift case with linear affine dynamics". In: *IEEE Conference on Decision and Control*. 2015, pp. 2574–2581.

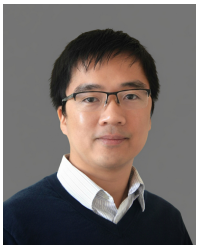
[104] M. Penrose. *Random Geometric Graphs*. Oxford Univ. Press, May 2003.

[105] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski. "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering". In: *International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2017, pp. 37–42.

[106] I. A. Şucan, M. Moll, and L. E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). https://ompl.kavrakilab.org.

[107] C. Chamzas, C. Quintero-Peña, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki. "MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets". In: *IEEE Robotics and Automation Letters* 7.2 (Apr. 2022), pp. 882–889.

[108] E. Coumans and Y. Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2021.



Thai Duong received his Ph.D. degree in Electrical and Computer Engineering from the University of California San Diego, La Jolla, CA, USA, in 2024.

He is currently a postdoctoral research associate at Rice University, Houston, TX, USA. His research interests include machine learning with applications to robotics, robot dynamics learning, motion planning and control.

Clayton W. Ramsey is a 4th-year Ph.D. student under Dr. Lydia E. Kavraki at Rice University.

He is a NASA NSTGRO Research Fellow, collaborating with the Dexterous Robotics team at NASA Johnson Space Center. His research specializes in hardware acceleration for robot planning, especially for long-horizon planning in dynamic environments.

Zachary Kingston received his Ph.D. degree in Computer Science from Rice University, Houston, TX, USA, in 2021 under the supervision of Dr. Lydia E. Kavraki.

He is currently an Assistant Professor of Computer Science at Purdue University, West Lafayette, IN, USA. His research interests include robot motion planning, hardware-accelerated planning, and planning for systems under constraints.

Wil Thomason received the Ph.D. degree in Computer Science from Cornell University, Ithaca, NY, USA, in 2021.

He was funded by the National Defense Science and Engineering Graduate Fellowship for his Ph.D. He is a research scientist at the Robotics and AI Institute. He was a Postdoctoral Computing Research Association/Computing Community Consortium Computing Innovation Fellow working under the direction of Dr. Kavraki with the Department of Computer Science, Rice University, Houston, TX, USA.

Lydia E. Kavraki (Fellow, IEEE) received the Ph.D. degree in computer science from Stanford University, Stanford, CA, USA, in 1995.

She is the Kenneth and Audrey Kennedy University Professor of Computing at Rice University, Houston, TX, USA, and Director of the Ken Kennedy Institute. She coauthored *Principles of Robot Motion* (MIT Press, 2005), pioneered sampling-based motion planning, and leads development of the Open Motion Planning Library. Her research interests include motion planning, human-robot collaboration, and robotics/AI methods for computational biomedicine.

Dr. Kavraki is a Fellow of AAAI, AIMBE, ACM, and AAAS, and a Member of the National Academies of Engineering, Sciences, and Medicine.