IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Global Tensor Motion Planning

An T. Le[1], Kay Pompetzki[1], João Carvalho[1], Joe Watson[1,2],
Julen Urain[1,5], Armin Biess[6], Georgia Chalvatzaki[1,5] and Jan Peters[1,2,3,4]
[1]Intelligent Autonomous Systems Lab, TU Darmstadt, Germany; [2]German Research Center for AI (DFKI);
[3]Hessian.AI; [4]Centre for Cognitive Science; [5]Interactive Robot Perception & Learning Lab, TU Darmstadt
[6]Autobrains, Israel
Corresponding author: An T. Le, an@robot-learning.de

***Abstract*—Batch planning is increasingly necessary to quickly produce diverse and quality motion plans for downstream learning applications, such as distillation and imitation learning. This paper presents Global Tensor Motion Planning (GTMP)—a sampling-based motion planning algorithm comprising only tensor operations. We introduce a novel discretization structure represented as a random multipartite graph, enabling efficient vectorized sampling, collision checking, and search. We provide a theoretical investigation showing that GTMP exhibits probabilistic completeness while supporting modern GPU/TPU. Additionally, by incorporating smooth structures into the multipartite graph, GTMP directly plans smooth splines without requiring gradient-based optimization. Experiments on lidar-scanned occupancy maps and the MotionBenchMaker dataset demonstrate GTMP's computation efficiency in batch planning compared to baselines, underscoring GTMP's potential as a robust, scalable planner for diverse applications and large-scale robot learning tasks.**

***Index Terms*—Motion and Path Planning, Manipulation Planning**

I. Introduction

Motion planning with probabilistic completeness has been a foundation of robotics research, with seminal works like PRM [1] and RRTConnect [2] serving as cornerstone methods for years [3]. However, as the complexity of robotic tasks increases, there is a growing demand for batch-planning methods. Several factors drive this interest: (i) the need to gather large datasets for policy learning [4]–[6], (ii) the inherent non-linearity of task objectives that lead to multiple viable solutions [7]–[9], and (iii) the increasing availability of powerful GPUs/TPUs for accelerated planning [10], [11]. Despite these advances, batching traditional sampling-based planners, such as RRT/PRM and their variants, remains an ongoing challenge [12]–[15]. Their underlying discretization techniques, such as the incremental graph construction of RRT/PRM or the search mechanism of A* [16], [17], are not conducive to efficient vectorization over planning instances.

This paper revisits classical motion planning, where we plan from a single start configuration to multiple goal configurations. We introduce a simple yet effective discretization structure with layers of waypoints, which can be represented as tensors, enabling GPU/TPU utilization. We propose Global Tensor Motion Planning (GTMP), which enables highly batchable operations on multiple planning instances, such as batch collision checking and batch Value Iteration (VI), inducing an easily vectorizable implementation with JAX [18]. This simplicity allows for differentiable planning and rapid integration with modern frameworks, making the algorithm particularly desirable for real-time applications and scalable data collection for robot learning. Our experimental results demonstrate much better batch efficiency planning than standard baseline implementations while achieving similar smoothness and better path diversity with the spline discretization structure.

Our contributions are twofold: i) we propose a *vectorizable sampling-based planner* exhibiting probabilistic completeness, which does not require simplification routines [19], and ii) we extend GTMP with a spline discretization structure, enabling batch spline planning with path quality comparable to trajectory optimizers.

II. Related Works

Vectorizing motion planning has been an active research topic for decades. Here, we briefly survey the most relevant works on vectorizing either at the *algorithmic*-level (e.g., collision-checking) or *instance*-level (e.g., batch trajectory planning).

Sampling-based Vectorization. Recognizing the importance of planning parallelization, the earliest works [12], [13], [15], [20], [21] propose a *vectorizable* collision-checking data structure. State-of-the-art work on leveraging CPU-based *single instruction, multiple data* [22] (i.e., VAMP) has pushed collision-checking efficiency to *microseconds*. In a different vein, a body of works [23]–[28] proposes a learning heuristic or batch-sampling strategies to inform or refine the search-graph with new samples, effectively reducing collision checking. Despite the hardware or algorithmic acceleration efforts, past works still resort to discretization structures such as trees for RRT variants or graphs for PRM variants [29], which are unsuitable for instance-level vectorization.

Vectorizing Trajectory Optimization. Vectorizing optimization-based planner with GPU-acceleration [7], [10], [30]–[32] gained traction recently due to their computational efficiency, the solutions' multi-modality, and their robustness to bad local-minima. However, these local methods are sensitive to initial conditions and may get stuck in large infeasible regions, thereby the need for warmstarting the sampling-based global solutions [11]. GTMP addresses this issue by proposing a layerwise discretization structure,

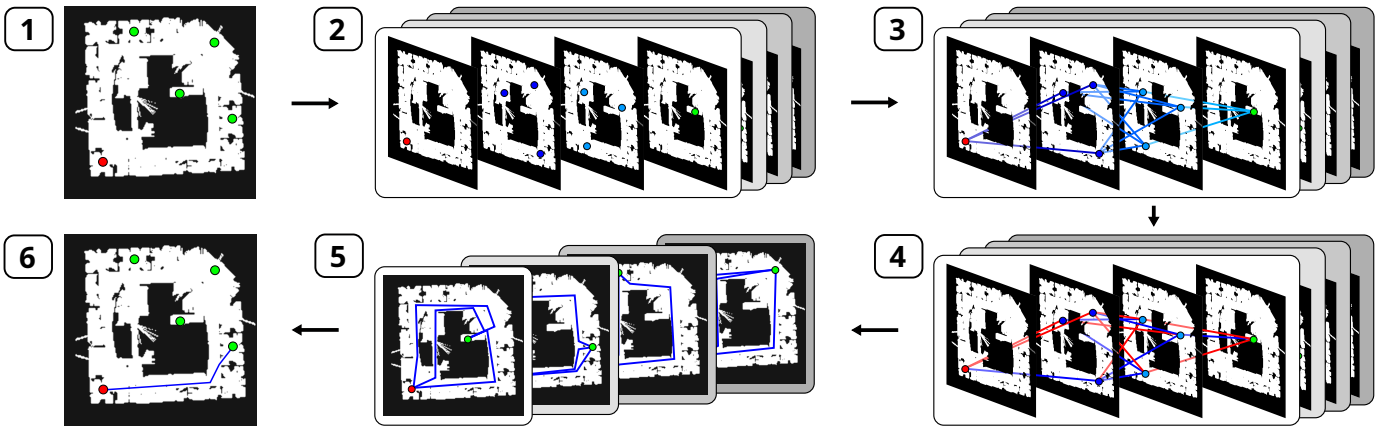

Fig. 1. GTMP can plan with multiple goals or `vmap` over goals. For clarity, we present an example of performing JAX `vmap` on GTMP (M=2, N=3) over the batch of $B = 3$ seeds. **(1)** The objective is to find a batch of feasible paths from the start (red) to the goals (green). **(2, 3)** In each seed, we sample a multipartite graph and form a tensor (Algorithm 1, Line 1). **(4)** A batch of collision checks is performed and stored into cost matrices (Algorithm 1, Line 2). **(5)** Then, per seed, we execute finite value iterations (Algorithm 1, Line 5-7) and trace the optimal path from the optimal value matrices (Algorithm 1, Line 8-13). **(6)** For execution, we can select the best path in terms of exemplary shortest path criteria. More information can be found on https://sites.google.com/view/gtmp.

enabling vectorization in sampling and search operations while having better global solutions.

III. Tensorizing Motion Planning

We consider the path planning problem [33] for a configuration $\boldsymbol{q} \in \mathcal{C} \subset \mathbb{R}^d$ in a compact d-dimensional space, with $\mathcal{C}_{\text{coll}}$ being the collision space such that $\mathcal{C} \setminus \mathcal{C}_{\text{coll}}$ is open. Let $\mathcal{C}_{\text{free}} = \mathsf{Cl}(\mathcal{C} \setminus \mathcal{C}_{\text{coll}})$ be the free space, with $\mathsf{Cl}(\cdot)$ the set closure. Denote the start configuration $\boldsymbol{q}_0$ and a set of goal configurations $\mathcal{G}$. Let $f : [0,1] \to \mathcal{C}$, we can define its total variation as its arc length

$$\text{TV}(f) = \sup_{M \in \mathbb{N}, 0=t_0,\dots,t_M=1} \sum_{i=1}^{M} \left\| \boldsymbol{f}(t_i) - \boldsymbol{f}(t_{i-1}) \right\|, \quad (1)$$

Definition 1 (Feasible Path)**.** *The function $f : [0,1] \to \mathcal{C}$ with $TV(f) < \infty$ is*

- *a path, if it is continuous.*
- *a feasible path, if and only if $\forall t \in [0,1], \boldsymbol{f}(t) \in \mathcal{C}_{free}$, $\boldsymbol{f}(0) = \boldsymbol{q}_0$, $\boldsymbol{f}(1) \in \mathcal{G}$.*

Let $\mathcal{F}$ be the set of all paths. We denote $\mathcal{F}_{\text{free}}$ as the set of feasible paths for a feasible planning problem. Here, we do not consider dynamic constraints (e.g., velocity or acceleration limits).

Problem 1 (Batch Path Planning)**.** *Given a planning problem $(\mathcal{C}_{free}, \boldsymbol{q}_0, \mathcal{G})$ and cost function $c : \mathcal{F} \to \mathbb{R}_{>0}$, find a batch of $B > 0$ feasible paths and report failure if no feasible path exists.*

This problem definition is standard for several robotic settings, such as serial manipulators with joint limits. We propose to solve Problem 1 with probabilistic completeness, striving to discover multiple solution modes.

Practical Motivation. In essence, GTMP leverages a fixed discretization structure over the whole search space, represented by fixed-shape tensors, to enable efficient planning vectorization with JAX `vmap` operation [18]. This approach contrasts with the incremental discretization structures of classical motion planning algorithms, which procedurally expand the search space during planning.

A. Discretization Structure

We introduce the random multipartite graph as a novel configuration discretization structure designed to represent planning problems as tensors.

Definition 2 (Random Multipartite Graph Discretization)**.** *Consider a geometric graph $G = (\mathcal{V}, \mathcal{E})$ on configuration space $\mathcal{C}$, the node set $\mathcal{V}$ is represented by $\{\boldsymbol{q}_s, \mathcal{M}, \mathcal{G}\}$, where $\mathcal{M} = \{\mathcal{L}_m\}_{m=1}^{M}$ is a set of M layers. Each layer $\mathcal{L}_m = \{\boldsymbol{q}_i \in \mathcal{C} \mid \boldsymbol{q}_i \sim p_m\}_{i=1}^{N}$ contains N waypoints sampled by an associated proposal distribution p_m on $\mathcal{C}$. The edge set $\mathcal{E}$ is defined by the union of (forward) pair-wise connections between the start and first layer $\{(\boldsymbol{q}_s, \boldsymbol{q}) \mid \forall \boldsymbol{q} \in \mathcal{L}_1\}$, between layers in $\mathcal{M}$*

$$\{(\boldsymbol{q}_m, \boldsymbol{q}_{m+1}) \mid \forall \boldsymbol{q}_m \in \mathcal{L}_m,\ \boldsymbol{q}_{m+1} \in \mathcal{L}_{m+1},\ 1 \le m < M\},$$

and between the last layer and goals $\{(\boldsymbol{q}, \boldsymbol{q}_g) \mid \forall \boldsymbol{q} \in \mathcal{L}_M,\ \boldsymbol{q}_g \in \mathcal{G}\}$, leading to a complete $(M+2)$-partite directed graph.

We typically set $p_m = \mathcal{U}(\mathcal{C})$ as uniform distributions over configuration space (bounded by configuration limits cf. Fig. 1). Consequently, the graph nodes are represented as the waypoint tensors for all layers $\boldsymbol{Q} \in \mathbb{R}^{M \times N \times d}$ and the goal configuration $\boldsymbol{G} \in \mathbb{R}^{|\mathcal{G}| \times d}$ from $\mathcal{G}$, within the state limits. Extending Definition 2 to *spline discretization structure* by replacing the straight line with the cubic polynomials, representing any edge $(\boldsymbol{q}, \boldsymbol{q}') \in \mathcal{E}$, is straightforward with Akima spline [34] (cf. Section B).

Definition 3 (Path In G)**.** *A path $f : [0,1] \to \mathcal{C}$ in G exists if $\boldsymbol{f}(0) = \boldsymbol{q}_0$, $\boldsymbol{f}(1) \in \mathcal{G}$ and its piecewise linear segments correspond to edges in $\mathcal{E}$ connecting $\boldsymbol{q}_0$ through the layers to some $\boldsymbol{q}_g \in \mathcal{G}$.*

B. State Machine On Graph

The graph G is represented by the state machine $(\mathcal{V}, \mathcal{E}, c, t)$ [35], where the state set is the node set of G, the action set is equivalent to the edge set $\mathcal{E}$, the transition cost function $c : \mathcal{V} \times \mathcal{E} \to \mathbb{R}$, deterministic state transition $t(\boldsymbol{q}' \mid \boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) = 1$, $(\boldsymbol{q}, \boldsymbol{q}') \in \mathcal{E}$. The goal set $\mathcal{G} \subset \mathcal{V}$ is the terminal set with terminal costs $c_g(\boldsymbol{q})$, $\boldsymbol{q} \in \mathcal{G}$. A policy $\pi : \mathcal{V} \to \mathcal{E}$ depicts the decision to transition to the next layer, given the current state at the current layer.

We use unbounded occupancy collision costs

$$c_{\text{coll}}(\boldsymbol{q}) = 0 \text{ if } \boldsymbol{q} \in \mathcal{C}_{\text{free}}, \text{ else } \infty, \tag{2}$$

which merges the planning and verification steps (cf. Proposition 2). Then, the transition cost function can be defined

$$c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) = \underbrace{\int_a^b c_{\text{coll}}(\boldsymbol{f}(t)) f' dt}_{\text{collision}} + \underbrace{\|\boldsymbol{q} - \boldsymbol{q}'\|}_{\text{smoothness}}, \tag{3}$$

where the collision term is a straight-line integral with $f' = 1/\|\boldsymbol{q}' - \boldsymbol{q}\|$ between $\boldsymbol{f}(a) = \boldsymbol{q}$ and $\boldsymbol{f}(b) = \boldsymbol{q}'$. Finding the optimal value function on G is straightforward by iterating the Bellman optimality operator

$$\begin{aligned} v_G(\boldsymbol{q}) &\leftarrow \min_{(\boldsymbol{q}, \boldsymbol{q}')} \sum_{\boldsymbol{q}'} t(\boldsymbol{q}' \mid \boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) \left(c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) + v_G(\boldsymbol{q}')\right) \\ &\leftarrow \min_{(\boldsymbol{q}, \boldsymbol{q}')} \left(c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) + v_G(\boldsymbol{q}')\right) \end{aligned} \tag{4}$$

with a finite number of iterations $K = M + 1$. The optimal policy is extracted by tracing the optimal value function

$$\pi^*(\boldsymbol{q}) = \operatorname*{argmin}_{(\boldsymbol{q}, \boldsymbol{q}')} \left(c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) + v_G^*(\boldsymbol{q}')\right), \tag{5}$$

from $\boldsymbol{q}_0$ until $\boldsymbol{q}' \in \mathcal{G}$ [35]. This produces a sequence of edges $\mathcal{P} = \{(\boldsymbol{q}_0, \boldsymbol{q}_1), \ldots, (\boldsymbol{q}_M, \boldsymbol{q}_g) \mid \boldsymbol{q}_g \in \mathcal{G}\}$.

Proposition 1. *By following any policy on $(\mathcal{V}, \mathcal{E}, c, t)$ from $\boldsymbol{q}_0$, $\mathcal{P}$ has a constant cardinality of $M + 1$.*

Proof. By construction of graph G, each application of Eq. (5) increases the layer number m strictly monotonically, since $t(\boldsymbol{q}_{m+1} \mid \boldsymbol{q}_m, \pi(\boldsymbol{q}_m)) = t(\boldsymbol{q}_{m+1} \mid \boldsymbol{q}_m, (\boldsymbol{q}_m, \boldsymbol{q}_{m+1})) = 1$, $(\boldsymbol{q}_m, \boldsymbol{q}_{m+1}) \in \mathcal{E}$. Hence, $|\mathcal{P}| = M + 1$. □

Finding optimal paths by finite VI over a discretization structure has been a common practice and widely applied in different settings [36]. However, to our knowledge, applying VI over a random multipartite graph, enabling batching mechanisms over planning instances, is novel, as we present in the next section.

C. Batching The Planner

In practice, we do not need to construct an explicit graph data structure due to G's multipartite structure. Observing the deterministic state transition and the equal cardinality of layers, we just need to compute and maintain the transition cost matrices $\boldsymbol{C}_s \in \mathbb{R}^N$, $\boldsymbol{C}_h \in \mathbb{R}^{(M-1)\times N\times N}$, $\boldsymbol{C}_l \in \mathbb{R}^{N\times|\mathcal{G}|}$ and value matrices $V_s \in \mathbb{R}$, $\boldsymbol{V}_h \in \mathbb{R}^{M\times N}$, $\boldsymbol{V}_g \in \mathbb{R}^{|\mathcal{G}|}$, where $\boldsymbol{C}_s$ is the transition costs from $\boldsymbol{q}_0$ to the first layer; $\boldsymbol{C}_h, \boldsymbol{C}_l$ hold transition costs between middle layers and last layer to goals; $V_s, \boldsymbol{V}_h, \boldsymbol{V}_g = \boldsymbol{C}_g$ hold values of start, layers costs and terminal goal costs. Given the uniformly-sampled waypoint tensors $\boldsymbol{Q} \in \mathbb{R}^{M\times N\times d}$ and the goals $\boldsymbol{G} \in \mathbb{R}^{|\mathcal{G}|\times d}$, the cost-to-go term of the transition costs Eq. (3) is approximately computed by first probing an H number of equidistant points on all edges, evaluating them in batches, and taking the mean values over the probing dimension. We assume all cost functions are batch-wise computable.

The GTMP algorithm is compactly presented in Algorithm 1. Note that Line 6 is a matrix-reduced `min` operation on the last dimension, while the `sum` is broadcasted to the middle dimension of the cost matrix $\boldsymbol{C}_h \in \mathbb{R}^{(M-1)\times N\times N}$ from the value matrix $\boldsymbol{V}_h \in \mathbb{R}^{M\times 1\times N}$. After $M + 1$ Bellman iterations (Line 5-7), given the converged value matrix $\boldsymbol{V}_h^*$, a sequence of waypoints is traced over the layers to the goals (Line 11-13). Notice that all component matrices can be straightforwardly vectorized by adding the batch dimension B for all matrices, and the whole algorithm can be JAX `vmap` over sampling seeds on line 1. Note that [12]–[14], [22] focus on vectorizing collision checking or forward kinematics in a single planning instance, while we can ensure that Algorithm 1 can be vectorized at the instance-level [7] by Proposition 1.

Complexity Analysis. The Bellman matrix update (Line 5-7) is an asynchronous update in batches (i.e., updates based on values of previous iteration) and also known to converge [37]. Considering the layer number M, waypoint number per layer N, and probing number H, we assume that the Bellman matrix update is executed on P processor units, an estimate of time complexity per VI iteration is $\mathcal{O}(MN^2/P)$ due to the broadcasted `sum` and `min` operator on Line 6 Algorithm 1. Hence, the overall worst-case time complexity is $\mathcal{O}(M^2N^2/P)$, with a fixed number of $M+1$ VI iterations. The collision-checking time complexity is $\mathcal{O}(MN^2H/P)$, and thus, the overall time complexity is $\mathcal{O}(MN^2(H+M)/P)$. The space complexity is $\mathcal{O}(MN^2H)$ due to the collision checking.

Algorithm 1: Global Tensor Motion Planning

Input: Start $\boldsymbol{q}_0$, Goals $\boldsymbol{G} \in \mathbb{R}^{|\mathcal{G}|\times d}$
1 Uniformly sample $\boldsymbol{Q} \in \mathbb{R}^{M\times N\times d}$
2 Compute cost matrices $\boldsymbol{C}_s, \boldsymbol{C}_h, \boldsymbol{C}_l$ as Eq. (3)
3 Init $V_s \in \mathbb{R}$, $\boldsymbol{V}_h \in \mathbb{R}^{M\times N}$, $\boldsymbol{V}_g \in \mathbb{R}^{|\mathcal{G}|}$ `// finite VI`
4 **for** $1 \le k \le M + 1$ **do**
5 $\quad \boldsymbol{V}_h[M-1] \leftarrow \min(\boldsymbol{C}_l + \boldsymbol{V}_g)$
6 $\quad \boldsymbol{V}_h[:M-1] \leftarrow \min(\boldsymbol{C}_h + \boldsymbol{V}_h[1:], \text{axis} = -1)$
7 $\quad V_s \leftarrow \min(\boldsymbol{C}_s + \boldsymbol{V}_h[0])$
8 $i \leftarrow \operatorname{argmin}(\boldsymbol{C}_s + \boldsymbol{V}_h^*[0])$ `// path tracing`
9 $\mathcal{P} = \{i\}$
10 **for** $1 \le m \le M - 1$ **do**
11 $\quad i \leftarrow \operatorname{argmin}(\boldsymbol{C}_h[m-1, i] + \boldsymbol{V}_h^*[m])$
12 $\quad$ Append $\boldsymbol{Q}[m, i]$ to $\mathcal{P}$
13 $i \leftarrow \operatorname{argmin}(\boldsymbol{C}_l[i] + \boldsymbol{V}_g)$ and append $\boldsymbol{G}[i]$ to $\mathcal{P}$
Output: $\mathcal{P}$

D. Theoretical Guarantees

Proposition 2 (Feasibility Check)**.** *For any planning problem, $v_G^*(\boldsymbol{q}_0) < \infty$ if and only if there exists a feasible path in G.*

Proposition 2 follows from the unbounded collision costs Eq. (2) and is useful to filter collided paths after VI. Under uniform sampling per layer and the existence of a feasible path with positive collision margin, we establish:

Theorem 1 (Probabilistic Completeness)**.** *For a feasible planning problem $(\mathcal{C}_{free}, \boldsymbol{q}_0, \mathcal{G})$, with G having $M \geq M_m$ layers (M_m is the minimum number of segments for piecewise linear feasibility), there exist constants $a, R, L > 0$ depending only on $\mathcal{C}_{free}$ and $\mathcal{G}$, such that*

$$\mathbb{P}\left(v_G^*(\boldsymbol{q}_0) < \infty\right) > 1 - M \exp\left(-a\left(R - \frac{L}{M+1}\right)^d N\right). \tag{6}$$

The bound implies a minimum $M > [L/R] - 1$ and an optimal M^*; increasing M helps then harms N-sample efficiency. The full proof with supporting lemmas is provided in Section A.

IV. Experiment Results

We assess the performance of GTMP and its smooth extension on batch planning and single planning capability compared to popular baselines and collision-checking mechanisms. Hence, we investigate the following questions for batch trajectory generation, or for finding the global solution: i) how does GTMP with JAX/GPU-implementation compare to highly optimized probabilistic-complete planners implemented in PyBullet/OMPL [19], [38] or in VAMP [22]?, ii) how does GTMP-Akima compare to popular gradient-based smooth trajectory optimizers such as CHOMP [39] or GPMP [8]?, and iii) Are the empirical results consistent with the theoretical guarantees (Theorem 1)?

Settings. We run all CPU-based planners (RRTC, BKPIECE) on AMD Ryzen 5900X clocked at 3.7GHz and GPU-based planners (GTMP, CHOMP, GPMP, cuRobo) on a single Nvidia RTX 3090. Note that GTMP, CHOMP, and GPMP are implemented in JAX [18], and the planning times are measured after JIT. We use cuRobo's official PyTorch implementation. We initialize CHOMP and GPMP with samples from a high-variance Gaussian process prior [32] connecting from the start to the goals. We set a default probing $H = 10$ and used uniform sampling for all GTMP runs. For all CPU-based planners, we give a timeout of one minute and report metrics after simplification routines. Planning time per task is the sum of all planning instances, which includes simplification time for CPU-based planners, while GTMP does not need path simplification.

Metrics. The metrics are chosen for comparing across probabilistically-complete planners and trajectory optimizers: (i) *Planning Time (s)* in seconds of a batch of paths given a task, (ii) *Collision Free (CF %)* percentage of paths in a batch (failure cases are either in collision or timeout), (iii) *Minimum Cosine Similarities (Min Cosim)* over consecutively path segments and averaging over the batch of paths in a task, (iv) *Paths Diversity (PD)* as the mean of pairwise Sinkhorn [40] distances in a batch having B paths

$$PD = \frac{1}{B(B-1)} \sum_{\substack{i,j \in \{1,\dots,B\} \\ i \neq j}} \mathrm{OT}_\lambda(\mathcal{P}_i, \mathcal{P}_j), \tag{7}$$

where we treat the path $\mathcal{P} = \{\boldsymbol{q}_0, \dots \boldsymbol{q}_T\}$ as empirical distribution with uniform weights, and different paths can have different horizons T. The entropic scalar $\lambda = 5 \times 10^{-3}$ is constant. The metric *Min Cosim* measures the worst/average rough turns over path segments, which represents worst-case jerks since baselines plan different trajectory dynamic orders. The PD measures the spread of solution paths correlating to solutions' modes discovery.

A. Batch Planning Comparison

Fig. 2 (top-row) compares GTMP and GTMP-Akima with OMPL implementation of (single-query) RRTConnect [2] and BKPIECE [41]. The environments are planar occupancy maps of Intel Lab, ACES3 Austin, Orebro, Freiburg Campus, and Seattle UW Campus generated from the Radish dataset [42]. The maps are chosen to include narrow passages, large spaces, and noisy occupancies (cf. Fig. 1). We randomly sample 100 start-goal pairs as tasks on each map and plan 100 paths per task. We observe a comparable Min Cosim (i.e., similar statistics of rough turns) and PD of GTMP (M=200, N=4) compared to baselines across maps and in aggregated statistics over maps. With JIT and GPU utilization, GTMP consistently produces batch paths with a fixed number of segments and $10^4\times$ less wall-clock time compared to baselines across maps.

We choose the MotionBenchMaker (MBM) dataset [43] of 7-DoF Franka Emika Panda tasks such as table-top manipulation (*table pick and table under pick*), reaching (*bookshelf small, tall and thin*), and highly-constrained reaching (*box and cage*). Each task is pre-generated with 100 problems available publicly. We implement our collision-checking in JAX via primitive shape approximation, such as a Panda spherized model, oriented cubes, and cylinders representing tasks in MBM. The default hyperparameters and compilation configurations for VAMP/RRTC, OMPL/RRTC, and OMPL/BKPIECE are also adopted following [22][1]. CHOMP and GPMP plan first-order trajectories having a horizon of $T = 32$. All algorithms are compared on the planning performance of a batch of $B = 50$ paths for all tasks.

[1]We use default *shortcut* simplification for OMPL planners while using default *shortcut and B-spline smoothing* for VAMP/RRTC.

TABLE I
Aggregated Statistics Of MπNets Dataset

Algorithms	PT ↓ (ms)	Success ↑ (%)	Path Length ↓	Min Cosim ↑	PD ↑
GTMP (N=30, M=2)	0.11	99.6	4.8	−0.3	7.7
GTMP-Akima (N=30, M=2)	0.10	97.1	7.3	−0.1	**7.8**
VAMP/RRTC [22]	**0.09**	**100.0**	3.6	**0.1**	-
cuRobo [11]	43.1	99.7	**2.8**	0.0	-

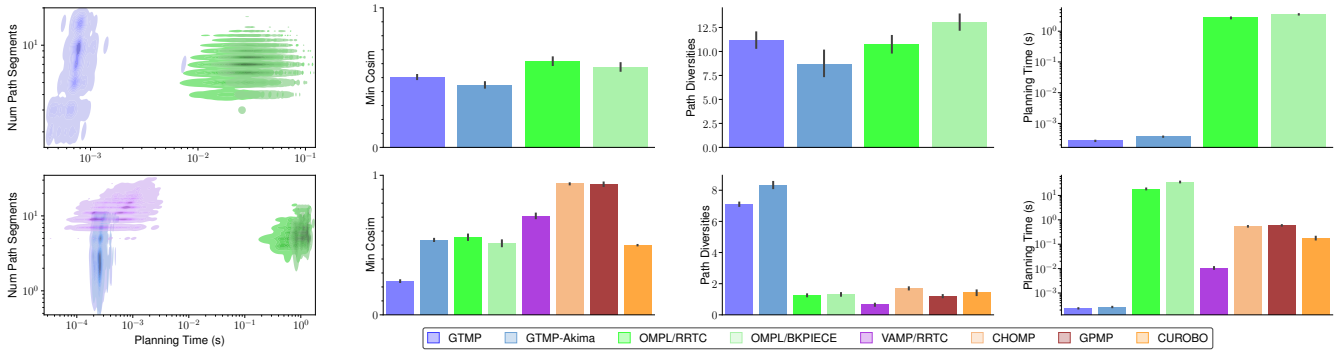

Fig. 2. Aggregated statistics of comparison experiments on Planar Occupancy (top-row) and Panda MBM dataset (bottom-row). We note the log scale on the Planning Time axes. The batch planning time is the sum of instance time for sequential planners (last column). All plotted data points are based on successful path statistics.

Fig. 2 (bottom-row) shows the planning performance comparisons between GTMP (N=30, M=2) and baseline probabilistically-complete planners and gradient-based trajectory optimizers. We see that GTMP consistently has the best diversity (PD) and worst rough turn statistics (Min Cosim) in all tasks. This is due to the maximum exploration behavior of GTMP by sampling uniformly over configuration space, which increases the risk of rough paths. In principle, increasing points per layer N while having minimum solving layers M would improve Min Cosim due to having more chances to discover smoother paths with fewer segments, as long as GPU memory allows (cf. Fig. 3). Compared with gradient-based optimizers, GTMP-Akima with spline discretization construction has a similar Min Cosim to cuRobo while not requiring gradients from the planning costs. Note that cuRobo additionally considers dynamical constraints, which increases planning time but improves metrics such as maximum model jerk. On batch planning efficiency, GTMP and GTMP-Akima achieve $50\times$ faster than state-of-the-art VAMP/RRTC implementation while being $2500\times$ faster than CHOMP/GPMP/cuRobo and $10^5\times$ faster than the OMPL implementation with PyBullet collision checking. We leave the investigation of combining GTMP with the VAMP collision checking for future work.

Fig. 2 (first-column) shows the distributions of single-instance planning time versus number of path segments, reflecting inherent algorithmic differences between GTMP and RRTC implementations. RRTC blobs are spread due to differences in randomized graph explorations between planning instances and are separated due to differences in collision-checking efficiency [22]. GTMP vectorizes planning via layered structure, resulting in predictable narrow distribution due to fixed-segment path planning.

B. Single Plan Comparison

We compare GTMP and GTMP-Akima to the strong baselines such as VAMP/RRTC [22] and cuRobo [11], in terms of single planning for execution, on the MπNets dataset [44] of diverse 7-DoF Franka Emika Panda tasks. We set $B = 50$ for GTMP/GTMP-Akima and select the lowest path length for execution. We plan a single instance for VAMP/RRTC and cuRobo. Table I shows that GTMP achieves a similar success rate to the baselines (i.e., at least one successful path in the batch) while having similar planning time to the state-of-the-art VAMP/RRTC. However, due to the maximum exploration nature, GTMP performs worse regarding path quality. Future works on better sampling strategy per layer could improve GTMP path quality while increasing the sample efficiency on M, N for low-memory planning.

C. Ablation Study

This section explores various aspects of GTMP by sweeping the number of layer M and number of points per layer N. Fig. 3 shows the sweeping statistics of $M \in \{2, 3, \ldots, 80\}, N \in \{10, 11, \ldots, 100\}$ on the Intel Lab occupancy map with a fixed start-goal pair to experimentally confirm the probabilistic completeness Theorem 1. In Fig. 3, Planning Time heatmap shows an experimentally infinitesimal increase in polynomial planning time-complexity over increasing M, N (due to JIT-ing finite VI loops and efficient batch collision-checking, cf. Section III). Then, the CF(%) heatmap directly reflects the path existence probability Eq. (6). Notice that the minimum layer $M_m = 3$ must be set for collision-free paths in the batch. Interestingly, M_m is also the optimal number of layers to achieve non-zero CF(%) with a minimal point per layer N (red star), which confirms the observation in Section III-D. Next, further observations on Min Cosim also confirm that with less M, the paths are smoother. Finally, higher path diversity is induced by having higher CF(%), corresponding to the top-right heatmap.

V. Discussion & Conclusions

GTMP offers several advantages algorithmically, as it is *vectorizable* over a large number of planning instances, it does not require *joint-limit enforcement* (i.e., sampling points in the limits), *gradients* or *simplification routines*. On the practical side, GTMP is *easy to implement* (i.e., only tensor manipulation), *easy to tune* (i.e., hyperparameter M, N, H), and *easy to incorporate* motion planning objectives in Eq. (14).

GTMP is designed to be efficient in batch planning representing multiple instances of the same planning problem. The

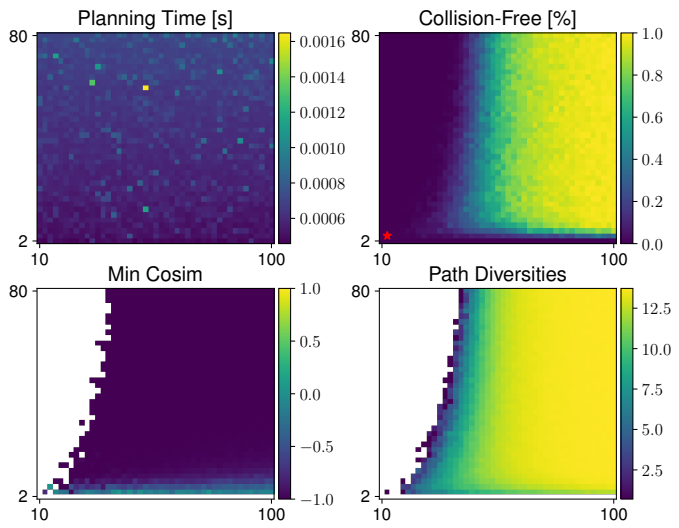

Fig. 3. For each M (y-axis), N (x-axis), we set the number of probing $H = 30$ and plan the batch of $B = 200$ paths. The red star denotes the minimum number of layers M_m, corresponding to the minimum requirement of N to discover some solutions experimentally.

batch dimension representing the multiple GTMP planning instances can be interpreted as multiple replanning attempts. Indeed, Theorem 1 depicts probabilistic completeness over the batch dimension and M, N, contrasting with probabilistic completeness over exploration nodes as in RRT* [45]. Beyond GTMP, since Algorithm 1 is cheap in common case, we could also derive an outer loop gradually increasing M, N until some solutions in the batch are found.

GTMP addresses global exploration challenges but comes with memory requirements, especially for GPU acceleration. In contrast, local methods such as CHOMP or GPMP leverage gradient-based, more memory-efficient trajectory optimization. GTMP-Akima, for instance, avoids the need for gradients while delivering smooth velocity trajectories by a *spline discretization structure*, making it a viable initialization for methods like GPMP, potentially combining the strengths of both approaches.

Variants of GTMP emphasize maximum exploration while maintaining smooth trajectory structures. Exploring further smooth discretization structures for higher-order planning is exciting, as the current Akima discretization structure only provides a C^1 spline grid. Furthermore, we are eager to adopt the efficient collision-checking of VAMP [22] for GTMP, when the VAMP batching configuration collision-checking becomes available, extending GTMP to CPU-based vectorization. Lastly, GTMP suggests the direction of probabilistically-complete batch planners, serving as a differentiable global planner or a competent oracle for learning.

APPENDIX A
PROOFS OF THEORETICAL RESULTS

Notation. Let $\mathcal{R}$ be the set of all paths in G. The path cost is $c(g) = \sum_{m=0}^{M} c(\boldsymbol{q}_m, \boldsymbol{q}_{m+1}) + c_g(\boldsymbol{g}(1))$, $g \in \mathcal{R}$.

Assumption 1. *We assume that all associated proposal distributions at each layer are uniformly distributed on the configuration space* $\forall 1 \le m \le M$, $p_m := \mathcal{U}(\mathcal{C})$.

Assumption 2. *Consider a feasible planning problem, there exists a feasible path* $f : [0,1] \to \mathcal{C}_{free}$ *having margin* $R = \inf_{t\in[0,1]} \|\boldsymbol{f}(t) - \boldsymbol{q}\|, \boldsymbol{q} \in \mathcal{C}_{coll}$, *such that* $R > 0$.

These assumptions are common in path planning, where the free-path set is not zero-measure $\mu(\mathcal{C}_{\text{free}}) \neq 0$.

Proof of Proposition 2. By Bellman optimality, $v_G^*(\boldsymbol{q}_0) = \min_g\{c(g) \mid g \in \mathcal{R}\}$ is the minimum path cost. The smoothness term in Eq. (3) is bounded since $\text{TV}(g) < \infty$. Thus, $c(\mathcal{P}) = \infty$ iff $\exists t$, $\boldsymbol{f}(t) \in \mathcal{C}_{\text{coll}}$. Hence, $v_G^*(\boldsymbol{q}_0) < \infty$ iff $\exists \mathcal{P} \in \mathcal{R}$, $c(\mathcal{P}) < \infty$. □

Lemma 1 (Solvability In Finite Path Segments)**.** *If Assumption 2 holds, there exists a minimum number of segments* $M_m \in \mathbb{N}_{>0}$ *for piecewise linear paths to be feasible.*

Proof. We construct a piecewise linear $g : [0,1] \to \mathcal{C}$ with $\|f - g\|_\infty < R$ by dividing $[0,1]$ into M subintervals of length at most $\delta > 0$. On each $[t_m, t_{m+1}]$, define

$$\boldsymbol{g}(t) = \boldsymbol{f}(t_m) + \frac{\boldsymbol{f}(t_{m+1}) - \boldsymbol{f}(t_m)}{t_{m+1} - t_m}(t - t_m). \tag{8}$$

Since f is continuous on compact $[0,1]$, by Heine-Cantor, f is uniformly continuous: $\exists \delta > 0$ s.t. $|a - b| < \delta \Rightarrow \|\boldsymbol{f}(a) - \boldsymbol{f}(b)\|_\infty < R$. Choosing δ sufficiently small ensures $\|f - g\|_\infty < R$, so $\exists M_m$ such that g is feasible. □

Lemma 2. *Let piecewise linear path* $g : [0,1] \to \mathcal{C}$ *having* n *equal subintervals approximating* $f : [0,1] \to \mathcal{C}$. *The error upper bound is* $\|f - g\|_\infty \le L/n$, *where* $L = TV(f)$.

Proof. Using subinterval length $u = 1/n$, the interpolation error on a segment $t \in [t_m, t_{m+1}]$ satisfies

$$\|\boldsymbol{f}(t) - \boldsymbol{g}(t)\| \le \omega_f(u), \omega_f(u) = \sup_{|a-b|\le u} \|\boldsymbol{f}(a) - \boldsymbol{f}(b)\|.$$

The global error is $\|f - g\|_\infty = \max_{0\le m\le n-1} \sup_{t\in[t_m,t_{m+1}]} \|\boldsymbol{f}(t) - \boldsymbol{g}(t)\|$. For arc-length parameterized paths with bounded variation, $\omega_f(1/n) \le L/n$ on $[0,1]$. Hence, $\|f - g\|_\infty \le L/n$. □

Lemma 3. *Let* g_1, g_2 *be piecewise linear with the same partition points* $\{t_m\}_{m=0}^M$, $\|g_1 - g_2\|_\infty < \delta$ *iff* $\|\boldsymbol{g}_1(t_m) - \boldsymbol{g}_2(t_m)\| < \delta$, $0 \le m \le M$.

Proof. **Sufficiency.** Since g_1, g_2 are piecewise linear, interpolation between partition points ensures the difference is maximized at the partition points:

$$\begin{aligned}\|\boldsymbol{g}_1(t) - \boldsymbol{g}_2(t)\| \le &\max\{\|\boldsymbol{g}_1(t_m) - \boldsymbol{g}_2(t_m)\|, \\ &\|\boldsymbol{g}_1(t_{m+1}) - \boldsymbol{g}_2(t_{m+1})\|\} < \delta\end{aligned}$$

Hence, $\|g_1 - g_2\|_\infty < \delta$. **Necessity** is immediate from evaluation at t_m. □

Proof of Theorem 1. From Lemma 1, if $M \geq M_m$, there exists a feasible piecewise linear path g with $M+1$ equal segments approximating f. Let $R = \inf_t\{\|\boldsymbol{f}(t) - \boldsymbol{q}\|, \boldsymbol{q} \in \mathcal{C}_{\text{coll}}\}$, $r = \inf_t\{\|\boldsymbol{g}(t) - \boldsymbol{q}\|, \boldsymbol{q} \in \mathcal{C}_{\text{coll}}\}$ be collision margins, and $\mathcal{B}_\delta(\boldsymbol{q})$ an open δ-ball. We bound the margin r:

$$\begin{aligned} r &\geq \inf_{t\in[0,1]} \{\|\boldsymbol{f}(t) - \boldsymbol{q}\|, \boldsymbol{q} \in \mathcal{C}_{\text{coll}}\} - \sup_{t\in[0,1]} \|\boldsymbol{g}(t) - \boldsymbol{f}(t)\| \\ &\geq R - \frac{L}{M+1}, \end{aligned}$$

where the last inequality follows from Lemma 2 with $L = \text{TV}(f)$. Let $r_h = R - \frac{L}{M+1}$. From Lemma 3, the event $\|h - g\|_\infty < r_h$ for a sampled path h in G requires that for each layer $1 \leq m \leq M$, at least one point is sampled inside $\mathcal{B}_{r_h}(\boldsymbol{g}(t_m))$. Sampling N points uniformly over $\mathcal{C}$ per layer (Assumption 1), the failing probability is

$$\begin{aligned} \mathbb{P}(\|h - g\|_\infty \geq r_h) &\leq \sum_{m=1}^{M} \left(1 - \frac{\mu(\mathcal{B}_{r_h}(\boldsymbol{g}(t_m)))}{\mu(\mathcal{C})}\right)^N \\ &\leq M \exp\left(-\frac{\alpha_d}{\mu(\mathcal{C})}\left(R - \frac{L}{M+1}\right)^d N\right) \end{aligned}$$

using $1 - x \leq e^{-x}$ and $a = \alpha_d/\mu(\mathcal{C})$, with α_d the d-ball volume constant. Since the event $\exists h$ in G with $\|h - g\|_\infty < r_h$ is a subset of the event $\exists$ a feasible path in G, and by Proposition 2:

$$\begin{aligned} \mathbb{P}(v_G^*(\boldsymbol{q}_0) < \infty) &\geq \mathbb{P}(\|h - g\|_\infty < r_h) \\ &> 1 - M \exp\left(-a\left(R - \frac{L}{M+1}\right)^d N\right). \end{aligned}$$

□

APPENDIX B
AKIMA SPLINE EXTENSION

The Akima spline [34] is a piecewise cubic interpolation method that exhibits C^1 smoothness by using local points to construct the spline, avoiding oscillations or overshooting in other interpolation methods, such as cubic splines or B-splines.

Definition 4 (Akima Spline)**.** *Given a point set $\{\boldsymbol{q}_i \in \mathcal{C}\}_{i=1}^P$, the Akima spline constructs a piecewise cubic polynomial $f(t)$ for each interval $[t_i, t_{i+1}]$*

$$f_i(t) = \boldsymbol{d}_i(t - t_i)^3 + \boldsymbol{c}_i(t - t_i)^2 + \boldsymbol{b}_i(t - t_i) + \boldsymbol{a}_i, \tag{9}$$

where the coefficients $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{d}_i \in \mathcal{C}$ are determined from the conditions of smoothness and interpolation. Let $\boldsymbol{m}_i = (\boldsymbol{q}_{i+1} - \boldsymbol{q}_i)/(t_{i+1} - t_i)$ at t_i, the spline slope is computed from m_{i-1}, m_{i+1}

$$\boldsymbol{s}_i = \frac{|\boldsymbol{m}_{i+1} - \boldsymbol{m}_i|\boldsymbol{m}_{i-1} + |\boldsymbol{m}_{i-1} - \boldsymbol{m}_{i-2}|\boldsymbol{m}_i}{|\boldsymbol{m}_{i+1} - \boldsymbol{m}_i| + |\boldsymbol{m}_{i-1} - \boldsymbol{m}_{i-2}|}. \tag{10}$$

The spline slopes for the first two points at both ends are $\boldsymbol{s}_1 = \boldsymbol{m}_1, \boldsymbol{s}_2 = (\boldsymbol{m}_1 + \boldsymbol{m}_2)/2, \boldsymbol{s}_{P-1} = (\boldsymbol{m}_{P-1} + \boldsymbol{m}_{P-2})/2, \boldsymbol{s}_P = \boldsymbol{m}_{P-1}$. Then, the polynomial coefficients are uniquely defined

$$\begin{aligned} &\boldsymbol{a}_i = \boldsymbol{q}_i, \quad \boldsymbol{b}_i = \boldsymbol{s}_i, \\ &\boldsymbol{c}_i = (3\boldsymbol{m}_i - 2\boldsymbol{s}_i - \boldsymbol{s}_{i+1})/(t_{i+1} - t_i), \\ &\boldsymbol{d}_i = (\boldsymbol{s}_i + \boldsymbol{s}_{i+1} - 2\boldsymbol{m}_i)/(t_{i+1} - t_i)^2. \end{aligned} \tag{11}$$

The Akima spline slope is determined by the local behavior of the data points, preventing oscillations that can occur when using global information. Interpolating with Akima spline does not require solving large systems of linear equations, making it computationally efficient as an ideal extension to Definition 2 to *a spline discretization structure*.

Definition 5 (Akima Spline Graph)**.** *Given a geometric graph $G = (\mathcal{V}, \mathcal{E})$ (cf. Definition 2), the Akima Spline graph G_A has the edge set $\mathcal{E}$ geometrically augmented by cubic polynomials. In particular, consider an edge $(\boldsymbol{q}_{m,i}, \boldsymbol{q}_{m+1,j}) \in \mathcal{E}$ with i, j are respective indices of points at layers $\mathcal{L}_m, \mathcal{L}_{m+1}$, the spline slope is defined with $\boldsymbol{m}_{m,i,j} = (\boldsymbol{q}_{m+1,j} - \boldsymbol{q}_{m,i})/(t_{m+1} - t_m)$ as Modified Akima interpolation [34]*

$$\begin{aligned} \boldsymbol{s}_{m,i,j} &= \frac{\boldsymbol{w}_{m,i,j}\boldsymbol{m}_{m-1,i,j} + \boldsymbol{w}_{m-1,i,j}\boldsymbol{m}_{m,i,j}}{\boldsymbol{w}_{m,i,j} + \boldsymbol{w}_{m-1,i,j}} \\ \boldsymbol{w}_{m,i,j} &= \left|\frac{1}{N^2}\sum_{i,j} \boldsymbol{m}_{m+1,i,j} - \boldsymbol{m}_{m,i,j}\right| \\ &\quad + \frac{1}{2}\left|\frac{1}{N^2}\sum_{i,j} \boldsymbol{m}_{m+1,i,j} + \boldsymbol{m}_{m,i,j}\right| \\ \boldsymbol{w}_{m-1,i,j} &= \left|\boldsymbol{m}_{m-1,i,j} - \frac{1}{N^2}\sum_{i,j} \boldsymbol{m}_{m-2,i,j}\right| \\ &\quad + \frac{1}{2}\left|\boldsymbol{m}_{m-1,i,j} + \frac{1}{N^2}\sum_{i,j} \boldsymbol{m}_{m-2,i,j}\right|. \end{aligned} \tag{12}$$

Then, the augmented cubic polynomial $f_{i,j}(t)$, $t \in [t_m, t_{m+1}]$ is computed following Eq. (11)

$$\begin{aligned} &\boldsymbol{s}_m = \frac{1}{N^2}\sum_{i,j} \boldsymbol{s}_{m,i,j},\ \boldsymbol{a}_{m,i,j} = \boldsymbol{q}_{m,i},\ \boldsymbol{b}_{m,i,j} = \boldsymbol{s}_m, \\ &\boldsymbol{c}_{m,i,j} = (3\boldsymbol{m}_{m,i,j} - 2\boldsymbol{s}_m - \boldsymbol{s}_{m+1})/(t_{m+1} - t_m), \\ &\boldsymbol{d}_{m,i,j} = (\boldsymbol{s}_m + \boldsymbol{s}_{m+1} - 2\boldsymbol{m}_{m,i,j})/(t_{m+1} - t_m)^2. \end{aligned} \tag{13}$$

The original Akima interpolation computes equal weight to the points on both sides, evenly dividing an undulation. When two flat regions with different slopes meet, this modified Akima interpolation [34] gives more weight to the side where the slope is closer to zero, thus giving priority to the side that is closer to horizontal, which avoids overshoot. Notice that after pre-computing $\boldsymbol{m}_{m,i,j}$ for every edge in G_A, every polynomial segment Eq. (13) can be computed in batch for G_A. Furthermore, given a batch of graphs G_A, adding a batch dimension for these equations is straightforward. The transition cost is then defined

$$c(\boldsymbol{q}, \boldsymbol{q}') = \int_a^b (c_{\text{coll}}(f(t)) + 1)\,\|f'(t)\|\,dt, \tag{14}$$

where $f(t)$ is the cubic polynomial representing the edge $(\boldsymbol{q}, \boldsymbol{q}') \in G_A$.

Remark 1. *With some algebra derivations, one can verify the cubic polynomial $f_{i,j}(t)$, $t \in [t_m, t_{m+1}]$ representing any edge $(\boldsymbol{q}_{m,i}, \boldsymbol{q}_{m+1,j}) \in G_A$ satisfying four conditions of continuity*

$$\begin{aligned} \boldsymbol{f}_{i,j}(t_m) &= \boldsymbol{q}_{m,i},\ \boldsymbol{f}_{i,j}(t_{m+1}) = \boldsymbol{q}_{m+1,j}, \\ \boldsymbol{f}'_{i,j}(t_m) &= \boldsymbol{s}_m,\ \boldsymbol{f}'_{i,j}(t_{m+1}) = \boldsymbol{s}_{m+1}, \end{aligned} \tag{15}$$

for any $m \in \{0, \dots, M+1\}$, $i, j \in \{1, \dots, N\}$. Hence, any path $f \in G_A$ is an Akima spline.

The Akima spline provides C^1-continuity for first-order planning; however, the second derivative is not necessarily continuous. Note that Theorem 1 does not necessarily hold for Akima Spline Graph G_A and is left for future work.

ACKNOWLEDGMENT

An T. Le was funded by the German Research Foundation project METRIC4IMITATION (PE 2315/11-1). Kay Pompetzki received funding from the German Research Foundation project CHIRON (PE 2315/8-1).

REFERENCES

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 1996.

[2] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE ICRA*, 2000.

[3] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[4] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *IEEE/RSJ IROS*, 2023.

[5] J. Wang, T. Zhang, N. Ma, Z. Li, H. Ma, F. Meng, and M. Q.-H. Meng, "A survey of learning-based robot motion planning," *IET Cyber-Systems and Robotics*, vol. 3, no. 4, pp. 302–314, 2021.

[6] M. Reuss, M. Li, X. Jia, and R. Lioutikov, "Goal conditioned imitation learning using score-based diffusion policies," in *R:SS*, 2023.

[7] A. T. Le, G. Chalvatzaki, A. Biess, and J. R. Peters, "Accelerating motion planning via optimal transport," *NeurIPS*, vol. 36, 2024.

[8] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *IJRR*, 2018.

[9] T. Osa, "Multimodal trajectory optimization for motion planning," *IJRR*, 2020.

[10] 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 *CoRL*. PMLR, 2022.

[11] 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 *IEEE ICRA*, 2023.

[12] J. Pan and D. Manocha, "Gpu-based parallel collision detection for fast motion planning," *IJRR*, 2012.

[13] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the rrt and the rrt," in *IEEE/RSJ IROS*, 2011.

[14] J. Blankenburg, R. Kelley, D. Feil-Seifer, R. Wu, L. Barford, and F. C. Harris, "Towards gpu-accelerated prm for autonomous navigation," in *ITNG*. Springer, 2020.

[15] 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 ICRA*, 2012.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[17] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.

[18] J. Bradbury *et al.*, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[19] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, 2012.

[20] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *IEEE ICRA*, 1999.

[21] E. Plaku *et al.*, "Sampling-based roadmap of trees for parallel motion planning," *IEEE TRO*, 2005.

[22] W. Thomason, Z. Kingston, and L. E. Kavraki, "Motions in microseconds via vectorized sampling-based planning," in *IEEE ICRA*, 2024.

[23] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *IJRR*, 2020.

[24] 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," *IJRR*, 2015.

[25] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in *IEEE ICRA*, 2020.

[26] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural rrt*: Learning-based optimal path planning," *IEEE Transactions on Automation Science and Engineering*, 2020.

[27] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4274–4289, 2021.

[28] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *IEEE ICRA*, 2018.

[29] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, 2023.

[30] V. K. Adajania, A. Sharma, A. Gupta, H. Masnavi, K. M. Krishna, and A. K. Singh, "Multi-modal model predictive control through batch non-holonomic trajectory optimization: Application to highway driving," *IEEE RA-L*, 2022.

[31] A. Lambert, A. Fishman, D. Fox, B. Boots, and F. Ramos, "Stein variational model predictive control," *arXiv:2011.07641*, 2020.

[32] J. Urain, A. T. Le, A. Lambert, G. Chalvatzaki, B. Boots, and J. Peters, "Learning implicit priors for motion optimization," in *IEEE/RSJ IROS*, 2022.

[33] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[34] H. Akima, "A method of bivariate interpolation and smooth surface fitting based on local procedures," *Communications of the ACM*, vol. 17, no. 1, pp. 18–20, 1974.

[35] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[36] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 4.

[37] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[38] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[39] M. Zucker *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *IJRR*, 2013.

[40] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," *NeurIPS*, 2013.

[41] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *WAFR*. Springer, 2009.

[42] A. Howard, N. Roy, C. Stachniss, G. Grisetti, D. Haehnel, H. Andreasson, P. Larsson, T. Duckett, and P. Beeson, "The robotics data set repository (radish)," 2003. [Online]. Available: http://radish.sourceforge.net/

[43] C. Chamzas, C. Quintero-Pena, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "Motionbenchmaker: A tool to generate and benchmark motion planning datasets," *IEEE RA-L*, 2021.

[44] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox, "Motion policy networks," in *CoRL*. PMLR, 2023, pp. 967–977.

[45] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, 2011.