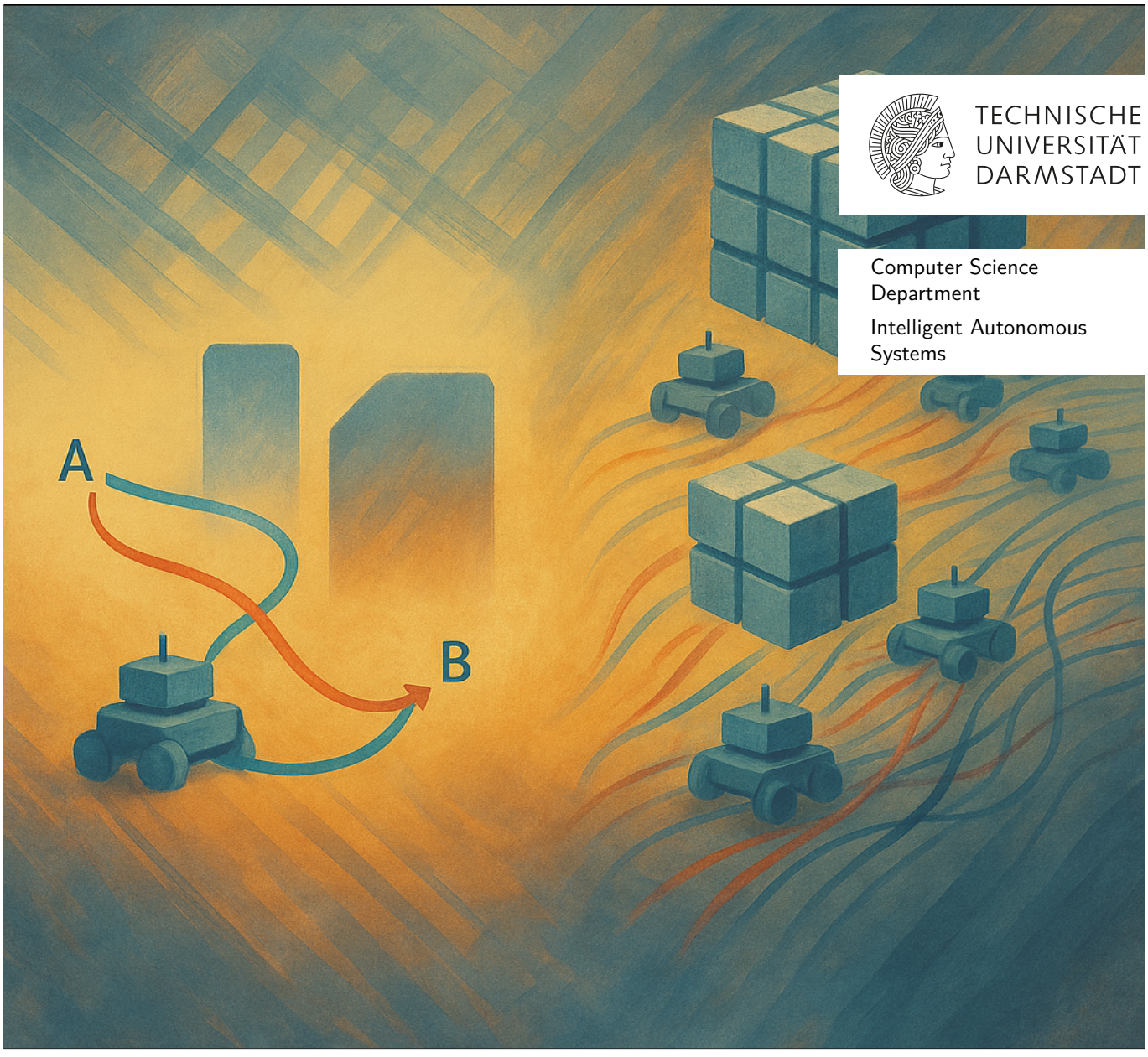
Tensor Search Methods For Vectorizing Motion Planning

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von An T. Le
Tag der Einreichung: 29.08.2025, Tag der Prüfung: 17.07.2025

1. Gutachten: Prof. Jan Peters, Ph.D.
2. Gutachten: Prof. Siddhartha Srinivasa, Ph.D.
Darmstadt, Technische Universität Darmstadt



TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Intelligent Autonomous Systems

Tensor Search Methods For Vectorizing Motion Planning

Accepted doctoral thesis by An T. Le

Date of submission: 29.08.2025
Date of thesis defense: 17.07.2025

Darmstadt, Technische Universität Darmstadt

Dành cho gia đình tôi.

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 29.08.2025

An T. Le

Abstract

Motion planning is one of the most important cornerstones in robotics. Through decades of algorithmic innovation, most classical methods remain inherently sequential and computationally expensive. This limitation is especially pronounced in high-dimensional or time-critical scenarios, where a robot must generate or evaluate many planning candidates in parallel—across different environments, task configurations, or dynamic model perturbations. While recent advances in hardware accelerators (e.g., GPUs, TPUs) and vectorized machine learning frameworks like JAX and PyTorch have revolutionized batched computation for learning and simulation, the planning community has yet to fully utilize these benefits. This thesis addresses this gap by introducing a new class of *tensor search methods* that reformulate motion planning as a series of fixed-shape tensor operations, thereby enabling full pipeline vectorization across multiple planning instances.

The first contribution of this thesis is **Motion Planning via Optimal Transport (MPOT)**, a local trajectory optimization method built on zero-order updates via entropic optimal transport. By constructing a randomly rotated polytope around each trajectory waypoint and solving for the optimal barycentric projection using the Sinkhorn algorithm, MPOT produces vectorized trajectory updates without relying on gradients. Notably, the whole MPOT pipeline consists only of matrix multiplications. This formulation supports simultaneous optimization of hundreds of trajectories in parallel, enabling fast multi-trajectory refinement.

The second contribution is **Global Tensor Motion Planning (GTMP)**, a global planner that leverages a multipartite graph representation to enable structured path sampling and batched graph search. Rather than constructing a search tree, GTMP samples a layered, complete graph with fixed topology and uses dynamic programming to solve batched shortest-path problems via tensor operations. The approach allows simultaneous planning over batches of start-goal pairs and randomized environments. A spline extension produces smooth paths via interpolation, and theoretical guarantees ensure probabilistic completeness under graph growth. GTMP demonstrates strong empirical performance compared to classical and GPU-based baselines, while maintaining compatibility with JAX's `vmap` and `jit` compilation for efficient execution on modern GPUs/TPUs.

The third contribution is **Model Tensor Planning (MTP)**, a fully vectorized framework

for sampling-based Model Predictive Control (MPC) under online domain randomization. MTP introduces high-entropy control trajectory generation through structured tensor sampling. By sampling over randomized multipartite graphs and interpolating control trajectories with B-splines and Akima splines, MTP ensures smooth and globally diverse control candidates. To tame the noise of exploration, MTP utilizes a β-mixing strategy that blends local and global samples within the modified Cross-Entropy Method (CEM) update, balancing control refinement and exploration. MTP supports robust sim-to-real transfer by solving multiple perturbed dynamics models in parallel at runtime and outperforms strong baselines on manipulation and locomotion tasks.

Together, these contributions suggest a general framework for tensorized planning that aligns with the core design principles of modern accelerator-based computing: static tensor shapes and batched execution, which further comes with differentiability inherited from the machine learning libraries implying differentiable planning. This thesis demonstrates that by treating motion planning as tensor computation, we can unlock scalable, high-throughput solutions that not only match but often exceed the performance of traditional methods—while opening the door to new applications in synthetic data generation, policy learning, and real-time sim-to-real transfer.

Zusammenfassung

Die Bewegungsplanung ist eines der wichtigsten Fundamente der Robotik. Trotz jahrzehntelanger algorithmischer Innovationen bleiben die meisten klassischen Methoden inhärent sequenziell und rechnerisch aufwendig. Diese Einschränkung wird besonders in hochdimensionalen oder zeitkritischen Szenarien deutlich, in denen ein Roboter viele Planungskandidaten parallel generieren oder bewerten muss – über unterschiedliche Umgebungen, Aufgabenstellungen oder dynamische Modellvarianten hinweg. Während aktuelle Fortschritte bei Hardwarebeschleunigern (z. B. GPUs, TPUs) und vektorisierten Machine-Learning-Frameworks wie JAX und PyTorch die batched Verarbeitung für Lernen und Simulation revolutioniert haben, werden diese Vorteile in der Planungs-Community bislang kaum ausgeschöpft. Diese Dissertation schließt diese Lücke durch die Einführung einer neuen Klasse von *Tensor-Suchmethoden*, welche Bewegungsplanung als eine Abfolge fester Tensoroperationen reformulieren und dadurch eine vollständige Vektorisierung der gesamten Pipeline über viele Planungsinstanzen hinweg ermöglichen.

Der erste Beitrag dieser Arbeit ist **Motion Planning via Optimal Transport (MPOT)**, eine Methode zur lokalen Trajektorienoptimierung, die auf Nullordnungs-Updates mittels entropischem Optimaltransport basiert. Durch die Konstruktion eines zufällig rotierten Polytops um jeden Trajektorienpunkt und die Berechnung der optimalen baryzentrischen Projektion mittels des Sinkhorn-Algorithmus erzeugt MPOT vektorisierte Trajektorienupdates ohne Gradienten. Bemerkenswert ist, dass die gesamte MPOT-Pipeline ausschließlich aus Matrixmultiplikationen besteht. Diese Formulierung erlaubt die gleichzeitige Optimierung von Hunderten Trajektorien in Parallelität und ermöglicht eine schnelle Multi-Trajektorien-Verfeinerung.

Der zweite Beitrag ist **Global Tensor Motion Planning (GTMP)**, ein globaler Planer, der eine multipartite Graphstruktur nutzt, um strukturierte Pfadabtastung und batched Graphsuche zu ermöglichen. Anstelle eines Suchbaums erzeugt GTMP einen geschichteten, vollständigen Graphen mit fixer Topologie und verwendet dynamische Programmierung, um batched Kürzeste-Wege-Probleme mittels Tensoroperationen zu lösen. Dieser Ansatz ermöglicht gleichzeitige Planung über viele Start-Ziel-Paare und randomisierte Umgebungen hinweg. Eine Spline-Erweiterung erzeugt glatte Pfade durch Interpolation, und theoretische Garantien sichern probabilistische Vollständigkeit bei wachsender Graphgröße. GTMP zeigt eine starke empirische Leistung im Vergleich zu klassischen und GPU-basierten Methoden und ist vollständig kompatibel mit der `vmap`- und `jit`-Kompilation von JAX zur effizienten Ausführung auf modernen GPUs/TPUs.

Der dritte Beitrag ist **Model Tensor Planning (MTP)**, ein vollständig vektorisierter Ansatz zur sampling-basierten Model Predictive Control (MPC) unter Online-Domain-Randomisierung. MTP ermöglicht die Erzeugung hochentropischer Steuertrajektorien durch strukturiertes Tensorsampling. Durch Sampling über randomisierte multipartite Graphen und Interpolation von Steuertrajektorien mittels B-Splines und Akima-Splines stellt MTP glatte und global diverse Steuerungskandidaten sicher. Zur Kontrolle des Explorationsrauschens wird eine β-Mischstrategie verwendet, die lokale und globale Samples im modifizierten Cross-Entropy-Methoden-Update kombiniert und so zwischen Verfeinerung und Exploration ausbalanciert. MTP unterstützt robuste Sim-to-Real-Übertragung, indem es zur Laufzeit viele gestörte Dynamikmodelle parallel löst, und übertrifft starke Baselines in Manipulations- und Lokomotionsaufgaben.

Insgesamt schlagen diese Beiträge einen allgemeinen Rahmen für tensorisierte Planung vor, der mit den Grundprinzipien moderner beschleunigter Berechnung übereinstimmt: feste Tensorformen, batched Ausführung und – durch die ML-Bibliotheken – Differenzierbarkeit, was zukünftig differenzierbare Planung ermöglicht. Diese Arbeit zeigt, dass durch die Behandlung von Bewegungsplanung als Tensorberechnung skalierbare Hochdurchsatzlösungen möglich sind, die klassische Methoden nicht nur erreichen, sondern oft übertreffen – und zugleich neue Anwendungen in der synthetischen Datengenerierung, im Policy Learning und in der Echtzeit-Sim-to-Real-Übertragung eröffnen.

Acknowledgments

First and foremost, I owe a tremendous amount of gratitude to *Jan*, for trusting me with wild ideas, tolerating my many detours, and always reminding me to aim higher—even when I was foolishly tinkering with motion planning problems that had supposedly been "solved" thirty years ago. His curiosity, candor, and life lessons set the gold standard for what research should feel like: a serious pursuit with space for serious mischief. I am deeply thankful to *Sidd*, who immediately and generously agreed to be on my Ph.D. committee. I am also grateful for *Stefan Roth*, *Rolf Findeisen*, and *Felix Wolf* for their work in the committee. I have no other words than a sincere and simple: thank you.

To my labmates at the IAS: thank you for being the quirky, brilliant, and unreasonably caffeinated crew that made every day in the lab both intellectually stimulating and delightfully chaotic. To *Theo, Daniel, Aryaman, Firas, and Tim*, I will never forget the stunts and havoc we've unleashed together through many trips. To *Kay, Joao, and Niklas*, I salute the relentless "pushes" that led to Permanent Head Damage (Ph.D. indeed). To *Junning and Puze*, my culinary comrades who shared the never-ending quest to escape the tyranny of Mensa. And to everyone else at IAS, you guys are indeed IAS that forever imprinted in my memories. Special thanks to my collaborators and fellow wanderers through the land of tensors: *Georgia*, for endless discussions that turned confusion into clarity; *Julen, Joe, and Pascal*, for together seeing structure in the madness; *Ali*, for wildest discussions in the quantum worlds; *Armin Biess*, for his precise thinking and infinite patience; *Sasha Lambert*, for his sharp insights and even sharper feedback; and *Ho Minh Duy Nguyen*, for being a kindred spirit in code and curry. I cannot name all the collaborators and friends at PEARL, LiteRL, and other labs: your ideas, code reviews, comments, and occasional memes kept this thesis alive.

To my family, who quietly carried my worries so I could carry on with this work—your unwavering support means more than words can express. And to my wife–*Oanh*, my co-conspirator in life, who has stood by me (online!) through every failed experiment and every small victory: thank you for keeping me grounded, fed, and loved. This thesis is as much yours as it is mine (but I'm still the first author). Finally, to all the unnamed sources of inspiration—books, papers, GitHub repositories, bugs, and badly drawn draft papers/whiteboard diagrams—thank you for keeping the chaos interesting.

Contents

1. Introduction

The recent advancements in computing hardware, notably the significant evolution of modern Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) since 2015 [1, 2], have significantly enhanced computational throughput due to their Single-Instruction, Multiple-Data (SIMD) architectures [3]. GPU and TPU have evolved to handle massive parallel operations, capitalizing on thousands of cores designed to execute identical instructions concurrently across multiple data streams. These hardware innovations have drastically reduced computation times and revolutionized computationally intensive tasks in fields such as machine learning [4, 5, 6, 7, 8, 9], vectorizing simulation for robot learning [10, 11, 12, 13], and differentiable physics [14, 15, 16, 17]. The growth in computational power has directly impacted machine learning capabilities, enabling training models on larger datasets and larger model capacity at unprecedented speeds [18, 19]. Despite these rapid advancements, fully harnessing the computational potential of modern GPU and TPU remains an ongoing research area, raising questions about how effectively parallelism can be embedded within complex computational pipelines.

Concurrently, the development and widespread adoption of machine learning frameworks such as JAX [20] and PyTorch [21] have revolutionized the machine learning landscape by fully leveraging SIMD architectures. These frameworks provide powerful abstractions and optimizations specifically tailored for parallel hardware, enabling efficient tensor computations and automatic differentiation through batch operations. As a result, JAX and PyTorch facilitate streamlined, scalable, and highly parallel training pipelines that accelerate model convergence and support vectorizing model variants (e.g., neural architecture search [22]). Moreover, their versatile integration with different modern hardware has led to significant improvements in computational efficiency, greatly enhancing the accessibility and effectiveness of deep learning models across applications, including large models [9, 23] or edge devices [24]. However, while these frameworks have significantly advanced training and inference pipelines, there remains an ongoing effort regarding their optimal integration and performance tuning for specific application domains, outside of the data fitting paradigm, such as robotics planning and real-time control systems [25, 26, 27].

Classical motion planning algorithms—such as trajectory optimization [28], path planning with probabilistic completeness [29, 30], and sampling-based Model Predictive Control (MPC) [31]—have traditionally been designed with a strong dependency on

sequential computations [32, 33]. Indeed, sequential computation naturally imposes restrictions on the computational throughput achievable in real-time and high-dimensional scenarios, as many motion planning tasks inherently comprise numerous parallelizable subproblems, such as collision checking and sampling operations. The ongoing development of GPUs, which offer thousands of processing cores capable of handling massively parallel computations, has created an opportunity to scale these traditionally sequential motion planning tasks dramatically [34, 35]. Although previous attempts at parallelizing motion planning have targeted specific components like collision checking or tree sampling [36, 37], a comprehensive approach to fully exploiting GPU architectures throughout the entire planning pipeline, enabling vectorization mapping over multiple, holistic planning instances, remains largely unexplored [38].

This gap strongly motivates the definition of *instance-level planning vectorization*, which parallelizes the complete planning pipeline (cf. Fig. 1.1). Instance-level vectorization proposes a holistic approach to motion planning by ensuring the entire pipeline—ranging from sampling and search processes to model evaluation and solution extraction—has fixed input/output tensor shapes, thereby enabling the vectorization of multiple planning problems. This holistic approach is particularly beneficial for multi-solution discovery, facilitating the efficient generation of diverse datasets and supporting global execution strategies [39, 40, 41, 38]. Furthermore, complete pipeline vectorization uniquely enables real-time online domain randomization [42, 43], thereby enhancing the robustness and transferability of simulation-based control strategies to real-world scenarios, a critical requirement in robotics and autonomous systems (cf. Fig. 1.6). This leads to critical research inquiries:

> **Can vectorized, gradient-free and fast algorithmic designs-inducing efficient control loops-solve a wide range of motion planning problems, from reactive tasks and stabilization under uncertainty to sparse-cost exploration and multi-solution extraction?**

This question challenges the prevailing reliance on heavy gradient-based planning pipelines by asking whether lightweight, gradient-free, and batched computation can yield practical and robust planning behavior [41]. Especially in latency-critical applications like autonomous driving or agile robotics, the ability to generate plans in microseconds—and to do so across many models, scenarios, or constraints in parallel—demands rethinking motion planning through the lens of vectorized design.

To address this core question, this thesis proposes a class of *tensor search methods*—a new formulation of motion planning centered on fixed-shape tensor operations. While modern simulators like MuJoCo XLA and Isaac Sim, as well as control update routines

such as MPPI, CEM, and gradient-based trajectory optimization, are already architected to exploit vectorized execution on accelerators, the *search mechanism itself*—including sampling, decision-making, and multi-query planning—has remained largely sequential at instance-level [35, 34, 37, 38]. Tensor search methods fill this gap by tensorizing the search process across multiple planning instances, enabling simultaneous exploration of solution spaces, environments, and model perturbations. This thesis introduces tensor-based sampling strategies throughout the planning pipeline, with local methods using randomly rotated polytopes for trajectory optimization (Chapter 2), and global methods using multipartite graphs for structured, high-entropy exploration (Chapter 3 and Chapter 4). By ensuring a homogeneous tensor structure across all planning components, we unlock full compatibility with modern Just-in-Time (JIT) compilation and vectorized mapping tools such as JAX's `vmap` and `jit` [20]. Beyond implementation, the thesis presents theoretical analysis to highlight the structural properties and performance guarantees of these methods. Extensive empirical evaluations further demonstrate their practical benefits—achieving faster computation, higher-quality and more diverse solutions, and improved robustness under real-time and high-dimensional constraints. Altogether, tensor search provides a principled, scalable framework for motion planning that aligns with the strengths of modern accelerator hardware and software ecosystems.

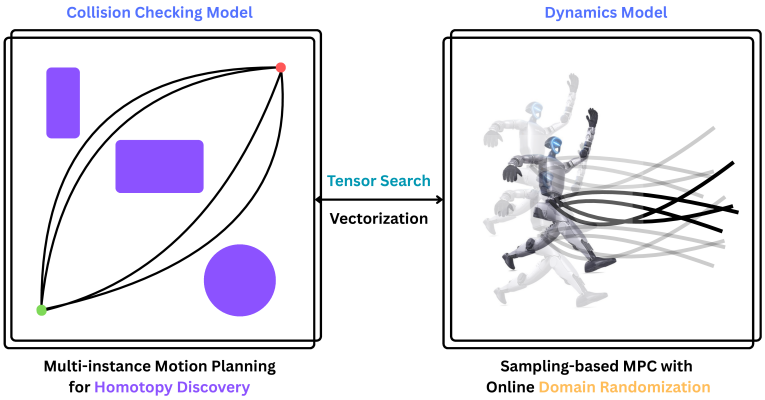


Figure 1.1.: The duality of instance-level planning vectorization. (Left) Vectorized motion planning from A to B through multiple homotopy classes using batched collision checking and environment probing, illustrating multi-instance path discovery around obstacles. (Right) Online domain randomization via batch rollouts across perturbed model variants in sampling-based MPC, leveraging vectorized dynamics simulation and parallel solution evaluation.

Before outlining the main contributions of this thesis, we briefly survey and connect the historical development of linear algebra as an efficient formalism for solving structured problems, from early arithmetic and geometric applications to modern scientific computing. This foundation not only enabled analytical solutions to classical problems but also shaped how we express and solve computational tasks today. In particular, we draw a line from the matrix-based representation of problems in classical linear algebra to their modern generalization as tensors, which are crucial for achieving parallelism through vectorized computation. By understanding this evolution (cf. Fig. 1.2), we establish the motivation for tensorizing classical computing problems—such as motion planning—to leverage modern tools like `vmap` and `jit` in JAX and PyTorch for scalable, high-performance execution.

1.1. From Classical Linear Algebra To Modern Parallelism

> Science is what we understand well enough to explain to a computer. Art is everything else we do.
>
> *Donald Knuth*

The evolution from solving daily arithmetic problems to scaling modern scientific computation can be traced through the formalization of linear algebra, effectively representing daily problems as structured forms. The earliest roots of linear algebra can be found in ancient civilizations. Around 200 BCE, Chinese mathematicians used methods similar to modern Gaussian elimination to solve systems of linear equations, as documented in the classic text *The Nine Chapters on the Mathematical Art* [44]. Similarly, in the 3rd century CE, Greek mathematicians Euclid and Diophantus studied equations that laid the groundwork for algebraic thinking [45]. The symbolic notation that forms the foundation of modern algebra was developed much later, particularly during the 16th and 17th centuries, with mathematicians like François Viète and René Descartes contributing to algebraic symbolism and the Cartesian coordinate system, respectively. These developments paved the way for representing geometric problems algebraically. Linear algebra began to take a more formal shape in the 18th and 19th centuries. In 1750, Gabriel Cramer introduced *Cramer's Rule* for solving systems of linear equations using determinants. In the early 19th century, Augustin-Louis Cauchy contributed significantly to matrix theory and determinant properties. The term *matrix* itself was coined by James Joseph Sylvester in 1850, while Arthur Cayley formalized

matrix operations, including multiplication and the concept of the inverse. The notion of a *vector space* and *linear transformation* was solidified in the late 19th and early 20th centuries through the work of mathematicians such as Giuseppe Peano, who defined abstract vector spaces, and Hermann Grassmann, whose *Ausdehnungslehre* (i.e., Theory of Extension) laid the groundwork for multilinear algebra. In the 20th century, linear algebra became an essential tool in many branches of mathematics and engineering disciplines, including quantum mechanics, statistics, and numerical analysis [46].

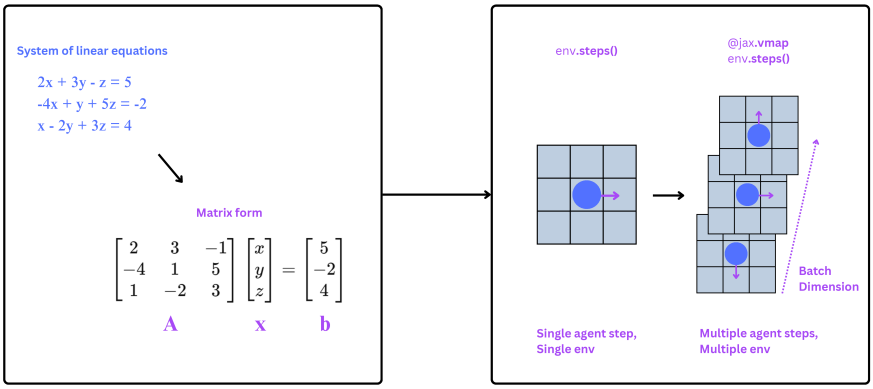


Figure 1.2.: The evolution from representing ancient problems as matrix forms to representing computing problems as tensors.

Nowadays, the rise of computing hardware capacity has further entrenched linear algebra at the heart of scientific computing, with matrix operations becoming core building blocks of modern algorithms and frameworks. The introduction of `NumPy` [47] marked a pivotal moment in this evolution. Developed by Travis Oliphant in 2005 as a successor to `Numeric` and `numarray`, `NumPy` unified and extended these earlier libraries, providing a robust and efficient N-dimensional array object (`ndarray`) that serves as the foundational data structure for numerical operations in Python. With its seamless integration of low-level C and Fortran libraries, `NumPy` offers high-performance capabilities for array operations, linear algebra, random sampling, and Fourier transforms, among others. Its concise syntax and performance close to natively compiled code made Python a viable alternative to traditionally faster languages like C and MATLAB for scientific computing. Moreover, `NumPy` established the de facto standard API for tensor computation in the Python ecosystem, influencing the design of major libraries such as `SciPy`, `Pandas`, `Scikit-learn`, `TensorFlow`, and `PyTorch`. As such, `NumPy` has played a foundational role in the rapid expansion of Python as the language

of choice for data science, machine learning, and numerical computing.

Building on the foundational semantics of `NumPy`, the JAX library [48] introduces a powerful approach to program transformation and hardware acceleration through high-level tracing and vectorization. Central to this capability is the `vmap` function, which enables automatic vectorization by transforming functions to operate over batched inputs without the need for manual looping. This aligns closely with the SIMD paradigm and modern GPU architectures, which thrive on concurrent execution of identical operations across different data instances. By leveraging XLA (Accelerated Linear Algebra), JAX compiles pure, statically composed subroutines into highly optimized GPU/TPU kernels, offering substantial speedups while maintaining Pythonic programmability. A critical requirement for JIT compilation in JAX is the use of *statically shaped* matrices and tensors. This means that all input arrays must have fixed shapes at compile time so that the underlying XLA compiler can generate monomorphic, highly optimized kernels. Any change in the shape of the inputs triggers a recompilation, which can incur overhead. Thus, maintaining consistent tensor shapes across the computation pipeline is essential for performance and kernel fusion, especially when applying JAX transformations such as `jit`, `vmap`, and `grad`.

The modern landscape of scientific computing demonstrates how reformulating mathematical problems in terms of matrices and tensors unlocks the full power of vectorized mapping.

Examples: Markov Chains. A classical example is found in the computation of stationary distributions for Markov chains. Given a Markov chain with n nodes, instead of solving the stationary distribution by looping over nodes, we can formulate a transition matrix $P \in \mathbb{R}^{n \times n}$ representing pairwise transition probability with marginal constraints, the stationary distribution $\boldsymbol{\pi} \in \mathbb{R}^n$ satisfies:

$$\boldsymbol{\pi} = \boldsymbol{\pi} P, \quad \text{s.t.} \quad \sum_{i=1}^{n} \pi_i = 1.$$

This formulation effectively transforms the problem into solving a linear equation system, boiling down to matrix operations. This matrix form also benefits greatly from `vmap`, which can apply the method in parallel across many different transition matrices—enabling solving multiple stationary distributions of multiple stochastic systems or parallel environment models.

Examples: Neural Networks. Similarly, in Multilayer Perceptron (MLP), each layer

computes a transformation of the form:

$$\mathbf{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}),$$

where $\boldsymbol{W}$ is a weight matrix, $\boldsymbol{b}$ is a bias vector, $\boldsymbol{x}$ is the input, and $f(\cdot)$ is a nonlinear activation function. Using `vmap`, these computations can be batched over multiple data inputs, allowing for fully vectorized forward and backward passes across a minibatch of dataset $\{\boldsymbol{x}_i\}_{i=1}^{B}$ or ensemble of models $\{(\boldsymbol{W}_j, \boldsymbol{b}_j)\}_{j=1}^{K}$. This classical network design ensures consistent tensor shapes, which is crucial for JIT compilation and kernel fusion, thereby unlocking high-throughput execution on modern accelerators.

In the context of this thesis, we adopt this tensor-centric computational paradigm to reframe motion planning—traditionally a sequential process—as a fully vectorizable pipeline. By formulating motion planning components such as sampling, search, optimization, and evaluation in tensor algebra and ensuring fixed tensor input/output shapes, we enable instance-level vectorization of planning problems. This not only facilitates the parallel solution of multiple planning queries simultaneously but also ensures compatibility with modern hardware acceleration techniques like `jit` and `vmap`. Our goal is to systematically explore the theoretical and practical implications of this transformation, demonstrating that structured tensor representations can unify and accelerate complex planning algorithms through scalable, high-throughput parallelism.

1.2. Contributions

This thesis presents three core contributions corresponding to distinct paradigms of the planning pipeline, each demonstrating how classical motion planning techniques can be reformulated in a tensorized, vectorizable representation, enabling vectorized mapping over planning instances for multi-mode discovery or online domain randomization. These contributions align with the chapters of this thesis and collectively establish a foundation for *instance-level vectorization* using modern GPU-accelerated frameworks (cf. Fig. 1.3).

1.2.1. Local Tensor Search for Vectorizing Trajectory Optimization

In Chapter 2, we introduce *Motion Planning via Optimal Transport* (MPOT), a novel trajectory optimization method built on a fully tensorized update mechanism, namely *the Sinkhorn Step*. This method reframes trajectory optimization as a zero-order, gradient-free problem and introduces a batch-wise, parallelizable update rule suitable

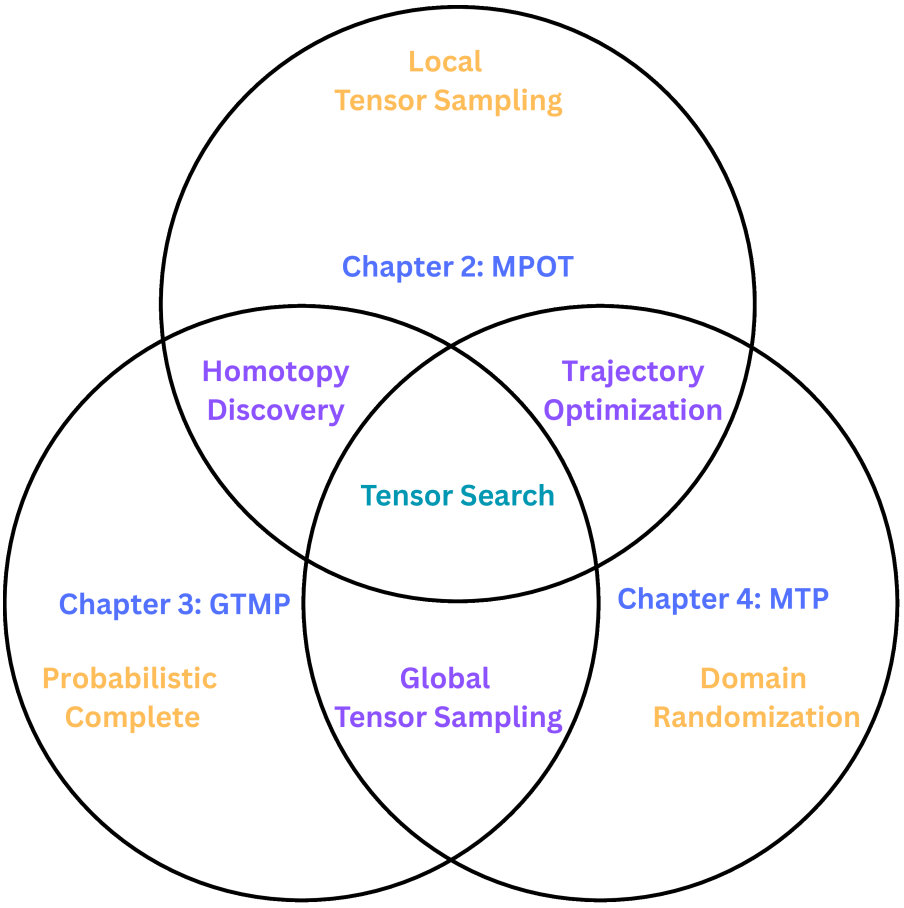


Figure 1.3.: Outline of core algorithm contributions mapping to thesis chapters. The overlap areas represent the common property of the algorithms (purple) w.r.t. the counterpart property of the other algorithm (orange).

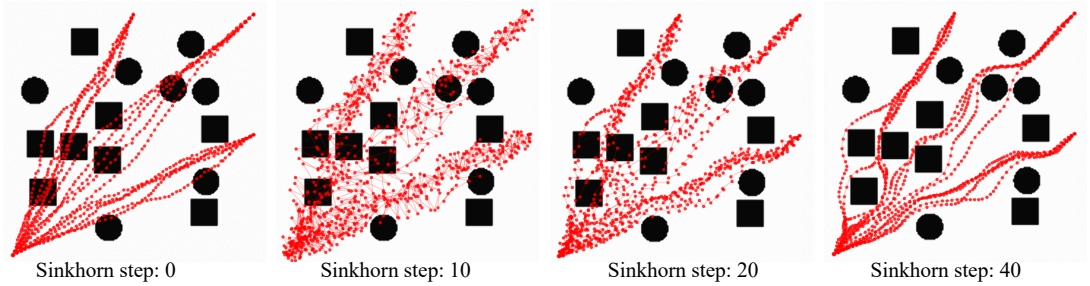


Figure 1.4.: Example of Motion Planning via Optimal Transport (MPOT) in the multimodal planar navigation scenario with three different goals. For each goal, we sample five initial trajectories from a Gaussian Process (GP) prior. We illustrate four snapshots of our proposed Sinkhorn Step that updates a batch of waypoints from multiple trajectories over multiple goals. For this example, the **total planning time was 0**.12s.

for GPU acceleration. Specifically, each trajectory is discretized into waypoints, and for each waypoint, we construct a randomly rotated regular polytope that serves as a local trust region. The update direction is determined by solving an entropic-regularized Optimal Transport (OT) problem, where the cost matrix is evaluated by probing the environment's local cost structure along the polytope vertices.

The result of this OT problem is an optimal transport plan that defines a barycentric projection of each waypoint toward lower-cost regions. This process is computed efficiently using the Sinkhorn-Knopp algorithm and can be batched across multiple waypoints and multiple trajectories simultaneously. We show that this formulation naturally induces tensorized update steps across hundreds of trajectory plans (cf. Fig. 1.4).

The method exhibits strong convergence properties and supports structured exploration via entropic regularization, making it particularly effective in high-dimensional, non-convex motion planning problems. The contribution also includes a theoretical investigation of the Sinkhorn Step convergence in an exemplary condition of square matrix, empirical benchmarking on manipulation and mobile tasks, and implementation considerations for vectorized execution in modern accelerator environments.

1.2.2. Global Tensor Search for Vectorizing Global Path Planning

In Chapter 3, we present *Global Tensor Motion Planning* (GTMP), a fully tensorized sampling-based motion planner designed for efficient instance-level vectorization. Unlike traditional planners based on trees or incrementally constructed graphs—which are inherently sequential and difficult to parallelize—GTMP introduces a novel discretization structure–a *random multipartite graph*. This graph represents the planning problem as a fixed-shape tensor, enabling efficient batch sampling, forward kinematics, collision checking, and graph search.

Each planning problem is encoded as a layered graph with uniformly sampled waypoints organized in multiple layers. These layers form a complete multipartite graph, which allows the use of tensorized dynamic programming to solve for optimal paths. Crucially, GTMP performs batch Bellman updates via finite-horizon value iteration, computing the cost-to-go over all sampled nodes using batched min-sum operations, and then extracts optimal paths via tracing the value table—all using pure tensor operations.

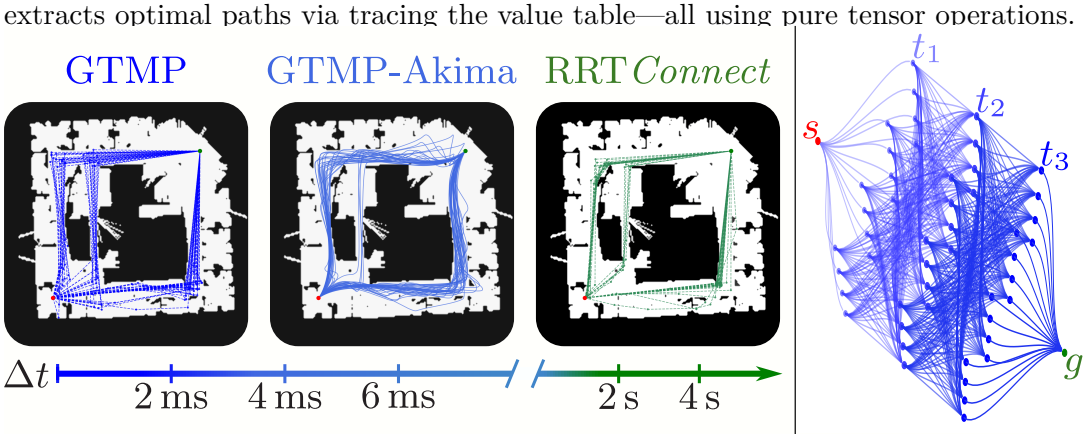


Figure 1.5.: (Left) Planning comparison of a batch 100 instances on Intel Lab occupancy map. We vectorize planning with GTMP and GTMP-Akima on an RTX3090, filtering out collided paths, taking less than one millisecond after just-in-time compilation (JIT) [20]. OMPL's RRTConnect [49] is run in a loop with a single core of AMD Ryzen 5900X, which takes a few seconds (including simplification time and can be divided by the total number of CPU cores if using CPU parallelization). (Right) Spline discretization graph example of $M = 3$ layers. The spline structure computation can be vectorized over planning instances with the Akima interpolation method [50].

This formulation supports instance-level vectorization, where multiple planning problems—differing in start-goal pairs or environments—are solved in parallel by simply

adding a batch dimension. GTMP is implemented in JAX using `vmap` and `jit`, enabling high-throughput GPU execution and fixed memory footprints. We provide theoretical analysis demonstrating GTMP's probabilistic completeness and practical guarantees on path feasibility as a function of layer count and waypoint density.

Furthermore, GTMP extends naturally to smooth path generation via Akima spline interpolation [50], allowing the algorithm to produce smooth, continuous trajectories without requiring gradients or simplification routines (cf. Fig. 1.5). Our experiments validate that GTMP and its spline extension achieve strong planning efficiency, smoothness, and path diversity, surpassing classical baselines such as RRTConnect [30] and BKPIECE [51] implemented in OMPL/Pybullet implementation [49], and matching or exceeding state-of-the-art microsecond planning RRTConnect/VAMP [37] and modern GPU-based planners like cuRobo [52] in planning efficiency of manipulators.

1.2.3. Global Tensor Search for Vectorizing Sampling-based MPC

In Chapter 4, we introduce *Model Tensor Planning* (MTP), a novel framework for sampling-based Model Predictive Control (MPC) that achieves high-entropy control generation through a tensorized planning structure. Traditional sampling-based MPC often relies on locally perturbing nominal trajectories, limiting global exploration and causing frequent mode collapse, especially in high-dimensional or contact-rich robotic tasks. MTP addresses this by reformulating control trajectory generation as a structured tensor sampling problem over a randomized multipartite graph.

At the core of MTP is a fixed-shape control waypoint tensor, sampled across multiple layers and interpolated into smooth control trajectories using linear, B-spline, or Akima-spline interpolators. These control sequences are evaluated in batch using a parallelized simulator, enabling fast rollout and cumulative cost estimation. To balance local refinement and global exploration, MTP introduces a simple yet effective β-mixing strategy, blending local Gaussian samples with globally exploratory tensor samples. The update rule follows a modified Cross-Entropy Method (CEM), incorporating softmax-weighted updates that enable smoother convergence across timesteps.

The entire MTP pipeline is designed and implemented using JAX with full support for `jit` compilation and vectorized mapping `vmap`. This allows the method to run efficiently on modern accelerators like GPUs while maintaining static tensor shapes compatible with XLA. Crucially, this design supports online domain randomization by vectorizing over perturbed models in batch, offering viable sim-to-real transfer (cf. Fig. 1.6).

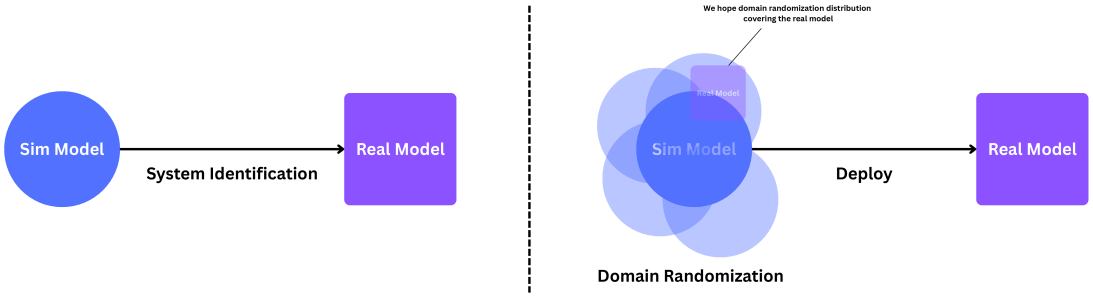


Figure 1.6.: (Left) Typically, classical motion planning requires system identification to set up the precise model for planning. (Right) With vectorized simulators, we perturb models in parallel, effectively vectorizing rollouts over model variants and evaluating risk strategies.

Theoretically, we show that tensor sampling approximates maximum entropy over the control trajectory space and achieves asymptotic path coverage in the limit of infinite tensor width and depth. Empirically, MTP outperforms strong baselines—including MPPI [53], Predictive Sampling [54], and modern Evolutionary Strategies [55, 56]—on a range of robotic benchmarks from dexterous manipulation to humanoid locomotion. These results highlight the strength of tensorized high-entropy sampling for scalable and robust control in modern robotics.

Together, these contributions present a cohesive methodology for transforming classical planning algorithms into a tensorized formulation, unlocking their compatibility with modern parallel hardware and enabling scalable, real-time performance through vectorized computation.

2. Accelerating Motion Planning via Optimal Transport

Motion planning is still an open problem for many disciplines, e.g., robotics, autonomous driving, due to their need for high computational resources that hinder real-time, efficient decision-making. A class of methods striving to provide smooth solutions is gradient-based trajectory optimization. However, those methods usually suffer from bad local minima, while for many settings, they may be inapplicable due to the absence of easy-to-access gradients of the optimization objectives. In response to these issues, we introduce Motion Planning via Optimal Transport (MPOT)—a *gradient-free* method that optimizes a batch of smooth trajectories over highly nonlinear costs, even for high-dimensional tasks, while imposing smoothness through a Gaussian Process dynamics prior via the planning-as-inference perspective. To facilitate batch trajectory optimization, we introduce an original zero-order and highly-parallelizable update rule—-the Sinkhorn Step, which uses the regular polytope family for its search directions. Each regular polytope, centered on trajectory waypoints, serves as a local cost-probing neighborhood, acting as a *trust region* where the Sinkhorn Step "transports" local waypoints toward low-cost regions. We theoretically show that Sinkhorn Step guides the optimizing parameters toward local minima regions of non-convex objective functions. We then show the efficiency of MPOT in a range of problems from low-dimensional point-mass navigation to high-dimensional whole-body robot motion planning, evincing its superiority compared to popular motion planners, paving the way for new applications of optimal transport in vectorizing motion planning.

2.1. Introduction

Motion planning is a fundamental problem for various domains, spanning robotics [57], autonomous driving [58], space-satellite swarm [59], protein docking [60]. etc., aiming to find feasible, smooth, and collision-free paths from start-to-goal configurations. Motion planning has been studied both as sampling-based search [29, 30] and as an optimization problem [61, 62, 63]. Both approaches have to deal with the complexity of high-dimensional configuration spaces, e.g., when considering high-degrees of freedom (DoF) robots, the multi-modality of objectives due to multiple constraints at both

configuration and task space, and the requirement for smooth trajectories that low-level controllers can effectively execute. Sampling-based methods sample the high-dimensional manifold of configurations and use different search techniques to find a feasible and optimal path [30, 64, 29], but suffer from the complex sampling process and the need for large computational budgets to provide a solution, which increases w.r.t. the complexity of the problem (e.g., highly redundant robots and narrow passages) [65]. Optimization-based approaches work on a trajectory level, optimizing initial trajectory samples either via covariant gradient descent [61, 66] or through probabilistic inference [63, 62, 67]. Nevertheless, as with every optimization pipeline, trajectory optimization depends on initialization and can get trapped in bad local minima due to the non-convexity of complex objectives. Moreover, in some problem settings, objective gradients are unavailable or expensive to compute. Indeed, trajectory optimization is difficult to tune and is often avoided in favor of sampling-based methods with probabilistic completeness. We refer to Section 2.6 for an extensive discussion of related works.

To address these issues of trajectory optimization, we propose a zero-order, fast, and highly parallelizable update rule—the Sinkhorn Step. We apply this novel update rule in trajectory optimization, resulting in *MPOT* – a gradient-free trajectory optimization method optimizing a batch of smooth trajectories. MPOT optimizes trajectories by solving a sequence of entropic-regularized Optimal Transport (OT) problems, where each OT instance is solved efficiently with the celebrated Sinkhorn-Knopp algorithm [68]. In particular, MPOT discretizes the trajectories into waypoints and structurally probes a local neighborhood around each of them, which effectively exhibits a *trust region*, where it "transports" local waypoints towards low-cost areas given the local cost approximated by the probing mechanism. Our method is simple and does not require computing gradients from cost functions propagating over long kinematics chains. Crucially, the planning-as-inference perspective [69, 67] allows us to impose constraints related to transition dynamics as planning costs, additionally imposing smoothness through a GP prior. Delegating complex constraints to the planning objective allows us to locally resolve trajectory update as an OT problem at each iteration, updating the trajectory waypoints towards the local optima, thus effectively optimizing for complex cost functions formulated in configuration and task space. We also provide a preliminary theoretical analysis of the Sinkhorn Step, highlighting its core properties that allow optimizing trajectories toward local minima regions.

Further, our empirical evaluations on representative tasks with high-dimensionality and multimodal planning objectives demonstrate an increased benefit of MPOT, both in terms of planning time and success rate, compared to notable trajectory optimization methods. Moreover, we empirically demonstrate the convergence of MPOT in a 7-DoF

robotic manipulation setting, showcasing a fast convergence of MPOT, reflected also in its dramatically reduced planning time w.r.t. baselines. The latter holds even for 36-dimensional, highly redundant mobile manipulation systems in long-horizon fetch and place tasks (cf. Fig. 2.3).
Our **contribution** is twofold. *(i)* We propose the Sinkhorn Step - an efficient zero-order update rule for optimizing a batch of parameters, formulated as a barycentric projection of the current points to the polytope vertices. *(ii)* We, then, apply the Sinkhorn Step to motion planning, resulting in a novel trajectory optimization method that optimizes a batch of trajectories by efficiently solving a sequence of linear programs. It treats every waypoint across trajectories equally, enabling fast batch updates of multiple trajectories-waypoints over multiple goals by solving a single OT instance while retaining smoothness due to integrating the GP prior as cost function.

2.2. Preliminaries

Entropic-regularized optimal transport. We briefly introduce discrete OT. For a thorough introduction, we refer to [70, 71, 72].
Notation. Throughout the paper, we consider the optimization on a d-dimensional Euclidean space $\mathbb{R}^d$, representing the parameter space (e.g., a system state space). $\mathbf{1}_d$ is the vector of ones in $\mathbb{R}^d$. The scalar product for vectors and matrices is $x, y \in \mathbb{R}^d$, $\langle x, y\rangle = \sum_{i=1}^{d} x_i y_i$; and $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$, $\langle \boldsymbol{A}, \boldsymbol{B}\rangle = \sum_{i,j=1}^{d} \boldsymbol{A}_{ij}\boldsymbol{B}_{ij}$, respectively. $\|\cdot\|$ is the l_2-norm, and $\|\cdot\|_{\boldsymbol{M}}$ denotes the Mahalanobis norm w.r.t. some positive definite matrix $\boldsymbol{M} \succ 0$. For two histograms $\boldsymbol{n} \in \Sigma_n$ and $\boldsymbol{m} \in \Sigma_m$ in the simplex $\Sigma_d := \{\boldsymbol{x} \in \mathbb{R}^d_+ : \boldsymbol{x}^\intercal \mathbf{1}_d = 1\}$, we define the set $U(\boldsymbol{n}, \boldsymbol{m}) := \{\boldsymbol{W} \in \mathbb{R}^{n\times m}_+ \mid \boldsymbol{W}\mathbf{1}_m = \boldsymbol{n}, \boldsymbol{W}^\intercal \mathbf{1}_n = \boldsymbol{m}\}$ containing $n \times m$ matrices with row and column sums $\boldsymbol{n}$ and $\boldsymbol{m}$ respectively. Correspondingly, the entropy for $\boldsymbol{A} \in U(\boldsymbol{n}, \boldsymbol{m})$ is defined as $H(\boldsymbol{A}) = -\sum_{i,j=1}^{n,m} a_{ij} \log a_{ij}$.
Let $\boldsymbol{C} \in \mathbb{R}^{n\times m}_+$ be the positive cost matrix, the OT between $\boldsymbol{n}$ and $\boldsymbol{m}$ given cost $\boldsymbol{C}$ is $\mathrm{OT}(\boldsymbol{n}, \boldsymbol{m}) := \min_{\boldsymbol{W}\in U(\boldsymbol{n},\boldsymbol{m})} \langle \boldsymbol{W}, \boldsymbol{C}\rangle$. Traditionally, OT does not scale well with high dimensions. To address this, Cuturi [73] proposes to regularize its objective with an entropy term, resulting in the entropic-regularized OT

$$\mathrm{OT}_\lambda(\boldsymbol{n}, \boldsymbol{m}) := \min_{\boldsymbol{W}\in U(\boldsymbol{n},\boldsymbol{m})} \langle \boldsymbol{W}, \boldsymbol{C}\rangle - \lambda H(\boldsymbol{W}). \tag{2.1}$$

Solving (2.1) with Sinkhorn-Knopp [73] has a complexity of $\tilde{O}(n^2/\epsilon^3)$ [74], where ϵ is the approximation error w.r.t. the original OT. Higher λ enables a faster but "blurry" solution, and vice versa.

Trajectory optimization. Given a parameterized trajectory by a discrete set of support states and control inputs $\boldsymbol{\tau} = [\mathbf{x}_0, \boldsymbol{u}_0, ..., \mathbf{x}_{T-1}, \boldsymbol{u}_{T-1}, \mathbf{x}_T]^\intercal$, trajectory optimization aims to find the optimal trajectory $\boldsymbol{\tau}^*$, which minimizes an objective function $c(\boldsymbol{\tau})$, with $\mathbf{x}_0$ being the start state. Standard motion planning costs, such as goal cost c_g defined as the distance to a desired goal-state $\boldsymbol{x}_g$, obstacle avoidance cost c_{obs}, and smoothness cost c_{sm} can be included in the objective. Hence, trajectory optimization can be expressed as the sum of those costs while obeying the dynamics constraint

$$\boldsymbol{\tau}^* = \arg\min_{\boldsymbol{\tau}} \left[c_{obs}(\boldsymbol{\tau}) + c_g(\boldsymbol{\tau}, \mathbf{x}_g) + c_{sm}(\boldsymbol{\tau})\right] \text{ s.t. } \quad \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \text{ and } \boldsymbol{\tau}[0] = \boldsymbol{x}_0. \tag{2.2}$$

For many manipulation tasks with high-DoF robots, this optimization problem is typically highly nonlinear due to many complex objectives and constraints. Besides c_{obs}, c_{sm} is crucial for finding smooth trajectories for better tracking control. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [61] designs a finite difference matrix $\boldsymbol{M}$ resulting to the smoothness cost $c_{sm} = \boldsymbol{\tau}^\intercal \boldsymbol{M} \boldsymbol{\tau}$. This smoothness cost can be interpreted as a penalty on trajectory derivative magnitudes. Mukadam et al. [62] generalizes the smoothness cost by incorporating a GP prior as cost via the planning-as-inference perspective [69, 75], additionally constraining the trajectories to be dynamically smooth. Recently, an emergent paradigm of multimodal trajectory optimization [76, 77, 67, 78] is promising for discovering different modes for non-convex objectives, thereby exhibiting robustness against bad local minima. Our work contributes to this momentum by proposing an efficient batch update-rule for vectorizing waypoint updates across timesteps and number of plans.

2.3. Sinkhorn Step

To address the problem of batch trajectory optimization in a gradient-free setting, we propose *Sinkhorn Step*—a zero-order update rule for a batch of optimization variables. Our method draws inspiration from the free-support barycenter problem [79], where the mean support of a set of empirical measures is optimized w.r.t. the OT cost. Consider an optimization problem with some objective function without easy access to function derivatives. This barycenter problem can be utilized as a parameter update mechanism, i.e., by defining a set of discrete target points (i.e., local search directions) and a batch of optimizing points as two empirical measures, the barycenter of these empirical measures acts as the updated optimizing points based on the objective function evaluation at the target points.

With these considerations in mind, we introduce *Sinkhorn Step*, consisting of two components: a polytope structure defining the unbiased search-direction bases, and a weighting distribution for evaluating the search directions. Particularly, the weighting distribution has row-column unit constraints and must be efficient to compute. Following the motivation of [79], the entropic-regularized OT fits nicely into the second component, providing a solution for the weighting distribution as an OT plan, which is solved extremely fast, and its solution is unique [73]. In this section, we formally define Sinkhorn Step and perform a preliminary theoretical analysis to shed light on its connection to *directional-direct search* methods [80, 81], thereby motivating its algorithmic choices and practical implementation proposed in this paper.

2.3.1. Problem formulation

We consider the batch optimization problem

$$\min_X f(X) = \min_X \sum_{i=1}^{n} f(\boldsymbol{x}_i), \tag{2.3}$$

where $X = \{\boldsymbol{x}_i\}_{i=1}^n$ is a set of n optimizing points, $f : \mathbb{R}^d \to \mathbb{R}$ is non-convex, differentiable, bounded below, and has L-Lipschitz gradients.

Assumption 2.1. *The objective f is L-smooth with $L > 0$*

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$$

and bounded below by $f(\boldsymbol{x}) \geq f_* \in \mathbb{R}$, $\forall \boldsymbol{x} \in \mathbb{R}^d$.

Throughout the paper, we assume that function evaluation is implemented batch-wise and is cheap to compute. Function derivatives are either expensive or impossible to compute. At the first iteration, we sample a set of initial points $X_0 \sim \mathcal{D}_0$, with its matrix form $\boldsymbol{X}_0 \in \mathbb{R}^{n\times d}$, from some prior distribution $\mathcal{D}_0$. The goal is to compute a batch update for the optimizing points, minimizing the objective function. This problem setting suits trajectory optimization described in Section 2.4.

2.3.2. Sinkhorn Step formulation

Similar to directional-direct search, Sinkhorn Step typically evaluates the objective function over a search-direction-set D, ensuring descent with a sufficiently small stepsize. The search-direction-set is typically a vector-set requiring to be a *positive spanning set* [82], i.e., its conic hull is $\mathbb{R}^d = \{\sum_i w_i \boldsymbol{d}_i,\ \boldsymbol{d}_i \in D,\ w_i \geq 0\}$, ensuring that every point

(including the extrema) in $\mathbb{R}^d$ is reachable by a sequence of positive steps from any initial point.

Regular Polytope Search-Directions. Consider a $(d-1)$-unit hypersphere $S^{d-1} = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\}$ with the center at zero.

Definition 2.1 (Regular Polytope Search-Directions)**.** *Let us denote the regular polytope family $\mathcal{P} = \{$simplex, orthoplex, hypercube$\}$. Consider a d-dimensional polytope $P \in \mathcal{P}$ with m vertices, the search-direction set $D^P = \{\boldsymbol{d}_i \mid \|\boldsymbol{d}_i\| = 1\}_{i=1}^m$ is constructed from the vertex set of the regular polytope P inscribing S^{d-1}.*

The d-dimensional regular polytope family $\mathcal{P}$ has all of its dihedral angles equal and, hence, is an unbiased sparse approximation of the circumscribed $(d-1)$-sphere, i.e., $\sum_i \boldsymbol{d}_i = 0, \|\boldsymbol{d}_i\| = 1 \, \forall i$. There also exist other regular polytope families. However, the regular polytope types in $\mathcal{P}$ exist in every dimension (cf. [83]). Moreover, the algorithmic construction of general polytope is not trivial [84]. Vertex enumeration for $\mathcal{P}$ is straightforward for vectorization and simple to implement, which we found to work well in our settings–see also Appendix A.1.6. We state the connection between regular polytopes and the positive spanning set in the following proposition.

Proposition 2.1. *$\forall P \in \mathcal{P}$, D^P forms a positive spanning set.*

This property ensures that any point $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x} = \sum_i w_i \boldsymbol{d}_i$, $w_i \geq 0$, $\boldsymbol{d}_i \in D^P$ can be represented by a positively weighted sum of the set of directions defined by the polytopes.

Batch Update Rule. At an iteration k, given the current optimizing points X_k and their matrix form $\boldsymbol{X}_k \in \mathbb{R}^{n \times d}$, we first construct the direction set from a chosen polytope P, and denote the direction set $\boldsymbol{D}^P \in \mathbb{R}^{m \times d}$ in matrix form. Similar to [79], let us define the prior histograms reflecting the importance of optimizing points $\boldsymbol{n} \in \Sigma_n$ and the search directions $\boldsymbol{m} \in \Sigma_m$, then the constraint space $U(\boldsymbol{n}, \boldsymbol{m})$ of OT is defined. With these settings, we define Sinkhorn Step.

Definition 2.2 (Sinkhorn Step)**.** *The batch update rule is the barycentric projection (Remark 4.11, [71]) that optimizes the free-support barycenter of the optimizing points and the batch polytope vertices*

$$\begin{aligned} \boldsymbol{X}_{k+1} &= \boldsymbol{X}_k + \boldsymbol{S}_k, \; \boldsymbol{S}_k = \alpha_k diag(\boldsymbol{n})^{-1} \boldsymbol{W}_\lambda^* \boldsymbol{D}^P \\ s.t. \; \boldsymbol{W}_\lambda^* &= \arg\min_{\boldsymbol{W} \in U(\boldsymbol{n}, \boldsymbol{m})} \langle \boldsymbol{W}, \boldsymbol{C} \rangle - \lambda H(\boldsymbol{W}), \end{aligned} \tag{2.4}$$

with $\alpha_k > 0$ as the stepsize, $\boldsymbol{C} \in \mathbb{R}^{n\times m}$, $\boldsymbol{C}_{i,j} = f(\boldsymbol{x}_i + \alpha_k \boldsymbol{d}_j)$, $\boldsymbol{x}_i \in X_k, \boldsymbol{d}_j \in D^P$ is the local objective matrix evaluated at the linear-translated polytope vertices.

Observe that the matrix $\text{diag}(\boldsymbol{n})^{-1}\boldsymbol{W}_\lambda^*$ has n row vectors in the simplex Σ_m. The batch update *transports* $\boldsymbol{X}$ to a barycenter shaping by the polytopes with weights defined by the optimal solution $\boldsymbol{W}_\lambda^*$. However, in contrast with the barycenter problem [79], the target measure supports are constructed locally at each optimizing point, and, thus, the points are transported in accordance with their local search directions. By Proposition 2.1, D^P is a positive spanning set, thus, $\boldsymbol{W}_\lambda^*$ forms a *generalized barycentric coordinate*, defined w.r.t. the finite set of polytope vertices. This property implies any point in $\mathbb{R}^d$ can be reached by a sequence of Sinkhorn Steps. For the d-simplex case, any point inside the convex hull can be identified with a unique barycentric coordinate [85], which is not the case for d-orthoplex or d-cube. However, coordinate uniqueness is not required for our analysis in this paper, given the following assumption.

Assumption 2.2. *At any iteration $k > 0$, the prior histogram on the optimizing points and the search-direction set is uniform $\boldsymbol{n} = \boldsymbol{m} = \mathbf{1}_n/n$, having the same dimension $n = m$. Additionally, the entropic scaling approaches zero $\lambda \to 0$.*

Assuming uniform prior importance of the optimizing points and their search directions is natural since, in many cases, priors for stepping are unavailable. However, our formulation also suggests a conditional Sinkhorn Step, which is interesting to study in future work. This assumption allows performing an analysis on Sinkhorn Step on the original OT solution.
With these assumptions, we can state the following theorem for each $\boldsymbol{x}_k \in X_k$ separately, given that they follow the Sinkhorn Step rule.

Theorem 2.1 (Main result)**.** *If assumption 2.1 and assumption 2.2 hold and the stepsize is sufficiently small $\alpha_k = \alpha$ with $0 < \alpha < 2\mu_P\epsilon/L$, then with a sufficient number of iterations*

$$K \geq k(\epsilon) := \frac{f(\boldsymbol{x}_0) - f_*}{(\mu_P\epsilon - \frac{L\alpha}{2})\alpha} - 1, \tag{2.5}$$

we have $\min_{0\leq k\leq K} \|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$, $\forall \boldsymbol{x}_k \in X_k$.

Note that we do not make any additional assumptions on f besides the smoothness and boundedness, and the analysis is performed on non-convex settings. Theorem 2.1 only

guarantees that the gradients of some points in the sequence of Sinkhorn Steps are arbitrarily small, i.e., in the proximity of local minima. If in practice, we implement the sufficient decreasing condition $f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq c\alpha_k^2$, then $f(\boldsymbol{x}_K) \leq f(\boldsymbol{x}_i)$, $\|\nabla f(\boldsymbol{x}_i)\| \leq \epsilon$ holds. However, this sufficient decrease check may waste some iterations and worsen the performance. We show in the experiments that the algorithm empirically exhibits convergence behavior without this condition checking. If L is known, then we can compute the optimal stepsize $\alpha = \mu_P \epsilon / L$, leading to the complexity bound $k(\epsilon) = \frac{2L(f(\boldsymbol{x}_0) - f_*)}{\mu_P^2 \epsilon^2} - 1$. Therefore, the complexity bounds for d-simplex, d-orthoplex and d-cube are $O(d^2/\epsilon^2)$, $O(d/\epsilon^2)$, and $O(1/\epsilon^2)$, respectively. The d-cube case shows the same complexity bound $O(1/\epsilon^2)$ as the well-known gradient descent complexity bound on the L-smooth function [86]. These results are detailed in Appendix A.1.1. Generally, we perform a preliminary study on Sinkhorn Step with assumption 2.1 and assumption 2.2 to connect the well-known directional-direct search literature [80, 81], as many unexplored theoretical properties of Sinkhorn Step remain in practical settings described in Section 2.4.

2.4. Motion Planning via Optimal Transport

Here, we introduce MPOT - a method that applies *Sinkhorn Step* to solve the batch trajectory optimization problem, where we realize waypoints in a set of trajectories as optimizing points. Due to Sinkhorn Step's properties, MPOT does not require gradients propagated from cost functions over long kinematics chains. It optimizes trajectories by solving a sequence of strictly convex linear programs with a maximum entropy objective (cf. Definition 2.2), smoothly transporting the waypoints according to the local polytope structure. To promote smoothness and dynamically feasible trajectories, we incorporate the GP prior as a cost via the planning-as-inference perspective.

2.4.1. Planning As Inference With Empirical Waypoint Distribution

Let us consider general discrete-time dynamics $\boldsymbol{X} = F(\mathbf{x}_0, \boldsymbol{U})$, where $\boldsymbol{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_T]$ denotes the states sequence, $\boldsymbol{U} = [\boldsymbol{u}_0, ..., \boldsymbol{u}_T]$ is the control sequence, and $\mathbf{x}_0$ is the start state. The target distribution over control trajectories $\boldsymbol{U}$ can be defined as the posterior [87]

$$q(\boldsymbol{U}) = \frac{1}{Z} \exp\left(-\eta E(\boldsymbol{U})\right) q_0(\boldsymbol{U}), \tag{2.6}$$

with $E(\boldsymbol{U})$ the energy function representing control cost, $q_0(\boldsymbol{U}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ a zero-mean normal prior, η a scaling term (temperature), and Z the normalizing scalar.

Assuming a first-order trajectory optimization[1], the control sequence can be defined as a time-derivative of the states $\boldsymbol{U} = [\dot{\mathbf{x}}_0, ..., \dot{\mathbf{x}}_T]$. The *target posterior distribution* over both state-trajectories and their derivatives $\boldsymbol{\tau} = (\boldsymbol{X}, \boldsymbol{U}) = \{\boldsymbol{x}_t \in \mathbb{R}^d : \boldsymbol{x}_t = [\mathbf{x}_t, \dot{\mathbf{x}}_t]\}_{t=0}^{T}$ is defined as

$$q^*(\boldsymbol{\tau}) = \frac{1}{Z} \exp\big(-\eta c(\boldsymbol{\tau})\big) q_F(\boldsymbol{\tau}), \tag{2.7}$$

which is similar to Eq. (2.6) with the energy function $E = c \circ F(\mathbf{x}_0, \boldsymbol{U})$ being the composition of the cost c over $\boldsymbol{\tau}$ and the dynamics F. The dynamics F is also now integrated into the prior distribution $q_F(\boldsymbol{\tau})$. The absorption of the dynamics into the prior becomes evident when we represent the prior as a zero-mean constant-velocity GP prior $q_F(\boldsymbol{\tau}) = \mathcal{N}(\mathbf{0}, \boldsymbol{K})$, with a constant time-discretization Δt and the time-correlated trajectory covariance $\boldsymbol{K}$, as described in Appendix A.1.2.

Now, to apply Sinkhorn Step, consider the trajectory we want to optimize $\boldsymbol{\tau} = \{\boldsymbol{x}_t\}_{t=1}^{T}$, we can define the *proposal trajectory distribution* as a waypoint empirical distribution

$$p(\boldsymbol{x}; \boldsymbol{\tau}) = \sum_{t=1}^{T} p(t) p(\boldsymbol{x}|t) = \sum_{t=1}^{T} n_t \delta_{\boldsymbol{x}_t}(\boldsymbol{x}), \tag{2.8}$$

with the histogram $\boldsymbol{n} = [n_1, \ldots, n_T]$, $p(t) = n_t = 1/T$, and $\delta_{\boldsymbol{x}_t}$ the Dirac on waypoints at time steps t. In this case, we consider the model-free setting for the proposal distribution. Indeed, this form of proposal trajectory distribution typically assumes no temporal or spatial (i.e., kinematics) correlation between waypoints. This assumption is also seen in [61, 63] and can be applied in a wide range of robotics applications where the system model is fully identified. We leverage this property for batch-wise computations and batch updates over all waypoints. The integration of model constraints in the proposal distribution is indeed interesting but is deferred for future work.

Following the planning-as-inference perspective, the motion planning problem can be formulated as the minimization of a Kullback–Leibler (KL) divergence between the proposal trajectory distribution $p(\boldsymbol{x}; \boldsymbol{\tau})$ and the target posterior distribution Eq. (2.7)

[1]We describe first-order formulation for simplicity. However, this work can be extended to second-order systems similar to [62].

(i.e., the I-projection)

$$
\begin{aligned}
\boldsymbol{\tau}^* &= \arg\min_{\boldsymbol{\tau}} \left\{ \mathrm{KL}\left(p(\boldsymbol{x};\boldsymbol{\tau}) \,||\, q^*(\boldsymbol{\tau})\right) = \mathbb{E}_p\left[\log q^*(\boldsymbol{\tau})\right] - H(p) \right\} \\
&= \arg\min_{\boldsymbol{\tau}} \sum_{t=0}^{T-1} \mathbb{E}_p \left[\eta c(\boldsymbol{\tau}) + \frac{1}{2} \|\boldsymbol{\tau}\|^2_{\boldsymbol{K}} - \log Z \right] \\
&= \arg\min_{\boldsymbol{\tau}} \sum_{t=0}^{T-1} \eta \underbrace{c(\boldsymbol{x}_t)}_{\text{state cost}} + \underbrace{\frac{1}{2} \|\boldsymbol{\Phi}_{t,t+1}\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|^2_{\boldsymbol{Q}^{-1}_{t,t+1}}}_{\text{transition model cost}},
\end{aligned}
\tag{2.9}
$$

with $\boldsymbol{\Phi}_{t,t+1}$ the state transition matrix, and $\boldsymbol{Q}_{t,t+1}$ the covariance between time steps t and $t+1$ originated from the GP prior (cf. Appendix A.1.2), and the normalizing constant of the target posterior Z is absorbed. Note that the entropy of the empirical distribution is constant $H(p) = -\int_{\boldsymbol{x}\in\mathbb{R}^d} \frac{1}{T} \sum_{t=1}^{T} \delta_{\boldsymbol{x}_t}(\boldsymbol{x}) \log p(\boldsymbol{x};\boldsymbol{\tau}) = \log T$. Evidently, KL objective Eq. (2.9) becomes a standard motion planning problem Eq. (2.2) with the defined waypoint empirical distributions. Note that this objective is not equivalent to Eq. (2.3) due to the second coupling term. However, we demonstrate in Section 2.5.2 that MPOT still exhibits convergence. Indeed, investigating Sinkhorn Step in a general graphical model objective [88] is vital for future work. We apply Sinkhorn Step to Eq. (2.9) by realizing the trajectory as a batch of optimizing points $\boldsymbol{\tau} \in \mathbb{R}^{T\times d}$. This realization also extends naturally to a batch of trajectories described in the next section.

The main goal of this formulation is to naturally inject the GP dynamics prior to MPOT, benefiting from the GP sparse Markovian structure resulting in the second term of the objective Eq. (2.9). This problem formulation differs from the moment-projection objective [87, 62, 67], which relies on importance sampling from the proposal distribution to perform parameter updates. Contrarily, we do not encode the model in the proposal distribution and directly optimize for the trajectory parameters, enforcing the model constraints in the cost.

2.4.2. Practical considerations for applying Sinkhorn Step

For the practical implementation, we make the following realizations to the Sinkhorn Step implementation for optimizing a trajectory $\boldsymbol{\tau}$. First, we define a set of probe points for denser function evaluations (i.e., cost-to-go for each vertex direction). We populate equidistantly *probe* points along the directions in D^P outwards till reaching a *probe radius* $\beta_k \geq \alpha_k$, resulting in the *probe set* H^P with its matrix form $\boldsymbol{H}^P \in \mathbb{R}^{m\times h\times d}$

with h probe points for each direction (cf. Fig. 2.1). Second, we add stochasticity in the search directions by applying a random d-dimensional rotation $\boldsymbol{R} \in SO(d)$ to the polytopes to promote local exploration (computation of $\boldsymbol{R} \in SO(d)$ is discussed in Appendix A.1.7). Third, to further decouple the correlations between the waypoints updates, we sample the rotation matrices in batch and then construct the direction sets from the rotated polytopes, resulting in the tensor $\boldsymbol{D}^P \in \mathbb{R}^{T\times m\times d}$. Consequently, the *probe set* is also constructed in batch for every waypoint $\boldsymbol{H}^P \in \mathbb{R}^{T\times m\times h\times d}$. The Sinkhorn Step is computed with the *einsum* operation along the second dimension (i.e., the m-dimension) of $\boldsymbol{D}^P$ and $\boldsymbol{H}^P$. In intuition, the second and third considerations facilitate random permutation of the rows of the OT cost matrix.

With these considerations, the element of the t^{th}-waypoint and i^{th}-search directions in the OT cost matrix $\boldsymbol{C} \in \mathbb{R}^{T\times m}$ is the mean of probe point evaluation along a search direction (i.e., cost-to-go)

$$\boldsymbol{C}_{t,i} = \frac{1}{h}\sum_{j=1}^{h} \eta\, c(\boldsymbol{x}_t + \boldsymbol{y}_{t,i,j}) + \tfrac{1}{2}\|\boldsymbol{\Phi}_{t,t+1}\boldsymbol{x}_t - (\boldsymbol{x}_{t+1} + \boldsymbol{y}_{t+1,i,j})\|^2_{\boldsymbol{Q}^{-1}_{t,t+1}}, \tag{2.10}$$

with the probe point $\boldsymbol{y}_{t,i,j} \in H^P$. Then, we ensure the cost matrix positiveness for numerical stability by subtracting its minimum value. With uniform prior histograms $\boldsymbol{n} = \boldsymbol{1}_T/T$, $\boldsymbol{m} = \boldsymbol{1}_m/m$, the problem $\boldsymbol{W}^* = \arg\min \mathrm{OT}_\lambda(\boldsymbol{n}, \boldsymbol{m})$ is instantiated and solved with the log-domain stabilization version [89, 90] of the Sinkhorn algorithm. By setting a moderately small $\lambda = 0.01$ to balance between performance and blurring bias, the update does not always collapse towards the vertices of the polytope, but to a conservative one inside the polytope convex hull. In fact, the Sinkhorn Step defines an *explicit trust region*, which bounds the update inside the polytope convex hull. More discussions of log-domain stabilization and trust region properties are in Appendix A.1.5 and Appendix A.1.4. In the trajectory optimization experiments, we typically do not assume any cost structure (e.g., non-smooth, non-convex). In MPOT, assumption 2.2 is usually violated with $T \gg m$, but MPOT still works well due to the soft assignment of Sinkhorn distances. We observe that finer function evaluations, randomly rotated polytopes, and moderately small λ increase the algorithm's robustness against practical conditions. Note that these implementation technicalities do not incur much overhead due to the efficient batch computation of modern GPUs.

The Sinkhorn Step typically does not assume any planning cost structure (e.g., non-smooth, non-convex), and the observed cost approximates the true planning cost region with the randomly rotated polytope (cf. Fig. 2.1). Due to these properties, the Sinkhorn Step can perform batched parameter updates in long computational chain scenarios using only a single forward pass and is suitable for gradient-free settings.

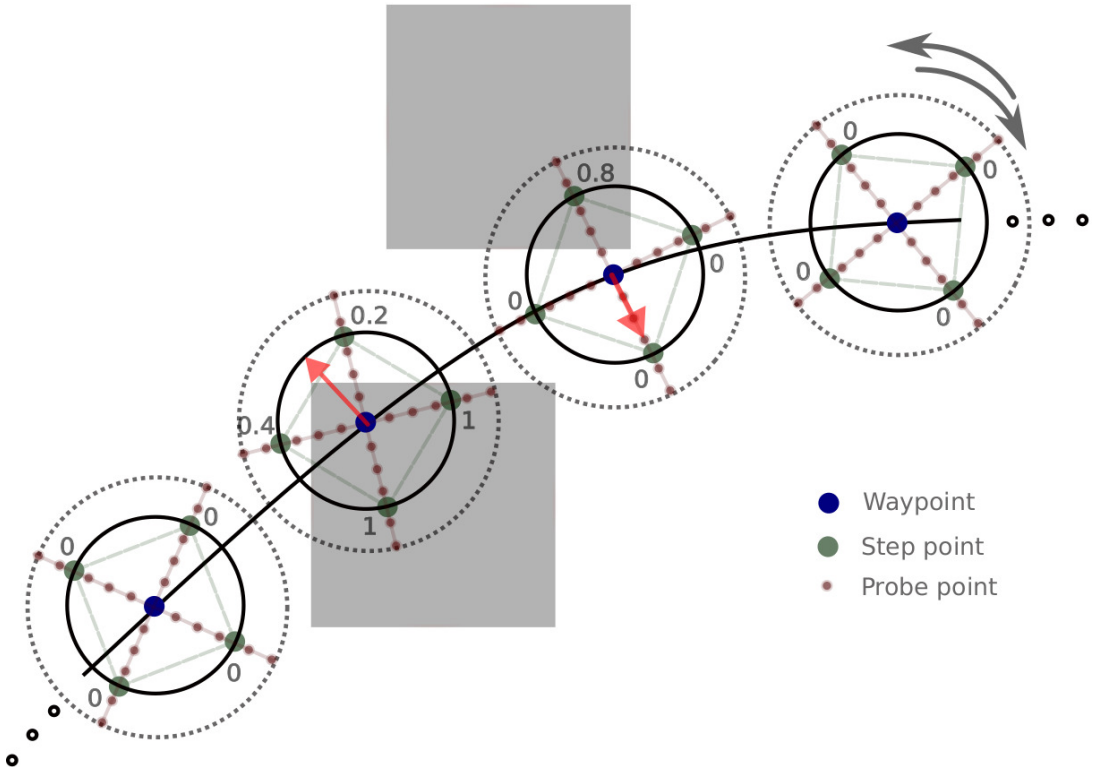


Figure 2.1.: Graphical illustration of Sinkhorn Step with practical considerations. In this point-mass example, we zoom-in one part of the discretized trajectory. The search-direction sets are constructed from randomly rotated 2-cube vertices at each iteration, depicted by the gray arrows and the green points. The gray numbers are the averaged costs over the red probe points in each vertex direction. Note that for clarity, we only visualize an occupancy obstacle cost. The red arrows describe the updates that transport the waypoints gradually out of the obstacles, depending on the (solid inner) polytope circumcircle α_k and (dotted outer) probe circle β_k. More demos can be found on `https://sites.google.com/view/sinkhorn-step/`

Algorithm 1: Motion Planning via Optimal Transport

$\mathcal{T}^0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{K}_0)$ and $\boldsymbol{n} = \mathbf{1}_N/N$, $\boldsymbol{m} = \mathbf{1}_m/m$
while *termination criteria not met* **do**
  (Optional) $\alpha \leftarrow (1-\epsilon)\alpha$, $\beta \leftarrow (1-\epsilon)\beta$ `// Epsilon Annealing`
  Construct randomly rotated D^P, H^P and compute the cost matrix $\boldsymbol{C}$ as in Eq. (2.10)
  Perform Sinkhorn Step $\mathcal{T} \leftarrow \mathcal{T} + \mathbf{S}$

2.4.3. Batch trajectory optimization

We leverage our Sinkhorn Step to optimize multiple trajectories in parallel, efficiently providing many feasible solutions for multi-modal planning problems. Specifically, we implement MPOT using PyTorch [21] for vectorization across different motion plans, randomly rotated polytope constructions, and *probe set* cost evaluations. For a problem instance, we consider N_p trajectories of horizon T, and thus, the trajectory set $\mathcal{T} = \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{N_p}\}$ is the parameter to be optimized. We can flatten the trajectories into the set of $N = N_p \times T$ waypoints. Now, the tensors of search directions and *probe set* $\boldsymbol{D}^P \in \mathbb{R}^{N \times m \times d}$, $\boldsymbol{H}^P \in \mathbb{R}^{N \times m \times h \times d}$ can be efficiently constructed and evaluated by the state cost function $c(\cdot)$, provided that the cost function is implemented with batch-wise processing (e.g., neural network models in PyTorch). Similarly, the model cost term in Eq. (2.9) can also be evaluated in batch by vectorizing the computation of the second term in Eq. (2.10).

At each iteration, it is optional to anneal the stepsize α_k and *probe radius* β_k. Often we do not know the Lipschitz constant L in practice, so the optimal stepsize cannot be computed. Hence, the Sinkhorn Step might oscillate around some local minima. It is an approximation artifact that can be mitigated by reducing the radius of the ball-search over time, gradually changing from an exploratory to an exploitative behavior. Annealing the ball search radius while keeping the number of probe points increases the chance of approximating better ill-conditioned cost structure, e.g., large condition number locally.

To initialize the trajectories, we randomly sample from the discretized GP prior $\mathcal{T}^0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{K}_0)$, where $\boldsymbol{\mu}_0$ is a constant-velocity, straight-line trajectory from start-to-goal state, and $\boldsymbol{K}_0 \in \mathbb{R}^{(T\times d)\times(T\times d)}$ is a large GP covariance matrix for exploratory initialization [91, 92] (cf. Appendix A.1.2). In practice, the initial GP covariance $\boldsymbol{K}_0$ is set higher than the covariance $\boldsymbol{K}$ in the cost, thus increasing the likelihood of covering more modes in the solution space. In execution, we select the lowest cost trajectory

$\boldsymbol{\tau}^* \in \mathcal{T}^*$. For collecting a trajectory dataset, all collision-free trajectories $\mathcal{T}^*$ are stored along with contextual data, such as occupancy map, goal state, etc. See Algorithm 1 for an overview of MPOT. Further discussions on batch trajectory optimization are in Appendix A.1.3.

2.5. Experiments

We experimentally evaluate MPOT in PyBullet simulated tasks, which involve high-dimensional state space, multiple objectives, and challenging costs. First, we benchmark our method against strong motion planning baselines in a densely cluttered 2D-point-mass and a 7-DoF robot arm (Franka Emika Panda) environment. Subsequently, we study the convergence of MPOT empirically. Finally, we demonstrate the efficacy of our method on high-dimensional mobile manipulation tasks with TIAGo++. Additional ablation studies on the design choices of MPOT, and gradient approximation capability on a smooth function of Sinkhorn Step w.r.t. different hyperparameter settings are in the Appendix A.1.9.

2.5.1. Experimental setup

In all experiments, all planners optimize first-order trajectories with positions and velocities in configuration space. The batch trajectory optimization dimension is $N \times T \times d$, where d is the full-state concatenating position and velocity.

For the *point-mass* environment, we populate 15 square and circle obstacles randomly and uniformly inside x-y limits of $[-10, 10]$, with each obstacle having a radius or width of 2 (cf. Fig. 1.4). We generate 100 environment-seeds, and for each environment-seed, we randomly sample 10 collision-free pairs of start and goal states, resulting in 1000 planning tasks. We plan each task in parallel 100 trajectories of horizon 64. A trajectory is considered successful if it is collision-free.

For the *Panda* environment, we also generate 100 environment-seeds. Each environment-seed contains randomly sampled 15 obstacle-spheres having a radius of 10cm inside the x-y-z limits of $[[-0.7, 0.7], [-0.7, 0.7], [0.1, 1.]]$, ensuring that the Panda's initial configuration has no collisions (cf. Appendix A.1.8). Then, we sample 5 random collision-free (including self-collision-free) target configurations, resulting in 500 planning tasks, and plan in parallel 10 trajectories containing 64 timesteps.

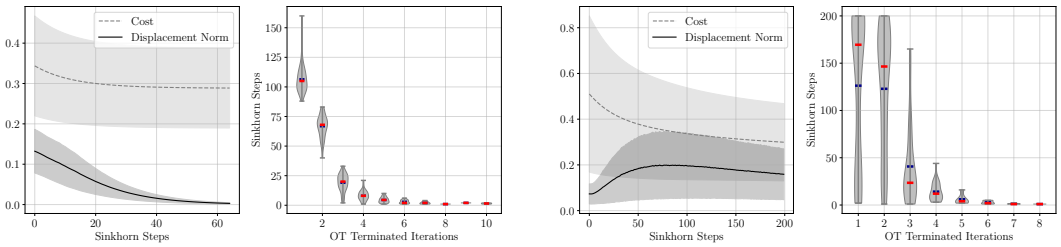
In the last experiment, we design a realistic high-dimensional mobile manipulation task in PyBullet (cf. Fig. 2.3). The task comprises two parts: the *fetch* part and *place* part; thus, it requires solving two planning problems. Each plan contains 128 timesteps, and we plan a single trajectory for each planner due to the high-computational and memory demands. We generate 20 seeds by randomly spawning the robot in the room, resulting in 20 tasks.

The motion planning costs are the $SE(3)$ goal, obstacle, self-collision, and joint-limit costs. The state dimension (configuration position and velocity) is $d = 4$ for the point-mass experiment, $d = 14$ for the Panda experiment, and $d = 36$ (3 dimensions for the base, 1 for the torso, and 14 for the two arms) for the mobile manipulation experiment. As for polytope settings, we choose a 4-cube for the point-mass case, a 14-othorplex for Panda, and a 36-othorplex for TIAGo++. Further experiment details are in Appendix A.1.8.

Baselines. We compare MPOT to popular trajectory planners, which are also straightforward to implement and vectorize in PyTorch for a fair comparison (even if the vectorization is not mentioned in their original papers). The chosen baselines are gradient-based planners: CHOMP [61] and GPMP2 (no interpolation) [62]; sampling-based planners: RRT* [30, 64] and its informed version I-RRT* [93], Stochastic Trajectory Optimization for Motion Planning (STOMP) [63], and the recent work Stochastic Gaussian Process Motion Planning (SGPMP) [67]. We implemented all baselines in PyTorch except for RRT* and I-RRT*, which are implemented in NumPy we plan with a loop using CPU.[2] We found that resetting the tree, rather than reusing it, is much faster for generating multiple trajectories; hence, we reset RRT* and I-RRT* when they find their first solution.

Metrics. Comparing various aspects among different types of motion planners is challenging. We aim to benchmark the capability of planners to parallelize trajectory optimization under dense environment settings. The metrics are T[s] - *planning time* until convergence; SUC[%] - *success rate* over tasks in an environment-seed, where success means there is at least one successful trajectory found each task; GOOD[%] - success percentage of total parallelized plans in each environment-seed, reflecting the *parallelization quality*; S - *smoothness* measured by the norm of the finite difference of trajectory velocities, averaged over all trajectories and horizons; PL - *path length*.

[2]To the best of our knowledge, instance-level vectorization of RRT* is non-trivial and still an open problem.



(a) (b)

Figure 2.2.: Convergence analysis of MPOT in Panda benchmark. The plots show the cost convergence when applying *step radius* annealing $\epsilon = 0.035$ and without. The plots imply that by following the Sinkhorn Steps, even without annealing, the cost converges exponentially (w.r.t. the update step size shown by the displacement norm). Slower convergence is observed without annealing. The right plots depict the number of iterations for solving the inner OT problem, whose stopping threshold is set at 10^{-5}. The mean and median of the violin plots are shown in blue and red, respectively. This shows that later iterations require fewer OT iterations, which attributes to the efficiency of MPOT.

Table 2.1.: Trajectory generation benchmarks in densely cluttered environments. RRT* and I-RRT* success and collision-free rates depict the maximum achievable values for all planners. S and PL statistics are computed on successful trajectories only.

	Point-mass Experiment					Panda Experiment				
	T[s]	SUC[%]	GOOD[%]	S	PL	T[s]	SUC[%]	GOOD[%]	S	PL
RRT*	43.2 ± 15.2	100 ± 0.	100 ± 0.	0.43 ± 0.12	23.8 ± 4.6	186.9 ± 184.2	100 ± 0.	73.8 ± 26.7	0.17 ± 0.05	7.8 ± 2.9
I-RRT*	43.6 ± 13.8	100 ± 0.	100 ± 0.	0.43 ± 0.11	23.9 ± 4.8	184.2 ± 166.0	100 ± 0.	74.6 ± 29.0	0.17 ± 0.05	7.6 ± 3.2
STOMP	2.2 ± 0.1	31.4 ± 13.9	10.5 ± 25.7	**0.01** ± 0.01	**17.0** ± 1.4	4.3 ± 0.1	50.8 ± 28.3	35.3 ± 42.0	**0.01** ± 0.0	**4.5** ± 0.8
SGPMP	6.5 ± 0.9	98.6 ± 4.5	**74.9** ± 28.9	0.03 ± 0.01	18.3 ± 2.0	5.0 ± 0.2	67.8 ± 23.5	58.1 ± 45.8	**0.01** ± 0.0	**4.5** ± 0.9
CHOMP	0.5 ± 0.1	70.9 ± 16.7	38.6 ± 40.7	0.03 ± 0.0	17.7 ± 1.7	3.1 ± 0.3	63.0 ± 25.5	51.6 ± 46.2	0.02 ± 0.0	4.6 ± 0.8
GPMP2	2.8 ± 0.1	98.3 ± 4.9	**74.9** ± 32.1	0.07 ± 0.05	20.3 ± 3.1	3.3 ± 0.2	66.0 ± 25.2	53.2 ± 42.3	**0.01** ± 0.0	4.9 ± 0.8
MPOT	**0.4** ± 0.0	**99.2** ± 3.1	73.6 ± 26.7	0.06 ± 0.03	19.3 ± 2.3	**0.8** ± 0.1	**71.6** ± 23.2	**60.2** ± 44.4	**0.01** ± 0.01	4.6 ± 0.9

2.5.2. Benchmarking results

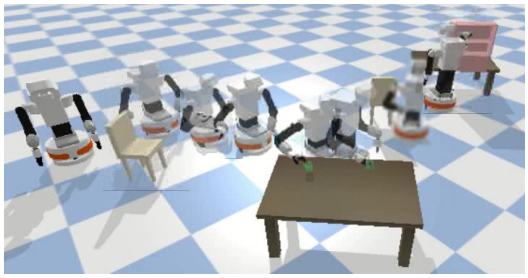
We present the comparative results between MPOT and the baselines in Table 2.1. While RRT* and I-RRT* achieve perfect results on success criteria, their planning time is dramatically high, which reconfirms the issues of RRT* in narrow passages and high-dimensional settings. Moreover, solutions of RRT* need postprocessing to improve smoothness. For GPMP2, the success rate is comparable but requires computational

effort. CHOMP, known for its limitation in narrow passages [94], requiring a small stepsize to work. This parallelization setting requires a larger step size for all trajectories to make meaningful updates, which incurs its inherent instability. With STOMP and SGPMP the comparison is "fairer," as they are both gradient-free methods. However, the sampling variance of STOMP is too restrictive, leading to bad solutions along obstacles near the start and goal configuration. Only SGPMP is comparable in success rate and parallelization quality. Nevertheless, we observe that tuning the proposal distribution variances is difficult in dense environments since they do not consider an obstacle model and cannot sample meaningful "sharp turns", hence requiring small update step size, more samples per evaluation, and longer iterations to optimize.

MPOT achieves better planning time, success rate, and parallelization quality, some with large margins, especially for the Panda experiments, while retaining smoothness due to the GP cost. We observe that MPOT performs particularly well in narrow passages, since waypoints across all trajectories are updated independently, thus avoiding the observed diminishing stepsize issue of the gradient-based planners in parallelization settings. Thanks to the explicit trust region property (cf. Appendix A.1.4), it is easier to tune the stepsize since it ensures that the update bound of each waypoint is the polytope convex hull. Notably, the MPOT planning time scales well with dimensionality. As seen in Section 2.5.1, solving OT is even more rapid at later Sinkhorn Steps; as the waypoints approach local minima, the OT cost matrix becomes more uniform and can be solved with only one or two Sinkhorn iterations. Empirically, we always observe MPOT converging to local minima.

2.5.3. Mobile manipulation experiment

We design a long-horizon, high-dimensional whole-body mobile manipulation planning task to stress-test our algorithm. This task requires designing many non-convex costs, e.g., signed-distance fields for gradient-based planners. Moreover, the task space is huge while the $SE(3)$ goal is locally small (i.e., the local grasp-pose, while having a hyper-redundant mobile robot, meaning the whole-body IK solution may be unreliable); hence, it typically requires long-horizon configuration trajectories and a small update step-size. Notably, the RRTs fail to find a solution, even with a very high time budget of 1000 seconds, signifying the environment's difficulty. These factors also add to the worst performance of GPMP2 in planning time (Fig. 2.3). Notably, CHOMP performs worse than GPMP2 and takes more iterations in a cluttered environment in Table 2.1. However, CHOMP beats GPMP2 in runtime complexity, in this case due to its simpler update rule. STOMP exploration mechanism is restrictive, and we could not tune it to



	TF[s]	SUC[%]	S	PL
RRT*	1000 ± 0.00	0	-	-
I-RRT*	1000 ± 0.00	0	-	-
STOMP	-	0	-	-
SGPMP	27.75 ± 0.29	25	**0.010** ± 0.001	**6.69** ± 0.38
CHOMP	16.74 ± 0.21	40	0.015 ± 0.001	8.60 ± 0.73
GPMP2	40.11 ± 0.08	40	0.012 ± 0.015	8.63 ± 0.53
MPOT	**1.49** ± 0.02	**55**	0.022 ± 0.003	10.53 ± 0.62

Figure 2.3.: (a) TIAGo++ fetches a cup from a table and places it on a shelf while avoiding chairs. (b) Mobile fetch & place results. TF[s] is the time to first solution. S and PL are computed from successful trials.

work in this environment. MPOT achieves much better planning times by avoiding the propagation of gradients in a long computational chain while retaining the performance with the efficient Sinkhorn Step, facilitating individual waypoint exploration. However, due to the sparsity of the 36-othorplex ($m = 72$) defining the search direction bases in this high-dimensional case, it becomes hard to balance success rate and smoothness when tuning the algorithm, resulting in worse smoothness than the baselines.

Limitations. MPOT is backed by experimental evidence that its planning time scales distinctively with high-dimensional tasks in the parallelization setting while optimizing reasonably smooth trajectories. Our experiment does not imply that MPOT should replace prior methods. MPOT has limitations in some aspects. First, the entropic-regularized OT has numerical instabilities when the cost matrix dimension is huge (i.e., huge number of waypoints and vertices). We use log-domain stabilization to mitigate this issue [89, 90]. However, in rare cases, we still observe that the Sinkhorn scaling factors diverge, and MPOT would terminate prematurely. Normalizing the cost matrix, scaling down the cost terms, and slightly increasing the entropy regularization λ helps. Second, on the theoretical understanding, we only perform preliminary analysis based on assumption 2.2 to connect directional-direct search literature. Analyzing Sinkhorn Steps in other conditions for better understanding, e.g., Sinkhorn Step gradient approximation analysis with arbitrary $\lambda > 0$, Sinkhorn Step on convex functions for sharper complexity bounds, etc., is desirable. Third, learning motion priors [67, 78] can naturally complement MPOT to provide even better initializations, as currently, we only use GP priors to provide random initial smooth trajectories.

2.6. Related Works

Motion optimization. While sampling-based motion planning algorithms have gained significant traction [29, 30], they are typically computationally expensive, hindering their application in real-world problems. Moreover, these methods cannot guarantee smoothness in the trajectory execution, resulting in jerky robot motions that must be post-processed before executing them on a robot [65]. To address the need for smooth trajectories, a family of gradient-based methods was proposed [61, 75, 62] for finding locally optimal solutions. These methods require differentiable cost functions, effectively requiring crafting or learning signed-distance fields of obstacles. CHOMP [61] and its variants [95, 96, 97] optimize a cost function using covariant gradient descent over an initially suboptimal trajectory that connects the start and goal configuration. However, such approaches can easily get trapped in local minima, usually due to bad initializations. Stochastic trajectory optimizers, e.g., STOMP [63] sample candidate trajectories from proposal distributions, evaluate their cost, and weigh them for performing updates [53, 67]. Although gradient-free methods can handle discontinuous costs (e.g., planning with surface contact), they may cause oscillatory behavior or failure to converge, requiring additional heuristics for acquiring better performance [98]. Schulman et al. [99] addresses the computational complexity of CHOMP and STOMP, which require fine trajectory discretization for collision checking, proposing a sequential quadratic program with continuous time collision checking. Gaussian Process Motion Planning (GPMP) [62] casts motion optimization as a probabilistic inference problem. A trajectory is parameterized as a function of continuous-time that maps to robot states, while a GP is used as a prior distribution to encourage trajectory smoothness, and a likelihood function encodes feasibility. The trajectory is inferred via Maximum-a-Posteriori (MAP) estimation from the posterior distribution of trajectories, constructed out of the GP prior and the likelihood function. In this work, we perform updates on waypoints across multiple trajectories concurrently. This view is also considered in methods that resolve trajectory optimization via collocation [100].

Optimal transport in robot planning. While OT has several practical applications in problems of resource assignment and machine learning [71], its application to robotics is scarce. Most applications consider swarm and multi-robot coordination [101, 102, 103, 104, 105], while OT can be used for exploration during planning [106] and for curriculum learning [107]. A comprehensive review of OT in control is available in [108]. Recently, Le et al. [109] proposed a method for re-weighting Riemannian motion policies [110] using an unbalanced OT at the high level, leading to fast reactive robot motion generation that effectively escapes local minima.

2.7. Conclusions and Broader Impacts

We presented MPOT—a gradient-free and efficient batch motion planner that optimizes multiple high-dimensional trajectories over non-convex objectives. In particular, we proposed the Sinkhorn Step—a zero-order batch update rule parameterized by a local optimal transport plan with a nice property of cost-agnostic step bound, effectively updating waypoints across trajectories independently. We demonstrated that in practice, our method converges, scales very well to high-dimensional tasks, and provides practically smooth plans. This work opens multiple exciting research questions, such as investigating further polytope families that can be applied for scaling up to even more high-dimensional settings, conditional batch updates, or different strategies for adapting the step-size. Furthermore, while classical motion planning considers single planning instance for each task, which under-utilizes the modern GPU capability, this work encourages future work that benefits from vectorization in the algorithmic choices, providing multiple plans and covering several modes, leading to execution robustness or even for dataset collection for downstream learning tasks. At last, we foresee potential applications of Sinkhorn Step to sampling methods or variational inference.

3. Global Tensor Motion Planning

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 MotionBenchMarker 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.

3.1. Introduction

Motion planning with probabilistic completeness has been a foundation of robotics research, with seminal works like PRM [29] and RRTConnect [30] serving as cornerstone methods for years [111]. 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 [78, 112, 113], (ii) the inherent non-linearity of task objectives that lead to multiple viable solutions [41, 62, 76], and (iii) the increasing availability of powerful GPUs/TPUs for accelerated planning [98, 52]. Despite these advances, batching traditional sampling-based planners, such as RRT/PRM and their variants, remains an ongoing challenge [35, 34, 114, 115]. Their underlying discretization techniques, such as the incremental graph construction of RRT/PRM or the search mechanism of A* [32, 116], are not conducive to efficient vectorization over planning instances.

This paper revisits classical motion planning, introducing 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

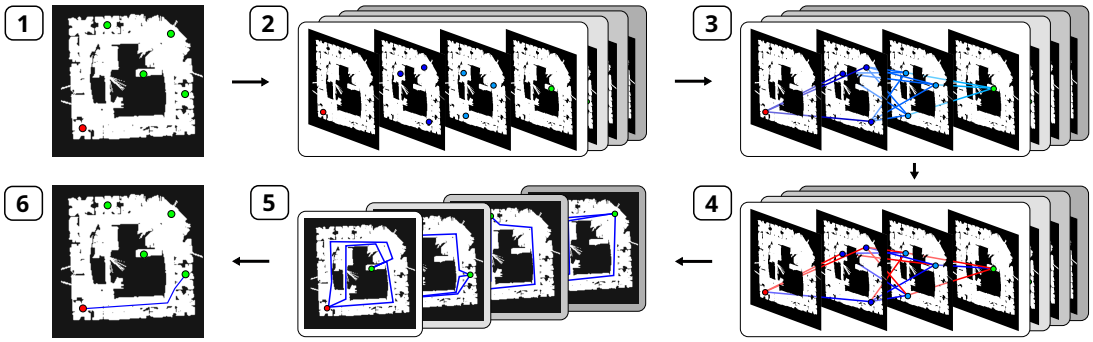


Figure 3.1.: GTMP can plan with multiple goals or `vmap` over goals. For clarity, we present an example of performing JAX `vmap` on GTMP (M=2, N=3) over the batch of $B = 3$ seeds. **(1)** The objective is to find a batch of feasible paths from the start (red) to the goals (green). **(2, 3)** In each seed, we sample a multipartite graph and form a tensor (Algorithm 2, Line 1). **(4)** A batch of collision checks is performed and stored into cost matrices (Algorithm 2, Line 2). **(5)** Then, per seed, we execute finite value iterations (Algorithm 2, Line 5-7) and trace the optimal path from the optimal value matrices (Algorithm 2, 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`.

collision checking and batch Value Iteration (VI) while maintaining an easily vectorizable implementation with JAX [20]. 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 [49], and ii) we extend GTMP with a spline discretization structure, enabling batch spline planning with path quality comparable to trajectory optimizers.

3.2. 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, earliest works [117, 118, 34, 115, 35] propose a *vectorizable* collision-checking data structure. State-of-the-art work on leveraging CPU-based *single instruction, multiple data* [37] (i.e., VAMP) has pushed collision-checking efficiency to *microseconds*. In a different vein, a body of works [119, 120, 121, 122, 123, 124] 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 [125], which are unsuitable for instance-level vectorization.

Vectorizing Trajectory Optimization. Vectorizing optimization-based planner with GPU-acceleration [126, 77, 98, 67, 41] 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 [52]. GTMP addresses this issue by proposing a layerwise discretization structure, enabling vectorization in sampling and search operations while having better global solutions.

3.3. Tensorizing Motion Planning

We consider the path planning problem [33] for a configuration $\boldsymbol{q}$ in compact space $\boldsymbol{q} \in \mathcal{C} \subset \mathbb{R}^d$ having d-dimensions, 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 $f : [0, 1] \to \boldsymbol{q}$, $\boldsymbol{f}(t) \in \mathcal{C}$, we can define its total variation as its arc length

$$\text{TV}(f) = \sup_{M \in \mathbb{N}, 0 = t_0, \ldots, t_M = 1} \sum_{i=1}^{M} \|\boldsymbol{f}(t_i) - \boldsymbol{f}(t_{i-1})\|, \tag{3.1}$$

Definition 3.1 (Feasible Path)**.** *The function $f : [0, 1] \to \boldsymbol{q}$ 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, invalid configurations that violate collision constraints, and configuration limits.

Problem 3.1 (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 path f 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 [20]. This approach contrasts with the incremental discretization structures of classical motion planning algorithms, which procedurally expand the search space during planning.

3.3.1. Discretization Structure

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

Definition 3.2 (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 $\mathcal{M}$*

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

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. 3.1). 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 3.2 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 [50] (cf. Section 3.4).

Definition 3.3 (Path In G)**.** *A path $f : [0, 1] \to \boldsymbol{q}$ in G exists if it $\boldsymbol{f}(0) = \boldsymbol{q}_0$, $\boldsymbol{f}(1) \in \mathcal{G}$ and its piecewise linear segments correspond to edges connecting $\boldsymbol{q}_0$ and $\boldsymbol{q}_g \in \mathcal{G}$.*

3.3.2. State Machine On Graph

The graph G is represented by the state machine $(\mathcal{V}, \mathcal{E}, c, t)$ [127], 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

$$c_{\text{coll}}(\boldsymbol{q}) = 0 \text{ if } \boldsymbol{q} \in \text{int}_\delta(\mathcal{C}_{\text{free}}), \text{ else } \infty, \tag{3.2}$$

which merges the planning and verification steps (cf. Proposition 3.2). Then, the transition cost function can be defined

$$c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) = \underbrace{\int_a^b c_{\text{coll}}(\boldsymbol{f}(t)) f' dt}_{\text{collision}} + \underbrace{\|\boldsymbol{q} - \boldsymbol{q}'\|}_{\text{smoothness}}, \tag{3.3}$$

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

$$\begin{aligned} v_G(\boldsymbol{q}) &\leftarrow \min_{(\boldsymbol{q}, \boldsymbol{q}')} \sum_{\boldsymbol{q}'} t(\boldsymbol{q}' \mid \boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) \left(c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) + v_G(\boldsymbol{q}') \right) \\ &\leftarrow \min_{(\boldsymbol{q}, \boldsymbol{q}')} \left(c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) + v_G(\boldsymbol{q}') \right) \end{aligned} \tag{3.4}$$

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

$$\pi^*(\boldsymbol{q}) = \arg\min_{(\boldsymbol{q}, \boldsymbol{q}')} \left(c(\boldsymbol{q}, (\boldsymbol{q}, \boldsymbol{q}')) + v_G^*(\boldsymbol{q}') \right), \tag{3.5}$$

from $\boldsymbol{q}_0$ until $\boldsymbol{q}' \in \mathcal{G}$ [127]. 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 3.1. *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. (3.5) 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 [128]. 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.

3.3.3. 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 $\boldsymbol{C}_s \in \mathbb{R}^N$, $\boldsymbol{C}_h \in \mathbb{R}^{(M-1)\times N\times N}$, $\boldsymbol{C}_l^{N\times|\mathcal{G}|}$ 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. (3.3) 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 2. 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 [35, 34, 114, 37] focus on vectorizing collision checking or forward kinematics in a single planning instance, while we can ensure that Algorithm 2 can be vectorized at the instance-level [41] by Proposition 3.1.

Complexity Analysis. The Bellman matrix update (Line 5-7) is an asynchronous update in batches (i.e., updates based on values of the previous iteration) and also known to converge [129]. 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 2. 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. Theoretical investigations regarding GTMP probabilistic completeness are presented in Section 3.6.

Algorithm 2: Global Tensor Motion Planning

Input: Start $\boldsymbol{q}_0$, Goals $\boldsymbol{G} \in \mathbb{R}^{|\mathcal{G}| \times d}$

Uniformly sample $\boldsymbol{Q} \in \mathbb{R}^{M \times N \times d}$ on $\mathcal{C}$.

Compute cost matrices $\boldsymbol{C}_s, \boldsymbol{C}_h, \boldsymbol{C}_l$ as Eq. (3.3)

`// Value iteration as planning on` $(M+2)$`-partite graph`

Init $V_s \in \mathbb{R}$, $\boldsymbol{V}_h \in \mathbb{R}^{M \times N}$, $\boldsymbol{V}_g \in \mathbb{R}^{|\mathcal{G}|}$

for $1 \leq k \leq M+1$ **do**

  $\boldsymbol{V}_h[M-1] \leftarrow \min(\boldsymbol{C}_l + \boldsymbol{V}_g)$

  $\boldsymbol{V}_h[:M-1] \leftarrow \min(\boldsymbol{C}_h + \boldsymbol{V}_h[1:], \text{axis} = -1)$

  $V_s \leftarrow \min(\boldsymbol{C}_s + \boldsymbol{V}_h[0])$

`// Extract the optimal path by tracing over layers`

$i \leftarrow \arg\min(\boldsymbol{C}_s + \boldsymbol{V}_h^*[0])$

$\mathcal{P} = \{i\}$

for $1 \leq m \leq M-1$ **do**

  $i \leftarrow \arg\min(\boldsymbol{C}_h[m-1, i] + \boldsymbol{V}_h^*[m])$

  Append $\boldsymbol{Q}[m, i]$ to $\mathcal{P}$

$i \leftarrow \arg\min(\boldsymbol{C}_l[i] + \boldsymbol{V}_g)$ and append $\boldsymbol{G}[i]$ to $\mathcal{P}$

Output: $\mathcal{P}$

3.4. Extension: Akima Spline

The Akima spline [50] 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 3.4 (Akima Spline)**.** *Given a point set* $\{\boldsymbol{q}_i | \boldsymbol{q} \in \mathcal{C}\}_{i=1}^{P}$*, the Akima spline constructs a piecewise cubic polynomial* $f(t)$ *for each interval* $[t_i, t_{i+1}]$

$$f_i(t) = \boldsymbol{d}_i(t - t_i)^3 + \boldsymbol{c}_i(t - t_i)^2 + \boldsymbol{b}_i(t - t_i) + \boldsymbol{a}_i, \tag{3.6}$$

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}

$$\boldsymbol{s}_i = \frac{|\boldsymbol{m}_{i+1} - \boldsymbol{m}_i|\boldsymbol{m}_{i-1} + |\boldsymbol{m}_{i-1} - \boldsymbol{m}_{i-2}|\boldsymbol{m}_i}{|\boldsymbol{m}_{i+1} - \boldsymbol{m}_i| + |\boldsymbol{m}_{i-1} - \boldsymbol{m}_{i-2}|}. \tag{3.7}$$

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*

$$\begin{aligned} \boldsymbol{a}_i &= \boldsymbol{q}_i, \quad \boldsymbol{b}_i = \boldsymbol{s}_i, \\ \boldsymbol{c}_i &= (3\boldsymbol{m}_i - 2\boldsymbol{s}_i - \boldsymbol{s}_{i+1})/(t_{i+1} - t_i), \\ \boldsymbol{d}_i &= (\boldsymbol{s}_i + \boldsymbol{s}_{i+1} - 2\boldsymbol{m}_i)/(t_{i+1} - t_i)^2. \end{aligned} \tag{3.8}$$

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 4.1 to *a spline discretization structure.*

Definition 3.5 (Akima Spline Graph)**.** *Given a geometric graph* $G = (\mathcal{V}, \mathcal{E})$ *(cf. Definition 4.1), 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* $\boldsymbol{m}_{m,i,j} = (\boldsymbol{q}_{m+1,j} - \boldsymbol{q}_{m,i})/(t_{i+1} - t_i)$ *as Modified Akima interpolation [50]*

$$\begin{aligned} \boldsymbol{s}_{m,i,j} &= \frac{\boldsymbol{w}_{m,i,j}\boldsymbol{m}_{m-1,i,j} + \boldsymbol{w}_{m-1,i,j}\boldsymbol{m}_{m,i,j}}{\boldsymbol{w}_{m,i,j} + \boldsymbol{w}_{m-1,i,j}} \\ \boldsymbol{w}_{m,i,j} &= \left| \frac{1}{N^2} \sum_{i,j} \boldsymbol{m}_{m+1,i,j} - \boldsymbol{m}_{m,i,j} \right| + \frac{1}{2} \left| \frac{1}{N^2} \sum_{i,j} \boldsymbol{m}_{m+1,i,j} + \boldsymbol{m}_{m,i,j} \right| \\ \boldsymbol{w}_{m-1,i,j} &= \left| \boldsymbol{m}_{m-1,i,j} - \frac{1}{N^2} \sum_{i,j} \boldsymbol{m}_{m-2,i,j} \right| + \frac{1}{2} \left| \boldsymbol{m}_{m-1,i,j} + \frac{1}{N^2} \sum_{i,j} \boldsymbol{m}_{m-2,i,j} \right|. \end{aligned} \tag{3.9}$$

Then, the augmented cubic polynomial $f_{i,j}(t), t \in [t_m, t_{m+1}]$ *is computed following Eq.* (3.8)

$$\begin{aligned} \boldsymbol{s}_m &= \frac{1}{N^2} \sum_{i,j} \boldsymbol{s}_{m,i,j}, \, \boldsymbol{a}_{m,i,j} = \boldsymbol{q}_{m,i}, \, \boldsymbol{b}_{m,i,j} = \boldsymbol{s}_m, \\ \boldsymbol{c}_{m,i,j} &= (3\boldsymbol{m}_{m,i,j} - 2\boldsymbol{s}_m - \boldsymbol{s}_{m+1})/(t_{m+1} - t_m), \\ \boldsymbol{d}_{m,i,j} &= (\boldsymbol{s}_m + \boldsymbol{s}_{m+1} - 2\boldsymbol{m}_{m,i,j})/(t_{m+1} - t_m)^2. \end{aligned} \tag{3.10}$$

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 [50] 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. (3.10) 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

$$c(\boldsymbol{q}, \boldsymbol{q}') = \int_a^b \left(c_{\text{coll}}(f(t)) + 1\right) \|f'(t)\| dt, \tag{3.11}$$

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

Remark 3.1. *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*

$$\begin{aligned} \boldsymbol{f}_{i,j}(t_m) &= \boldsymbol{q}_{m,i},\ \boldsymbol{f}_{i,j}(t_{m+1}) = \boldsymbol{q}_{m+1,j}, \\ \boldsymbol{f}'_{i,j}(t_m) &= \boldsymbol{s}_m,\ \boldsymbol{f}'_{i,j}(t_{m+1}) = \boldsymbol{s}_{m+1}, \end{aligned} \tag{3.12}$$

for any $m \in \{0, \ldots, M+1\}$, $i, j \in \{1, \ldots, N\}$. 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 3.1 does not necessarily hold for Akima Spline Graph G_A and is left for future work.

3.5. 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 [49, 130] or in VAMP [37]?, ii) how does GTMP-Akima compare to popular gradient-based smooth trajectory optimizers such as CHOMP [94] or GPMP [62]?, and iii) Are the empirical results consistent with the theoretical guarantees (Theorem 3.1)?

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 [20], 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 [67] 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 [73] distances in a batch having B paths

$$PD = \frac{1}{B(B-1)} \mathrm{OT}_\lambda(\mathcal{P}_i, \mathcal{P}_j),\ i, j \in \{1, \dots, B\}, \tag{3.13}$$

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 $\lambda = 5e^{-3}$ 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.

3.5.1. Batch Planning Comparison

Fig. 3.2 (top-row) compares GTMP and GTMP-Akima with OMPL implementation of (single-query) RRTConnect [30] and BKPIECE [51]. The environments are planar occupancy maps of Intel Lab, ACES3 Austin, Orebro, Freiburg Campus, and Seattle UW Campus generated from the Radish dataset [131]. The maps are chosen to include narrow passages, large spaces, and noisy occupancies (cf. Fig. 3.1). We randomly sample 100 start-goal pairs as tasks on each map and plan 100 paths per task. We clearly see 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

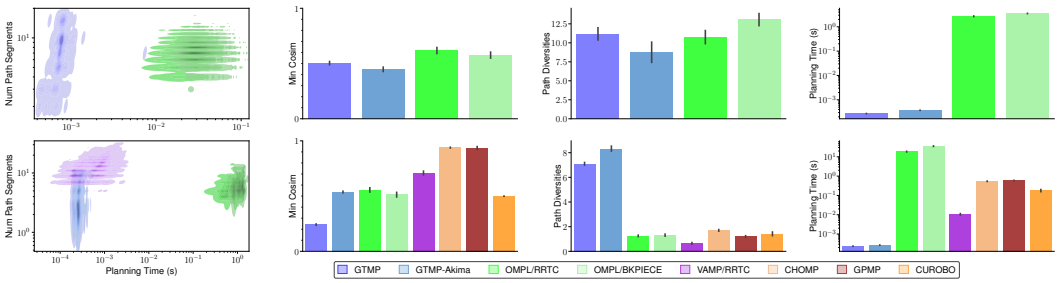


Figure 3.2.: Aggregated statistics of comparison experiments on Planar Occupancy (top-row) and Panda MBM dataset (bottom-row). We note the log scale on the Planning Time axes. The batch planning time is the sum of instance time for sequential planners (last column). All plotted data points are based on successful path statistics.

over maps. With JIT and GPU utilization, GTMP consistently produces batch paths with a fixed number of segments and x10000 less wall-clock time compared to baselines across maps.

Table 3.1.: 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 [37]	**0.09**	**100.0**	3.6	**0.1**	-
cuRobo [52]	43.1	99.7	**2.8**	0.0	-

We choose the MotionBenchMaker (MBM) dataset [132] 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 [37] [1]. CHOMP and GPMP plan

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

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.

Fig. 3.2 (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. 3.3). 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 x50 faster than state-of-the-art VAMP/RRTC implementation while being x2500 faster than CHOMP/GPMP/cuRobo and x100000 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 3.2 (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 [37]. GTMP vectorizes planning via layered structure, resulting in predictable narrow distribution due to fixed-segment path planning.

3.5.2. Single Plan Comparison

We compare GTMP and GTMP to the strong baselines such as VAMP/RRTC [37] and cuRobo [52], in terms of single planning for execution, on the MπNets dataset [40] 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 3.1 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.

3.5.3. Ablation Study

Probabilistic Completeness Ablation. This section explores various aspects of GTMP by sweeping the number of layer M and number of points per layer N. Fig. 3.3 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 3.1. In Fig. 3.3, 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 3.3). Then, the CF(%) heatmap directly reflects the path existence probability Eq. (3.16). 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 3.6. 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.

GTMP Hyperparameter Characteristics. In Table 3.2, we ask how many tasks in the Panda MBM dataset can be solved with just one or more layers. Note that $M = 1$ is a trivial realization of GTMP (i.e., VI is not required at $M = 1$). Solving a task means at least one collision-free path exists in a batch of $B = 50$ paths per problem. We report for GTMP and GTMP-Akima the aggregated similar planning times of $272 \pm 80\mu s$ across hyperparameters. Although not comparable to CHOMP and GPMP in smoothness, one-layer GTMP solves many tasks due to maximum exploration characteristics, compared to local methods such as trajectory optimizers requiring gradients.

3.5.4. Real-world experiment

To validate GTMP in real settings, we conducted a real-world experiment with a UR5 performing reactive obstacle avoidance in the presence of a moving obstacle. This experiment demonstrates GTMP's reactive capability to operate as a high-frequency replanner under strict time constraints.

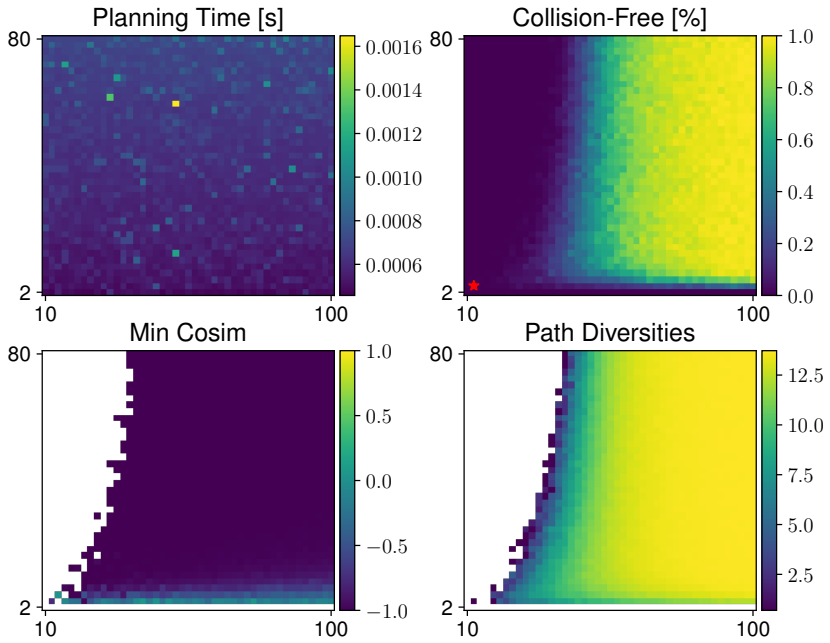


Figure 3.3.: For each M (y-axis), N (x-axis), we set the number of probing $H = 30$ and plan the batch of $B = 200$ paths. The red star denotes the minimum number of layers M_m, corresponding to the minimum requirement of N to discover some solutions experimentally.

Experiment Setup. We consider an UR5 robot arm operating in a constrained tabletop workspace. We use two cylinder sticks as moving obstacles, whose poses are estimated by a motion tracker updated at 380Hz. The goal of the robot was to move its end-effector between two fixed poses on opposite sides of the table while avoiding the obstacle, which moved unpredictably at a moderate speed.

We use GTMP-Akima as the global planner in a loop. At each iteration, GTMP-Akima directly recomputes the global path using the current configuration space, performing a full batch search ($B = 20$) with updated collision costs. We select the shortest path in the resulting collision-free Akima spline batch. Then, given a constant time discretization and path duration, we send the path derivatives to the low-level velocity controller for execution. We use ($M = 4, N = 50, H = 20$) and all planning computations were performed on an NVIDIA RTX 4090.

Table 3.2.: Success Rate Ablation Over Motion Bench Maker Dataset

Algorithms	*bookshelf_small*	*bookshelf_tall*	*bookshelf_thin*	*box*	*cage*	*table_pick*	*table_under_pick*
GTMP (N=200, M=1)	**100**	**100**	**100**	**100**	57	62	**100**
GTMP-Akima (N=200, M=1)	85	94	78	95	**83**	91	29
GTMP (N=30, M=2)	**100**	**100**	**100**	**100**	26	**100**	**100**
GTMP-Akima (N=30, M=2)	91	98	86	96	67	95	67
GTMP (N=30, M=5)	100	100	100	100	25	100	100
GTMP-Akima (N=30, M=5)	82	99	89	96	63	95	67
CHOMP	83	96	98	76	0	93	22
GPMP	56	72	47	16	51	51	0

Throughout a 3-minute experiment, GTMP-Akima maintained a consistent average runtime of 1.1 ms per replanning cycle (approximately 900Hz), comfortably within the real-time requirements of reactive control. The robot successfully avoided the moving obstacle with no collisions observed. GTMP-Akima produced diverse homotopy classes as the obstacle moved into different regions of the workspace, demonstrating GTMP's ability to generate multiple valid solutions under dynamic conditions.

3.6. Theoretical Analysis

We formally investigate the probabilistic completeness property of GTMP under linear interpolation. Further investigation with other spline structures is deferred for future work.

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

Assumption 3.1. *We assume that all associated proposal distributions at each layer are uniformly distributed on the configuration space* $\forall 1 \leq m \leq M$, $p_m := \mathcal{U}(\mathcal{C})$.

Assumption 3.2. *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 applications, where the free-path set is not zero-measure $\mu(\mathcal{F}_{\text{free}}) \neq 0$.

Proposition 3.2 (Feasibility Check)**.** *For any planning problem, $v_G^*(\boldsymbol{q}_0) < \infty$ if and only if there exists a feasible path in G.*

Proof. According to Bellman optimality, $v_G^*(\boldsymbol{q}_0) = \min_g\{c(g) \mid g \in \mathcal{R}\}$ is the minimum path cost reaching the goals. By definition, the smoothness term in Eq. (3.3) is bounded with $\forall \boldsymbol{q} \in \mathcal{C}$, since $\mathrm{TV}(g) < \infty$. Thus, any unbounded path cost $c(\mathcal{P}) = \infty$ occurs, if and only if $\exists t$, $\boldsymbol{f}(t) \in \mathcal{C}_{\text{coll}}$. Hence, $v_G^*(\boldsymbol{q}_0) = \min_{\mathcal{P}}\{c(\mathcal{P}) \mid \mathcal{P} \in \mathcal{R}\} < \infty$, if and only if $\exists \mathcal{P} \in \mathcal{R}$, $c(\mathcal{P}) < \infty$. □

Proposition 3.2 is useful to filter collided paths after VI.

Lemma 3.1 (Solvability In Finite Path Segments)**.** *If Assumption 3.2 holds, there exists a minimum number of segments $M_m \in \mathbb{N}_{>0}$ for piecewise linear paths to be feasible.*

Proof. We first show that there exists a piecewise linear path $g : [0, 1] \to \mathcal{C}$ such that $\|f - g\|_\infty < r$, where $\|f - g\|_\infty = \sup_{t\in[0,1]} \|\boldsymbol{f}(t) - \boldsymbol{g}(t)\|$. We construct g by dividing the interval $[0, 1]$ into M subintervals with length less than $\delta > 0$, i.e., $[t_0, t_1], \ldots, [t_{M-1}, t_M]$ with $0 = t_0 < t_1 < \ldots < t_M = 1$. On each subinterval $[t_m, t_{m+1}]$, we define the corresponding segment of g to approximate f

$$\boldsymbol{g}(t) = \boldsymbol{f}(t_m) + \frac{\boldsymbol{f}(t_{m+1}) - \boldsymbol{f}(t_m)}{t_{m+1} - t_m}(t - t_m),\ t \in [t_m, t_{m+1}]. \tag{3.14}$$

Since by definition the path f is continuous on a compact interval $[0, 1]$, then by Heine-Cantor theorem, f is also uniformly continuous, i.e., $\exists \delta > 0$ for any $a, b \in [0, 1]$, $|a - b| < \delta$, then $\|\boldsymbol{f}(a) - \boldsymbol{f}(b)\|_\infty < r$. Then, by the construction of g and uniform continuity of f, we can choose a δ sufficiently small such that $\|f - g\|_\infty < r$. This implies that there exists a sufficiently large number of segments M_m such that δ is sufficiently small, hence, g is a feasible path. □

Lemma 3.1 implies that any path planning algorithm producing a piecewise linear feasible path, then it must have a minimum number of segments.

Lemma 3.2. *Let piecewise linear path $g : [0, 1] \to \mathcal{C}$ having n equal subintervals approximating a path $f : [0, 1] \to \mathcal{C}$. The error lowerbound is $\|f - g\|_\infty > L/n$, where $L = TV(f)$ is the total variation of f.*

Proof. Denoting the subinterval length $u = 1/n$ and reusing the notations from Lemma 3.1 proof, we define g as a piecewise linear function 3.14. Since f, g are uniformly continuous, the linear interpolation error lower bound can be expressed using the modulus of continuity on a segment $t \in [t_m, t_{m+1}]$

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

And, the global error over all segments is

$$\|\boldsymbol{f}(t) - \boldsymbol{g}(t)\|_\infty = \max_{0\leq m\leq n-1} \sup_{t\in[t_m,t_{m+1}]} \|\boldsymbol{f}(t) - \boldsymbol{g}(t)\|$$

By definition, f is uniformly continuous and of bounded variation, the modulus of continuity $\omega_f(u)$ provides a lower bound for the error on each segment. Therefore, $\|f - g\|_\infty > \omega_f(1/n)$ on $[0, 1]$. For functions of bounded variation, the modulus of continuity can be bounded in terms of the total variation $\omega_f(1/n) > L/n$ on $[0, 1]$. Hence, $\|f - g\|_\infty > L/n$. □

Lemma 3.3. *Let g_1, g_2 be a piecewise linear function having the same number of partition points $\{\boldsymbol{g}_1(t_m)\}_{m=0}^M, \{\boldsymbol{g}_2(t_m)\}_{m=0}^M$ with $0 = t_0 < \ldots, t_M = 1$, $\|g_1 - g_2\|_\infty < \delta$, if and only if $\|\boldsymbol{g}_1(t_m) - \boldsymbol{g}_2(t_m)\| < \delta$, $0 \leq m \leq M$.*

Proof. **Sufficiency.** Given $\|\boldsymbol{g}_1(t_m) - \boldsymbol{g}_2(t_m)\| < \delta, \forall 1 \leq m \leq M$, since g_1, g_2 are piecewise linear functions, the linear interpolation between partition points t_m, t_{m+1} ensures that the difference between g_1, g_2 is maximized at the partition points. Consider g_1, g_2 on a segment $[t_m, t_{m+1}]$

$$\begin{aligned}\|\boldsymbol{g}_1(t) - \boldsymbol{g}_2(t)\| \leq &\max\{\|\boldsymbol{g}_1(t_m) - \boldsymbol{g}_2(t_m)\|,\\ &\|\boldsymbol{g}_1(t_{m+1}) - \boldsymbol{g}_2(t_{m+1})\|\} < \delta\end{aligned} \tag{3.15}$$

Hence, $\|g_1 - g_2\|_\infty = \max_{t\in[0,1]} \|\boldsymbol{g}_1(t) - \boldsymbol{g}_2(t)\| < \delta$.

Necessity. Given $\|g_1 - g_2\|_\infty < \delta$, then $\|\boldsymbol{g}_1(t_m) - \boldsymbol{g}_2(t_m)\| < \delta$, $0 \leq m \leq M$. □

Theorem 3.1 (Probabilistic Completeness)**.** *If Assumption 3.1 and Assumption 3.2 hold, for a feasible planning problem $(\mathcal{C}_{free}, \boldsymbol{q}_0, \mathcal{G})$, with G having $M \geq M_m$ layers, there exist constants $a, R, L > 0$ depending only on $\mathcal{C}_{free}$ and $\mathcal{G}$, such that*

$$\mathbb{P}\left(v_G^*(\boldsymbol{q}_0) < \infty\right) > 1 - M \exp\left(-a\left(R - \frac{L}{M+1}\right)^d N\right). \tag{3.16}$$

Proof. From Lemma 3.1, if $M \geq M_m$, there exists a feasible piecewise linear path g having $M+1$ segments with $0 = t_0 < \ldots < t_{M+1} = 1$ approximating a feasible path f. Let $R = \inf_{t\in[0,1]} \{\|\boldsymbol{f}(t) - \boldsymbol{q}\|, \boldsymbol{q} \in \mathcal{C}_{\text{coll}}\}, r = \inf_{t\in[0,1]} \{\|\boldsymbol{g}(t) - \boldsymbol{q}\|, \boldsymbol{q} \in \mathcal{C}_{\text{coll}}\}$ be collision margins of f, g, $\mathcal{B}_\delta(\boldsymbol{q}) = \{\|\boldsymbol{q}' - \boldsymbol{q}\| < \delta,\, \delta > 0\}$ is an open δ-ball around $\boldsymbol{q}$. Now, let g have equal subinterval.

First, we compute the probability of the event that a sampled graph G has at least a piecewise linear path h with $M+1$ segments such that h approximates g. h is feasible when $\|h - g\|_\infty < r$. We have

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

where the first inequality due to $\mathcal{C} \subset \mathbb{R}^d$, and last inequality from Lemma 3.2 and $L = \text{TV}(f)$.

From Lemma 3.3, since by definition h, g has the same number of segments, the event $\|h - g\|_\infty < r < r_h = R - \frac{L}{M+1}$ is the event that, given start and goals fixed, for each layer $1 \leq m \leq M$, there is at least one point $\boldsymbol{h}(t_m)$ is sampled inside the ball $\mathcal{B}_{r_h}(\boldsymbol{g}(t_m))$. Then, by sampling N points uniformly over $\mathcal{C}$ per layer (Assumption 3.1), and the fact that there are pairwise connections between layers, we have the failing probability

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

where we use the inequality $1 - x \leq e^{-x}$, $x \geq 0$, and $a = \alpha_d/\mu(\mathcal{C})$, where α_d is the constant term computing volume of a d-ball.

The event of h approximating g having equal intervals is a subset of the event of h approximating g having arbitrary intervals. The event that at least a path h in G having $\|h - g\|_\infty < r_h$ is a subset of the event $\exists$a feasible path in G, since

there might exist multiple feasible paths and their corresponding piecewise linear approximations have $M+1$ segments. From Proposition 3.2, $v_G^*(\boldsymbol{q}_0) < \infty$ is equivalent to $\exists$ a feasible path in G. We have

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

□

The lower bound is intuitive since it directly implies a minimum number of layers $M > [L/R] - 1$ (cf. Lemma 3.1) for the exponent coefficient to be strictly positive. It also implies the existence of an optimal number M^*; increasing M helps then harms N sample efficiency, depending on the planning problem (cf. Fig. 3.3).

3.7. 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 set (M, N, H)), and *easy to incorporate* motion planning objectives in Eq. (3.11).

GTMP is designed to be efficient in batch planning representing multiple instances of the same planning problem. The batch dimension representing the multiple GTMP planning instances can be interpreted as multiple replanning attempts. Indeed, Theorem 3.1 depicts probabilistic completeness over the batch dimension and M, N, contrasting with probabilistic completeness over exploration nodes as in RRT* [64]. Beyond GTMP, since Algorithm 2 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 [37] 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.

4. Model Tensor Planning

Sampling-based model predictive control (MPC) offers strong performance in nonlinear and contact-rich robotic tasks, yet often suffers from poor exploration due to locally greedy sampling schemes. We propose *Model Tensor Planning* (MTP), a novel sampling-based MPC framework that introduces high-entropy control trajectory generation through structured tensor sampling. By sampling over randomized multipartite graphs and interpolating control trajectories with B-splines and Akima splines, MTP ensures smooth and globally diverse control candidates. We further propose a simple β-mixing strategy that blends local exploitative and global exploratory samples within the modified Cross-Entropy Method (CEM) update, balancing control refinement and exploration. Theoretically, we show that MTP achieves asymptotic path coverage and maximum entropy in the control trajectory space in the limit of infinite tensor depth and width.

Our implementation is fully vectorized using JAX and compatible with MuJoCo XLA, supporting *Just-in-time* (JIT) compilation and batched rollouts for real-time control with online domain randomization. Through experiments on various challenging robotic tasks, ranging from dexterous in-hand manipulation to humanoid locomotion, we demonstrate that MTP outperforms standard MPC and evolutionary strategy baselines in task success and control robustness. Design and sensitivity ablations confirm the effectiveness of MTP's tensor sampling structure, spline interpolation choices, and mixing strategy. Altogether, MTP offers a scalable framework for robust exploration in model-based planning and control.

4.1. Introduction

Sampling-based Model Predictive Control (MPC) [31, 133, 53] has emerged as a powerful framework for controlling nonlinear and contact-rich systems. Unlike gradient-based or linearization approaches, sampling-based MPC is model-agnostic and does not require differentiable dynamics, making it well-suited for high-dimensional, complex systems such as legged robots [134] and dexterous manipulators [135]. Moreover, its inherent parallelism enables efficient deployment on modern hardware (e.g., GPUs), allowing for high-throughput simulation and online domain randomization in real-time control pipelines [43].

Despite these advantages, a fundamental limitation remains: sampling-based MPC is typically local in its search behavior. Most methods perturb a nominal trajectory or refine the current best samples, which makes them susceptible to local minima and unable to consistently discover globally optimal solutions [136]. While the Cross-Entropy Method (CEM) [137] has shown promise in high-dimensional control and sparse-reward settings [138], it still suffers from mode collapse when sampling locally [139], leading to suboptimal behaviors. The curse of dimensionality exacerbates this issue, as the number of samples required to explore control spaces grows exponentially with the planning horizon and control dimension, posing a bottleneck if compute or memory is limited. These challenges motivate the need for a more effective, high-entropy sampling mechanism for control generation.

Evolutionary Strategies (ES) [140, 141, 55] have also been applied in sampling-based MPC settings to improve sampling exploration. While they improve over purely local strategies in some tasks, our experiments (cf. Section 4.4.1) reveal that ES still fails to systematically explore multimodal control landscapes, often yielding inconsistent performance on tasks requiring long-term coordinated actions.

In this work, we introduce *Model Tensor Planning* (MTP), a novel sampling-based MPC framework that enables globally exploratory control generation through structured tensor sampling. MTP reformulates control sampling as tensor operations over randomized multipartite graphs, enabling efficient generation of diverse control sequences with high entropy. To balance exploration and exploitation, we propose a simple yet effective β-mixing mechanism that combines globally exploratory samples with locally exploitative refinements. We also provide a theoretical analysis under bounded-variation assumptions, showing that our sampling scheme achieves asymptotic path coverage, approximating maximum entropy in trajectory space.

MTP is designed with real-time applicability with matrix-based formulation, which is compatible with *Just-in-time* (JIT) compilation and vectorized mapping (e.g., via JAX `vmap` [20]), allowing high-throughput sampling, batch rollout evaluation, and online domain randomization on modern simulators. Our main contributions are as follows:

- We propose *tensor sampling*, a novel structured sampling strategy for control generation, and provide theoretical justification via asymptotic path coverage.
- We introduce a simple β-mixing mechanism that effectively balances exploration and exploitation by blending high-entropy and local samples within the modified CEM update rule.

- We demonstrate that MTP is highly compatible with modern vectorized simulators, enabling efficient batch rollout evaluation and robust real-time control in high-dimensional, contact-rich environments.

4.2. Preliminary

Notations and Assumptions. We consider the problem of sampling-based MPC. Given a dynamics model $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$, we consider the path sampling problem in the control space $\mathbb{U} \subset \mathbb{R}^n$ with the control $\boldsymbol{u} \in \mathbb{U}$ having n-dimensions at the current system state $\boldsymbol{x} \in \mathbb{X} \subset \mathbb{R}^d$. Typically, a batch of control trajectories is sampled, rolled out through the dynamics model, and evaluated using a cost function. Let a control path be $u : [0, 1] \to \boldsymbol{u}$, $\boldsymbol{u}(t) \in \mathbb{U}$, we can define the path arc length as

$$\mathrm{TV}(u) = \sup_{M \in \mathbb{N}^+, 0 = t_1, \ldots, t_M = 1} \sum_{i=1}^{M-1} \|\boldsymbol{u}(t_{i+1}) - \boldsymbol{u}(t_i)\|. \tag{4.1}$$

We define $\mathbb{F}$ as the set of all control paths that are uniformly continuous with bounded variation $TV(u) < \infty$, $u \in \mathbb{F}$. This assumption is common in many control settings, where the control trajectories are bounded in time and control space (i.e., both time and control spaces are compact). Throughout this paper, we narrate the preliminary and the tensor sampling method in matrix definitions, discretizing continuous paths with equal time intervals.

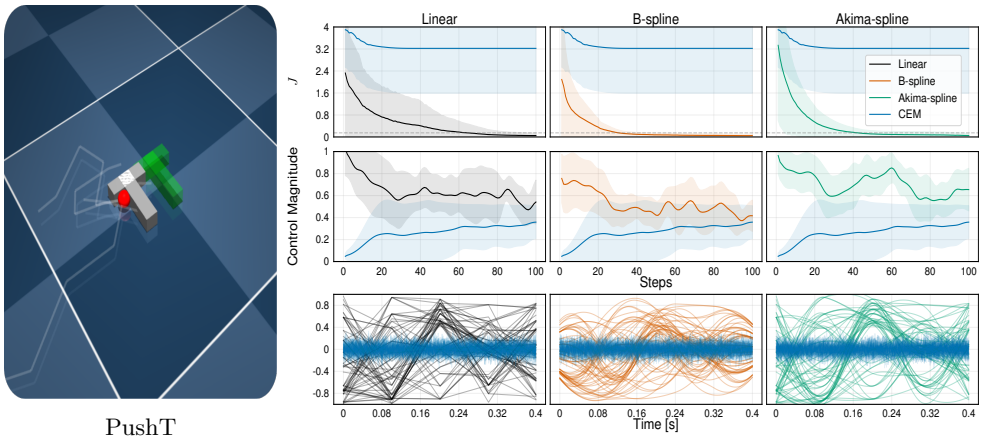
4.2.1. Cross-Entropy Method for Sampling-based MPC

Consider a discretized dynamical system $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad t = 0, \ldots, T-1$ with horizon T, where $\boldsymbol{x}_t \in \mathbb{R}^d, \boldsymbol{u}_t \in \mathbb{R}^n$ denotes the state the control at time step t. The objective is to minimize a cumulative cost function

$$J(\boldsymbol{\tau}, \boldsymbol{U}) = \sum_{t=0}^{T-1} c(\boldsymbol{x}_t, \boldsymbol{u}_t) + c_T(\boldsymbol{x}_T), \text{ s.t. } \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{4.2}$$

where $\boldsymbol{\tau} = [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T] \in \mathbb{R}^{(T+1)\times d}, \boldsymbol{U} = [\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{T-1}] \in \mathbb{R}^{T \times n}$ are the dynamics rollout, $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the immediate cost at each time step, and $c_T(\boldsymbol{x}_T)$ is the terminal cost.

CEM optimizes the control sequence $\boldsymbol{U}$ iteratively by approximating the optimal control distribution using a parametric probability distribution, typically Gaussian. Initially,



PushT

Figure 4.1.: Comparison of MTP interpolation methods versus CEM on `PushT` environment. The cost of `PushT` is the sum of the position and orientation error to the green target (without the guiding contact cost), and the initial T pose is randomized. The first row depicts the cost convergence over 10 seeds. In most seeds, CEM struggles to push the object due to the lack of exploration (e.g., mode collapsing), while MTP variants always find the correct contact point to achieve the task. Note that the control magnitude of MTP is high due to the global explorative samples (see second & third rows), compared to the white noise samples (blue). B-spline helps regulate the control magnitude due to its barycentric weightings, while retaining exploration behaviors. The last row illustrates the control trajectories between 64 tensor samples and 64 white noise trajectories. Experiment videos and open-source implementations are publicly available at `https://sites.google.com/view/tensor-sampling/` and `https://github.com/anindex/mtp`.

the control inputs are sampled from an initial distribution parameterized by mean $\boldsymbol{\mu} \in \mathbb{R}^{T \times n}$ and standard deviation $\boldsymbol{\sigma} \in \mathbb{R}^{T \times n}$. At each iteration, CEM performs the following steps iteratively:

Sampling. Draw B candidate control sequences from the current Gaussian distribution:

$$\boldsymbol{U}^{(k)} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)), \quad k = 1, \dots, B. \tag{4.3}$$

Evaluation. Rollout $\boldsymbol{\tau}^{(k)}$ from the dynamics model and compute the cost $J(\boldsymbol{\tau}^{(k)}, \boldsymbol{U}^{(k)})$ for each sampled control sequence by simulating the system dynamics.

Elite Selection. Choose the top-$E \leq B$ elite candidates of control sequences that have the lowest cost, forming an elite set $\mathcal{E} = \{\boldsymbol{U}^{(k)}\}_{k=1}^{E}$.

Distribution Update. Update the parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ based on elite samples:

$$\boldsymbol{\mu}_{\text{new}} = \frac{1}{E} \sum_{\boldsymbol{U} \in \mathcal{E}} \boldsymbol{U}, \quad \boldsymbol{\sigma}^2_{\text{new}} = \frac{1}{E-1} \sum_{\boldsymbol{U} \in \mathcal{E}} (\boldsymbol{U} - \boldsymbol{\mu}_{\text{new}})^2. \tag{4.4}$$

An exponential smoothing update rule can be used for stability:

$$\boldsymbol{\mu} \leftarrow \alpha \boldsymbol{\mu} + (1-\alpha)\boldsymbol{\mu}_{\text{new}}, \quad \boldsymbol{\sigma} \leftarrow \alpha \boldsymbol{\sigma} + (1-\alpha)\boldsymbol{\sigma}_{\text{new}}, \tag{4.5}$$

where $\alpha \in [0, 1)$ is a smoothing factor.

Termination Criterion. The iterative process continues until a convergence criterion is met or a maximum number of iterations is reached. The optimal control sequence is approximated by the final mean $\boldsymbol{\mu}$ of the Gaussian distribution.

4.2.2. Spline-based Controls

Splines provide a powerful representation for trajectory generation in MPC due to their flexibility, continuity properties, and ease of parameterization [142, 143]. In this work, we focus on spline-parametrization of control trajectories. Spline-based trajectories ensure smooth and feasible control inputs that satisfy constraints and objectives inherent to MPC frameworks.

A spline is defined as a piecewise polynomial function $\boldsymbol{u}(t) : [0, T] \to \mathbb{U}$, which is polynomial within intervals divided by knots $t_1, \ldots, t_M$, with continuity conditions enforced at these knots. In particular,

- **Knots.** A knot $t_i \in [0, T]$ is a time point where polynomial pieces join. We have a non-decreasing sequence of knots $0 = t_1 \leq \ldots \leq t_M = T$, which partition the time interval $[0, T]$ into pieces $[t_i, t_{i+1}]$ so that the path is polynomial in each piece. We may often have double or triple knots, meaning that several consecutive knots $t_i = t_{i+1}$ are equal, especially at the beginning and end, as this can ensure boundary conditions for zero higher-order derivatives.
- **Waypoints.** A waypoint $\boldsymbol{u}(t) \in \mathbb{U}$ is a point on the path, typically corresponding to $\boldsymbol{u}(t_i)$.
- **Control Points.** A set of control points $\mathcal{Z} = \{\boldsymbol{z}_i | \boldsymbol{z}_i \in \mathbb{R}^n\}_{i=1}^K$ parametrizes the spline via basis functions.

B-Spline Parameterization. In B-splines [144], the path $\boldsymbol{u}$ is expressed as a linear combination of control points $\boldsymbol{z}_i \in \mathcal{Z}$

$$\boldsymbol{u}(t) = \sum_{i=1}^{K} B_{i,p}(t)\boldsymbol{z}_i, \quad \text{s.t.} \quad \sum_{i=1}^{K} B_{i,p}(t) = 1, \tag{4.6}$$

where $B_{i,p} : \mathbb{R} \to \mathbb{R}$ maps the time t to the weighting of the i^{th} control point, depicting the i^{th} control point weight for blending (i.e., as with a probability distribution over i). Hence, the path point $\boldsymbol{u}(t)$ is always in the convex hull of control points. The B-spline functions $B_{i,p}(t)$ are fully specified by a non-decreasing series of time knots $0 = t_1 \leq \ldots \leq t_M = T$ and the integer polynomial degree $p \in \{0, 1, \ldots\}$ by

$$\begin{aligned} B_{i,0}(t) &= [t_i \leq t \leq t_{i+1}], \quad \text{for } 1 \leq i \leq M-1, \\ B_{i,p}(t) &= \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t), \quad \text{for } 1 \leq i \leq M-p-1. \end{aligned} \tag{4.7}$$

$B_{i,0}$ are binary indicators of $t \in [t_i, t_{i+1}]$ with $1 \leq i \leq M-1$. The 1st-degree B-spline functions $B_{i,1}$ have support in $t \in [t_i, t_{i+2}]$ with $1 \leq i \leq M-2$, such that $\sum_{i=1}^{M-2} B_{i,1}(t) = 1$ holds. In general, degree p B-spline functions $B_{i,p}$ have support in $t \in [t_i, t_{i+p+1}]$ with $1 \leq i \leq M-p-1$. We need $K = M-p-1$ control points $\boldsymbol{z}_{1:K}$, which ensures the normalization property $\sum_{i=1}^{K} B_{i,p}(t) = 1$ for every degree.

Akima-Spline Parameterization. The Akima spline [50] 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 B-splines. In other words, an Akima spline is a piecewise cubic spline constructed to pass through control points with C^1 smoothness. Given the control point set $\mathcal{Z}$ with $K = M$, the Akima spline constructs a piecewise cubic polynomial $u(t)$ for each interval $[t_i, t_{i+1}]$

$$\boldsymbol{u}_i(t) = \boldsymbol{d}_i(t - t_i)^3 + \boldsymbol{c}_i(t - t_i)^2 + \boldsymbol{b}_i(t - t_i) + \boldsymbol{a}_i, \tag{4.8}$$

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

$$\boldsymbol{s}_i = \frac{|\boldsymbol{m}_{i+1} - \boldsymbol{m}_i|\boldsymbol{m}_{i-1} + |\boldsymbol{m}_{i-1} - \boldsymbol{m}_{i-2}|\boldsymbol{m}_i}{|\boldsymbol{m}_{i+1} - \boldsymbol{m}_i| + |\boldsymbol{m}_{i-1} - \boldsymbol{m}_{i-2}|}. \tag{4.9}$$

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

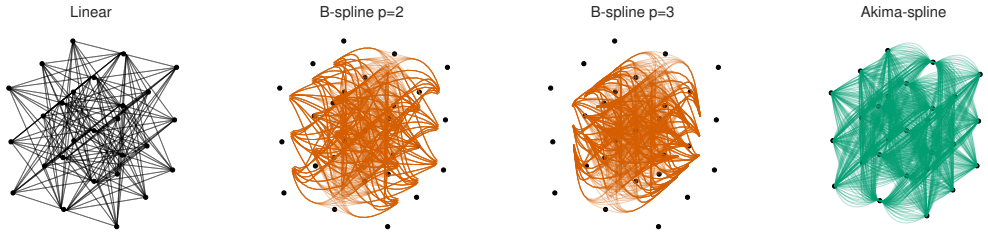


Figure 4.2.: Illustration of different tensor interpolations on evenly spaced graph with $M = 3, N = 9$. With a higher B-spline degree, the control trajectories exhibit more smoothness and conservative behavior, while Akima-spline aggressively and smoothly tracks control-waypoints. Note that we do not consider boundary conditions for B-spline interpolation in tensor sampling.

are uniquely defined

$$\boldsymbol{a}_i = \boldsymbol{u}_i,\ \boldsymbol{b}_i = \boldsymbol{s}_i,\ \boldsymbol{c}_i = (3\boldsymbol{m}_i - 2\boldsymbol{s}_i - \boldsymbol{s}_{i+1})/(t_{i+1} - t_i),\ \boldsymbol{d}_i = (\boldsymbol{s}_i + \boldsymbol{s}_{i+1} - 2\boldsymbol{m}_i)/(t_{i+1} - t_i)^2. \tag{4.10}$$

Motivation. Spline representation provides several benefits. (i) It ensures smooth control trajectories with guaranteed continuity of positions, velocities, and accelerations under mild assumptions of the dynamics. (ii) Spline simplifies complex trajectories through a few control points and efficiently incorporates constraints and boundary conditions. (iii) It enables easy numerical optimization thanks to differentiable and convex representations.

4.3. Method

We first propose *tensor sampling* – a batch control path sampler having high explorative behavior, and investigate its path coverage property over the compact control space. Then, we incorporate the tensor sampling with the modified CEM update rule, balancing exploration and exploitation in cost evaluation, forming an overall vectorized sampling-based MPC algorithm.

4.3.1. Tensor Sampling

Inspired by Definition 3.2 [38], we utilize the random multipartite graph as a tensor discretization structure to approximate global path sampling.

Definition 4.1 (Random Multipartite Graph Control Discretization)**.** *Consider a graph* $\mathcal{G}(M, N) = (V, W)$ *on control space* $\mathbb{U}$*, the node set* $V = \{L_i\}_{i=1}^{M}$ *is a set of* M *layers. Each layer* $L_i = \{\boldsymbol{u}_j \in \mathbb{U} \mid \boldsymbol{u}_j \sim \textit{Uniform}(\mathbb{U})\}_{j=1}^{N}$ *contains* N *control-waypoints sampled from a uniform distribution over control space (i.e., bounded by control limits). The edge set* W *is defined by the union of (forward) pair-wise connections between layers*

$$W = \{(\boldsymbol{u}_i, \boldsymbol{u}_{i+1}) \mid \forall \boldsymbol{u}_i \in L_i,\ \boldsymbol{u}_{i+1} \in L_{i+1}, 1 \leq i < M\},$$

leading to a complete M*-partite directed graph.*

Sampling from $\mathcal{G}(M, N)$. The graph nodes are represented as the control-waypoint tensor for all layers $\boldsymbol{Z} \in \mathbb{R}^{M \times N \times n}$, within the control limits. To sample a batch of $B \in \mathbb{N}^+$ control paths with a horizon T, we subsample with replacement $\boldsymbol{C} \in \mathbb{R}^{B \times M \times n}$ from the set of all combinatorial paths in $\mathcal{G}$ (cf. algorithm 3) and further interpolate $\boldsymbol{C}$ into control trajectories $\boldsymbol{U} \in \mathbb{R}^{B \times T \times n}$ with different smooth structures, e.g., using Eq. (4.6) or Eq. (4.8). Sampling with replacement is cheap $\mathcal{O}(MN)$, while sampling without replacement is $\mathcal{O}(N^M)$. Sampling without replacement adds overheads due to re-indexing or tree-traversing to get batch of sequence indices (depending on low-level implementation of sampling) but offers better diversity. In practice, we use sampling with replacement, which does not really affect diversity (see last row in Fig. 4.1), is faster and scales well with JAX vectorized operations. To see this, we have N^M combinatorial paths in $\mathcal{G}(M, N)$, each path in $\mathcal{G}(M, N)$ has uniform $1/N^M$ mass, and the probability of sampling the same paths is small.

Algorithm 3: Sampling Paths From $\mathcal{G}(M, N)$

Input: Control waypoints $\boldsymbol{Z} \in \mathbb{R}^{M \times N \times n}$, number of paths B
Output: Sampled control-waypoints $\boldsymbol{C} \in \mathbb{R}^{B \times M \times n}$

`// batch sample with replacement from` $1, \ldots, N$ `with shape` (B, M)
$\boldsymbol{I} \leftarrow$ `randint`$((B, M), 1, N)$.
`// extract waypoints from sampled indices into` $\boldsymbol{C} \in \mathbb{R}^{B \times M \times n}$
$\boldsymbol{C} \leftarrow$ `parse_index`$(\boldsymbol{Z}, \boldsymbol{I})$.

Control Path Interpolation. Straight-line interpolation can be realized straightforwardly by simply probing an $H = \lfloor T/M \rfloor$ number of equidistant points between layers, forming the linear coverage trajectories $\boldsymbol{U} \in \mathbb{R}^{B \times T \times n}$. However, there exist discontinuities at control waypoints using linear interpolation. Hence, we motivate spline tensor interpolations for sampling smooth control paths (cf. Fig. 4.2).

Definition 4.2 (B-spline Control Trajectories)**.** *Given two time sequences* $t_i = i/(M+p+1)$, $i \in \{0, \ldots, M+p\}$ *and* $t_j = j/T$, $j \in \{0, \ldots, T-1\}$, *the B-Spline matrix* $\boldsymbol{B}_p \in \mathbb{R}^{M \times T}$ *can be constructed recursive following Eq.* (4.7)*, with index* i, j *corresponding to the element* $\boldsymbol{B}_{i,p}(t_j)$*. Then, the control trajectories can be interpolated by performing* `einsum` *on the* M *dimension of* $\boldsymbol{B}_p \in \mathbb{R}^{M \times T}, \boldsymbol{C} \in \mathbb{R}^{B \times M \times n}$*, resulting* $\boldsymbol{U} \in \mathbb{R}^{B \times T \times n}$*.*

B-spline control trajectories exhibit conservative behavior as they are strictly inside the control point convex hull. Alternatively, they can be forced to pass through all control points by adding a multiplicity of p per knot [144]. However, this method wastes computation by increasing the B-spline matrix size to $Mp \times N$, and still cannot avoid the overshooting problem. Thus, we further propose the Akima-spline control interpolation.

Definition 4.3 (Akima-spline Control Trajectories [38])**.** *Given the time sequences* $t_i = i/M$, $i \in \{0, \ldots, M-1\}$ *representing the* M *layer time slices and the control points* $\boldsymbol{C} \in \mathbb{R}^{B \times M \times n}$*, the Akima polynomial parameters* $\boldsymbol{A} \in \mathbb{R}^{B \times (M-1) \times 4 \times n}$ *can be computed following Eq.* (4.10) *in batch. Then, given the time sequence* $t_j = j/T$, $j \in \{0, \ldots, T-1\}$*, the control trajectories* $\boldsymbol{U} \in \mathbb{R}^{B \times T \times n}$ *are interpolated following polynomial interpolation Eq.* (4.8)*.*

In the next section, we investigate whether the path distribution support of tensor sampling approximates the support of all possible paths in the control space, with linear interpolation. B-spline and Akima-spline variants are deferred for future work.

4.3.2. Path Coverage Guarantee

We analyze the path coverage property of $\mathcal{G}(M, N)$ for sampling control paths with linear interpolation. In particular, we investigate that any feasible path in the control space can be approximated arbitrarily well by a path in the random multipartite graph $\mathcal{G}(M, N)$ as the number of layers M and the number of waypoints per layer N approaches infinity.

Theorem 4.1 (Asymptotic Path Coverage)**.** *Let* $u \in \mathbb{F}$ *be any control path and* $\mathcal{G}(M, N)$ *be a random multipartite graph with* M *layers and* N *uniform samples per layer (cf. Definition 4.1). Assuming a time sequence (i.e., knots)* $0 = t_1 < t_2 < \ldots < t_M = 1$ *with equal intervals, associating with layers* $L_1, \ldots, L_M \in \mathcal{G}(M, N)$ *respectively, then*

$$\lim_{M,N \to \infty} \min_{g \in \mathcal{G}(M,N)} \|u - g\|_\infty = 0.$$

In intuition, Theorem 4.1 states the support of path distribution $\mathcal{G}(M, N)$ approximates $\mathbb{F}$ and converges to $\mathbb{F}$ as $M, N \to \infty$. Thus, sampling paths from $\mathcal{G}(M, N)$ provides a tensorized mechanism to efficiently sample any possible path from $\mathbb{F}$, which allows vectorized sampling.

Remark 4.1. *As $M, N \to \infty$, for any control path $g \in \mathbb{F}$, then $g \in \mathcal{G}(M, N)$. Hence, $\mathcal{G}(M, N)$ represents all homotopy classes in the limit $M, N \to \infty$, and sampling paths from $\mathcal{G}(M, N)$ approximate sampling from all possible paths.*

To quantify the exploration level of tensor sampling, one standard way is to investigate its path distribution entropy. In intuition, when $M, N \to \infty$, tensor sampling entropy also approaches infinity due to uniform sampling per layer, which is further discussed in Appendix A.2.2.

Practical Settings. We typically only set $M < T$. In principle, increasing N should enable finer-grained exploration over the trajectory space. However, we observe diminishing returns when N increases while keeping the total number of sampled trajectories B fixed (cf. Fig. A.2.2). Intuitively, the underlying graph becomes denser, the number of explored paths remains constant, resulting in only marginal performance improvements. Therefore, we recommend choosing N proportionally to B and within the bounds of available GPU memory, in order to maintain computational efficiency without oversampling from a small sample size.

4.3.3. Algorithm

Here, we present the overall algorithm combining tensor and local sampling with smooth structure options in algorithm 4. We propose a simple mixing mechanism with $\beta \in [0, 1]$ by concatenating explorative and exploitative samples, forming a control trajectory tensor $\boldsymbol{U}$ (Line 4-8). We include the current nominal control for system stability at the fixed-point states (e.g., for tracking tasks) [54]. Using simulators that allow for parallel runs [145, 146], rollout and cost evaluation can be efficiently vectorized (Line 9).

To tame the noise induced by tensor sampling, we modify the CEM update with `softmax` weighting on the elite set, for computing the new weighted control means and covariances similar to the MPPI update rule [53]. We observe that this greatly smoothens the update over timesteps (Line 11-13) (cf. Fig. A.2.3). Finally, similar to Howell et al. [54], we send the first control of the best candidate, since this control

trajectory is evaluated in the simulator rather than the updated mean $\boldsymbol{\mu}$. Notice that we have fixed tensor shapes based on hyperparameters, for all subroutines of sampling, rolling out, and control distribution updates. algorithm 4 can be JAX `jit` and `vmap` over a number of R model perturbations $\{f_j(\boldsymbol{x}, \boldsymbol{u})\}_{j=1}^{R}$, for efficient online domain randomization, while maintaining real-time control (cf. Appendix A.2.5).

4.4. Experiments

In this section, we investigate our proposed approach with the following research questions/points:

- How does MTP's performance with Akima/B-spline control variants compare to standard sampling-based MPC baselines, and strong-exploratory evolution strategies baselines?
- How does MTP's performance vary with the number of elites and mixing rate β on interpolation methods (Linear, B-spline, Akima-spline)?
- How does MTP's performance vary with (i) mixing rate β associating with levels of MTP exploration on complex environments, and (ii) MTP-Bspline degree versus planning performance.

We study the cumulative cost $J(\boldsymbol{\tau}, \boldsymbol{U})$ Eq. (4.2) over the control timestep for each task. For each experiment, we take the minimum cumulative cost in batch rollouts at each timestep, and plot the mean and standard deviation over 5 seeds. Further ablations on sweeping graph parameters M, N, `softmax` weighting, and JAX planning performance benchmark are presented in Appendix A.2.5.

Practical Settings. All algorithms and environments are implemented in MuJoCo XLA [145, 26], to utilize the `jit` and `vmap` JAX operators for efficient vectorized sampling and rollout on multiple model instances. All experiment runs are *sim-to-sim* evaluated (MuJoCo XLA to MuJoCo). In particular, we introduce some modeling errors in MuJoCo XLA. Then, for online domain randomization, we randomize a set of R models $\{f_j(\boldsymbol{x}, \boldsymbol{u})\}_{j=1}^{R}$, then we perform batch sampling and rollout of $R \times B$ trajectories. Finally, the cost evaluation is averaged on the R domain randomization dimension. Note that, for this paper, we deliberately design the task costs to be simple and set sufficiently short planning horizons to benchmark the exploratory capacity of algorithms. In practice, one may design dense guiding costs to achieve the tasks. Further task details are presented in Appendix A.2.3.

Algorithm 4: Model Tensor Planning

Input: Model $f(\boldsymbol{x}, \boldsymbol{u})$, graph params M, N, num samples B, mixing rate $\beta \in [0, 1]$, planning horizon T. CEM params $\alpha \in [0, 1), \lambda > 0, \sigma_m > 0, E$, which are smooth and temperature scalar, minimum variance, elite number, respectively.

Choose interpolation type **Linear**, or **B-spline** (Definition 4.2), or **Akima** (Definition 4.3).

Init the nominal control $\boldsymbol{\mu} \in \mathbb{R}^{T \times n}$ and variance $\mathrm{diag}(\boldsymbol{\sigma}^2), \boldsymbol{\sigma} \in \mathbb{R}^{T \times n}$.

while *Task is not complete* **do**

`// tensor sampling`

Uniformly sample $\boldsymbol{Q} \in \mathbb{R}^{M \times N \times d}$ on control space $\mathbb{U}$.

Sample control waypoints $\boldsymbol{C} \in \mathbb{R}^{P \times M \times n}$ with $P = \lfloor \beta B \rfloor$, using algorithm 3.

Interpolate $\boldsymbol{C}$ using with chosen interpolation method into control trajectories $\boldsymbol{U}_{\mathcal{G}} \in \mathbb{R}^{P \times T \times n}$.

`// local sampling`

Sample $B - P - 1$ local trajectories $\boldsymbol{U}_{\text{Local}} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$.

Stack $\boldsymbol{U}_{\text{Local}}, \boldsymbol{U}_{\mathcal{G}}, \mu$ into $\boldsymbol{U} \in \mathbb{R}^{B \times T \times n}$

`// Update routine using vectorized simulator`

Batch rollout $\boldsymbol{X} \in \mathbb{R}^{B \times T \times d}$ from model $f(\boldsymbol{x}, \boldsymbol{u})$, and evaluate cost matrix $\boldsymbol{S}(\boldsymbol{X}, \boldsymbol{U}) \in \mathbb{R}^{B \times T}$.

Sum cost $\boldsymbol{s} = \sum_t \boldsymbol{S}_{:,t} \in \mathbb{R}^B$ and sort top-E elite candidate indices $\boldsymbol{i}_E$.

Select candidate $\boldsymbol{U} \leftarrow \texttt{parse\_index}(\boldsymbol{U}, \boldsymbol{i}_E)$ and compute candidate weights $\boldsymbol{w} = \frac{\exp\left(-\frac{1}{\lambda}\boldsymbol{s}[\boldsymbol{i}_E]\right)}{\sum \exp\left(-\frac{1}{\lambda}\boldsymbol{s}[\boldsymbol{i}_E]\right)}$.

Compute new mean $\boldsymbol{\mu}' = \boldsymbol{w}\boldsymbol{U}$ and var. $\boldsymbol{\sigma}' = \max(\boldsymbol{w} \sum_{\boldsymbol{u} \in \boldsymbol{U}} (\boldsymbol{u} - \boldsymbol{\mu}')^2, \sigma_m)$.

Update $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}' + \alpha(\boldsymbol{\mu} - \boldsymbol{\mu}')$ and $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}' + \alpha(\boldsymbol{\sigma} - \boldsymbol{\sigma}')$.

`// Send the best evaluated control` $\boldsymbol{u}^* \in \mathbb{R}^n$

$\boldsymbol{u}^* \leftarrow \boldsymbol{U}_{0,:}$

Baselines. For explorative baselines, we choose OpenAI-ES (with antithetic sampling) [55], which shows parallelization capability with a high number of workers in high-dimensional model-based RL settings. Additionally, we choose the recently proposed Diffusion Evolution (DE) [56], bridging the evolutionary mechanism with a diffusion process [147], which demonstrates superior performances over classical baselines such as CMA-ES [140]. Both are implemented in `evosax` [148]. For methods that take into account local information (exploitation methods), we compare MTP with standard MPPI [53] and Predictive Sampling (PS) [54] to sanity check on task completion in sim-to-sim scenarios.

4.4.1. Motivating Example

Here, we provide an experimental analysis of the baselines' exploration capacity on `Navigation` environment (cf. Fig. 4.3), where the point-mass agent is controlled by an axis-aligned 2-dim velocity controller. We compare MTP-Bspline and MTP-Akima to evolutionary algorithms and standard MPC baselines, with maximum sampling noise settings (cf. Fig. 4.3). In particular, given the control limits $[-1, 1]$ on x-y axes, we set the standard deviation $\sigma = 1$ for sampling noise of MPPI and PS, and population generation noise for OpenAI-ES and DE. Fig. 4.3 also shows the cost convergence

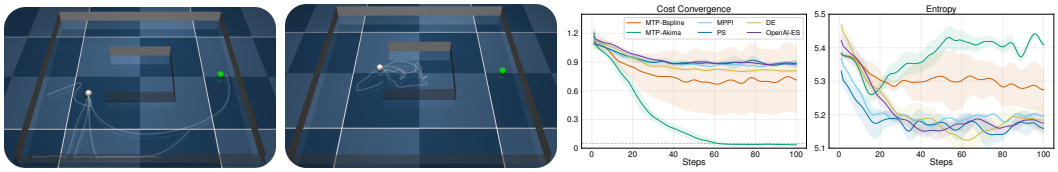


Figure 4.3.: Motivation comparison of MTP methods versus baselines with $B = 256$ on `Navigation` environment. The environment is designed to be challenging to reach the green goal, requiring strong exploration to avoid large local minima in the middle (see task details in Appendix A.2.3). We plan with $T = 20$ with $\Delta t = 0.05s$. The figures show 5 random traces of white rollouts. (Left) MTP-Akima rollouts reach the green goal very early due to high-entropy tensor sampling, while (Right) OpenAI-ES struggles to generate a rollout exploring the way out of large local minima.

and the cost entropy curves over timesteps, in which the entropy is computed as $H = -\sum_{j=1}^{B} P_j \log P_j$, $P_j = \exp(J_j)/(\sum_l \exp(J_l))$, where $\{J_j\}_{j=1}^{B}$ is the batch of cumulative rollout costs. The entropy represents the diversity of rollout evaluation, implying the exploration capability of algorithms. We observe that MTP-Akima has the highest entropy curve, inducing the lowest cost convergence over timesteps, while

other baselines converge to the middle minima. In this case, MTP-Bspline also struggles due to conservative interpolation in tensor sampling, achieving moderate entropy, yet higher than evolutionary strategies.

4.4.2. Comparison Experiments

We analyze the performance comparison of MTP and the baselines over various robotics tasks representing different dynamics and planning cost settings (cf. Fig. 4.4). In each task, we tune the baselines and set the same white noise standard deviation for MTP/MPPI/PS to study the performance gain by tensor sampling, while using the default hyperparameters for evolutionary baselines.

Comparison Environments. `PushT` [149], `Cube-In-Hand` [150] require intricate manipulation environments where robust exploration is critical due to complex contact dynamics and precise multi-step manipulation requirements. `G1-Standup, G1-Walk` represent high-dimensional robotic tasks demanding substantial computational resources and sophisticated control strategies. `Crane`, `Walker` [151] present underactuated and nonlinear dynamic challenges. To ensure fair comparison, for all baselines, we fix the same number of rollouts $B = 16$ on `Crane`, and $B = 128$ for all other tasks. All tasks are implemented in `hydrax` [26]. Further experiment details are in Appendix A.2.4. The `PushT` task, which involves pushing a T-shaped object precisely to a target location, particularly highlights the advantage of MTP variants. While MPPI and PS frequently encounter mode collapse due to insufficient exploration, resulting in suboptimal or even failed attempts at solving the task, MTP-Bspline and MTP-Akima consistently achieve low-cost convergence. This underscores the significant benefit of strong exploration enabled by tensor sampling. Evolutionary algorithms like OpenAI-ES and DE perform similarly to MTP in `PushT`, inherently show better exploration than MPPI and PS, but still fall short compared to MTP variants in `Cube-In-Hand` due to noisy rollouts. `Cube-In-Hand` requires strong exploration while maintaining intricate control to avoid the cube falling, thus emphasizing the effectiveness of the MTP β-mixing strategy.

In `Crane` environment, we apply heavy modeling errors of mass, inertia, and pulley/joint damping. MTP variants excel by maintaining stable control trajectories with smooth transitions, effectively navigating the nonlinear dynamics and underactuation. The B-spline interpolation's conservative nature helps avoid overshooting and instability prevalent in these tasks, thus outperforming both evolutionary algorithms and standard MPC methods that tend to produce erratic control inputs. The `Walker` task exhibits a rather simple dynamics model and is less sensitive to the sampling distribution due

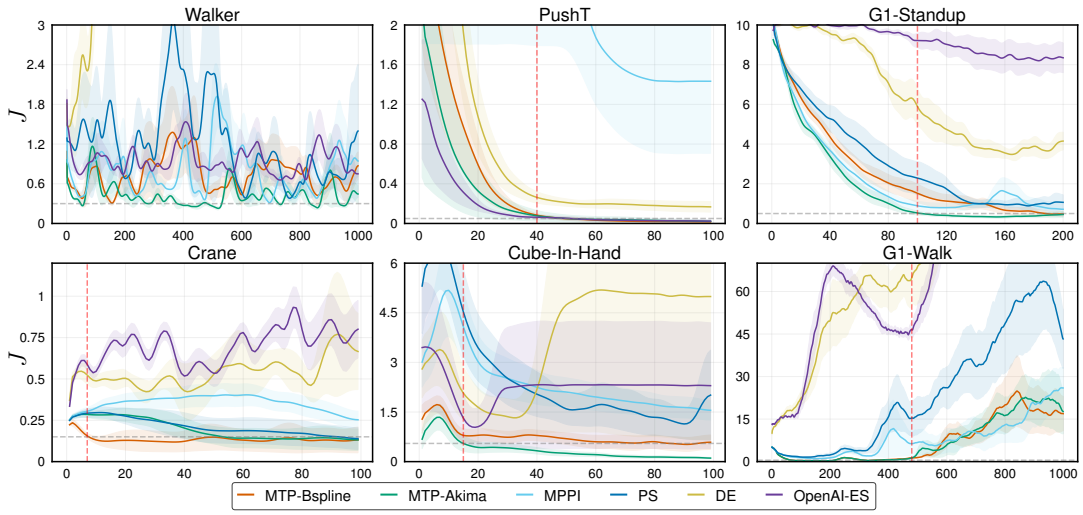


Figure 4.4.: Performance comparison of MTP variants against standard MPC methods (MPPI, PS) and evolutionary algorithms (OpenAI-ES, DE). The (horizontal) gray dashed line depicts the task success, while the (vertical) red line represents the timestep such that the first algorithm statistically succeeds the task, or fails last in `G1-Walk`.

to its relatively simple contact model. We apply no modeling error as a sanity check. Indeed, MTP and classical MPPI/PS perform similarly in simple cases.

In `G1-Standup`, MTP-Akima demonstrates effective humanoid standup due to its aggressive yet smooth trajectory interpolation, enabling efficient exploration and rapid convergence. In contrast, OpenAI-ES and DE struggle with the dimensionality, often yielding higher cumulative costs and failing to adequately sample feasible trajectories, resulting in significant performance gaps. MTP variants have marginally higher performance than standard MPC baselines in `G1-Standup`, but show better control stability for longer `G1-Walk` before falling. These results underline the capability of MTP to balance exploration and exploitation in high-dimensional tasks systematically.

4.4.3. Design Ablation

We conduct an ablation study analyzing the effect of varying two crucial hyper-parameters—the number of elites E and the mixing rate β—on MTP algorithmic performance. Fig. 4.5 presents heatmaps indicating accumulated cost over time for each task and interpolation method (MTP-Linear, MTP-Bspline, MTP-Akima). Each heatmap illustrates distinct algorithmic realizations at its corners. Specifically, the

bottom-left corner represents PS, characterized by a single elite and purely white noise sampling. Conversely, the bottom-right corner corresponds to Predictive Tensor Sampling (TS-PS), maintaining a single elite but employing full tensor sampling. Due to the `softmax` update (algorithm 4 Line 11-12), the top-left corner realizes MPPI (with adaptive sampling covariance), leveraging all candidate samples with local noise sampling, while the top-right corner reflects Tensor Sampling-MPPI (TS-MPPI), utilizing all candidates and full tensor sampling for maximum exploration.

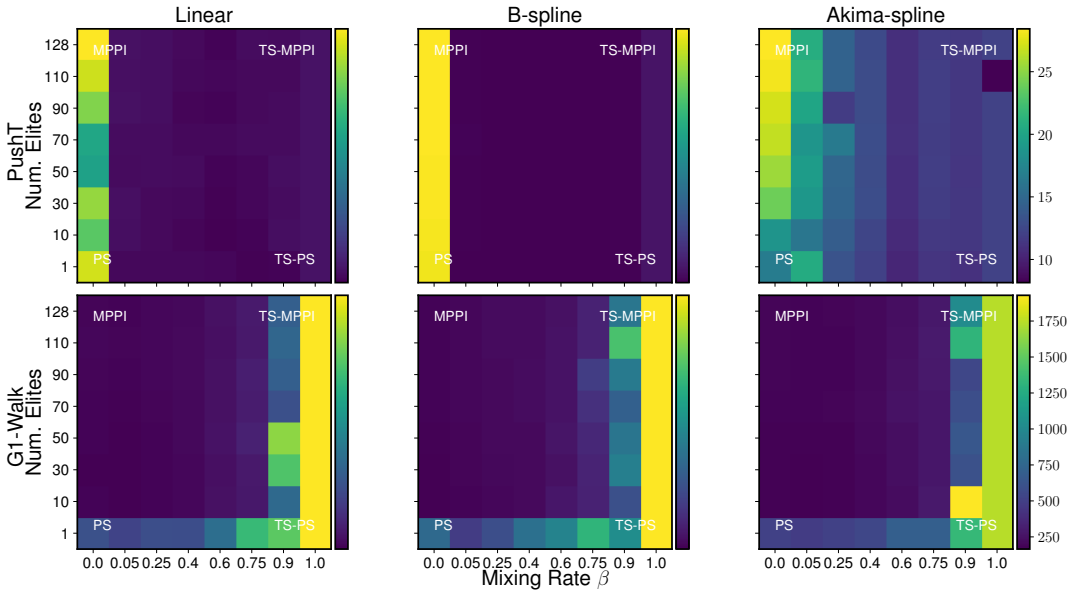


Figure 4.5.: Mixing rate β and number of elites E sweep on `PushT, G1-Walk` tasks with $B = 128$ to investigate the algorithmic update rule algorithm 4 Line 9-12. The heatmap indicates accumulated cost over timesteps at termination, and the heat value range is fixed for each task/row.

We observe the consistent pattern that MTP performance degrades at the extremes of β. In `PushT`, moderate values of β lead to significantly lower costs, while the absence of tensor sampling ($\beta = 0$) yields poor performance due to inadequate exploration. This observation reinforces the effectiveness of the β-mixing strategy for balancing global and local sampling contributions. Interestingly, the number of elites E has a limited impact in `PushT`, likely due to the task's insensitivity to control stability. However, in `G1-Walk`, the choice of E is crucial. Using a single elite ($E = 1$) , corresponding to the PS control scheme, leads to unstable and jerky behavior, which aligns with the

poor performance observed in Fig. 4.4. At the other extreme, with $\beta = 1$ (full tensor sampling), performance also degrades. This is attributed to the fixed rollout budget B; as M increases, global samples become too sparse to effectively capture the fine-grained control required for stable gait tracking. In this case, exploitation with local samples is essential to maintain intricate motion tracking control.

4.4.4. Sensitivity Ablation

Here, we perform sensitivity analyses for various critical algorithmic hyperparameters. Fig. 4.6 evaluates sensitivity to the mixing rate β for different tasks. For the `PushT` environment, results show minimal sensitivity across mixing rates, as the task inherently lacks significant failure modes, ensuring consistent success regardless of the exploration-exploitation balance. In contrast, the `Cube-In-Hand` task demonstrates high sensitivity, with larger mixing rates causing instability due to the cube falling out of grasp frequently. Optimal performance is thus achieved with lower β values, suggesting careful management of exploration intensity. Furthermore, for the high-dimensional `G1-Standup` task, a smaller mixing rate helps stabilize the control, enabling the robot to achieve a more consistent and stable stand-up performance.

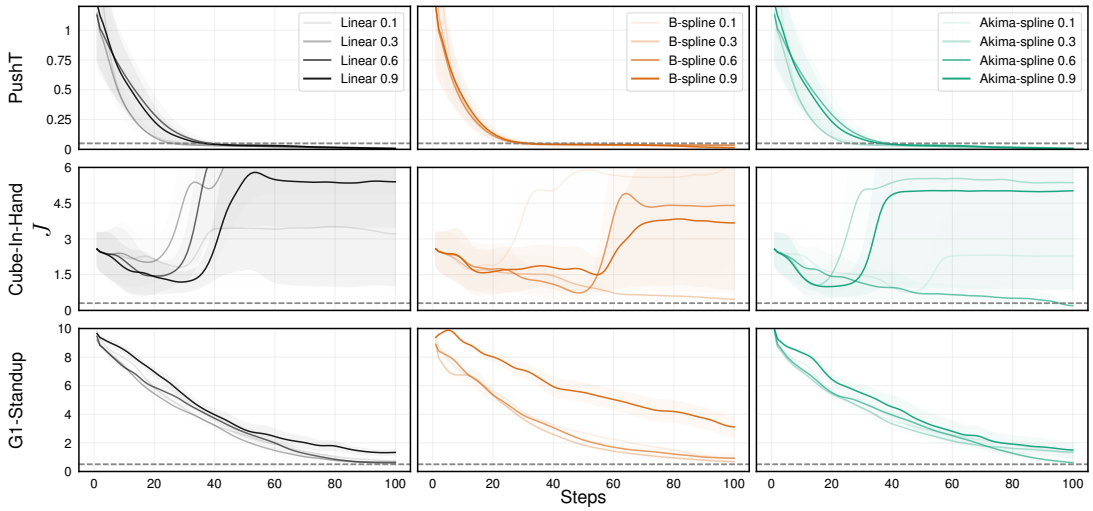


Figure 4.6.: Mixing scalar β sweep on `PushT, Cube-In-Hand, G1-Standup` environments with $B = 128$ to investigate the sensitivity of MTP on explorative level. The dashed line represents the successful bar. In `Cube-In-Hand`, some of the cost curves increase due to the cube falling out of the LEAP hand.

4.4.5. Real-World Stand-Up on Unitree G1

We evaluate MTP on a real Unitree G1 humanoid performing a stand-up maneuver from a kneeling/supine configuration. This task emphasizes online feasibility under changing contact conditions and rapid replanning, and benefits from MTP's batch rollouts over perturbed models (online domain randomization) and high-entropy control proposals.

Hardware Setup. This experiment is conducted on a Unitree G1 humanoid in a $3\,\mathrm{m} \times 3\,\mathrm{m}$ area with a flat, grass floor. A multi-camera Nokov motion capture system with a 380Hz sampling rate provides global pose for torso pose for height reward, and foot poses for contact estimations. All timestamps from proprioception and motion capture sensors are synchronized. A safety filter enforces joint-limit, torque, and base-tilt bounds and triggers a soft-stop when violated.

Digital Twin with MuJoCo MJX. We build a *digital twin* MuJoCo MJX, matching the G1's link inertias, kinematic tree, joint limits, actuator gains, and continuously updating the proprioception joint poses. The MJX model is used inside the MTP loop for parallel rollouts. To cope with model uncertainty, we maintain an ensemble of M perturbed models (masses $\pm 10\%$, COM shifts $\pm 2.5\,\mathrm{cm}$, joint damping $\times[0.5, 2.0]$, actuator delay $[5, 12]\,\mathrm{ms}$, ground friction $\mu \in [0.5, 1.0]$). Each MTP iteration evaluates control tensors across this ensemble using `jit`-compiled MJX dynamics with static shapes.

Control Architecture. Low-level whole-body control runs at $1\,\mathrm{kHz}$ with torque limits and rate limiters; MTP provides a reference joint-space trajectory over a short horizon (H=1.0 s, Δt=0.2 s). Controls are parameterized by K=5 knots and interpolated by Akima to ensure smoothness. Each planning iteration samples B control sequences as a fixed-shape tensor, rolls them out on the M-model ensemble, computes costs, and applies a β-mixing CEM-style update selecting top-k elites. All steps are vectorized with `vmap`/`jit` in JAX (static batch/sequence shapes).

Reward Design. Table 4.1 summarizes the task cost (negative reward) terms used in MTP rollouts. We denote pelvis height $h(q)$, target height $h^\star$, base rotation $R(q) \in \mathrm{SO}(3)$, world up $\hat{\mathbf{z}}$, joint torques $\boldsymbol{\tau}$, and foot tangential velocities $\mathbf{v}^{\parallel}_{\mathrm{foot},i}$.

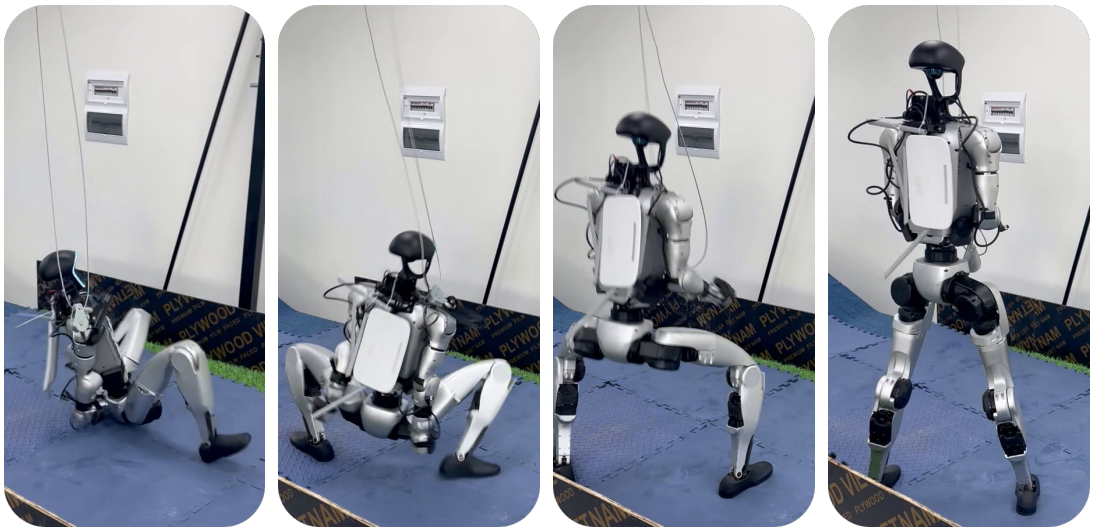
Table 4.1.: Reward (cost) terms for G1 stand-up. Negative signs indicate penalties.

Term	**Coeff.** w_j	**Expression / Rationale**
Uprightness	w_{up}	$-\hat{\mathbf{z}}^\top R(q)\,\hat{\mathbf{z}}$ (maximize alignment of torso with gravity).
Pelvis height tracking	w_h	$\lvert\, h(q) - h^\star \,\rvert$ (reach and stabilize target height).
COM lateral deviation	w_{com}	$\lVert \mathbf{p}^{xy}_{\text{COM}}(q) \rVert_2$ (keep COM over support).
Foot slip penalty	w_{slip}	$\sum_{i\in\{\text{L,R}\}} \lVert \mathbf{v}^{\parallel}_{\text{foot},i} \rVert_2$ when contact is active.
Torque effort	w_τ	$\lVert \boldsymbol{\tau} \rVert_2^2$ (energy/thermal management).
Control smoothness	$w_{\Delta u}$	$\lVert u_t - u_{t-1} \rVert_2^2$ (reduce jerks/oscillation).
Joint-limit hinge	w_{lim}	Hinge loss on normalized limit violations (safety).
Back/hand ground contact	w_{touch}	Penalty if non-foot links contact floor (task hygiene).
Progress shaping	w_{prog}	$-(h(q_t) - h(q_{t-1}))$ to encourage monotone rise early.

Practical Implementation: Height-Staged Tuning. Standing up involves distinct phases with different priorities. We therefore use *multiple reward weight sets* $\{w^{(s)}\}$ for staged heights:

$$\begin{aligned}
\text{Stage 1 (kneel)}: &\quad h^\star{=}0.30\text{ m}, \quad w^{(1)}: \text{ large } w_{\text{prog}}, w_{\text{slip}}, \text{moderate } w_\tau \\
\text{Stage 2 (crouch)}: &\quad h^\star{=}0.55\text{ m}, \quad w^{(2)}: \text{ increase } w_{\text{up}}, w_{\text{com}}, \text{reduce } w_{\text{prog}} \\
\text{Stage 3 (stand)}: &\quad h^\star{=}0.85\text{ m}, \quad w^{(3)}: \text{ dominant } w_{\text{up}}, w_{\text{com}}, \text{tight } w_{\Delta u}, w_\tau
\end{aligned}$$

A finite-state gate promotes to stage $s{+}1$ when $h(q) > h^\star - \epsilon$ for $\tau_{\min}$ seconds and base pitch/roll remain below thresholds. The β-*mixing* schedule (global/local proposal blend) is reset per stage and annealed ($\beta : 0.7 \to 0.3$) over iterations to emphasize refinement near the target.



(a) Sit (b) Crouch (c) Rise (d) Stand

Figure 4.7.: Unitree G1 stand-up with MTP.

Performance. Over N=10 stand-up trials from varied initial poses (kneel/supine, random arm placements), MTP achieves (cf. Fig. 4.7)

- **Success rate:** 80.0% (8/10) without human intervention; safety stops on fail runs.
- **Median time-to-stand (pelvis $h \geq 0.85$ m):** 3.1 s (IQR 2.8–3.6 s) with MTP-Akima 3.6 s.
- **Online compute:** median MTP planning latency 4.3 ms.

In ablations (same hardware), removing model-ensemble randomization reduced success to 0% and increased slip events.

Discussion. MTP's tensorized sampling and MJX-based parallel rollouts deliver consistent real-time performance and robust behavior on hardware. The height-staged reward tuning makes the optimization easier (different priorities per phase) while preserving a fixed-shape program for `jit` compilation. MTP-Akima highlights an interpretable exploration–exploitation trade-off: Akima attains fast ascent but with higher torque transients. Domain randomization across an ensemble of MJX models

substantially improves hardware robustness by immunizing against mis-modeled inertias, friction, and delays.

4.5. Related Works

We review related efforts across two major directions: the vectorization of sampling-based MPC and sampling-based motion planning. While these approaches originate from different planning paradigms—dynamics-aware MPC versus collision-free geometric planning—they share a common structure: interacting with an agent-environment model (e.g., dynamics model or collision checker) that can be vectorized for efficient, batched computation. Both domains benefit from high-throughput sampling, making them increasingly amenable to modern GPUs/TPUs.

Sampling-based MPC Vectorization. Sampling-based MPC [31] has been successfully applied to high-dimensional, contact-rich control problems. Methods such as Predictive Sampling (PS) [54], Model Predictive Path Integral (MPPI) [53], and CEM-based MPC [138] rely on parallel sampling of control trajectories and subsequent rollouts using a system dynamics model. These methods naturally benefit from vectorized simulation backends, and recent works have extended them toward more structured and efficient exploration. For instance, inspired by the diffusion process, DIAL-MPC [136] enhances exploration coverage and local refinement simultaneously, achieving high-precision quadruped locomotion and outperforming reinforcement learning policies [152] in climbing tasks. STORM [98] demonstrates GPU-accelerated joint-space MPC for robotic manipulators, achieving real-time performance while handling task-space and joint-space constraints. Other recent efforts integrate GPU-parallelizable simulators, such as IsaacGym [146], into the MPC loop [43] for online domain randomization, removing the need for explicit modeling and enabling real-time contact-rich control. In another line, CoVO-MPC [153] improves convergence speed by optimizing the covariance matrix in sampling, resulting in performance gains in both simulated and real-world quadrotor tasks. These advances demonstrate that structured, parallel control sampling can be effectively deployed in high-stakes robotics applications using vectorized dynamic models.

Motion Planning Vectorization. Recent advances in sampling-based motion planning have shown that classical methods such as RRT [30] can be significantly accelerated using parallel computation, while preserving theoretical guarantees like probabilistic completeness, which is another form of maximum exploration. Early work focused on accelerating specific subroutines like collision checking [34, 35], but more recent

efforts have restructured planners for full GPU-native execution. Examples include GMT* [36], VAMP [37], pRRTC [154], and Kino-PAX [155], which achieve millisecond-scale planning in high-dimensional configuration spaces by parallelizing sampling, forward kinematics, and tree expansions. GTMP [38] pushes this even further by implementing the sampling, graph-building, and search pipeline as tensor operations over batch-planning instances, showcasing the feasibility of real-time planning across multiple environments.

Complementing sampling-based planning, trajectory optimization methods such as batch CHOMP [94], Stochastic-GPMP [67], cuRobo [52], and MPOT [41] have embraced vectorization to solve hundreds of trajectory refinement problems in parallel. Many of these systems are further enhanced by high-entropy initialization with learned priors [78, 156, 157], allowing them to overcome challenging nonconvexities in cluttered environments. These developments collectively demonstrate that both motion planning and MPC can be reformulated as batched, tensor-based pipelines suitable for modern accelerators.

Our work draws on these insights to propose a unified sampling-based control framework that operates entirely through tensorized computation, blending global exploration and local refinement in a single batched planning loop.

4.6. Discussions and Conclusions

In this work, we introduced *Model Tensor Planning* (MTP), a robust sampling-based MPC approach designed to achieve global exploration via maximum entropy sampling. Theoretically, we demonstrated that in the limits of infinite layers M and samples per layer N, our tensor sampling method attains maximum entropy, thereby efficiently approximating the full trajectory space. Furthermore, MTP is intentionally designed to be practically feasible, enabling straightforward implementation for sampling high-entropy control trajectories (see Appendix A.2.5).

While evolutionary strategies algorithms offer improved exploration capabilities [55, 56], compared to traditional MPC methods, our experiments highlight their limitations. The inherently noisy mutation processes often fail to achieve consistent high-entropy exploration, limiting their effectiveness in robotics tasks. In contrast, MTP's tensor sampling consistently explores smooth control possibilities and achieves robust performance.

Spline-based interpolations are central to the practical implementation of MTP, notably B-spline and Akima-spline. These interpolation methods effectively address discontinuities in simple linear interpolation, ensuring the generation of smooth, continuous, and dynamically feasible control trajectories [134]. The experiments underscore the splines' critical role in enhancing trajectory quality, optimizing performance across diverse, complex tasks. We proposed a simple β-mixing strategy for exploration while retaining intricate controls, effectively balancing exploration and exploitation within sampling-based MPC. This flexible strategy allows easy algorithm tuning to various tasks, significantly improving performance stability and robustness across environments with different exploration needs.

From the vectorization standpoint, the matrix-based definition of MTP is specifically structured to leverage Just-in-time compilation `jit` and vectorized mapping `vmap` provided by JAX [20] and MuJoCo XLA [145]. This design choice dramatically accelerates computations, enabling real-time implementation and seamless integration with online domain randomization with `vmap`, crucial for robust control. Overall, MTP offers an efficient, scalable solution for various robotic tasks that demand high exploration capacity and precise control optimization.

Limitations. While MTP demonstrates strong performance across diverse control tasks, it inherits several limitations typical of sampling-based methods. First, its computational cost scales with the number of rollouts, making it challenging to deploy on hardware with limited parallel computing. Second, while our tensor-based sampler improves exploration coverage, it does not leverage task-specific priors or learning-based proposal distributions, which could further improve sample efficiency. Finally, MTP relies on a fixed dynamics model and does not currently support model uncertainty, limiting its robustness in partially observed or stochastic environments.

Broader Impact Statement This work contributes to the development of efficient sampling-based control by introducing a scalable, high-entropy sampling mechanism for model predictive control (MPC). Model Tensor Planning (MTP) opens a promising direction in the design of exploratory algorithms that go beyond local refinements, allowing for global reasoning over control spaces. By enabling maximum entropy exploration via structured tensor operations, MTP provides a framework that may benefit a wide range of decision-making systems requiring robust performance in underexplored, high-dimensional environments—such as dexterous manipulation, legged locomotion, or autonomous vehicles operating under partial observability and uncertainty.

From a real-world deployment perspective, MTP maintains controls within physical limits, but its high-rate, tensorized control sequences may induce rapid variations that practical motor systems must robustly execute. While this fast-changing control is beneficial for agility and responsiveness, it necessitates attention to actuator dynamics and hardware safety. Therefore, safety-aware control filtering or actuator-aware smoothness constraints may be incorporated as extensions for deployment.

Furthermore, like other MPC approaches, MTP assumes access to reliable state estimation for initializing planning rollouts in the control loop. In practical deployment, this typically requires a real-time state estimator and a simulation back-end that serves as a digital twin. For example, MuJoCo XLA can simulate hundreds of dynamics instances in parallel, making it suitable for real-time predictive control. However, realizing this in hardware introduces engineering challenges—such as ensuring low-latency communication between the physical robot and the simulator. We see this digital twin architecture as a promising frontier where algorithmic advances like MTP can be tightly integrated with system-level design for robust, real-time, and scalable autonomous control.

5. Conclusion

This thesis has motivated a novel paradigm of vectorizing motion planning at the instance-level, and introduced a class of tensor search methods, leveraging the computational advantages of modern accelerators such as GPUs and TPUs. By reformulating classical planning pipelines—from trajectory optimization and path planning to sampling-based MPC—into fixed-shape tensor operations, we demonstrate how motion planning can be lifted from traditionally sequential formulations to fully vectorized architectures suitable for Just-in-time compilation and automatic batching via `vmap` in JAX. Across three contributions—MPOT, GTMP, and MTP—we provide theoretical grounding, algorithmic development, and empirical validation that show not only improvements in efficiency but also robustness, diversity, and solution quality.

MPOT introduced a zero-order update rule via the Sinkhorn Step, enabling batch optimization over trajectories through barycentric projections defined by same-shape local polytopes. GTMP reformulated global path planning as Value Iteration on a batch of multipartite graphs, supporting structured sampling and smooth path synthesis in parallel. MTP extended this paradigm to the control domain, enabling sampling-based MPC to operate with high-entropy policy generation and online domain randomization—all within a batched, differentiable planning loop.

These contributions lay the significant indicator of a new class of planning algorithms that are inherently vectorizable, efficient, and compatible with the modern machine learning ecosystem. Yet, several promising directions remain open for future research.

Tensorizing Ergodic Search for Multi-Path Coverage. A natural extension of this thesis is to tensorize Stein Variational Ergodic Search [158, 159]. Ergodic control seeks trajectories whose time-averaged visitation matches a spatial distribution—critical in coverage planning, exploration, and information gathering. By discretizing space and time into tensors and evaluating ergodic metrics (e.g., Sobolev norms in Fourier space) across batches of trajectories and batches of model perturbations. This would enable multi-solution coverage of multi-goal or multi-region tasks while maintaining sim-to-real deployment possibility, which is crucial for dataset collection or decentralized multi-agent planning.

Tensor Tree Search for Symbolic Planning. Another compelling direction is the extension of tensor search to symbolic domains—where discrete logic, symbolic operators, or hierarchical task structures guide planning [160, 161]. Tree search or graph expansion, often relies on sequential branching and recursive evaluation, making them traditionally incompatible with SIMD execution. However, we can explore efficiently representing symbolic trees as fixed-shape tensors and their expansions as batched transitions, one could construct a differentiable or parallelizable version of symbolic planning. This approach could lead to the unification of task and motion planning (TAMP) under a tensor-based framework, wherein symbolic policies can be trained, evaluated, and searched using the same vectorized infrastructure as continuous motion planning.

Scalable Synthetic Data Collection. Tensor motion planning methods like GTMP and MPOT naturally support the generation of large, diverse synthetic planning datasets at scale, due to their homotopy discovery property. By solving thousands of planning problems in parallel with minimal overhead, we can collect massive batches of trajectory data for learning-based frameworks [39, 40] or foundational vision-language-action models [162], including supervised imitation learning, behavior cloning, and offline reinforcement learning methods. Moreover, structured sampling in these planners supports curriculum-style data generation, adaptive scenario difficulty, and multi-solution diversity—further enriching synthetic datasets for training robust robotic policies.

Differentiable Planning in End-to-End Learning Pipelines. As modern robotics increasingly adopts end-to-end learning systems, integrating differentiable planning modules becomes essential. The efficient planning property and statically shaped nature of MPOT, GTMP, and MTP make them ideal candidates for inclusion in neural pipelines—enabling backpropagation through the planner, policy fine-tuning with differentiable objectives, and seamless fusion with perception modules [163, 164, 165]. Future work can explore coupling these planners with differentiable scene understanding or inverse graphics networks, allowing perception-planning-control to be trained jointly with supervision from sparse demonstrations or high-level task rewards.

Online Domain Randomization via Tensor Search. MTP already demonstrates that tensor search supports online domain randomization, solving dozens of perturbed planning instances in parallel within the MPC loop [43, 134]. This opens the door

to robust sim-to-real transfer, where robots adapt their control strategy by solving batched planning problems across randomized mass, friction, or actuation parameters in real-time. Extending this idea, future planners could incorporate probabilistic world models or uncertainty-aware dynamics into tensorized planning, supporting fully Bayesian MPC, distributional robustness, or risk-sensitive control—all within a vectorized, real-time framework.

Concluding Remarks. In summary, this thesis revisits motion planning as a tensor computation problem, aligning classical geometric algorithms with the hardware and software demands of modern robotics and machine learning. The introduced tensor search methods not only accelerate existing pipelines but also unlock new capabilities in diversity, robustness, and integration with learning systems. By extending this direction toward symbolic reasoning, ergodic exploration, and real-world generalization, we move toward the next generation of general-purpose, differentiable, and scalable planning algorithms.

A. Supplementary Material

A.1. Appendix to Chapter 2

A.1.1. Theoretical Analysis & Proofs

We investigate the proposed Sinkhorn Step (Definition 2.2 in Section 2.3.2) properties for a non-convex and smooth objective function (assumption 2.1). Our preliminary analysis performs at arbitrary iteration $k > 0$ and depends on assumption 2.1 and assumption 2.2 stated in Section 2.3.

First, we state a proof sketch of Proposition 2.1. 2.1 $\forall P \in \mathcal{P}$, D^P forms a positive spanning set.

Proof. Observe that by construction of d-dimensional regular polytope $P \in \mathcal{P}$, the convex hull of its vertex set $\mathcal{V}^P$

$$\text{conv}(\mathcal{V}^P) = \left\{ \sum_i w_i \boldsymbol{v}_i \mid \boldsymbol{v}_i \in \mathcal{V}^P, \sum_i w_i = 1,\, w_i > 0,\, \forall i \right\}$$

has $\dim(\text{conv}(\mathcal{V}^P)) = d$ dimensions. Hence, trivially, the conic hull of D^P positively spans $\mathbb{R}^d$. □

Now, we can investigate the quality of D^P in the following lemma.

Lemma A.1. *For any $\boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{a} \neq \mathbf{0}$, $\exists \boldsymbol{d} \in D^P$ such that*

$$\langle \boldsymbol{a}, \boldsymbol{d} \rangle \geq \mu_P \|\boldsymbol{a}\|,\, 0 \leq \mu_p \leq 1$$

where $\mu_P = 1/\sqrt{d(d+1)}$ for $P = d$-simplex, $\mu_P = 1/\sqrt{d}$ for $P = d$-orthoplex, and $\mu_P = 1/\sqrt{2}$ for $P = d$-cube.

Proof. From Proposition 2.1, D^P is a positive spanning set, then for any $\boldsymbol{a} \in \mathbb{R}^d$, $\exists \boldsymbol{d} \in D^P$ such that $\langle \boldsymbol{a}, \boldsymbol{d} \rangle > 0$ (Theorem 2.6, [82]). This property results in the positive cosine measure of D^P(Proposition 7, [166])

$$1 \geq \mu_P := \min_{\mathbf{0} \neq \boldsymbol{a} \in \mathbb{R}^d} \max_{\boldsymbol{d} \in D^P} \frac{\langle \boldsymbol{a}, \boldsymbol{d} \rangle}{\|\boldsymbol{a}\| \|\boldsymbol{d}\|} > 0 \tag{A.1}$$

Equivalently, μ_P is the largest scalar such that $\langle \boldsymbol{a}, \boldsymbol{d} \rangle \geq \mu_P \|\boldsymbol{a}\| \|\boldsymbol{d}\| = \mu_P \|\boldsymbol{a}\|$, $\boldsymbol{d} \in D^P$.

Next, due to the symmetry of the regular polytope family $\mathcal{P}$, there exists an inscribing hypersphere S_r^{d-1} with radius r for any $P \in \mathcal{P}$ [83]. For $\mathcal{P}$, the tangent points of the inscribing hypersphere to the facets are also the centroid of the facets. Then, the centroid vectors pointing from the origin towards these tangent points form equal angles to all nearby vertex vectors. Thus, the cosine measure attains its saddle points Eq. (A.1) at these centroid vectors having the value

$$\mu_P = \frac{r}{R}$$

with $R = 1$ is the radius of the circumscribed unit hypersphere. The inradius r for d-simplex, d-orthoplex, and d-cube are $1/\sqrt{d(d+1)}, 1/\sqrt{d}, 1/\sqrt{2}$, respectively [83]. $\square$

This lemma has a straightforward geometric implication - for every $\boldsymbol{v} \neq \boldsymbol{0}, \boldsymbol{v} \in \mathbb{R}^d$, there exists a search direction $\boldsymbol{d} \in D^P$ such that the cosine angle between these vectors is acute (i.e., $\mu_P > 0$). Then, if we consider the negative gradient vector, which is unknown, there exists a direction in D^P that approximates it well with μ_P being the quality metric (i.e., larger μ_P is better). The values of μ_P for each polytope type also confirm the intuition that, for d-cube with an exponential number of vertices $m = 2^d$ has a constant cosine measure, while the cosine measure of d-simplex having $m = d+1$ vertices scales $O(1/d)$ with dimension. Now, we state the key lemma used to prove the main property of Sinkhorn Step.

Lemma A.2 (Key lemma)**.** *If assumption 2.1 and assumption 2.2 holds, then* $\forall \boldsymbol{x}_k \in X_k$, $\forall k > 0$

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \mu_P \alpha_k \|\nabla f(\boldsymbol{x}_k)\| + \frac{L}{2}\alpha_k^2 \tag{A.2}$$

Proof. If the assumption 2.2 holds, by Proposition 4.1 in [71], the OT solution $\boldsymbol{W}_\lambda \to \boldsymbol{W}_0$ converges to the optimal solution with maximum entropy in the set of solutions of the original problem

$$\min_{\boldsymbol{W} \in U(\boldsymbol{1}_n/n, \boldsymbol{1}_n/n)} \langle \boldsymbol{W}, \boldsymbol{C} \rangle.$$

Moreover, Birkhoff doubly stochastic matrix theorem [167] states that the set of extremal points of $U(\boldsymbol{1}_n/n, \boldsymbol{1}_n/n)$ is equal to the set of permutation matrices, and the fundamental theorem of linear programming (Theorem 2.7 in [168]) states that the minimum of a linear objective in a finite non-empty polytope is reached at a vertex or

a face of the polytope (i.e., the feasible space of the linear program), leading to the following two cases.

- **Case 1**: $\boldsymbol{W}_\lambda/n \to \boldsymbol{W}_0/n$ converges to a permutation matrix representing the bijective mapping between the optimizing points and the polytope vertices. There exists a vertex evaluation permutation forming the cost matrix such that, the update step $\boldsymbol{s}_k$ is a descending step for each optimizing point

$$\forall \boldsymbol{x}_k \in X_k,\ \boldsymbol{s}_k = \alpha_k \frac{1}{n} \boldsymbol{w}_0^* \boldsymbol{D}^P = \arg\min_{\boldsymbol{d} \in D^P} \{f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d})\} \tag{A.3}$$

  with $\boldsymbol{w}_0^*$ as a row in $\boldsymbol{W}_0^*$, then $\boldsymbol{w}_0^*/n$ is a one-hot vector. Let $\boldsymbol{a} = -\nabla f(\boldsymbol{x}_k)$, $\boldsymbol{s}_k$ is a descending step $f(\boldsymbol{x}_k + \boldsymbol{s}_k) \leq f(\boldsymbol{x}_k)$, then, by Lemma A.1, $\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{s}_k \rangle \leq -\alpha_k \mu_P \|\nabla f(\boldsymbol{x}_k)\|$.

- **Case 2**: $\boldsymbol{W}_\lambda/n \to \boldsymbol{W}_0/n$ converges to a linear interpolation between the permutation matrices defining the neighboring vertices of the polytope. In this case, there are infinite solutions as the linear interpolation between the two bijective maps. There still exists a vertex evaluation permutation forming the cost matrix such that, the update step $\boldsymbol{s}_k$ is the linear interpolation of multiple tied descending steps for each optimizing point, with $\boldsymbol{s}_k = \sum_i b_i \boldsymbol{d}_i$, $\sum_i b_i = 1$, $b_i \geq 0$, $\boldsymbol{d}_i = \arg\min_{\boldsymbol{d} \in D^P} \{f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d})\}$. Following the argument of Case 1, since $\boldsymbol{s}_k$ is the linear interpolation of descending steps, we also conclude that $\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{s}_k \rangle = \sum_i b_i \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}_i \rangle \leq -\sum_i b_i \alpha_k \mu_P \|\nabla f(\boldsymbol{x}_k)\| = -\alpha_k \mu_P \|\nabla f(\boldsymbol{x}_k)\|$.

Finally, starting the L-smooth property of f, we can write

$$\begin{aligned} \forall \boldsymbol{x} \in X,\ \forall k > 0,\ f(\boldsymbol{x}_{k+1}) = f(\boldsymbol{x}_k + \boldsymbol{s}_k) &\leq f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{s}_k \rangle + \frac{L}{2} \|\boldsymbol{s}_k\|^2 \\ &\leq f(\boldsymbol{x}_k) - \mu_P \alpha_k \|\nabla f(\boldsymbol{x}_k)\| + \frac{L}{2} \alpha_k^2 \end{aligned} \tag{A.4}$$

recalling that $\|\boldsymbol{d}\| = 1, \forall \boldsymbol{d} \in D^P$. □

If the sufficient decrease condition does not hold $f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) < c\alpha_k^2$ with some $c > 0$, then the iteration is deemed unsuccessful. In fact, Lemma A.2 is similar to (Lemma 10, [166]), which states that the gradients for these unsuccessful iterations are bounded above by a scalar multiplied with the stepsize. We can see this by

rewriting Eq. (A.2) as

$$\begin{aligned}\|\nabla f(\boldsymbol{x}_k)\| &\leq \frac{1}{\mu_P}\left(\frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1})}{\alpha_k} + \frac{L}{2}\alpha_k\right) \\ &< \frac{1}{\mu_P}\left(c + \frac{L}{2}\right)\alpha_k.\end{aligned}$$

We can implement a check if the sufficient decrease condition holds for ensuring monotonicity in each iteration, as a variant of the Sinkhorn Step.

Lemma A.2 also enables analyzing each optimizing point separately, and hence we can state the following main theorem separately for each $\boldsymbol{x}_k \in X_k$.

Theorem 2.1 (Main result)**.** *If assumption 2.1 and assumption 2.2 holds at each iteration and the stepsize is sufficiently small $\alpha_k = \alpha$ with $0 < \alpha < 2\mu_P\epsilon/L$, then with a sufficient number of iteration*

$$K \geq k(\epsilon) := \frac{f(\boldsymbol{x}_0) - f_*}{(\mu_P\epsilon - \frac{L\alpha}{2})\alpha} - 1,$$

we have $\min_{0\leq k\leq K} \|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$, $\forall \boldsymbol{x}_k \in X_k$.

Proof. We attempt the proof by contradiction, thus we assume $\|\nabla f(\boldsymbol{x}_k)\| > \epsilon$ for all $k \leq k(\epsilon)$. From Lemma A.2, we have $\forall \boldsymbol{x}_k \in X_k$, $\forall k > 0$

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \mu_P\alpha\|\nabla f(\boldsymbol{x}_k)\| + \frac{L}{2}\alpha^2.$$

From assumption 2.1, the objective is bounded below $f_* \leq f(\boldsymbol{x})$. Hence, we can write

$$\begin{aligned}f_* \leq f(\boldsymbol{x}_{K+1}) &< f(\boldsymbol{x}_K) - \mu_P\alpha\|\nabla f(\boldsymbol{x}_K)\| + \frac{L}{2}\alpha^2 \\ &\leq f(\boldsymbol{x}_{K-1}) - \mu_P\alpha(\|\nabla f(\boldsymbol{x}_K)\| + \|\nabla f(\boldsymbol{x}_{K-1})\|) + 2\frac{L}{2}\alpha^2 \\ &\leq f(\boldsymbol{x}_0) - \mu_P\alpha\sum_{k=0}^{K}\|\nabla f(\boldsymbol{x}_k)\| + (K+1)\frac{L}{2}\alpha^2 \\ &\leq f(\boldsymbol{x}_0) - (K+1)\mu_P\alpha\epsilon + (K+1)\frac{L}{2}\alpha^2 \\ &\leq f(\boldsymbol{x}_0) - (K+1)(\mu_P\alpha\epsilon - \frac{L}{2}\alpha^2) \\ &\leq f(\boldsymbol{x}_0) - (f(\boldsymbol{x}_0) - f_*) \\ &= f_*\end{aligned} \tag{A.5}$$

by applying recursively Lemma A.2 and the iteration lower bound at the second last line, which is a contradiction $f_* \leq f_*$. Hence, $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$ for some $k \leq k(\epsilon)$. □

If L is known, we can compute the optimal stepsize $\alpha = \mu_P \epsilon / L$. Then, the complexity bound is $k(\epsilon) = \frac{2L(f(\boldsymbol{x}_0)-f_*)}{\mu_P^2 \epsilon^2} - 1$. Note that Theorem 2.1 only guarantees the gradient of some points in the sequence of Sinkhorn Steps will be arbitrarily small. If in practice, we implement the sufficient decreasing condition $f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq c\alpha_k^2$, then $f(\boldsymbol{x}_K) \leq f(\boldsymbol{x}_i)$, $\|\nabla f(\boldsymbol{x}_i)\| \leq \epsilon$ holds. However, this sufficient decrease check may waste some iterations and worsen the performance. We show in the experiments that the algorithm exhibits convergence behavior without this condition checking. Finally, we remark on the complexity bounds when using different polytope types for Sinkhorn Step under assumption 2.1 and assumption 2.2, by substituting μ_P according to Lemma A.1.

Remark A.1. *By Theorem 2.1, with the optimal stepsize $\alpha = \mu_P \epsilon / L$, the complexity bounds for d-simplex, d-orthoplex and d-cube are $O(d^2/\epsilon^2)$, $O(d/\epsilon^2)$, and $O(1/\epsilon^2)$, respectively.*

The optimal stepsize with d-simplex reports the same complexity $O(d^2/\epsilon^2)$ as the best-known bound for directional-direct search [169]. Within the directional-direct search scope, d-cube reports the new best-known complexity bound $O(1/\epsilon^2)$, which is independent of dimension d since the number of search directions is also increased exponentially with dimension. However, in practice, solving a batch update with d-cube for each iteration is expensive since now the column-size of the cost matrix is 2^d.

A.1.2. Gaussian Process Trajectory Prior

To provide a trajectory prior with tunable time-correlated covariance for trajectory optimization, either as initialization prior or as cost, we introduce a prior for continuous-time trajectories using a GP [170, 171, 62]: $\boldsymbol{\tau} \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{K}(t, t'))$, with mean function $\boldsymbol{\mu}$ and covariance function $\boldsymbol{K}$. As described in [171, 172, 67], a GP prior can be constructed from a linear time-varying stochastic differential equation

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{u}(t) + \boldsymbol{F}(t)\mathbf{w}(t) \tag{A.6}$$

with $\boldsymbol{u}(t)$ the control input, $\boldsymbol{A}(t)$ and $\boldsymbol{F}(t)$ the time-varying system matrices, and $\mathbf{w}(t)$ a disturbance following the white-noise process $\mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \boldsymbol{Q}_c \delta(t - t'))$, where $\boldsymbol{Q}_c \succ 0$ is the power-spectral density matrix. With a chosen discretization time Δt, the

continuous-time GP can be parameterized by a mean vector of Markovian support states $\boldsymbol{\mu} = [\boldsymbol{\mu}(0), ..., \boldsymbol{\mu}(T)]^\intercal$ and covariance matrix $\boldsymbol{K} = [\boldsymbol{K}(i,j)]_{ij,0\leq i,j\leq T}$, $\boldsymbol{K}(i,j) \in \mathbb{R}^{d\times d}$, resulting in a multivariate Gaussian $q(\boldsymbol{\tau}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$. The inverse of the covariance matrix has a sparse structure $\boldsymbol{K}^{-1} = \boldsymbol{D}^\intercal \boldsymbol{Q}^{-1} \boldsymbol{D}$ (Theorem 1 in [171]) with

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{I} & & & & \\ -\boldsymbol{\Phi}_{1,0} & \boldsymbol{I} & & & \\ & & \cdots & & \\ & & & \boldsymbol{I} & \boldsymbol{0} \\ & & & -\boldsymbol{\Phi}_{T,T-1} & \boldsymbol{I} \\ & & & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}, \tag{A.7}$$

and the block diagonal time-correlated noise matrix $\boldsymbol{Q}^{-1} = \text{diag}(\boldsymbol{\Sigma}_s^{-1}, \boldsymbol{Q}_{0,1}^{-1}, \ldots, \boldsymbol{Q}_{T-1,T}^{-1}, \boldsymbol{\Sigma}_g^{-1})$. Here, $\boldsymbol{\Phi}_{t,t+1}$ is the state transition matrix, $\boldsymbol{Q}_{t,t+1}$ the covariance between time step t and $t+1$, and $\boldsymbol{\Sigma}_s, \boldsymbol{\Sigma}_g$ are the chosen covariance of the start and goal states. In this work, we mainly consider the constant-velocity prior (i.e., white-noise-on-acceleration model $\ddot{\mathbf{x}}(t) = \mathbf{w}(t)$), which can approximately represent a wide range of systems such as point-mass dynamics, gravity-compensated robotics arms [62], differential drive [171], etc., while enjoying its sparse structure for computation efficiency. As used in our paper, this constant-velocity prior can be constructed from Eq. (A.6) with the Markovian state representation $\boldsymbol{x} = [\mathbf{x}, \dot{\mathbf{x}}] \in \mathbb{R}^d$ and

$$\boldsymbol{A}(t) = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{d/2} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}, \quad \boldsymbol{u} = \boldsymbol{0}, \quad \boldsymbol{F}(t) = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I}_{d/2} \end{bmatrix} \tag{A.8}$$

Then, following [171, 62], the state transition and covariance matrix are

$$\boldsymbol{\Phi}_{t,t+1} = \begin{bmatrix} \boldsymbol{I}_{d/2} & \Delta t \boldsymbol{I}_{d/2} \\ \boldsymbol{0} & \boldsymbol{I}_{d/2} \end{bmatrix}, \quad \boldsymbol{Q}_{t,t+1} = \begin{bmatrix} \frac{1}{3}\Delta t^3 \boldsymbol{Q}_c & \frac{1}{2}\Delta t^2 \boldsymbol{Q}_c \\ \frac{1}{2}\Delta t^2 \boldsymbol{Q}_c & \Delta t \boldsymbol{Q}_c \end{bmatrix} \tag{A.9}$$

with the inverse

$$\boldsymbol{Q}_{t,t+1}^{-1} = \begin{bmatrix} 12\Delta t^{-3}\Delta t^3 \boldsymbol{Q}_c^{-1} & -6\Delta t^{-2}\Delta t^3 \boldsymbol{Q}_c^{-1} \\ -6\Delta t^{-2}\Delta t^3 \boldsymbol{Q}_c^{-1} & 4\Delta t^{-1} \boldsymbol{Q}_c^{-1} \end{bmatrix}, \tag{A.10}$$

which are used to compute $\boldsymbol{K}$.

Consider priors on start state $q_s(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$ and goal state $q_g(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$, the GP prior for discretized trajectory can be factored as follows [171, 62]

$$\begin{aligned} q_F(\boldsymbol{\tau}) &\propto \exp\big(-\frac{1}{2}\|\boldsymbol{\tau}-\boldsymbol{\mu}\|^2_{\boldsymbol{K}^{-1}}\big) \\ &\propto q_s(\boldsymbol{x}_0)\; q_g(\boldsymbol{x}_T) \prod_{t=0}^{T-1} q_t(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}), \end{aligned} \tag{A.11}$$

where each binary GP-factor is defined

$$q_t(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = \exp\Big\{-\frac{1}{2}\|\boldsymbol{\Phi}_{t,t+1}(\boldsymbol{x}_t - \boldsymbol{\mu}_t) - (\boldsymbol{x}_{t+1} - \boldsymbol{\mu}_{t+1})\|^2_{\boldsymbol{Q}^{-1}_{t,t+1}}\Big\}. \tag{A.12}$$

In the main paper, we use this constant-velocity GP formulation to sample initial trajectories. The initialization GP is parameterized by the constant-velocity straight line $\boldsymbol{\mu}_0$ connecting a start configuration $\boldsymbol{\mu}_s$ to a goal configuration $\boldsymbol{\mu}_g$, having moderately high covariance $\boldsymbol{K}_0$. For using this GP as the cost, we set the zero-mean $\boldsymbol{\mu} = \boldsymbol{0}$ to describe the uncontrolled trajectory distribution. The conditioning $q_g(\boldsymbol{x}_T)$ of the final waypoint to the goal configuration $\boldsymbol{\mu}_g$ is optional (e.g., when the goal configuration solution from inverse kinematics is sub-optimal), and we typically use the $SE(3)$ goal cost.

A.1.3. Additional Discussions Of Batch Trajectory Optimization

Direct implications of batch trajectory optimization. MPOT can be used as a strong oracle for collecting datasets due to the solution diversity covering various modes, capturing homotopy classes of the tasks and their associated contexts. For direct execution, with high variance initialization, an abundance of solutions vastly increases the probability of discovering good local minima, which we can select the best solution according to some criteria, e.g., collision avoidance, smoothness, model consistency, etc.

Solution diversity of MPOT. Batch trajectory optimization can serve as a strong oracle for collecting datasets or striving to discover a global optimal trajectory for execution. Three main interplaying factors contribute to the solution diversity, hence discovering better solution modes. They are

- the step radius α_k annealing scheme,

- the variances of GP prior initialization,
- the number of plans in a batch.

Additional sampling mechanism that promotes diversity, such as Stein Variational Gradient Descent (SVGD) [173] can be straightforwardly integrated into the trajectory optimization problem [77]. This is considered in the future version of this paper to integrate the SVGD update rule with the Sinkhorn Step (i.e., using the Sinkhorn Step to approximate the score function) for even more diverse trajectory planning.

Extension to optimizing batch of different trajectory horizons. Currently, for vectorizing the update of all waypoints across the batch of trajectories, we flatten the batch and horizon dimensions and apply the Sinkhorn Step. After optimization, we reshape the tensor to the original shape. Notice that what glues the waypoints in the same trajectory together after optimization is the log of the Gaussian Process as the model cost, which promotes smoothness and model consistency. Given this pretext, in case of a batch of different horizon trajectories, we address this case by setting maximum horizon $T_{\max}$ and padding with zeros for those trajectories having $T < T_{\max}$. Then, we also set zeros for all rows corresponding to these padded points in the cost matrix $\mathbf{C}^{T_{\max} \times m}$. The padded points are ignored after the barycentric projection. Another way is to maintain an index list of start and end indices of trajectories after flattening, then the cost computation also depends on this index list. Finally, the trajectories with different horizons can be extracted based on the index list. Intuitively, we just need to manipulate cost entries to dictate the behavior of waypoints.

A.1.4. Explicit Trust Region Of The Sinkhorn Step

In trajectory optimization, it is crucial to bound the trajectory update at every iteration to be close to the previous one for stability, and so that the updated parameter remains within the region where the linear approximations are valid. Given $\mathcal{F}(\cdot) : \mathbb{R}^{T \times d} \to \mathbb{R}$ to be the planning cost functional, prior works [61, 97, 62] apply a first-order Taylor expansion at the current parameter $\boldsymbol{\tau}_k$, while adding a regularization norm

$$\Delta \boldsymbol{\tau}^* = \arg\min \left\{ \mathcal{F}(\boldsymbol{\tau}_k) + \nabla \mathcal{F}(\boldsymbol{\tau}_k) \Delta \boldsymbol{\tau} + \frac{\beta}{2} \|\Delta \boldsymbol{\tau}_k\|_{\boldsymbol{M}} \right\}, \tag{A.13}$$

resulting in the following update rule by differentiating the right-hand side w.r.t. $\Delta \boldsymbol{\tau}$ and setting it to zero

$$\boldsymbol{\tau}_{k+1} = \boldsymbol{\tau}_k + \Delta \boldsymbol{\tau}^* = \boldsymbol{\tau}_k - \frac{1}{\beta} \boldsymbol{M}^{-1} \nabla \mathcal{F}(\boldsymbol{\tau}_k). \tag{A.14}$$

The metric $\boldsymbol{M}$ depends on the conditioning prior. Ratliff et al. [61] propose $\boldsymbol{M}$ to be the finite difference matrix, constraining the update to stay in the region of smooth trajectories (i.e., low-magnitude trajectory derivatives). Mukadam et al. [62] use the metric $\boldsymbol{M} = \boldsymbol{K}$ derived from a GP prior, also enforcing the dynamics constraint. It is well-known that solving for $\Delta\boldsymbol{\tau}$ in Eq. (A.13) is equivalent to minimizing the linear approximation within the ball of radius defined by the third term (i.e., the regularization norm) [174]. Hence, these mechanisms can be interpreted as implicitly shaping the *trust region* - biasing perturbation region by the prior, connecting the prior to the weighting matrix $\boldsymbol{M}$ in the update rule.

In contrast, the Sinkhorn Step approaches the *trust region* problem with a gradient-free perspective and provides a novel way to explicitly constrain the parameter updates inside a trust region defined by the regular polytope, without relying on Taylor expansions, where cost functional derivatives are not always available in practice (e.g., planning with only occupancy maps, planning through contacts). In this work, the bound on the trajectory update by the Sinkhorn Step is straightforward

$$\begin{aligned} \|\boldsymbol{\tau}_{k+1} - \boldsymbol{\tau}_k\| &= \left\|\alpha_k \mathrm{diag}(\boldsymbol{n})^{-1} \boldsymbol{W}_\lambda^* \boldsymbol{D}^P\right\| \\ &\leq \sum_{t=1}^{T} \left\|\alpha_k \frac{1}{n} \boldsymbol{w}_\lambda^* \boldsymbol{D}^P\right\| \\ &\leq \sum_{t=1}^{T} \|\alpha_k \boldsymbol{d}^*\| \leq T\alpha_k \end{aligned} \tag{A.15}$$

resulting from $\boldsymbol{D}^P$ being a regular polytope inscribing the $(d-1)$-unit hypersphere. In practice, one could scale the polytope in different directions by multiplying with $\boldsymbol{M}$ induced by priors, and, hence, shape the trust region in a similar fashion. Note that the bound in Eq. (A.15) does not depend on the local cost information.

For completeness of discussion, in sampling-based trajectory optimization, the regularization norm is related to the variance of the proposal distribution. The trajectory candidates are sampled from the proposal distribution and evaluated using the Model-Predictive Path Integral (MPPI) update rule [53]. For example, Kalakrishnan et al. [63] construct the variance matrix similarly to the finite difference matrix, resulting in a sampling distribution with low variance at the tails and high variance at the center. Recently, Urain et al. [67] propose using the same GP prior variance as in [62] to sample trajectory candidates for updates, leveraging them for tuning variance across timesteps.

A.1.5. The Log-Domain Stabilization Sinkhorn Algorithm

Following Proposition 4.1 in [71], for sufficiently small regularization λ, the approximate solution from the entropic-regularized OT problem

$$\boldsymbol{W}_\lambda^* = \arg\min \mathrm{OT}_\lambda(\boldsymbol{n}, \boldsymbol{m})$$

approaches the true optimal plan

$$\boldsymbol{W}^* = \underset{\boldsymbol{W} \in U(\boldsymbol{n}, \boldsymbol{m})}{\arg\min} \langle \boldsymbol{C}, \boldsymbol{W} \rangle.$$

However, small λ incurs numerical instability for a high-dimensional cost matrix, which is usually the case for our case of batch trajectory optimization. Too high λ, which leads to "blurry" plans, also harms the MPOT performance. Hence, we utilize the log-domain stabilization for the Sinkhorn algorithm.

We provide a brief discussion of this log-domain stabilization. For a full treatment of the theoretical derivations, we refer to [89, 90]. First, with the marginals $\boldsymbol{n} \in \Sigma_T$, $\boldsymbol{m} \in \Sigma_m$ and the exponentiated kernel matrix $\boldsymbol{P} = \exp(-\boldsymbol{C}/\lambda)$, the Sinkhorn algorithm aims to find a pair of scaling factors $\boldsymbol{u} \in \mathbb{R}_+^T$, $\boldsymbol{v} \in \mathbb{R}_+^m$ such that

$$\boldsymbol{u} \odot \boldsymbol{P}\boldsymbol{v} = \boldsymbol{n}, \quad \boldsymbol{v} \odot \boldsymbol{P}^\intercal \boldsymbol{u} = \boldsymbol{m}, \tag{A.16}$$

where $\odot$ is the element-wise multiplication (i.e., the Hadamard product). From a typical one vector initialization $\boldsymbol{v}^0 = \boldsymbol{1}_m$, the Sinkhorn algorithm performs a sequence of (primal) update rules

$$\boldsymbol{u}^{i+1} = \frac{\boldsymbol{n}}{\boldsymbol{P}\boldsymbol{v}^i}, \quad \boldsymbol{v}^{i+1} = \frac{\boldsymbol{m}}{\boldsymbol{P}^\intercal \boldsymbol{u}^{i+1}}, \tag{A.17}$$

leading to convergence of the scaling factors $\boldsymbol{u}^*, \boldsymbol{v}^*$ [175]. Then, the optimal transport plan can be computed by $\boldsymbol{W}_\lambda^* = \mathrm{diag}(\boldsymbol{u}^*)\boldsymbol{P}\mathrm{diag}(\boldsymbol{v}^*)$.

For small values of λ, the entries of $\boldsymbol{P}, \boldsymbol{u}, \boldsymbol{v}$ become either very small or very large, thus being susceptible to numerical problems (e.g., floating point underflow and overflow). To mitigate this issue, at an iteration i, Chizat et al. [89] suggests a redundant parameterization of the scaling factors as

$$\boldsymbol{u}^i = \tilde{\boldsymbol{u}}^i \odot \exp\Big(\boldsymbol{a}^i/\lambda\Big), \quad \boldsymbol{v}^i = \tilde{\boldsymbol{v}}^i \odot \exp\Big(\boldsymbol{b}^i/\lambda\Big), \tag{A.18}$$

Algorithm 5: Stabilized Sinkhorn Algorithm

$(\boldsymbol{a}^0, \boldsymbol{b}^0) \leftarrow (\mathbf{0}_T, \mathbf{0}_m),\ (\tilde{\boldsymbol{u}}^0, \tilde{\boldsymbol{v}}^0) \leftarrow (\mathbf{1}_T, \mathbf{1}_m),\ M = 10^3.$
Compute stabilized kernel $\boldsymbol{P}^0$ using Eq. (A.19).
while *termination criteria not met* **do**
    `// Sinkhorn iteration`
    $\tilde{\boldsymbol{u}}^{i+1} = \boldsymbol{n}/(\boldsymbol{P}^i \tilde{\boldsymbol{v}}^i), \quad \tilde{\boldsymbol{v}}^{i+1} = \boldsymbol{m}/(\boldsymbol{P}^{i\intercal} \tilde{\boldsymbol{u}}^{i+1}).$
    `// Check for numerical instabilities`
    if $\|\tilde{\boldsymbol{u}}^i\|_\infty < M \vee \|\tilde{\boldsymbol{v}}^i\|_\infty < M$ **then**
        `// Absorption.`
        $(\boldsymbol{a}^i, \boldsymbol{b}^i) \leftarrow (\boldsymbol{a}^i + \lambda \log(\tilde{\boldsymbol{u}}^i),\ \boldsymbol{b}^i + \lambda \log(\tilde{\boldsymbol{v}}^i)).$
        Compute stabilized kernel $\boldsymbol{P}^i$ using Eq. (A.19).
        $(\tilde{\boldsymbol{u}}^i, \tilde{\boldsymbol{v}}^i) \leftarrow (\mathbf{1}_T, \mathbf{1}_m).$
Return $\boldsymbol{W}_\lambda^* = \mathrm{diag}(\tilde{\boldsymbol{u}}^*)\boldsymbol{P}^*\mathrm{diag}(\tilde{\boldsymbol{v}}^*).$

with the purpose of keeping $\tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}$ bounded, while absorbing extreme values of $\boldsymbol{u}, \boldsymbol{v}$ into the log-domain via redundant vectors $\boldsymbol{a}, \boldsymbol{b}$. The kernel matrix $\boldsymbol{P}^i$ is also stabilized, having elements being modified as

$$\boldsymbol{P}_{tj}^i = \exp\left((\boldsymbol{a}_t^i + \boldsymbol{b}_j^i - \boldsymbol{C}_{tj})/\lambda\right), \tag{A.19}$$

such that large values in $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{C}$ cancel out before the exponentiation, which is crucial for small λ. With these ingredients, we state the log-domain stabilization Sinkhorn algorithm in Algorithm 5. Note that Algorithm 5 is mathematically equivalent to the original Sinkhorn algorithm, but the improvement in the numerical stability is significant.

Nevertheless, in practice, the extreme-value issues are still not resolved completely by Algorithm 5 due to the exponentiation of the kernel matrix $\boldsymbol{P}^i$. Moreover, we only check for numerical issues once per iteration for efficiency. Note that multiple numerical issue checks can be done in an iteration as a trade-off between computational overhead and stability. Hence, tuning for the cost matrix $\boldsymbol{C}$ magnitudes and λ, for the values inside the exp function to not become too extreme, is still required for numerical stability.

A.1.6. Uniform And Regular Polytopes

We provide a brief discussion on the d-dimensional uniform and regular polytope families used in the paper (cf. Section 2.3). For a comprehensive introduction, we refer to [83, 176]. In geometry, regular polytopes are the generalization in any dimensions of regular polygons (e.g., square, hexagon) and regular polyhedra (e.g., simplex, cube). The regular polytopes have their elements as j-facets ($0 \leq j \leq d$) - also called cells, faces, edges, and vertices - being transitive and also regular sub-polytopes of dimension $\leq d$ [83]. Specifically, the polytope's facets are pairwise congruent: there exists an isometry that maps any facet to any other facet.

To compactly identify regular polytopes, a *Schläfli symbol* is defined as the form $\{a, b, c, ..., y, z\}$, with regular facets as $\{a, b, c, ..., y\}$, and regular vertex figures as $\{b, c, ..., y, z\}$. For example,

- a polygon having n edges is denoted as $\{n\}$ (e.g., a square is denoted as $\{4\}$),
- a regular polyhedron having $\{n\}$ faces with p faces joining around a vertex is denoted as $\{n, p\}$ (e.g., a cube is denoted as $\{4, 3\}$) and $\{p\}$ is its *vertex figure* (i.e., a figure of an exposed polytope when one vertex is "sliced off"),
- a regular 4-polytope having cells $\{n, p\}$ with q cells joining around an edge is denoted as $\{n, p, q\}$ having vertex figure $\{p, q\}$, and so on.

A *d-dimensional uniform polytope* is a generalization of a regular polytope - only retaining the vertex-transitiveness (i.e., only vertices are pairwise congruent), and is bounded by its uniform facets. In fact, nearly every uniform polytope can be constructed by Wythoff constructions, such as *rectification*, *truncation*, and *alternation* from either regular polytopes or other uniform polytopes [176]. This implies a vast number of possible choices of vertex-transitive uniform polytopes that can be applied to the Sinkhorn Step. Further research in this direction is interesting.

We present three families of regular and uniform polytopes in Table A.1.1, which are used in this work due to their construction simplicity (see Fig. A.1.1), and their existence for any dimension. Note that there are regular and uniform polytope families that do not exist in any dimension [83]. The number of vertices is $n = d + 1$ for a d-simplex, $n = 2d$ for a d-orthoplex, and $n = 2^d$ for a d-cube.

Construction. We briefly discuss the vertex coordinate construction of d-regular polytopes $\mathcal{P}$ inscribing a $(d-1)$-unit hypersphere with its centroid at the origin. Note

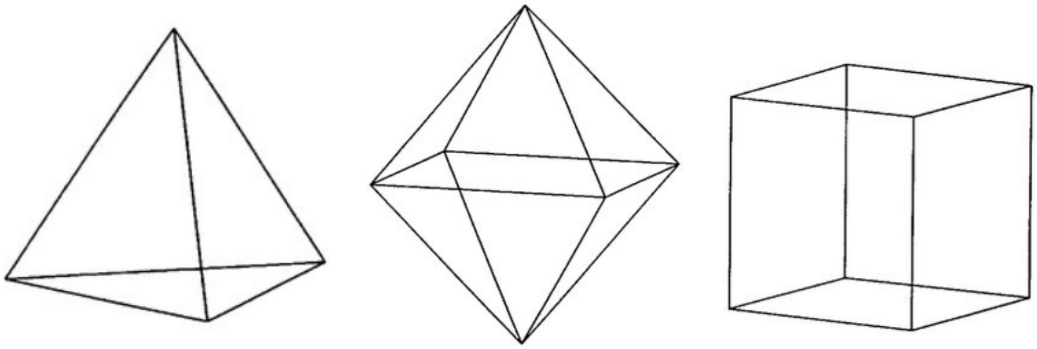


Figure A.1.1.: Examples of (left to right) 3-simplex, 3-orthorplex, 3-cube.

Table A.1.1.: Regular and uniform polytope families.

Dimension	Simplices	Orthoplexes	Hypercubes
$d = 2$	regular trigon $\{3\}$	square $\{4\}$	square $\{4\}$
$d = 3$	regular tetrahedron $\{3, 3\}$	regular octahedron $\{3, 4\}$	cube $\{4, 3\}$
Any d	d-simplex $\{3^{d-1}\}$	d-orthoplex $\{3^{d-2}, 4\}$	d-cube $\{4^{d-2}, 3\}$

that these constructions are GPU vectorizable. First, we denote the standard basis vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d$ for $\mathbb{R}^d$.

For a regular d-simplex, we begin the construction with the *standard* $(d-1)$-simplex, which is the convex hull of the standard basis vectors $\Delta^{d-1} = \{\sum_{i=1}^{d} w_i \boldsymbol{e}_i \in \mathbb{R}^d \mid \sum_{i=1}^{d} w_i = 1, w_i > 0, \text{for } i = 1, \ldots, d\}$. Now, we already got d vertices with the pairwise distance of $\sqrt{2}$. Next, the final vertex lies on the line perpendicular to the barycenter of the standard simplex, so it has the form $(a/d, \ldots, a/d) \in \mathbb{R}^d$ for some scalar a. For the final vertex to form regular d-simplex, its distances to any other vertices have to be $\sqrt{2}$. Hence, we arrive at two choices of the final vertex coordinate $\frac{1}{d}(1 \pm \sqrt{1+d})\mathbf{1}_d$. Finally, we shift the regular d-simplex centroid to zero and rescale the coordinate such that its circumradius is 1, resulting in two sets of $d+1$ coordinates

$$\left(\sqrt{1+\frac{1}{d}}\boldsymbol{e}_i - \frac{1}{d\sqrt{d}}(1 \pm \sqrt{d+1})\mathbf{1}_d\right) \text{ for } 1 \leq i \leq d, \text{ and } \frac{1}{\sqrt{d}}\mathbf{1}_d. \tag{A.20}$$

Note that we either choose two coordinate sets by choosing $+$ or $-$ in the computation.

For a regular d-orthoplex, the construction is trivial. The vertex coordinates are the positive-negative pair of the standard basis vectors, resulting in $2d$ coordinates

$$\boldsymbol{e}_1,\ -\boldsymbol{e}_1,\ \ldots,\ \boldsymbol{e}_d,\ -\boldsymbol{e}_d \tag{A.21}$$

For a regular d-cube, the construction is also trivial. The vertex coordinates are constructed by choosing each entry of the coordinate $1/2$ or $-1/2$, resulting in 2^d vertex coordinates.

A.1.7. d-Dimensional Random Rotation Operator

We describe the random d-dimensional rotation operator applied on polytopes mentioned in Section 2.4. Focusing on the computational perspective, we describe the rotation in any dimension through the lens of matrix eigenvalues. For any d-dimensional rotation, a (proper) rotation matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ acting on $\mathbb{R}^d$ is an orthogonal matrix $\boldsymbol{R}^\intercal = \boldsymbol{R}^{-1}$, leading to $\det(\boldsymbol{R}) = 1$. Roughly speaking, $\boldsymbol{R}$ does not apply contraction or expansion to the polytope convex hull $\text{vol}(\boldsymbol{D}^P) = \text{vol}(\boldsymbol{D}^P\boldsymbol{R})$.

For even dimension $d = 2m$, there exist d eigenvalues having unit magnitudes $\varphi = e^{\pm i\theta_l}$, $l = 1, \ldots, m$. There is no dedicated fixed eigenvalue $\varphi = 1$ depicting the axis of rotation, and thus no axis of rotation exists for even-dimensional spaces. For odd

dimensions $d = 2m + 1$, there exists at least one fixed eigenvalue $\varphi = 1$, and the axis of rotation is an odd-dimensional subspace. To see this, set $\varphi = 1$ in $\det(\boldsymbol{R} - \varphi \boldsymbol{I})$ as follows

$$\begin{aligned} \det(\boldsymbol{R} - \boldsymbol{I}) &= \det(\boldsymbol{R}^\intercal)\det(\boldsymbol{R} - \boldsymbol{I}) = \det(\boldsymbol{R}^\intercal \boldsymbol{R} - \boldsymbol{R}^\intercal) \\ &= \det(\boldsymbol{I} - \boldsymbol{R}) = (-1)^d \det(\boldsymbol{R} - \boldsymbol{I}) = -\det(\boldsymbol{R} - \boldsymbol{I}), \end{aligned} \tag{A.22}$$

with $(-1)^d = -1$ for odd dimensions. Hence, $\det(\boldsymbol{R} - \boldsymbol{I}) = 0$. This implies that the corresponding eigenvector $\boldsymbol{r}$ of $\varphi = 1$ is a fixed axis of rotation $\boldsymbol{R}\boldsymbol{r} = \boldsymbol{r}$. When there are some null rotations in the even-dimensional subspace orthogonal to $\boldsymbol{r}$, i.e., when fixing some $\theta_l = 0$, an even number of real unit eigenvalues appears, and thus the total dimension of rotation axis is odd. In general, the odd-dimensional $d = 2m + 1$ rotation is parameterized by the same number m of rotation angles as in the $2m$-dimensional rotation. As a remark, in $d \geq 4$, there exist pairwise orthogonal planes of rotations, each parameterized by a rotation angle θ. Interestingly, if we smoothly rotate a 4-dimensional object from a starting orientation and choose rotation angle rates such that $\theta_1 = w\theta_2$ with w is an irrational number, the object will never return to its starting orientation.

Construction. We only present random rotation operator constructions that are straightforward to vectorize. More methods on any dimensional rotation construction are presented in [177]. For an even-dimensional space $d = 2m$, by observing the complex conjugate eigenvalue pairs, the rotation matrix can be constructed as a block diagonal of 2×2 matrices

$$\boldsymbol{R}_l = \begin{bmatrix} \cos(\theta_l) & -\sin(\theta_l) \\ \sin(\theta_l) & \cos(\theta_l) \end{bmatrix}, \tag{A.23}$$

describing a rotation associated with the rotation angle θ_l and the pairs of eigenvalues $e^{\pm i\theta_l}$, $l = 1, \ldots, m$. In fact, this construction constitutes a *maximal torus* in the special orthogonal group $SO(2m)$ represented as $T(m) = \{\mathrm{diag}(e^{i\theta_1}, \ldots, e^{i\theta_m}),\ \forall l,\ \theta_l \in \mathbb{R}\}$, describing the set of all simultaneous component rotations in any fixed choice of m pairwise orthogonal rotation planes [178]. This is also a maximal torus for odd-dimensional rotations $SO(2m+1)$, where the group action fixes the remaining direction. For instance, the maximal tori in $SO(3)$ are given by rotations about a fixed axis of rotation, parameterized by a single rotation angle. Hence, we construct a random $d \times d$ rotation matrix by first uniformly sampling the angle vector $\boldsymbol{\theta} \in [0, 2\pi]^m$, then computing in batch the 2×2 matrices Eq. (A.23), and finally arranging them as block diagonal matrix.

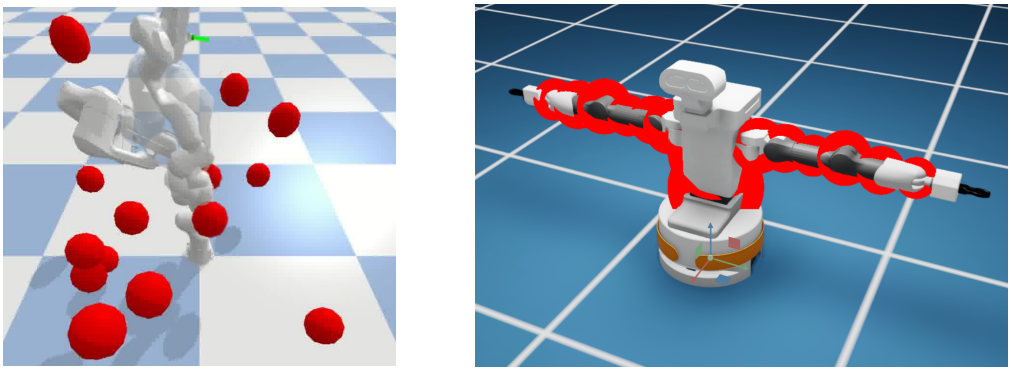


Figure A.1.2.: (Left) An example of the Panda arm plan execution for three simulation frames. The green line denotes a $SE(3)$ goal. (Right) An example of red collision spheres attached to TIAGo++ mesh at a configuration. The collision spheres are transformed with the robot links via forward kinematics.

Fortunately, in this paper, planning in first-order trajectories always results in an even-dimensional state space. Hence, we do not need to specify the axis of rotation. For general construction of a uniformly random rotation matrix in any dimension $d \geq 2$, readers can refer to the Stewart method [179] and our implementation of Steward method at `https://github.com/anindex/ssax/blob/main/ssax/ss/rotation.py#L38`.

A.1.8. Additional Experimental Details

We elaborate on all additional experimental details omitted in the main paper. All experiments are executed in a single RTX3080Ti GPU and a single AMD Ryzen 5900X CPU. Note that due to the fact that all codebases are implemented in PyTorch (e.g., forward kinematics, planning objectives, collision checkings, environments, etc.), hence due to conformity reasons, we also implement RRT*/I-RRT* in PyTorch. However, we set using CPU when running RRT*/I-RRT* experiments and set using GPU for MPOT and the other baselines. An example of Panda execution for collision checking in PyBullet is shown in Fig. A.1.2.

For a fair comparison, we construct the initialization GP prior $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{K}_0)$ with a constant-velocity straight line connecting the start and goal configurations, and sample initial trajectories for all trajectory optimization algorithms. We use the constant-velocity GP prior Eq. (A.9), both in the cost term and for the initial trajectory

Table A.1.2.: Experiment hyperparameters of MPOT. α_0, β_0 are the initial stepsize and probe radius. h is the number of probe points per search direction. *eps* is the annealing rate. P is the polytope type, and λ is the entropic scaling of OT problem.

	α_0	β_0	h	ϵ	P	λ
Point-mass	0.38	0.5	10	0.032	d-cube	0.01
Panda	0.03	0.15	3	0.035	d-orthoplex	0.01
TIAGo++	0.03	0.1	3	0.05	d-orthoplex	0.01

samples. To the best of our knowledge, the baselines are not explicitly designed for batch trajectory optimization. Striving for a unifying experiment pipeline and fair comparison, we reimplement all baselines in PyTorch with vectorization design (beside RRT*) and fine-tune them with the parallelization setting, which is unavailable in the original codebases.

Notably, we use RRT*/I-RRT* as a feasibility indicator of the environments since they enjoy probabilistic completeness, i.e., at an infinite time budget if a solution exists these search-based methods will find the plan. Optimization-based motion planners, like MPOT, GPMP2, CHOMP, and STOMP are only local optimizers. Therefore, if a solution cannot be found by RRT*/I-RRT*, then it is not possible that optimization-based approaches can recover a solution.

MPOT Experiment Settings

For MPOT, we apply ϵ-annealing, normalize the configuration space limits (e.g., position limits, joint limits) into the $[-1, 1]$ range, and do the Sinkhorn Step in the normalized space. MPOT is cost-sensitive due to exponential terms inside the Sinkhorn algorithm, hence, in practice, we normalize the cost matrix to the range $[0, 1]$. The MPOT hyperparameters used in the experiments are presented in Table A.1.2.

Environments

For the *point-mass* environment, we populate 15 square and circle obstacles randomly and uniformly inside x-y limits of $[-10, 10]$, with each obstacle having a radius or width of 2 (cf. Fig. 1.4). We generate 100 environment-seeds, and for each environment-seed,

we randomly sample 10 collision-free pairs of start and goal states, resulting in 1000 planning tasks. We plan each task in parallel 100 trajectories of horizon 64. A trajectory is considered successful if collision-free.

For the *Panda* environment, we also generate 100 environment-seeds. Each environment-seed contains randomly sampled 15 obstacle-spheres having a radius of 10cm inside the x-y-z limits of $[[-0.7, 0.7], [-0.7, 0.7], [0.1, 1.]]$ (cf. Fig. A.1.2), ensuring that the Panda's initial configuration has no collisions. Then, we sample 5 random collision-free (including self-collision-free) configurations, we check with RRT* the feasibility of solutions connecting initial and goal configurations, and then compute the $SE(3)$ pose of the end-effector as a possible goal. Thus, we create a total of 500 planning tasks and plan in parallel 10 trajectories containing 64 timesteps. To construct the GP prior, we first solve inverse kinematics (IK) for the $SE(3)$ goal in PyBullet, and then create a constant-velocity straight line to that goal. A trajectory is considered successful when the robot reaches the $SE(3)$ goal within a distance threshold with no collisions.

In the *TIAGo++* environment, we design a realistic high-dimensional mobile manipulation task in PyBullet (cf. Fig. 2.3). The task comprises two parts: the fetch part and place part; thus, it requires solving two planning problems. Each plan contains 128 timesteps, and we plan a single trajectory for each planner due to the high computational and memory demands. We generate 20 seeds by randomly spawning the robot in the room, resulting in 20 tasks in total. To sample initial trajectories with the GP, we randomly place the robot's base at the front side of the table or the shelf and solve IK using PyBullet. We designed a holonomic base for this experiment. A successful trajectory finds collision-free plans, successfully grasping the cup and placing it on the shelf.

Metrics

Comparing various aspects among different types of motion planners is challenging. We aim to benchmark the capability of planners to parallelize trajectory optimization under dense environment settings. We tune all baselines to the best performance possible for the respective experimental settings and then set the convergence threshold and a maximum number of iterations for each planner.

In all experiments, we consider N_s environment-seeds and N_t tasks for each environment-seed. For each task, we optimize N_p plans having T horizon.

Planning Time. We aim to benchmark not only the success rate but also the *parallelization quality* of planners. Hence, we tune all baselines for each experiment, and then measure the planning time T[s] of trajectory optimizers until convergence or till maximum iteration is reached. T[s] is averaged over $N_s \times N_t$ tasks.

Success Rate. We measure the success rate of task executions over environment-seeds. Specifically, SUC[%] $= N_{st}/N_t \times 100$ with N_{st} being the number of successful task executions (i.e., having at least a successful trajectory in a batch). The success rate is averaged over N_s environment-seeds.

Parallelization Quality. We measure the parallelization quality, reflecting the success rate of trajectories in a single task. Specifically, GOOD[%] $= N_{sp}/N_p \times 100$ with N_{sp} being the number of successful trajectories in a task, and it is averaged over $N_s \times N_t$ tasks.

Smoothness. We measure changing magnitudes of the optimized velocities as smoothness criteria, reflecting energy efficiency. This measure can be interpreted as accelerations multiplied by the time discretization. Specifically, S $= \frac{1}{T}\sum_{t=0}^{T-1} \|\dot{\mathbf{x}}_{t+1} - \dot{\mathbf{x}}_t\|$. S is averaged over successful trajectories in $N_s \times N_t$ tasks.

Path Length. We measure the trajectory length, reflecting natural execution and also smoothness. Specifically, PL $= \sum_{t=0}^{T-1} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|$. PL is averaged over successful trajectories in $N_s \times N_t$ tasks.

Motion Planning Costs

For the obstacle costs, we use an occupancy map having binary values for gradient-free planners (including MPOT) while we implement signed distance fields (SDFs) of obstacles for the gradient-based planners. For self-collision costs, we use the common practice of populating with spheres the robot mesh and transforming them with forward kinematics onto the task space [61, 62]. To be consistent for all planners, joint limits are enforced as an L2 cost for joint violations. Besides the point-mass experiment, all collisions are checked by PyBullet. The differentiable forward kinematics implemented in PyTorch is used for all planners.

Goal Costs. For the $SE(3)$ goal cost, given two points $\boldsymbol{T}_1 = [\boldsymbol{R}_1, \boldsymbol{p}_1]$ and $\boldsymbol{T}_2 = [\boldsymbol{R}_2, \boldsymbol{p}_2]$ in $SE(3)$, we decompose a translational and rotational part, and choose the following distance as cost $d_{SE(3)}(\boldsymbol{T}_1, \boldsymbol{T}_2) = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\| + \|(\text{LogMap}(\boldsymbol{R}_1^\intercal \boldsymbol{R}_2))\|$, where LogMap$(\cdot)$ is the operator that maps an element of $SO(3)$ to its tangent space [180].

Collision Costs. Similar to CHOMP and GPMP2, we populate K *collision spheres* on the robot body (shown in Fig. A.1.2). Given differentiable forward kinematics implemented in PyTorch (for propagating gradients back to configuration space), the obstacle cost for any configuration $\boldsymbol{q}$ is

$$C_{\text{obs}}(\boldsymbol{q}) = \frac{1}{K} \sum_{j=1}^{K} c(\boldsymbol{x}(\boldsymbol{q}, S_j)) \tag{A.24}$$

with $\boldsymbol{x}(\boldsymbol{q}, S_j)$ is the forward kinematics position of the j^{th}-collision sphere, which is computed in batch. For gradient-based motion optimizers, we design the cost using the signed-distance function $d(\cdot)$ from the sphere center to the closest obstacle surface (plus the sphere radius) in the task space with a $\epsilon > 0$ margin

$$c(\boldsymbol{x}) = \begin{cases} d(\boldsymbol{x}) + \epsilon & \text{if } d(\boldsymbol{x}) \geq -\epsilon \\ 0 & \text{if } d(\boldsymbol{x}) < -\epsilon \end{cases}. \tag{A.25}$$

For gradient-free planners, we discretize the collision spheres into fixed probe points, check them in batch with the occupancy map, and then average the obstacle cost over probe points.

Self-collision Costs. We group the collision spheres that belong to the same robot links. Then, we compute the pair-wise link sphere distances. The self-collision cost is the average of the computed pair-wise distances.

Joint Limits Cost. We also construct a soft constraint on joint limits (and velocity limits) by computing the L2 norm violation as cost, with a $\epsilon > 0$ margin on each dimension i

$$C_{\text{limits}}(q_i) = \begin{cases} \|q_{\min} + \epsilon - q_i\| & \text{if } q_i < q_{\min} + \epsilon \\ 0 & \text{if } q_{\min} + \epsilon \leq q_i \leq q_{\max} - \epsilon \\ \|q_{\max} - \epsilon - q_i\| & \text{else} \end{cases}. \tag{A.26}$$

A.1.9. Ablation Study

In this section, we study different algorithmic aspects, horizons and number of paralleled plans, and also provide an ablation on polytope choices.

Algorithmic Ablations

We study the empirical convergence and the parallelization quality over Sinkhorn Steps between the main algorithm MPOT and its variants: MPOT-NoRot - no random rotation applied on the polytopes, and MPOT-NoAnnealing - annealing option is disabled. This ablation study is conducted on the point-mass experiment due to the extremely narrow passages and non-smooth, non-convex objective function, contrasting the performance difference between algorithmic options.

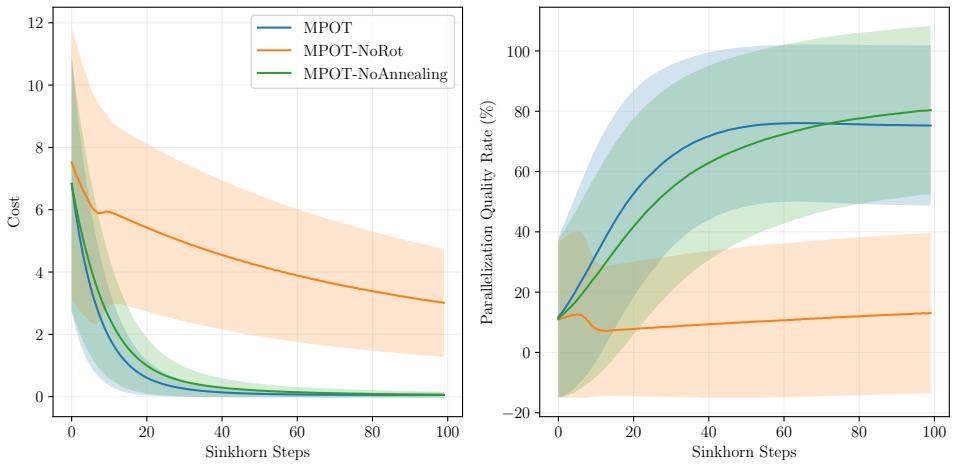


Figure A.1.3.: Ablation study on algorithmic choices in the point-mass environment. All planners are terminated at 100 Sinkhorn Steps. All statistics are evaluated on 1000 tasks as described in Section 2.5.2.

The performance gap between MPOT-NoRot and the others in Fig. A.1.3 is significant. The absence of random rotation on waypoint polytopes leads to biases in the planning cost approximation due to the fixed *probe set* H^P. This approximation bias from non-random rotation becomes more prominent in higher-dimensional tasks due to the sparse search direction set. This experiment result confirms the robustness gained from the random rotation for arbitrary objective function conditions. In other words, the fixed search direction set D^P causes biases in the direction bases, resulting in discarded step direction subspaces.

Between MPOT and MPOT-NoAnnealing, the performance gap depends on the context. MPOT has a faster convergence rate due to annealing the step and probe radius, which leads to a better approximation of local minima. However, it requires careful

tuning of the annealing rate to avoid premature convergence and missing better local minima. MPOT-NoAnnealing converges slower and thus takes more time, but eventually discovers more successful local minima (nearly 80%) than MPOT with annealing (cf. Table A.2.5). This is a trade-off between planning efficiency and parallelization quality with the annealing option.

Flattening Ablations

In both the point-mass and the Panda environments, we experiment with different horizons T and the number of parallel plans N_p. For each (T, N_p) combination, we tune MPOT to achieve a satisfactory success rate and then measure the planning time until convergence, as shown in Fig. A.1.4. The planning time heatmap highlights the batch computation property of MPOT, resulting in a nearly symmetric pattern. Despite long horizons and large batch trajectories, the planning time remains reasonable (under a minute) and can be run efficiently on a single GPU without excessive memory usage, making it suitable, for example, for collecting datasets for learning neural network models.

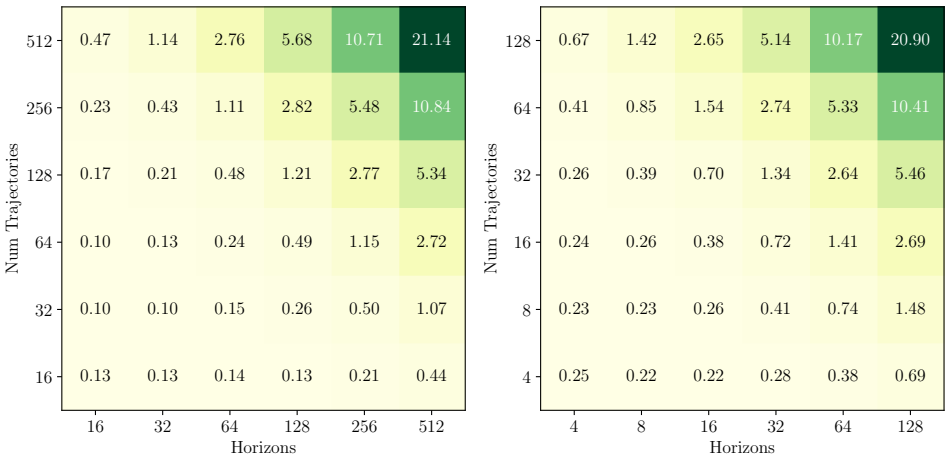


Figure A.1.4.: Planning time heatmap in seconds while varying the horizons and number of paralleled trajectories on both the point-mass (left) and Panda (right) environments.

Polytope Ablations

In Table A.1.3, we compare the performance of MPOT-Orthoplex (i.e., MPOT in the Panda experiments) with its variants: MPOT-Simplex using the d-simplex vertices as

Table A.1.3.: Polytope ablation study on the Panda environment. All statistics are evaluated on 500 tasks as described in Section 2.5.2.

	T[s]	SUC[%]	GOOD[%]	S	PL
MPOT-Random	2.5 $\pm$ 0.0	70.1 $\pm$ 23.7	58.3 $\pm$ 44.3	0.03 $\pm$ 0.01	4.7 $\pm$ 1.2
MPOT-Simplex	**0.5** $\pm$ 0.0	65.8 $\pm$ 24.5	52.1 $\pm$ 45.3	0.01 $\pm$ 0.01	4.6 $\pm$ 1.1
MPOT-Orthoplex	0.8 $\pm$ 0.1	**71.6** $\pm$ 23.2	**60.2** $\pm$ 44.4	0.01 $\pm$ 0.01	4.6 $\pm$ 0.9

search direction set D^P, and MPOT-Random, i.e., not using any polytope structure. For MPOT-Random, we generate 100 points on the 13-sphere ($d = 14$ for the Panda environment) as the search direction set D for each waypoint at each Sinkhorn Step, using the Marsaglia method [181]. As expected, since the d-simplex has fewer vertices than the d-orthoplex, MPOT-Simplex has better planning time but sacrifices some success rate due to a more sparse approximation. MPOT-Random, while achieving a comparable success rate, performs even worse in both planning time and smoothness criteria. We also observe that increasing the number of sampled points on the sphere improves the smoothness marginally. However, increasing the sample points worsens the planning time in general, inducing more matrix columns and instabilities in the already large dimension cost matrix (cf. Appendix A.1.5) of the OT problem. This ablation study highlights the significance of the polytope structure for the Sinkhorn Step in high-dimensional settings.

Smooth Gradient Approximation Ablations

We conduct an ablation on the gradient approximation of Sinkhorn Step w.r.t. different important hyperparameter settings for sanity check of Sinkhorn Step's optimization behavior on a smooth objective function. We choose the Styblinski-Tang function (cf. Fig. A.1.5) in 10D as the smooth objective function due to its variable dimension and multi-modality for non-convex optimization benchmark [182]. We target the most important hyperparameters of *polytope type* P, and entropic regularization scalar λ. These parameters sensitively affect the Sinkhorn Step's optimization performance. We set the other important hyperparameters of *step size* and *probe size* $\alpha = \beta = 0.1$ to be constant, the number of probing points per vertices to be 5 and turn off the annealing option for all optimization runs. The cosine similarity is defined for each particle

$\boldsymbol{x}_i \in X$ as follows:

$$\mathrm{CS}_i = \frac{\boldsymbol{s}(\boldsymbol{x}_i) \cdot (-\nabla f(\boldsymbol{x}_i))}{\|\boldsymbol{s}(\boldsymbol{x}_i)\| \|-\nabla f(\boldsymbol{x}_i)\|} \tag{A.27}$$

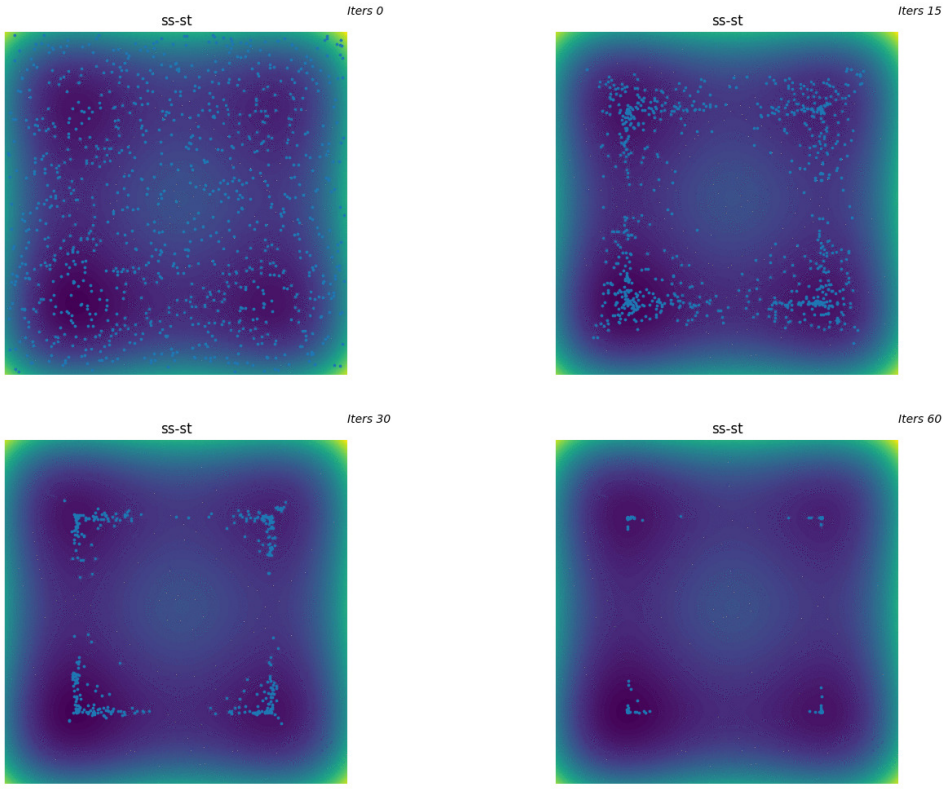


Figure A.1.5.: An example optimization run of 1000 points on the Styblinski-Tang function with Sinkhorn Step. The points are uniformly sampled at the start of optimization. This plot shows the projected optimization run in the first two dimensions.

Regarding this smooth objective, we observe the gradient approximation quality is consistent with Lemma A.1, with increasing cosine similarities for all curves from left to right column (cf. Fig. A.1.6). However, regarding entropic regularization scalar λ, we observe higher cosine similarity and lower curve variance for larger λ. Interestingly, this means higher λ induces both computational benefit solving entropic OT [73] and higher *entropic smoothing bias* [183], where the latter regularizes the gradient approximation

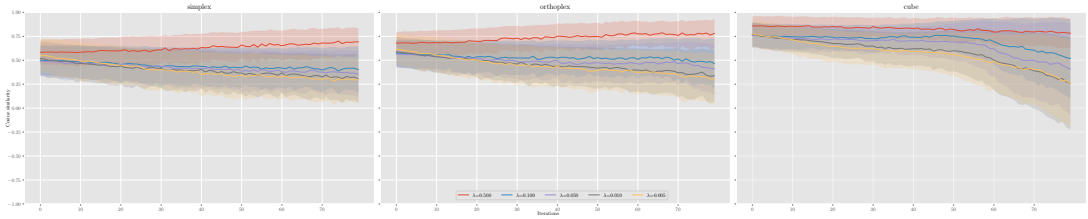


Figure A.1.6.: Ablation study on gradient approximation with cosine similarity between Sinkhorn Step directions and true gradients. We choose the Styblinski-Tang function as the test objective function. Each curve represents an optimization run of 1000 points w.r.t to entropic regularization scalar λ and polytope choice (corresponding to each column), where each iteration shows the mean and variance of cosine similarity of points w.r.t their true gradients. We conduct 50 seeds for each curve, where for all seeds we concatenate the cosine similarities of all optimizing points across the seeds at each iteration.

directions, while it contrarily blurs the result barycenters in the barycenter problem. Notably, this Sinkhorn Step smoothing effect is more necessary in the case of $P = \text{cube}$ toward the end of optimization (i.e., near the local minima/fixed points), where the gradients have small magnitudes and may be noisy while the Sinkhorn Step size is constant. High $\lambda = 0.5$ (red curve) keeps high cosine similarity toward fixed points (cf. Fig. A.1.6), while the lower/sharper λ exhibits degradation due to noisy random rotated 10-cube with constant-size having 2^{10} vertices.

Note that the conclusion drawn from this ablation may not apply to the motion planning application in the main paper since we are evaluating the Sinkhorn Step on smooth objective functions, while the motion planning costs may have an ill-formed cost landscape. Further investigation of (sub)-gradient approximation in various objective function conditions is very interesting for future work.

A.2. Appendix to Chapter 4

A.2.1. Proof of Asymptotic Path Coverage Theorem

Let $u \in \mathbb{F}$ be any path that is uniformly continuous and has bounded variation $TV(u) < \infty$. We begin by constructing a piecewise linear path approximating u, $g_M : [0,1] \to \mathbb{U}$ by dividing the interval $[0,1]$ into $M-1$ subintervals, i.e., $[t_1, t_2], \ldots, [t_{M-1}, t_M]$ with $0 = t_1 < t_2 < \ldots < t_M = 1$. On each subinterval $[t_i, t_{i+1}]$, we define the corresponding segment of g to approximate u

$$\boldsymbol{g}(t) = \boldsymbol{u}(t_i) + \frac{\boldsymbol{u}(t_{i+1}) - \boldsymbol{u}(t_i)}{t_{i+1} - t_i}(t - t_i),\ t \in [t_i, t_{i+1}]. \tag{A.28}$$

Then, we define a control path in $\mathcal{G}(M, N)$.

Definition A.1 (Path In $\mathcal{G}(M,N)$)**.** *A path $u : [0,1] \to \mathbb{U}$ is in $\mathcal{G}(M,N)$ (i.e., $u \in \mathcal{G}(M,N)$) if and only if u is piecewise linear having $M-1$ segments and $\forall 1 \leq i \leq M$, $\boldsymbol{u}(t_i) \in L_i$.*

Lemma A.3 (Piecewise Linear Path Approximation)**.** *Let g_1, g_2 be a piecewise linear function having the same number of partition points $\{\boldsymbol{g}_1(t_i)\}_{i=1}^{M}, \{\boldsymbol{g}_2(t_i)\}_{i=1}^{M}$ with $0 = t_1 < t_2 < \ldots < t_M = 1$, $\|g_1 - g_2\|_\infty < \epsilon$, if and only if $\|\boldsymbol{g}_1(t_i) - \boldsymbol{g}_2(t_i)\| < \epsilon$, $1 \leq i \leq M$.*

Proof. **Sufficiency.** Given $\|\boldsymbol{g}_1(t_i) - \boldsymbol{g}_2(t_i)\| < \epsilon$, $\forall 1 \leq i \leq M$, since g_1, g_2 are piecewise linear functions, the linear interpolation between partition points t_i, t_{i+1} ensures that the difference between g_1, g_2 is maximized at the partition points. Consider g_1, g_2 on a segment $[t_i, t_{i+1}]$

$$\|\boldsymbol{g}_1(t) - \boldsymbol{g}_2(t)\| \leq \max\{\|\boldsymbol{g}_1(t_i) - \boldsymbol{g}_2(t_i)\|, \|\boldsymbol{g}_1(t_{i+1}) - \boldsymbol{g}_2(t_{i+1})\|\} < \epsilon \tag{A.29}$$

Hence, $\|g_1 - g_2\|_\infty = \max_{t \in [0,1]} \|\boldsymbol{g}_1(t) - \boldsymbol{g}_2(t)\| < \epsilon$.

Necessity. Given $\|g_1 - g_2\|_\infty < \epsilon$, then $\|\boldsymbol{g}_1(t_i) - \boldsymbol{g}_2(t_i)\| < \epsilon$, $1 \leq i \leq M$. □

Now, we investigate that any piecewise linear path g with $M-1$ equal subintervals, approximating $u \in \mathbb{F}$, uniformly converges to u as $M \to \infty$.

Lemma A.4 (Convergence Of Linear Path Approximation)**.** *Let g_M be any piecewise linear path approximating $u \in \mathbb{F}$ having M equal subintervals of width $h = 1/M$. Then,*

$$\lim_{M\to\infty} \|u - g_M\|_\infty = 0.$$

Proof. Since u is uniformly continuous on $[0,1]$, for any $\epsilon > 0$, there exists $\delta > 0$ such that for all $t, s \in [0,1]$, if $|t-s| < \delta$, then

$$\|\boldsymbol{u}(t) - \boldsymbol{u}(s)\| < \frac{\epsilon}{2}. \tag{A.30}$$

Also, the variation of u within each subinterval approaches zero as $M \to \infty$ due to the uniformly continuous property. Hence, for sufficiently large M, each subinterval length $h = \frac{1}{M} < \delta$, and thus:

$$\begin{aligned}
\sup_{t\in[t_i,t_{i+1}]} \|\boldsymbol{u}(t) - \boldsymbol{g}_m(t)\| &= \sup_{t\in[t_i,t_{i+1}]} \left\| \boldsymbol{u}(t) - \boldsymbol{u}(t_i) + \frac{\boldsymbol{u}(t_{i+1}) - \boldsymbol{u}(t_i)}{h}(t - t_i) \right\| \\
&\leq \sup_{t\in[t_i,t_{i+1}]} \|\boldsymbol{u}(t) - \boldsymbol{u}(t_i)\| + \sup_{t\in[t_i,t_{i+1}]} \left\| \frac{\boldsymbol{u}(t_{i+1}) - \boldsymbol{u}(t_i)}{h}(t - t_i) \right\| \qquad \text{(A.31)} \\
&< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon. \qquad \text{(A.32)}
\end{aligned}$$

Taking the supremum over all $t \in [0,1]$ (i.e., over M equal subintervals), we obtain:

$$\|u - g_M\|_\infty < \epsilon. \tag{A.33}$$

Since $\epsilon > 0$ is arbitrary and $h \to 0$ as $M \to \infty$, it follows that:

$$\lim_{M\to\infty} \|u - g_M\|_\infty = 0. \tag{A.34}$$

□

We now prove that the random multipartite graph discretization is asymptotically dense in $\mathbb{F}$. Specifically, as the number of layers M and the number of samples per layer N approach infinity, the graph will contain a path that uniformly approximates any continuous path in $\mathbb{F}$.

Theorem 4.1 (Asymptotic Path Coverage)**.** *Let $u \in \mathbb{F}$ be any control path and $\mathcal{G}(M, N)$ be a random multipartite graph with M layers and N uniform samples per layer (cf. Definition 4.1). Assuming a time sequence (i.e., knots) $0 = t_1 < t_2 < \ldots < t_M = 1$ with equal intervals, associating with layers $L_1, \ldots, L_M \in \mathcal{G}(M, N)$ respectively, then*

$$\lim_{M,N\to\infty} \min_{g\in\mathcal{G}(M,N)} \|u - g\|_\infty = 0.$$

Proof. First, Lemma A.4 implies that there exists a sequence of linear piecewise g_M, having $M - 1$ equal intervals approximating u, converging to u as $M \to \infty$.

Let $\hat{g}_M \in \mathcal{G}(M, N)$ be a control path in $\mathcal{G}$ (cf. Definition A.1). Since the time sequence $0 = t_1 < t_2 < \ldots < t_M = 1$ corresponding to layers $L_1, \ldots, L_M \in \mathcal{G}(M, N)$ has equal intervals, we can consider $\hat{g}_M$ having $M - 1$ segments approximating g_M without loss of generality.

Since $\mathbb{U}$ is open, for each $i = 1, \ldots, M$, there exists a ball $B_\epsilon(\boldsymbol{u}(t_i)) \subset \mathbb{U}$, $\epsilon > 0$. By definition $\hat{g}_M, g_M$ has the same number of segments, the event $\|\hat{g}_M - g_M\|_\infty < \epsilon$ is the event that, for each layer $1 \leq i \leq M$, there is at least one point $\boldsymbol{g}_M(t_i)$ is sampled inside the ball $B_\epsilon(\hat{\boldsymbol{g}}_M(t_i))$. The probability that none of the N samples in layer L_i fall inside $B_\epsilon(\boldsymbol{g}_M(t_i))$ is

$$\left(1 - \frac{\mu(B_\epsilon(\boldsymbol{u}(t_i)))}{\mu(\mathbb{U})}\right)^N \leq e^{-cN} \tag{A.35}$$

for some $c > 0$. From Lemma A.3, the probability that every layer contains at least one such sample, such that $\|g_M - \hat{g}_M\|_\infty < \epsilon$, is at least $1 - Me^{-cN}$, which converges to 1 as $N \to \infty$.

From Lemma A.4, for sufficiently large M, we have $\|u - g_M\|_\infty < \epsilon$. Now, due to $\mathbb{U}$ is compact, we can apply the triangle inequality as $N \to \infty$

$$\|u - \hat{g}_M\|_\infty \leq \|u - g_M\|_\infty + \|g_M - \hat{g}_M\|_\infty < \epsilon + \epsilon = 2\epsilon. \tag{A.36}$$

Since ϵ was arbitrary, we conclude that

$$\lim_{M,N\to\infty} \min_{\hat{g}_M\in G(M,N)} \|u - \hat{g}_M\|_\infty = 0. \tag{A.37}$$

□

A.2.2. Exploration Versus Exploitation Discussion

We investigate the tensor sampling Definition 4.1 (with linear interpolation) versus MPPI sampling with horizon T, corresponding to global exploration versus local exploitation behaviors from the current system state. In particular, we remark on the entropy of path distributions in both methods in the discretized control setting with equal time intervals. Further investigation on the continuous control setting is left for future work.

Let a discrete control path be $\boldsymbol{\tau} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_T] \in \mathbb{R}^{T \times n}, \boldsymbol{u} \in \mathbb{U}$. Let $P(\boldsymbol{\tau})$ be the probability of sampling control path $\boldsymbol{\tau}$ under a given planning method. The entropy is then defined as

$$H(P) = -\sum_{\boldsymbol{\tau} \in \mathbb{F}_T} P(\boldsymbol{\tau}) \log P(\boldsymbol{\tau}), \tag{A.38}$$

where $\mathbb{F}_T \subset \mathbb{F}$ is the set of all possible discrete control paths $\boldsymbol{\tau}$ of length T.

Consider $\mathcal{G}(M, N)$, each node in layer L_i is sampled independently from a uniform distribution over $\mathbb{U}$, and path candidates are equivalent to sequences of node indices $\boldsymbol{\tau} \sim \tau = (i_1, i_2, \dots, i_M) \in \{1, \dots, N\}^M$ (cf. algorithm 3). Let $\mathcal{S}$ denote the set of all index sequences representing valid paths through the graph. The uniform distribution over $\mathcal{S}$ is given by $P_{\mathcal{G}}(\boldsymbol{\tau}) = 1/|\mathcal{S}|$, where $|\mathcal{S}| = N^M$. Hence, the entropy of tensor sampling is

$$H(P_{\mathcal{G}}) = -\sum_{\tau \in \mathcal{S}} (1/N^M) \log\Big(1/N^M\Big) = \log\Big(N^M\Big) = M \log N. \tag{A.39}$$

Indeed, as $M, N \to \infty$, the entropy $H(P_{\mathcal{G}}) \to \infty$, and the distribution over sampled paths in $\mathcal{G}$ becomes maximum entropy over $\mathbb{F}_T$ among all discrete path distributions. Theorem 4.1 implies that $\mathbb{F}_T \to \mathbb{F}$ as $M, N \to \infty$, and thus tensor sampling distribution becomes maximum entropy over $\mathbb{F}$.

Now, typical MPPI implementation generates control paths by perturbing a nominal trajectory $\bar{\boldsymbol{\tau}}$ with Gaussian noise [184] $\boldsymbol{u}_t = \bar{\boldsymbol{u}}_t + \boldsymbol{\epsilon}_t$, $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, and propagating the dynamics to generate a state trajectory. The path distribution P_{MPPI} concentrates around $\bar{\boldsymbol{\tau}}$ and is generally non-uniform. The entropy is constant and computed in closed form

$$H(P_{\text{MPPI}}) = \frac{Tn}{2}(1 + \log(2\pi)) + T \log\det(\boldsymbol{\Sigma}), \tag{A.40}$$

due to independent Gaussian noise over timestep (i.e., white noise kernel [185]).

In general, tensor sampling serves as a configurable high-entropy sampling mechanism over control trajectory space, offering maximum exploration, while MPPI targets local improvement around a nominal trajectory, thereby performing exploitation. This distinction motivates the hybrid method, where we mix explorative (smooth) controls with local controls sampled from a typical white noise kernel.

A.2.3. Task Details

Here, we provide task details on the task, cost definitions, and their domain randomization. There exist motion capture sensors in MuJoCo to implement the tasks. For this paper, we deliberately design the task costs to be simple and set sufficiently short planning horizons to benchmark the exploratory capacity of algorithms. In practice, one may design dense guiding costs to make tasks easier.

Crane. The agent controls a luffing crane via torque inputs to move a suspended payload to a target while minimizing oscillations. The cost function penalizes payload deviation and swing:

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 \|\boldsymbol{x}_t - \boldsymbol{x}_g\|^2 + \alpha_2 \|\dot{\boldsymbol{x}}_t\|^2, \tag{A.41}$$

where $\boldsymbol{x}_t$ is the payload tip position, $\boldsymbol{x}_g$ is the target point, and $\dot{\boldsymbol{x}}_t$ is the tip velocity. Success is achieved when the payload tip is within a small radius of the target location. This task is difficult due to heavy modeling errors and underactuation, which is common in real crane applications.

Cube-In-Hand. Using velocity control of a dexterous LEAP hand, the agent must rotate a cube to match a randomly sampled target orientation. The cost combines position and orientation error:

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 d_{\mathrm{SE}(3)}(\boldsymbol{x}_t, \boldsymbol{x}_g)^2 + \alpha_2 \|\dot{\boldsymbol{x}}_t\|^2, \tag{A.42}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{x}_g$ are the current cube and target poses, $d_{\mathrm{SE}(3)}$ is the SE(3) distance metric between poses, and $\boldsymbol{u}_t = \dot{\boldsymbol{x}}_t$. Success is defined when the combined position and orientation errors fall below a threshold. This task is difficult due to the high-dimensional, contact-rich, and failure mode of the falling cube.

G1-Walk. A Unitree G1 humanoid robot tracks a motion-captured walking trajectory using position control. The cost is defined as the deviation from reference joint positions:

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 \|\boldsymbol{x}_t - \boldsymbol{x}_{\mathrm{ref}}(t+k)\|^2, \tag{A.43}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{x}_{\text{ref}}(t+k)$ are the current joint and reference joint positions, given the current control iteration k. Success is not binary but is measured by minimizing deviation from the reference joint configurations. Note that this cost is not designed for stable locomotion. The main challenge is maintaining motion tracking locomotion over long horizons with complex joint couplings.

G1-Standup. The humanoid must rise from a lying pose to an upright standing posture. The cost penalizes deviation from upright pose and instability:

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 (h_t - h^*)^2 + \alpha_2 d_{\text{SO(3)}}(\boldsymbol{R}_{\text{torso}}, \boldsymbol{R}_g)^2 + \alpha_3 \|\boldsymbol{q}_t - \boldsymbol{q}_{\text{nominal}}\|^2, \tag{A.44}$$

where h_t, h^* is the torso height and the standing height threshold, $\boldsymbol{R}_t^{\text{torso}}, \boldsymbol{R}_g$ are the orientation of current and target torso. $h_t, \boldsymbol{R}_t^{\text{torso}}, \boldsymbol{q}_t$ are elements of $\boldsymbol{x}_t$. Success is defined when the height of the torso exceeds a target threshold. The task is difficult due to large initial instability and the need to achieve balance in high-dimensional dynamics.

PushT. A position-controlled end effector to push a T-shaped block to a goal pose. The cost measures block pose error

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 d_{\text{SE(3)}}(\boldsymbol{x}_t, \boldsymbol{x}_g)^2, \tag{A.45}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{x}_g$ are the current T-block and target poses. Success is achieved when the block's position and orientation errors are minimized. The task is challenging because contact dynamics is complex, requiring precise interaction strategies.

Walker. A planar biped must walk forward at a desired velocity while maintaining an upright torso. The cost function penalizes deviation from target velocity and orientation:

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 (h_t - h^*)^2 + \alpha_2 (\theta_t - \theta^*) + \alpha_3 (v_t - v^*)^2 + \alpha_4 \|\boldsymbol{u}_t\|^2, \tag{A.46}$$

where h_t, h^* is the torso height and the standing height threshold, v_t, v^* is the forward and target velocity, θ_t, θ^* is the torso and target pitch angle. h_t, θ_t, v_t are elements of $\boldsymbol{x}_t$. Success is measured by stable forward motion and velocity tracking. The difficulty lies in generating stable gaits without explicit foot placement planning.

Navigation. A planar point mass moves in a bounded 2D space via velocity commands to reach a target while avoiding collisions. The state cost is defined as

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha_1 \exp(-\lambda d_{\text{wall}}(\boldsymbol{x}_t)) + \alpha_2 \|\boldsymbol{x}_t - \boldsymbol{x}_g\|^2 + \alpha_3 \|\boldsymbol{u}_t\|^2, \tag{A.47}$$

where $d_{\text{wall}}(\cdot)$ is the distance to the closest wall, $\boldsymbol{x}_g$ is the goal position, and $\boldsymbol{u}_t$ is the velocity control. Success is defined when the agent's distance to the target is sufficiently small. This task is extremely difficult for sampling-based MPC due to large local minima near the starting point.

Table A.2.1 summarizes the environments used in our experiments, including their state and action space dimensions, control modalities, and whether domain randomization or task randomization was applied. We set the control horizon and sim step per plan such that they resemble realistic control settings.

Table A.2.1.: Summary of environment properties.

Task	State Dim	Action Dim	Control Type	Domain Randomization
`Navigation`	4	2	Velocity	Joint obs. noise, actuation gain, init position
`Crane`	24	3	Torque	Payload mass, inertia, joint damping, actuation gain
`Cube-In-Hand`	39	16	Velocity	Joint obs. noise, geom friction
`G1-Walk`	142	29	Position	Joint obs. noise, geom friction
`G1-Standup`	71	29	Position	Joint obs. noise, geom friction
`PushT`	14	2	Position	Geom friction, init pose
`Walker`	18	6	Torque	None (fixed init)

A.2.4. Experiment Details

Comparison Experiment. We summarize the simulation settings for comparison experiments (cf. Section 4.4.2) in Table A.2.2, and the hyperparameters used for MTP variants in Table A.2.3. Tables A.2.4 summarize the hyperparameters used for MPPI and PS (temperature is not relevant for PS). The same noise σ is used for MTP local samples. For evolutionary strategies (DE and OpenAI-ES), we use default hyperparameters in `evosax` [148].

Design Ablation. We conducted the sweep on $\beta \in [0, 1]$ and the number of elites to experimentally study the algorithmic design. The MTP hyperparameters in Section 4.4.3 are the same as in Table A.2.3. Each setting was evaluated with 4 random seeds.

Sensitivity Ablation. We conducted β-mixing rate ablation study (cf. Section 4.4.4) by varying the β across different MTP variants. The MTP hyperparameters in Section 4.4.4 are the same as in Table A.2.3. Each configuration was evaluated with 4 random seeds.

Experimental Discussions. In tasks with well-shaped or dense reward structures and moderately nonlinear dynamics, exploration becomes less critical and nominal sampling methods (e.g., MPPI, PS) often suffice—explaining MPPI's strong performance in

Table A.2.2.: Simulation Settings for Experiments

Task	Horizon Δt [s]	Horizon	Sim Step/Plan	Sim Hz	Num. Randomizations
`Navigation`	0.05	20	2	100	8
`Crane`	0.4	2	16	500	32
`Cube-In-Hand`	0.04	3	2	100	8
`G1-Walk`	0.1	4	1	100	4
`G1-Standup`	0.2	3	1	100	4
`PushT`	0.1	5	10	1000	4
`Walker`	0.15	4	15	200	1

Table A.2.3.: MTP Hyperparams

Task	M	N	$\sigma_{\min}$	Elites	β	α
`Navigation`	5	30	-	-	1.0	-
`Crane`	2	30	0.05	8	0.5	0.0
`Cube-In-Hand`	2	50	0.15	5	0.5	0.1
`G1-Standup`	2	100	0.2	100	0.05	0.0
`G1-Walk`	2	100	0.1	100	0.02	0.0
`PushT`	3	50	0.1	20	0.5	0.0
`Walker`	2	50	0.3	20	0.5	0.5

Table A.2.4.: PS/MPPI Hyperparams

Task	Noise Std. σ	Temperature
`Navigation`	1.0	0.1
`Crane`	0.05	0.1
`Cube-In-Hand`	0.15	0.1
`G1-Standup`	0.2	0.1
`G1-Walk`	0.1	0.01
`PushT`	0.3	0.1
`Walker`	0.3	0.1

`G1-Standup`, which benefits from fully actuated dynamics and informative rewards. However, in more challenging scenarios involving sparse rewards or highly nonlinear dynamics, such as `PushT` and under domain shifts in `Crane`, these locally guided strategies tend to struggle with inadequate exploration, often converging to suboptimal solutions or showing high performance variance. A representative case is the `Navigation` task (Figure 3), where the agent must discover velocity sequences to bypass obstacles and reach the goal—a setting in which local Gaussian sampling clearly fails by getting trapped in local minima. To address these challenges, MTP introduces a structured high-entropy sampling mechanism along with a simple yet effective β-mixing strategy that balances global exploration and local exploitation. With careful tuning of β, M, and N, MTP demonstrates robust and consistent performance across diverse tasks.

Mixing Rate Tuning. β determines the ratio between exploratory (tensor sampling) and exploitative (nominal sampling) samples, which is delicate to tune. As shown in our ablations in Fig. 4.6 and Fig. 4.5, high β values (e.g., 0.5) introduce strong

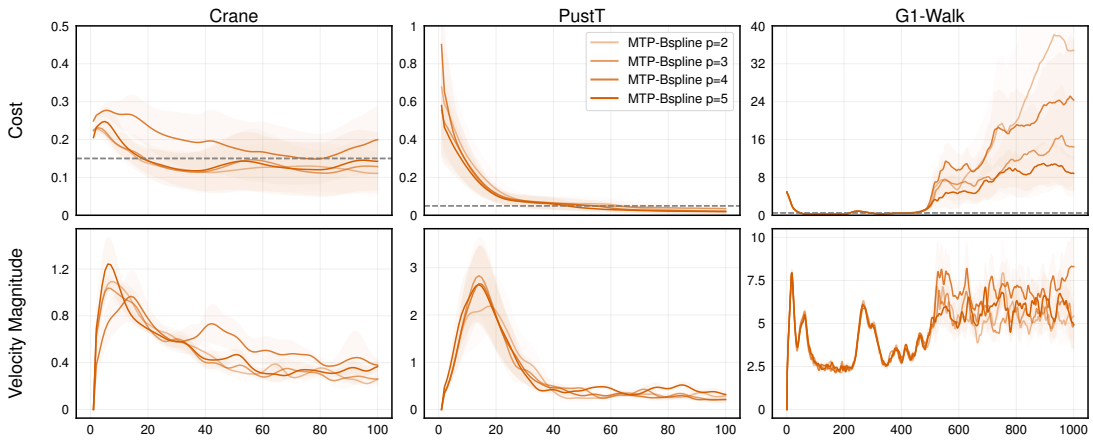


Figure A.2.1.: MTP-Bspline degree ablation. In `G1-Walk`, the Unitree G1 controlled Bspline degrees all roughly fall at 500 time steps.

exploration, which may benefit tasks with sparse rewards and not requiring delicate fixed-point stability (e.g., `PushT`, `Cube-In-Hand`, `Navigation`). Conversely, in tasks requiring high stability or precise actuation, such as `G1-Standup` or `G1-Walk`, lower β values (e.g., 0.05-0.1) tend to yield better performance by favoring consistent behavior while still injecting enough exploration to escape suboptimal solutions.

A.2.5. Additional Ablation & Performance Benchmarks

In this section, we conduct more MTP ablations to understand how hyperparameters affect MTP performance, and also briefly benchmark the baseline JAX implementations to confirm the real-time performance.

B-spline Degree Ablation. We investigate the sensitivity of MTP performance over B-spline interpolation degrees. The MTP hyperparameters are the same as in Table A.2.3. Each setting was evaluated using 5 random seeds.

In Fig. A.2.1, we investigate the sensitivity of MTP performance over B-spline interpolation degrees. Results consistently show minimal performance differences across degrees ($p = 2$ to $p = 5$) in terms of both cost and velocity magnitude curves across different tasks. Given this insensitivity, we select the lower computational complexity B-spline $p = 2$ as our default choice for the MTP-Bspline method.

Sweep M, N Ablation. We conducted an additional ablation to study the effect of varying M and N values of MTP variants on the `Navigation` task. We set $\beta = 1$

to use full tensor sampling. Each configuration was evaluated with 4 random seeds. According to Fig. A.2.2, there exists a sweet point in selecting the number of layers M

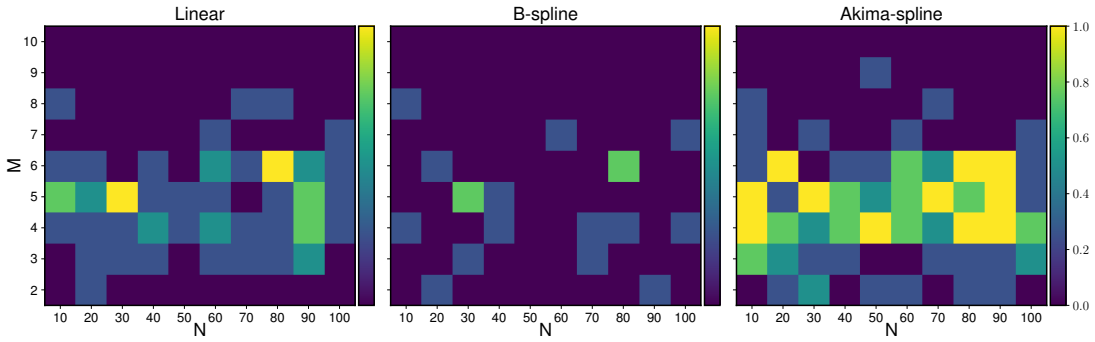


Figure A.2.2.: Sweep M, N on `Navigation` environment with $B = 256$ to investigate the interplay between number of batch sample B, number of layer M, and number of control-waypoints per layer N. Each data point is the success rate over 4 seeds. The environment setting is as in Appendix A.2.3.

in tensor sampling. Roughly, for all MTP variants, increasing M initially improves task performance, as more layers provide sufficient path complexity, allowing the planner to escape local minima and generate diverse, globally exploratory trajectories. However, beyond a certain point, further increasing M degrades performance. This is due to the exponential growth in the number of possible paths $\mathcal{O}(N^M)$, while the rollout budget B remains fixed. As a result, the sampled trajectory density becomes sparse relative to the vast number of paths, reducing effective coverage and leading to diminished exploration and performance. On the other hand, increasing N consistently improves performance by densifying the search at each layer. However, this comes at a higher computational cost. Therefore, a careful balance must be struck between M and N to maintain real-time control and effective control exploration.

Planning Performance. Table A.2.5 benchmarks the performance of our JAX-based implementation by measuring the wall-clock time of the JIT-compiled planning function on `G1-Standup` task. This function includes a single sampling, single trajectory rollout, single cost evaluation, and single parameter update. Note that this benchmark only serves as an exemplary simulated planning function performance with JAX JIT. In practice, we might have multiple search refinements, or multiple parameter updates in the control loops. The task setting is similar to Table A.2.2, but we set the sim step per plan to 1 with $B = 128$. The results show that the initial JIT compilation incurs a significant one-time cost, as expected for MuJoCo XLA pipelines.

However, after compilation, the per-step planning rates across MTP, MPPI, PS, and evolutionary baselines are roughly similar with the same batch sample B, as also reflected in Table A.2.6, Table A.2.7, and Table A.2.8. The results confirm that the JIT [s] and Planning Time [ms] are algorithm-agnostic, which depends slightly on batch size B (i.e., Planning Time [ms] logarithmically increases with B) and on the environment dynamics. All algorithms remain real-time feasible on GPU-accelerated hardware, when implemented with JAX and MuJoCo XLA.

Table A.2.5.: JAX implementation benchmark on `G1-Standup`, evaluated with 5 seeds on an Nvidia RTX 3090.

	MTP-Bspline	MTP-Akima	PS	MPPI	OpenAI-ES	DE
JIT Time [s]	76.4 ± 1.2	74.62 ± 2.5	72.35 ± 4.5	73.87 ± 4.2	69.87 ± 1.2	73.62 ± 3.1
Planning Time [ms]	2.7 ± 0.3	2.7 ± 0.4	3.1 ± 0.7	2.6 ± 0.2	2.9 ± 0.5	3.2 ± 0.6

Table A.2.6.: Planning performance of **MTP-Akima**. Averaged over 5 seeds on an Nvidia RTX 4090.

Batch Size B	**JIT Time [s]**				**Planning Time [ms]**			
	`PushT`	`Crane`	`Cube-In-Hand`	`G1-Walk`	`PushT`	`Crane`	`Cube-In-Hand`	`G1-Walk`
64	15.8	32.5	38.2	65.3	1.7 ± 0.1	1.8 ± 0.1	8.1 ± 0.3	1.4 ± 0.1
128	12.7	31.5	36.6	58.5	1.6 ± 0.1	1.9 ± 0.1	10.2 ± 0.7	1.5 ± 0.1
256	16.3	31.4	38.8	57.1	2.0 ± 0.2	2.0 ± 0.4	14.7 ± 0.8	1.5 ± 0.1

Table A.2.7.: Planning performance of **MPPI**. Averaged over 5 seeds on an Nvidia RTX 4090.

Batch Size B	**JIT Time [s]**				**Planning Time [ms]**			
	`PushT`	`Crane`	`Cube-In-Hand`	`G1-Walk`	`PushT`	`Crane`	`Cube-In-Hand`	`G1-Walk`
64	14.9	32.3	37.7	62.9	1.7 ± 0.1	1.7 ± 0.1	8.2 ± 0.3	1.4 ± 0.1
128	18.4	29.9	36.7	58.2	1.7 ± 0.1	1.8 ± 0.1	10.4 ± 0.5	1.4 ± 0.1
256	14.9	30.3	37.5	55.7	1.9 ± 0.1	1.8 ± 0.1	15.2 ± 0.9	1.4 ± 0.1

This demonstrates that MTP, despite its global exploration capabilities, remains suitable for real-time control applications. Our implementation benefits from efficient JIT and `vmap` vectorization in JAX and is compatible with MuJoCo's XLA backend. These design choices ensure that sampling, rollout, and learning components of MTP

Table A.2.8.: Planning performance of **OpenAI-ES**. Averaged over 5 seeds on an Nvidia RTX 4090.

Batch Size B	**JIT Time** [s]				**Planning Time** [ms]			
	`PushT`	`Crane`	`Cube-In-Hand`	`G1-Walk`	`PushT`	`Crane`	`Cube-In-Hand`	`G1-Walk`
64	15.1	32.1	39.2	61.5	1.7 ± 0.1	1.8 ± 0.1	8.2 ± 0.3	1.6 ± 0.1
128	19.0	30.1	38.4	57.9	1.7 ± 0.1	1.9 ± 0.1	10.3 ± 0.4	1.5 ± 0.1
256	16.1	30.9	42.3	55.1	2.0 ± 0.1	1.8 ± 0.1	14.9 ± 0.7	1.5 ± 0.1

are fully optimized and scalable, and they support advanced techniques such as online domain randomization. Overall, the benchmark confirms the practicality of deploying MTP in high-performance robotic control loops.

Softmax Update Effect. Fig. A.2.3 illustrates the impact of applying `softmax` weighting on elite candidates for updating the mean and standard deviation of control trajectories. The left plot demonstrates smoother and lower-variance control updates over time compared to updates without `softmax` weighting shown on the right. The smooth and stable updates afforded by `softmax` weighting are essential when effectively mixing global and local samples, highlighting its critical role in the MTP performance.

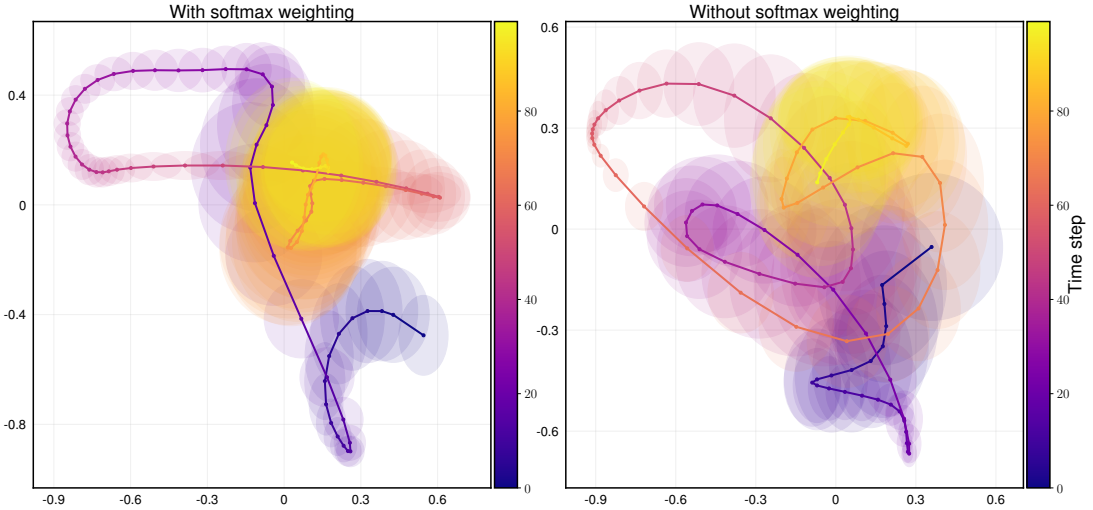


Figure A.2.3.: `Softmax` weighting ablation on `PushT` environment. Both control update trajectories converge to near-zero means with large variance at 100 timesteps, signifying task completion.

Publications

Journal Articles

An T Le et al. “Anytime Global Tensor Motion Planning”. In: *In preparation for IEEE Robotics and Automation Letters (RA-L)* (2025)

An T Le et al. “Accelerating motion planning via optimal transport”. In: *In preparation for IEEE Transactions on Robotics (TRO)* (2025)

An T. Le et al. “Model Tensor Planning”. In: *Transactions on Machine Learning Research (TMLR)* (2025)

An T Le et al. “Global Tensor Motion Planning”. In: *IEEE Robotics and Automation Letters (IEEE RA-L)* (2025)

Joe Watson et al. “Machine Learning with Physics Knowledge for Prediction: A Survey”. In: *Transactions on Machine Learning Research (TMLR)* (2025)

Joao Carvalho et al. “Grasp Diffusion Network: Learning Grasp Generators from Partial Point Clouds with Diffusion Models in SO (3) xR3”. In: *Submitted to IEEE Robotics and Automation Letters (IEEE RA-L)* (2024)

Joao Carvalho et al. “Motion planning diffusion: Learning and adapting robot motion planning with diffusion models”. In: *IEEE Transactions on Robotics (TRO)* (2025)

Georgia Chalvatzaki et al. “Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning”. In: *Frontiers in Robotics and AI* 10 (2023)

Conference Papers

Khang Nguyen et al. “TD-GRPC: Temporal Difference Learning with Group Relative Policy Constraint for Humanoid Locomotion”. In: *Submitted to IEEE-RAS International Conference on Humanoid Robots* (2025)

Kay Pompetzki et al. “Geometrically-Aware Goal Inference: Leveraging Motion Planning as Inference”. In: *German Robotics Conference (GRC)* (2025)

An T Le et al. “Kinematics Correspondence As Inexact Graph Matching”. In: *German Robotics Conference (GRC)* (2025)

Viet The Nguyen et al. “Persistent Homology-induced Graph Ensembles for Time Series Regressions”. In: *Submitted to NeurIPS* (2025)

Jiayun Li et al. “Constrained Gaussian Process Motion Planning via Stein Variational Newton Inference”. In: *arXiv preprint arXiv:2504.04936* (2025)

Khang Nguyen et al. “FlowMP: Learning Motion Fields for Robot Planning with Conditional Flow Matching”. In: *Submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2025)

An T Le* et al. “Dude: Dual distribution-aware context prompt learning for large vision-language model”. In: *Asian Conference on Machine Learning (ACML)* (2024)

Duy MH Nguyen et al. “Structure-aware e (3)-invariant molecular conformer aggregation networks”. In: *International Conference on Machine Learning (ICML)* (2024)

An T Le et al. “Accelerating motion planning via optimal transport”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 78453–78482

An T. Le et al. “Hierarchical Policy Blending As Optimal Transport”. In: *Learning for Dynamics and Control Conference*. PMLR. 2023, pp. 797–812

Joao Carvalho et al. “Motion planning diffusion: Learning and planning of robot motions with diffusion models”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 1916–1923

Julen Urain et al. “Learning implicit priors for motion optimization”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 7672–7679

An T Le et al. “Learning forceful manipulation skills from multi-modal human demonstrations”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 7770–7777

An T Le et al. “Hierarchical Human-Motion Prediction and Logic-Geometric Programming for Minimal Interference Human-Robot Tasks”. In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE. 2021, pp. 7–14

Curriculum Vitae

An T. Le

anthaile.com an@robot-learning.de
Room D202, Building S2|02 TU Darmstadt, FG IAS Hochschulstr. 10, 64289 Darmstadt

Education

Technische Universität Darmstadt *Darmstadt, Germany* **Dec 2021 – now**

Computer Science Ph.D.

- Part of the Intelligent Autonomous Systems group researching Robot Learning, supervised by Prof. Jan Peters.

Universität Stuttgart *Stuttgart, Germany* **Nov 2019 – Nov 2021**

Information Technology M.Sc. *First class*

- *Thesis*: Learning task-parameterized Riemannian motion policies.
- Modules include: Maths for Intelligent Systems, Image Processing, Network Systems

Frankfurt University of Applied Science *Frankfurt, Germany* **Nov 2015 – Oct 2019**

Electrical Engineering B.Eng. *First class*

- *Thesis*: Approaches to solve kidnapped robot problem

Honours

Info-Preis For Best Diploma Award 2022	Sony Research Awards 2021
Deutschlandstipendium (2020, 2021)	DAAD Scholarship 2019
eSilicon Scholarship (2017, 2018)	AmCham Scholarship 2017

Experience

Assistant Professor, Vin University *Hanoi, Vietnam* **September 2025 – Now**

Co-Founder, VinRobotics *Hanoi, Vietnam* **Jan 2025 – Now**

- Lead developing robotics software stacks of locomotion and manipulation tasks for humanoid robots.

Research Intern, Bosch AI *Renningen, Germany* **May 2020 – Dec 2020**

- Worked on forceful imitation learning applying to E-bike assembly task, hosted by Dr. Meng Guo in the robotics team.

Research Assistant, HLRS *Stuttgart, Germany* **Nov 2019 – Apr 2020**

- Implemented back-end functionalities in DASH project http://www.dash-project.org/
- Maintained and configure HPC systems in HLRS.

Engineer Intern, Intel Corporation *HCMC, Vietnam* **Jan 2017 – May 2017**

- Designed and implemented data analysis systems to process and analyze high-volume unit test data generated in the manufacturing line.
- Weekly validated and reported the quality of the Intel Thunderbolt Product manufacturing line.

Competencies

Software engineering (Python, C++ 14 & 17, UNIX, git), Machine Learning (PyTorch, Jax, Tensorflow V2, Hydra), Robotics (ROS2, Isaac Sim, MuJoCo, Genesis), Robots (TIAGo++, Unitree G1, Unitree H1-2, Franka Emika Panda, Optitrack, KUKA Rover), Design (Inkscape, LaTeX).

Academic Interests

Scaling robot planning, Dynamics generative models, Optimal Transport theory and gradient flows.

Academic Service

- Reviewed for CoRL (2023, 2024, 2025), IROS (2021, 2022, 2023, 2024, 2025), ICRA (2022, 2023, 2024), R:SS (2023, 2024, 2025), NeurIPS (2024, 2025), ICML (2024), ICLR (2024, 2025), IEEE Robotics and Automation Letters, IEEE Transaction on Robotics, Neurocomputing, and various ML & Robotics workshops.
- Served as Senior Reviewer for RLC 2024.
- Served as Program Committee member for AAAI (2022, 2023, 2024), ICRA @ RAP4Robots (2023), RSS @ EgoAct (2025)

Invited Talks

Rice University	Texas, US	Tensor Search Methods for Vectorizing Motion Planning	2025
HUST	Hanoi, Vietnam	Tensor Search Methods for Vectorizing Motion Planning	2025
Universität Mannheim	Mannheim, Germany	Accelerating Motion Planning via Optimal Transport	2023
Sony AI	Tokyo, Japan	Learning Implicit Priors for Motion Optimization	2022
Stanford University	Online Talk	Learning forceful manipulation skills from multi-modal human demonstrations	2021

Teaching

- Reinforcement Learning (SS 2022). Designed lectures & exercises on MDP, Dynamic programming methods, Deep actor-critic methods.
- Statistical Machine Learning (SS 2023, WS 2023/2024, SS 2024, WS 2024/2025). Designed lectures & exercises on Probabilistic Density Estimation, Regression, Gaussian Process, Expectation-Maximization, and Variational Inference.
- Probabilistic Methods for CS (WS 2024/2025). Designed lectures & exercises on Monte Carlo Algorithms.
- Robot Learning Integrated Project (WS 2024/2025). Organized matching between students and supervisors for robot learning projects and hosted report sessions.

Academic Supervision

SEMINAR	L. Liu, Y. Ouyang, J. Shi	Benchmark Neural Lyapunov Control Algorithms on Pendulum	(with J. Huang)
SEMINAR	C. Jin, L. Xiang, P. Yan	Hybrid Motion-Force Optimization	(with J. Huang)
MSC THESIS	D. Nandha	Leveraging Large Language Models For Autonomous Task Planning	(with A. Younes, G. Chalvatzaki)
MSC THESIS	M. Baierl	Score-based Planning Networks for Robot Motion Planning	(with J. Carvalho, J. Urain)
MSC THESIS	A. Gece	Leveraging Structured-Graph Correspondence in Imitation Learning	(with K. Hansel)
MSC THESIS	M. Zoeller	Enhancing Smoothness in Policy Blending with Gaussian Processes	(with K. Hansel)
MSC THESIS	C. Freitas	Graph Correspondence Diffusion For Imitation Learning	(with A. Younes, G. Chalvatzaki)
MSC THESIS	D. Andric	Learning Symplectic Manifold Of Dynamical Systems	(with G. Chalvatzaki)
MSC THESIS	N. Nonnengiesser	Graph Neural Network For Robotics Control	(with G. Chalvatzaki)
MSC THESIS	Q. Sun	SE(3) Trajectory Diffusion For Robotics Manipulation	(with J. Carvalho)
MSC THESIS	Sebastian Zach	Replanning via Stein Variational Inference on Graph Manifold	(with K. Hansel)
MSC THESIS	M. Dierking	Model Tensor Planning	(with J. Carvalho)

References available on request

List of Acronyms

BPS	Batch Polytope Search
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
CPU	Central Processing Unit
DoF	degrees of freedom
GP	Gaussian Process
GPMP	Gaussian Process Motion Planning
GPU	Graphics Processing Unit
KL	Kullback–Leibler
KL divergence	Kullback–Leibler Divergence
MAP	Maximum-a-Posteriori
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
MPOT	Motion Planning via Optimal Transport
OT	Optimal Transport
PoE	Product of Experts
RL	Reinforcement Learning
SDF	Signed Distance Field
SGPMP	Stochastic Gaussian Process Motion Planning
SIMD	Single-Instruction, Multiple-Data
STOMP	Stochastic Trajectory Optimization for Motion Planning
TPU	Tensor Processing Unit
VI	Value Iteration

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. "In-datacenter performance analysis of a tensor processing unit". In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017, pp. 1–12.

[2] William J Dally, Stephen W Keckler, and David B Kirk. "Evolution of the graphics processing unit (GPU)". In: *IEEE Micro* 41.6 (2021), pp. 42–51.

[3] Michael J Flynn. "Some computer organizations and their effectiveness". In: *IEEE transactions on computers* 100.9 (1972), pp. 948–960.

[4] Dave Steinkraus, Ian Buck, and Patrice Y Simard. "Using GPUs for machine learning algorithms". In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE. 2005, pp. 1115–1120.

[5] Rajat Raina, Anand Madhavan, Andrew Y Ng, et al. "Large-scale deep unsupervised learning using graphics processors." In: *Icml*. Vol. 9. 2009, pp. 873–880.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold". In: *nature* 596.7873 (2021), pp. 583–589.

[9] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[10] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[11] Firas Al-Hafez, Guoping Zhao, Jan Peters, and Davide Tateo. "LocoMuJoCo: A Comprehensive Imitation Learning Benchmark for Locomotion". In: *6th Robot Learning Workshop, NeurIPS*. 2023.

[12] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. “Gen2sim: Scaling up robot learning in simulation with generative models”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6672–6679.

[13] Genesis Authors. *Genesis: A Universal and Generative Physics Engine for Robotics and Beyond*. 2024.

[14] Michal Januszewski and Marcin Kostur. “Accelerating numerical solution of stochastic differential equations with CUDA”. In: *Computer Physics Communications* 181.1 (2010), pp. 183–188.

[15] Samuel Schoenholz and Ekin Dogus Cubuk. “Jax md: a framework for differentiable physics”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11428–11441.

[16] Mohammadmehdi Ataei and Hesam Salehipour. “XLB: A differentiable massively parallel lattice Boltzmann library in Python”. In: *Computer Physics Communications* 300 (2024), p. 109187.

[17] Miles Macklin. *Warp: A High-performance Python Framework for GPU Simulation and Graphics*. `https://github.com/nvidia/warp`. NVIDIA GPU Technology Conference (GTC). 2022.

[18] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[19] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. “Training compute-optimal large language models”. In: *arXiv preprint arXiv:2203.15556* (2022).

[20] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Koepf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: `1912.01703 [cs.LG]`.

[22] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. “A comprehensive survey of neural architecture search: Challenges and solutions”. In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–34.

[23] Yixin Liu, Kai Zhang, Yuan Li, Zhiling Yan, Chujie Gao, Ruoxi Chen, Zhengqing Yuan, Yue Huang, Hanchi Sun, Jianfeng Gao, et al. "Sora: A review on background, technology, limitations, and opportunities of large vision models". In: *arXiv preprint arXiv:2402.17177* (2024).

[24] Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. "A review on edge large language models: Design, execution, and applications". In: *ACM Computing Surveys* 57.8 (2025), pp. 1–35.

[25] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. "Brax–a differentiable physics engine for large scale rigid body simulation". In: *arXiv preprint arXiv:2106.13281* (2021).

[26] Vince Kurtz. *Hydrax: Sampling-based model predictive control on GPU with JAX and MuJoCo MJX*. https://github.com/vincekurtz/hydrax. 2024.

[27] Chung Min Kim, Brent Yi, Hongsuk Choi, Yi Ma, Ken Goldberg, and Angjoo Kanazawa. "PyRoki: A Modular Toolkit for Robot Kinematic Optimization". In: *arXiv preprint arXiv:2505.03728* (2025).

[28] John T Betts. "Survey of numerical methods for trajectory optimization". In: *Journal of guidance, control, and dynamics* 21.2 (1998), pp. 193–207.

[29] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* (1996).

[30] James J Kuffner and Steven M LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.

[31] David Q Mayne. "Model predictive control: Recent developments and future promise". In: *Automatica* 50.12 (2014), pp. 2967–2986.

[32] Peter E. Hart, Nils J. Nilsson, and Bertram 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.

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

[34] Joshua Bialkowski, Sertac Karaman, and Emilio Frazzoli. "Massively parallelizing the RRT and the RRT". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3513–3518.

[35] Jia Pan and Dinesh Manocha. "GPU-based parallel collision detection for fast motion planning". In: *The International Journal of Robotics Research* 31.2 (2012), pp. 187–200.

[36] R Connor Lawson, Linda Wills, and Panagiotis Tsiotras. "Gpu parallelization of policy iteration rrt". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 4369–4374.

[37] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. “Motions in microseconds via vectorized sampling-based planning”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 8749–8756.

[38] An T Le, Kay Hansel, João Carvalho, Joe Watson, Julen Urain, Armin Biess, Georgia Chalvatzaki, and Jan Peters. “Global Tensor Motion Planning”. In: *IEEE Robotics and Automation Letters (IEEE RA-L)* (2025).

[39] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C Yip. “Motion planning networks: Bridging the gap between learning-based and classical motion planners”. In: *IEEE Transactions on Robotics* 37.1 (2020), pp. 48–66.

[40] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. “Motion policy networks”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 967–977.

[41] An T Le, Georgia Chalvatzaki, Armin Biess, and Jan R Peters. “Accelerating motion planning via optimal transport”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 78453–78482.

[42] An T. Le, Khai Nguyen, Minh Nhat Vu, João Carvalho, and Jan Peters. “Model Tensor Planning”. In: *Transactions on Machine Learning Research (TMLR)* (2025).

[43] Corrado Pezzato, Chadi Salmi, Elia Trevisan, Max Spahn, Javier Alonso-Mora, and Carlos Hernández Corbato. “Sampling-based model predictive control leveraging parallelizable physics simulations”. In: *IEEE Robotics and Automation Letters* (2025).

[44] K. Shen, J.N. Crossley, A.W.C. Lun, and H. Liu. *The Nine Chapters on the Mathematical Art: Companion and Commentary*. Oxford University Press, 1999. ISBN: 9780198539360.

[45] Jean Christianidis and Jeffrey Oaks. “Practicing algebra in late antiquity: The problem-solving of Diophantus of Alexandria”. In: *Historia Mathematica* 40.2 (2013), pp. 127–163. ISSN: 0315-0860.

[46] Claude Brezinski, Gérard Meurant, and Michela Redivo-Zaglia. *A Journey through the History of Numerical Linear Algebra*. SIAM, 2022.

[47] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[48] Roy Frostig, Matthew James Johnson, and Chris Leary. “Compiling machine learning programs via high-level tracing”. In: *SysML Conference*. 2018.

[49] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. “The open motion planning library”. In: *IEEE Robotics & Automation Magazine* (2012).

[50] Hiroshi Akima. “A method of bivariate interpolation and smooth surface fitting based on local procedures”. In: *Communications of the ACM* 17.1 (1974), pp. 18–20.

[51] Ioan A Şucan and Lydia E Kavraki. “Kinodynamic motion planning by interior-exterior cell exploration”. In: *WAFR*. Springer. 2009.

[52] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. “Curobo: Parallelized collision-free robot motion generation”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 8112–8119.

[53] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. “Model predictive path integral control: From theory to parallel computation”. In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357.

[54] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. “Predictive sampling: Real-time behaviour synthesis with mujoco”. In: *arXiv preprint arXiv:2212.00541* (2022).

[55] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864* (2017).

[56] Yanbo Zhang, Benedikt Hartl, Hananel Hazan, and Michael Levin. “Diffusion Models are Evolutionary Algorithms”. In: *arXiv preprint arXiv:2410.02543* (2024).

[57] Jean-Paul Laumond et al. *Robot motion planning and control*. Vol. 229. Springer, 1998.

[58] Laurene Claussmann, Marc Revilloud, Dominique Gruyer, and Sébastien Glaser. “A review of motion planning for highway autonomous driving”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2019), pp. 1826–1848.

[59] Dario Izzo and Lorenzo Pettazzi. “Autonomous and distributed motion planning for satellite swarm”. In: *Journal of Guidance, Control, and Dynamics* 30.2 (2007), pp. 449–459.

[60] Ibrahim Al-Bluwi, Thierry Siméon, and Juan Cortés. “Motion planning algorithms for molecular simulations: A survey”. In: *Computer Science Review* 6.4 (2012), pp. 125–143.

[61] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 489–494.

[62] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. "Continuous-time Gaussian process motion planning via probabilistic inference". In: *IJRR* (2018).

[63] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. "STOMP: Stochastic trajectory optimization for motion planning". In: *IEEE ICRA*. 2011.

[64] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[65] Chonhyon Park, Jia Pan, and Dinesh Manocha. "High-dof robots in dynamic environments using incremental trajectory optimization". In: *International Journal of Humanoid Robotics* 11.02 (2014), p. 1441001.

[66] Anca D Dragan, Nathan D Ratliff, and Siddhartha S Srinivasa. "Manipulation planning with goal sets using constrained trajectory optimization". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4582–4588.

[67] Julen Urain, An T Le, Alexander Lambert, Georgia Chalvatzaki, Byron Boots, and Jan Peters. "Learning implicit priors for motion optimization". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 7672–7679.

[68] Richard Sinkhorn and Paul Knopp. "Concerning nonnegative matrices and doubly stochastic matrices". In: *Pacific Journal of Mathematics* 21.2 (1967), pp. 343–348.

[69] Marc Toussaint. "Robot trajectory optimization using approximate inference". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 1049–1056.

[70] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[71] Gabriel Peyré, Marco Cuturi, et al. "Computational optimal transport: With applications to data science". In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607.

[72] Alessio Figalli and Federico Glaudo. *An invitation to optimal transport, Wasserstein distances, and gradient flows*. EMS Press, 2021.

[73] Marco Cuturi. "Sinkhorn distances: Lightspeed computation of optimal transport". In: *Advances in neural information processing systems* 26 (2013).

[74] Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. "Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration". In: *Advances in neural information processing systems* 30 (2017).

[75] Marc Toussaint. "Newton methods for k-order Markov constrained motion problems". In: *arXiv preprint arXiv:1407.0414* (2014).

[76] Takayuki Osa. “Multimodal trajectory optimization for motion planning”. In: *The International Journal of Robotics Research* 39.8 (2020), pp. 983–1001.

[77] Alexander Lambert, Adam Fishman, Dieter Fox, Byron Boots, and Fabio Ramos. “Stein variational model predictive control”. In: *arXiv preprint arXiv:2011.07641* (2020).

[78] Joao Carvalho, An T Le, Mark Baierl, Dorothea Koert, and Jan Peters. “Motion planning diffusion: Learning and planning of robot motions with diffusion models”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 1916–1923.

[79] Marco Cuturi and Arnaud Doucet. “Fast computation of Wasserstein barycenters”. In: *International conference on machine learning*. PMLR. 2014, pp. 685–693.

[80] Robert Hooke and Terry A Jeeves. ““Direct Search”Solution of Numerical and Statistical Problems”. In: *Journal of the ACM (JACM)* 8.2 (1961), pp. 212–229.

[81] Jeffrey Larson, Matt Menickelly, and Stefan M Wild. “Derivative-free optimization methods”. In: *Acta Numerica* 28 (2019), pp. 287–404.

[82] Rommel G Regis. “On the properties of positive spanning sets and positive bases”. In: *Optimization and Engineering* 17.1 (2016), pp. 229–262.

[83] Harold Scott Macdonald Coxeter. *Regular polytopes*. Courier Corporation, 1973.

[84] Volker Kaibel and Marc E Pfetsch. “Some algorithmic problems in polytope theory”. In: *Algebra, geometry and software systems*. Springer. 2003, pp. 23–47.

[85] James R Munkres. *Elements of algebraic topology*. CRC press, 2018.

[86] Guillaume Garrigos and Robert M Gower. “Handbook of convergence theorems for (stochastic) gradient methods”. In: *arXiv preprint arXiv:2301.11235* (2023).

[87] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. “Information theoretic MPC for model-based reinforcement learning”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1714–1721.

[88] Martin J Wainwright, Michael I Jordan, et al. “Graphical models, exponential families, and variational inference”. In: *Foundations and Trends® in Machine Learning* 1.1–2 (2008), pp. 1–305.

[89] Lenaic Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. “Scaling algorithms for unbalanced optimal transport problems”. In: *Mathematics of Computation* 87.314 (2018), pp. 2563–2609.

[90] Bernhard Schmitzer. “Stabilized sparse scaling algorithms for entropy regularized transport problems”. In: *SIAM Journal on Scientific Computing* 41.3 (2019), A1443–A1481.

[91] Jan Peters, Katharina Mulling, and Yasemin Altun. “Relative entropy policy search”. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[92] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[93] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. “Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2997–3004.

[94] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. “Chomp: Covariant hamiltonian optimization for motion planning”. In: *The International journal of robotics research* 32.9-10 (2013), pp. 1164–1193.

[95] Keliang He, Elizabeth Martin, and Matt Zucker. “Multigrid CHOMP with local smoothing”. In: *IEEE-RAS 13th Humanoids*. 2013.

[96] Arunkumar Byravan, Byron Boots, Siddhartha S Srinivasa, and Dieter Fox. “Space-time functional gradient optimization for motion planning”. In: *IEEE ICRA*. 2014.

[97] Zita Marinho, Anca Dragan, Arun Byravan, Byron Boots, Siddhartha Srinivasa, and Geoffrey Gordon. “Functional gradient motion planning in reproducing kernel hilbert spaces”. In: *arXiv preprint arXiv:1601.03648* (2016).

[98] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron 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.

[99] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. “Motion planning with sequential convex optimization and convex collision checking”. In: *IJRR* (2014).

[100] Charles R Hargraves and Stephen W Paris. “Direct trajectory optimization using nonlinear programming and collocation”. In: *Journal of guidance, control, and dynamics* 10.4 (1987), pp. 338–342.

[101] Daisuke Inoue, Yuji Ito, and Hiroaki Yoshida. “Optimal transport-based coverage control for swarm robot systems: Generalization of the voronoi tessellation-based method”. In: *2021 American Control Conference (ACC)*. IEEE. 2021, pp. 3032–3037.

[102] Vishaal Krishnan and Sonia Martínez. “Distributed optimal transport for the deployment of swarms”. In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 4583–4588.

[103] Saptarshi Bandyopadhyay, Soon-Jo Chung, and Fred Y Hadaegh. “Probabilistic swarm guidance using optimal transport”. In: *2014 IEEE Conference on Control Applications (CCA)*. IEEE. 2014, pp. 498–505.

[104] Rabiul Hasan Kabir and Kooktae Lee. “Efficient, Decentralized, and Collaborative Multi-Robot Exploration using Optimal Transport Theory”. In: *2021 American Control Conference (ACC)*. 2021, pp. 4203–4208.

[105] Christina Frederick, Magnus Egerstedt, and Haomin Zhou. “Collective motion planning for a group of robots using intermittent diffusion”. In: *Journal of Scientific Computing* 90.1 (2022), pp. 1–20.

[106] Rabiul Hasan Kabir and Kooktae Lee. “Receding-horizon ergodic exploration planning using optimal transport theory”. In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 1447–1452.

[107] Pascal Klink, Haoyi Yang, Carlo D’Eramo, Jan Peters, and Joni Pajarinen. “Curriculum reinforcement learning via constrained optimal transport”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 11341–11358.

[108] Yongxin Chen, Tryphon T Georgiou, and Michele Pavon. “Optimal transport in systems and control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.1 (2021).

[109] An T. Le, Kay Hansel, Jan Peters, and Georgia Chalvatzaki. “Hierarchical Policy Blending As Optimal Transport”. In: *Learning for Dynamics and Control Conference*. PMLR. 2023, pp. 797–812.

[110] Kay Hansel, Julen Urain, Jan Peters, and Georgia Chalvatzaki. “Hierarchical policy blending as inference for reactive robot control”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10181–10188.

[111] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012.

[112] Jiankun Wang, Tianyi Zhang, Nachuan Ma, Zhaoting Li, Han Ma, Fei Meng, and Max Q-H Meng. “A survey of learning-based robot motion planning”. In: *IET Cyber-Systems and Robotics* 3.4 (2021), pp. 302–314.

[113] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. “Goal Conditioned Imitation Learning using Score-based Diffusion Policies”. In: *R:SS*. 2023.

[114] Janelle Blankenburg, Richard Kelley, David Feil-Seifer, Rui Wu, Lee Barford, and Frederick C Harris. “Towards GPU-Accelerated PRM for Autonomous Navigation”. In: *ITNG*. Springer. 2020.

[115] Sam Ade Jacobs, Kasra Manavi, Juan Burgos, Jory Denny, Shawna Thomas, and Nancy M Amato. “A scalable method for parallelizing sampling-based motion planning algorithms”. In: *IEEE ICRA*. 2012.

[116] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[117] Nancy M Amato and Lucia K Dale. “Probabilistic roadmap methods are embarrassingly parallel”. In: *IEEE ICRA*. 1999.

[118] Erion Plaku et al. “Sampling-based roadmap of trees for parallel motion planning”. In: *IEEE TRO* (2005).

[119] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. “Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search”. In: *IJRR* (2020).

[120] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions”. In: *IJRR* (2015).

[121] Marlin P Strub and Jonathan D Gammell. “Adaptively Informed Trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics”. In: *IEEE ICRA*. 2020.

[122] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q-H Meng. “Neural RRT*: Learning-based optimal path planning”. In: *IEEE Transactions on Automation Science and Engineering* (2020).

[123] Chenning Yu and Sicun Gao. “Reducing collision checking for sampling-based motion planning using graph neural networks”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4274–4289.

[124] Brian Ichter, James Harrison, and Marco Pavone. “Learning sampling distributions for robot motion planning”. In: *IEEE ICRA*. 2018.

[125] Andreas Orthey, Constantinos Chamzas, and Lydia E Kavraki. “Sampling-based motion planning: A comparative review”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 7 (2023).

[126] Vivek K Adajania, Aditya Sharma, Anish Gupta, Houman Masnavi, K Madhava Krishna, and Arun K Singh. “Multi-modal model predictive control through batch non-holonomic trajectory optimization: Application to highway driving”. In: *IEEE RA-L* (2022).

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

[128] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*. Vol. 4. Athena scientific, 2012.

[129] Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[130] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016.

[131] Andrew Howard, Nicholas Roy, Cyrill Stachniss, Giorgio Grisetti, Dirk Haehnel, Henrik Andreasson, Per Larsson, Tom Duckett, and Patrick Beeson. *The Robotics Data Set Repository (Radish)*. 2003.

[132] Constantinos Chamzas, Carlos Quintero-Pena, Zachary Kingston, Andreas Orthey, Daniel Rakita, Michael Gleicher, Marc Toussaint, and Lydia E Kavraki. "MotionBenchMaker: A tool to generate and benchmark motion planning datasets". In: *IEEE Robotics and Automation Letters* 7.2 (2021), pp. 882–889.

[133] Matthias Lorenzen, Fabrizio Dabbene, Roberto Tempo, and Frank Allgöwer. "Stochastic MPC with offline uncertainty sampling". In: *Automatica* 81 (2017), pp. 176–183.

[134] Juan Alvarez-Padilla, John Z Zhang, Sofia Kwok, John M Dolan, and Zachary Manchester. "Real-Time Whole-Body Control of Legged Robots with Model-Predictive Path Integral Control". In: *arXiv preprint arXiv:2409.10469* (2024).

[135] Albert H Li, Preston Culbertson, Vince Kurtz, and Aaron D Ames. "Drop: Dexterous reorientation via online planning". In: *arXiv preprint arXiv:2409.14562* (2024).

[136] Haoru Xue, Chaoyi Pan, Zeji Yi, Guannan Qu, and Guanya Shi. "Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing". In: *arXiv preprint arXiv:2409.15610* (2024).

[137] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. "A tutorial on the cross-entropy method". In: *Annals of operations research* 134 (2005), pp. 19–67.

[138] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. "Sample-efficient cross-entropy method for real-time planning". In: *Conference on Robot Learning*. PMLR. 2021, pp. 1049–1065.

[139] Zichen Zhang, Jun Jin, Martin Jagersand, Jun Luo, and Dale Schuurmans. "A simple decentralized cross-entropy method". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36495–36506.

[140] Nikolaus Hansen. "The CMA evolution strategy: A tutorial". In: *arXiv preprint arXiv:1604.00772* (2016).

[141] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. "Natural evolution strategies". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 949–980.

[142] Julius Jankowski, Lara Brudermüller, Nick Hawes, and Sylvain Calinon. "Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10125–10131.

[143] J. Carvalho, A. Le, P. Kicki, D. Koert, and J. Peters. *Motion Planning Diffusion: Learning and Adapting Robot Motion Planning with Diffusion Models*. 2024. arXiv: 2412.19948 [cs.RO].

[144] Carl de Boor. “Package for Calculating with B-Splines”. In: *SIAM Journal on Numerical Analysis* 14 (Oct. 1973), p. 57.

[145] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[146] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. “Isaac gym: High performance gpu-based physics simulation for robot learning”. In: *arXiv preprint arXiv:2108.10470* (2021).

[147] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[148] Robert Tjarko Lange. “evosax: Jax-based evolution strategies”. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 2023, pp. 659–662.

[149] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. “Diffusion policy: Visuomotor policy learning via action diffusion”. In: *The International Journal of Robotics Research* (2023), p. 02783649241273668.

[150] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[151] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. “Gymnasium: A Standard Interface for Reinforcement Learning Environments”. In: *arXiv preprint arXiv:2407.17032* (2024).

[152] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: `1707.06347 [cs.LG]`.

[153] Zeji Yi, Chaoyi Pan, Guanqi He, Guannan Qu, and Guanya Shi. “CoVO-MPC: Theoretical analysis of sampling-based MPC and optimal covariance design”. In: *6th Annual Learning for Dynamics & Control Conference*. PMLR. 2024, pp. 1122–1135.

[154] Chih H Huang, Pranav Jadhav, Brian Plancher, and Zachary Kingston. “pRRTC: GPU-Parallel RRT-Connect for Fast, Consistent, and Low-Cost Motion Planning”. In: *arXiv preprint arXiv:2503.06757* (2025).

[155] Nicolas Perrault, Qi Heng Ho, and Morteza Lahijanian. “Kino-PAX: Highly Parallel Kinodynamic Sampling-based Planner”. In: *IEEE Robotics and Automation Letters* (2025).

[156] Huang Huang, Balakumar Sundaralingam, Arsalan Mousavian, Adithyavairavan Murali, Ken Goldberg, and Dieter Fox. "Diffusionseeder: Seeding motion optimization with diffusion for rapid motion planning". In: *arXiv preprint arXiv:2410.16727* (2024).

[157] Khang Nguyen, An T Le, Tien Pham, Manfred Huber, Jan Peters, and Minh Nhat Vu. "FlowMP: Learning Motion Fields for Robot Planning with Conditional Flow Matching". In: *Submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2025).

[158] Darrick Lee, Cameron Lerch, Fabio Ramos, and Ian Abraham. "Stein Variational Ergodic Search". In: *arXiv preprint arXiv:2406.11767* (2024).

[159] Max Muchen Sun, Allison Pinosky, and Todd Murphey. "Flow Matching Ergodic Coverage". In: *arXiv preprint arXiv:2504.17872* (2025).

[160] Marc Toussaint. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning." In: *IJCAI*. 2015, pp. 1930–1936.

[161] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1470–1477.

[162] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. "Openvla: An open-source vision-language-action model". In: *arXiv preprint arXiv:2406.09246* (2024).

[163] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. "Differentiable convex optimization layers". In: *Advances in neural information processing systems* 32 (2019).

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

[165] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. "Differentiable mpc for end-to-end planning and control". In: *Advances in neural information processing systems* 31 (2018).

[166] Jakub Konečnỳ and Peter Richtárik. "Simple complexity analysis of simplified direct search". In: *arXiv preprint arXiv:1410.0390* (2014).

[167] Garrett Birkhoff. "Tres observaciones sobre el algebra lineal". In: *Univ. Nac. Tucuman, Ser. A* 5 (1946), pp. 147–154.

[168] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*. Vol. 6. Athena scientific Belmont, MA, 1997.

[169] Luís Nunes Vicente. "Worst case complexity of direct search". In: *EURO Journal on Computational Optimization* 1.1-2 (2013), pp. 143–153.

[170] Carl Edward Rasmussen. “Gaussian processes in machine learning”. In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[171] Tim D Barfoot, Chi Hay Tong, and Simo Särkkä. “Batch Continuous-Time Trajectory Estimation as Exactly Sparse Gaussian Process Regression.” In: *Robotics: Science and Systems*. Vol. 10. Citeseer. 2014, pp. 1–10.

[172] Simo Särkkä, Arno Solin, and Jouni Hartikainen. “Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering”. In: *IEEE Signal Processing Magazine* 30.4 (2013), pp. 51–61.

[173] Qiang Liu and Dilin Wang. “Stein variational gradient descent: A general purpose bayesian inference algorithm”. In: *Advances in neural information processing systems* 29 (2016).

[174] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[175] Richard Sinkhorn. “Diagonal equivalence to matrices with prescribed row and column sums”. In: *The American Mathematical Monthly* 74.4 (1967), pp. 402–405.

[176] Egon Schulte. “Symmetry of Polytopes and Polyhe Dra”. In: *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017, pp. 477–503.

[177] Andrew J Hanson. “4 Rotations for N-Dimensional Graphics”. In: *Graphics Gems V*. Elsevier, 1995, pp. 55–64.

[178] John Frank Adams. *Lectures on Lie groups*. University of Chicago Press, 1982.

[179] Gilbert W Stewart. “The efficient generation of random orthogonal matrices with an application to condition estimators”. In: *SIAM Journal on Numerical Analysis* 17.3 (1980), pp. 403–409.

[180] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. “A micro Lie theory for state estimation in robotics”. In: *arXiv preprint arXiv:1812.01537* (2018).

[181] George Marsaglia. “Choosing a point from the surface of a sphere”. In: *The Annals of Mathematical Statistics* 43.2 (1972), pp. 645–646.

[182] S. Surjanovic and D. Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. Retrieved October 26, 2023, from `http://www.sfu.ca/~ssurjano`. 2013.

[183] Hicham Janati, Marco Cuturi, and Alexandre Gramfort. “Debiased sinkhorn barycenters”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4692–4701.

[184] Bogdan Vlahov, Jason Gibson, Manan Gandhi, and Evangelos A Theodorou. “Mppi-generic: A cuda library for stochastic optimization”. In: *arXiv preprint arXiv:2409.07563* (2024).

[185] Joe Watson and Jan Peters. “Inferring smooth control: Monte carlo posterior policy iteration with gaussian processes”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 67–79.

[186] An T Le, Zachary Kingston, Wil Thomason, Joe Watson, and Jan Peters. “Anytime Global Tensor Motion Planning”. In: *In preparation for IEEE Robotics and Automation Letters (RA-L)* (2025).

[187] An T Le, Georgia Chalvatzaki, Armin Biess, and Jan R Peters. “Accelerating motion planning via optimal transport”. In: *In preparation for IEEE Transactions on Robotics (TRO)* (2025).

[188] Joe Watson, Chen Song, Oliver Weeger, Theo Gruner, An T Le, Kay Hansel, Ahmed Hendawy, Oleg Arenz, Will Trojak, Miles Cranmer, et al. “Machine Learning with Physics Knowledge for Prediction: A Survey”. In: *Transactions on Machine Learning Research (TMLR)* (2025).

[189] Joao Carvalho, An T Le, Philipp Jahr, Qiao Sun, Julen Urain, Dorothea Koert, and Jan Peters. “Grasp Diffusion Network: Learning Grasp Generators from Partial Point Clouds with Diffusion Models in SO (3) xR3”. In: *Submitted to IEEE Robotics and Automation Letters (IEEE RA-L)* (2024).

[190] Joao Carvalho, A Le, Piotr Kicki, Dorothea Koert, and Jan Peters. “Motion planning diffusion: Learning and adapting robot motion planning with diffusion models”. In: *IEEE Transactions on Robotics (TRO)* (2025).

[191] Georgia Chalvatzaki, Ali Younes, Daljeet Nandha, An Thai Le, Leonardo FR Ribeiro, and Iryna Gurevych. “Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning”. In: *Frontiers in Robotics and AI* 10 (2023).

[192] Khang Nguyen, Khai Nguyen, An T Le, Jan Peters, Manfred Huber, Ngo Anh Vien, and Minh Nhat Vu. “TD-GRPC: Temporal Difference Learning with Group Relative Policy Constraint for Humanoid Locomotion”. In: *Submitted to IEEE-RAS International Conference on Humanoid Robots* (2025).

[193] Kay Pompetzki, An T Le, Theo Gruner, Joe Watson, Georgia Chalvatzaki, and Jan Peters. “Geometrically-Aware Goal Inference: Leveraging Motion Planning as Inference”. In: *German Robotics Conference (GRC)* (2025).

[194] An T Le, Kay Pompetzki, Jan Peters, and Armin Biess. “Kinematics Correspondence As Inexact Graph Matching”. In: *German Robotics Conference (GRC)* (2025).

[195] Viet The Nguyen, Duy Anh Pham, An Thai Le, Jans Peter, and Gunther Gust. “Persistent Homology-induced Graph Ensembles for Time Series Regressions”. In: *Submitted to NeurIPS* (2025).

[196] Jiayun Li, Kay Pompetzki, An Thai Le, Haolei Tong, Jan Peters, and Georgia Chalvatzaki. “Constrained Gaussian Process Motion Planning via Stein Variational Newton Inference”. In: *arXiv preprint arXiv:2504.04936* (2025).

[197] An T Le*, Duy MH Nguyen*, Trung Q Nguyen, Nghiem T Diep, Tai Nguyen, Duy Duong-Tran, Jan Peters, Li Shen, Mathias Niepert, and Daniel Sonntag. "Dude: Dual distribution-aware context prompt learning for large vision-language model". In: *Asian Conference on Machine Learning (ACML)* (2024).

[198] Duy MH Nguyen, Nina Lukashina, Tai Nguyen, An T Le, TrungTin Nguyen, Nhat Ho, Jan Peters, Daniel Sonntag, Viktor Zaverkin, and Mathias Niepert. "Structure-aware e (3)-invariant molecular conformer aggregation networks". In: *International Conference on Machine Learning (ICML)* (2024).

[199] An T Le, Meng Guo, Niels van Duijkeren, Leonel Rozo, Robert Krug, Andras G Kupcsik, and Mathias Bürger. "Learning forceful manipulation skills from multi-modal human demonstrations". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 7770–7777.

[200] An T Le, Philipp Kratzer, Simon Hagenmayer, Marc Toussaint, and Jim Mainprice. "Hierarchical Human-Motion Prediction and Logic-Geometric Programming for Minimal Interference Human-Robot Tasks". In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE. 2021, pp. 7–14.