Contact-rich Manipulation with Tensor Networks : Structured Approximation for Efficient Planning, Control and Learning

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the registrar's office.

Thèse n. 1234 2026
présentée le 30 juillet 2026
à la Faculté des sciences de base
laboratoire SuperScience
programme doctoral en SuperScience
École polytechnique fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Teng XUE

acceptée sur proposition du jury :

Prof. C. N. Jones, président du jury
Prof. A. R. Zamir, Dr. S. Calinon, directeur de thèse
Prof. S. Coros, rapporteur
Prof. M. T. Mason, rapporteur
Prof. L. P. Kaelbling, rapporteur

Lausanne, EPFL, 2026

The purpose of computation is insight,
not numbers.
— Richard Hamming

To my father, who taught me to explore,
and my mother, who taught me to love.

Acknowledgements

The completion of this PhD thesis has been a long and challenging journey, shaped by the guidance, support, and encouragement of many people around me.
First and foremost, I am deeply grateful to my supervisors, Dr. Sylvain Calinon and Prof. Amir Zamir, for giving me the opportunity to pursue a doctorate jointly with the Idiap Research Institute and EPFL. I am especially grateful to Sylvain, who has not only taught me how to conduct rigorous and meaningful research but has also continuously encouraged and supported me throughout this journey. I am particularly thankful for his guidance, insightful feedback, and trust. His dedication, diligence, and passion for academic research have profoundly influenced me and will continue to inspire me in my future career.
I would like to express my sincere gratitude to the members of the jury committee, Prof. Colin Jones, Prof. Stelian Coros, Prof. Matthew T. Mason, and Prof. Leslie P. Kaelbling, for reviewing this thesis and providing insightful comments and valuable suggestions for improvement. Their thoughtful feedback has greatly contributed to enhancing the quality and clarity of this work.
I owe heartfelt thanks to all members of the RLI group, including Jeremy, Guillaume, João, Marco, Mattia, Boyang, Tobias, James, Martin, Jiacheng, Yongkang, and many others. This thesis would not have been possible without the many insightful discussions and feedback exchanged over countless tea breaks, lunches, and group meetings. I am particularly grateful to Teguh and Hakan for their mentorship during my first year, which helped me navigate the early stages of my doctoral studies. I would like to thank Suhan and Yan, with whom I shared many memorable experiences, from co-authoring papers to engaging in thoughtful discussions after work. I am also thankful to Julius, Ante, Cem, and Yiming for many inspiring discussions and excellent research insights.
Special thanks go to Amir, who has been by my side since the very beginning of my PhD, whether in the office or over lunch. We have shared countless moments together, from collaborating on papers to enjoying our daily coffee breaks and meals. I am deeply grateful for the many discussions, laughs, and memories we created along the way, which have made this experience all the more meaningful. My gratitude also extends to many other colleagues at Idiap and EPFL, including Florian, Haolin, Mingchi, Hengfei,

Fenglong, Yang, and many others whom I cannot name individually here. I am thankful for the lunches, hikes, social activities, and many casual moments we shared, which brought warmth and friendship to my doctoral journey.
I am also grateful to my colleagues at Meta Reality Labs, especially my mentors and peers, Dr. Amir Memar, Dr. Nicholas Colonnese, and Dr. Jiayi Shen, for giving me the opportunity to work on an exciting project and for the many insightful discussions throughout the internship.
I would also like to express my sincere gratitude to my supervisor and labmates from my master's studies. In particular, I am deeply thankful to Prof. Weiming Wang, whose constant encouragement to explore my interests and unwavering support inspired me to pursue my dreams, as well as to my labmates for the many unforgettable memories we shared. My heartfelt thanks also go to Prof. Marco Hutter, Dr. David Hoeller, and Dr. Martin Wermlinger for giving me the opportunity to visit ETHZ and for introducing me to reinforcement learning for the first time. In addition, I am truly grateful to Dr. Shenli Yuan and Prof. Kenneth Salisbury for offering me the opportunity to work on an exciting project at the Stanford AI Institute.
Finally, my deepest gratitude goes to my family for their unwavering support throughout this journey. I am deeply grateful to my parents, who have always encouraged me to pursue my dreams while respecting my choices. They created a loving and nurturing environment in which I could grow up carefree. My father taught me to stay curious about the world and to keep exploring, while my mother has always stood by my side, offering constant support and unconditional love. Their guidance and trust have shaped the way I approach both life and challenges, giving me the confidence to move forward even in times of uncertainty. I am also deeply grateful to my uncle, who filled my childhood with countless thoughtful and unique little items, each quietly sparking my curiosity and inspiring an early desire to explore the world beyond what I knew. My sincere thanks also go to my aunt, who taught me to be grounded, diligent, and persistent, and has always supported me in pursuing my dreams with determination. I also want to thank my parents-in-law for their kindness, understanding, and generous support. I am deeply grateful for the love and care they have given to our little family. Most importantly, I would like to thank my wife for always standing by my side, supporting and encouraging me through every challenge and difficult moment. Together, we have shared unforgettable years in Switzerland and welcomed our son into our lives, a joy beyond words. Their companionship has brought warmth, strength, and meaning to my PhD journey, making these years truly fulfilling and unforgettable.

Lausanne, July 30, 2026 Teng Xue

Abstract

Contact-rich manipulation lies at the core of human dexterity, yet remains a fundamental challenge in robotics. Such tasks can be naturally formulated as decision-making problems comprising two stages: task objective approximation and subsequent optimization. Classical optimal control approximates system dynamics to enable computational tractability, while learning-based methods, including reinforcement learning and imitation learning, have shifted toward approximating objective functions. However, both paradigms face significant challenges in achieving efficient decision-making due to the interplay of nonlinear dynamics, hybrid contact modes, and physical uncertainty inherent in contact-rich interactions.

This dissertation addresses these challenges by proposing a paradigm termed structured approximation. We show that the key to advancing contact-rich manipulation lies in approximating objective functions that are inherently amenable to subsequent optimization. Specifically, tensor networks, originally developed in quantum many-body physics, are leveraged to approximate objective functions in a structured manner, enabling efficient optimization in non-convex landscapes. In parallel, velocity fields are leveraged to geometrically represent action trajectories for imitation learning, enabling fast and reactive control under high-frequency tactile feedback.

Building on this perspective, the dissertation makes three main contributions. First, Tensor Train (TT), a specific class of tensor networks, is introduced as a general computational tool for planning and control under non-smooth contact dynamics, enabling scalable optimization across diverse manipulation tasks. Second, to address the combinatorial complexity arising from hybrid contact modes, TT is combined with symbolic reasoning and optimal control, enabling structured sequential skill planning for multi-stage contact-rich manipulation. Third, in the presence of contact uncertainty, TT is incorporated into robust policy learning, enabling rapid motor adaptation given probabilistic parameter distributions. Beyond TT, a tube-bounded action velocity field is leveraged as a geometric approximation of conventional action chunking, ensuring highly reactive visual–tactile control under contact variations and external disturbances.

The proposed methods are validated across a range of contact-rich manipulation

tasks, including planar pushing with face switching, whole-body bimanual manipulation, multi-stage sequential manipulation, and visual–tactile dexterous manipulation. Collectively, these results demonstrate that structured approximation provides a principled and scalable foundation for efficient optimization, long-horizon planning, robust control, and reactive policy learning in contact-rich robotic systems.

keywords: Contact-rich manipulation, Tensor Networks, Optimal Control, Task and Motion Planning, Robust Policy Learning, Visual-tactile Imitation Learning

Résumé

La manipulation riche en contacts est au cœur de la dextérité humaine, mais demeure un défi fondamental en robotique. Ces tâches peuvent être naturellement envisagées comme des problèmes de prise de décision reposant sur deux composantes étroitement liées : la représentation de la fonction objectif et son optimisation. Le contrôle optimal classique repose sur des fonctions de coût conçues manuellement afin de rendre les calculs tractables, tandis que les approches d'apprentissage, telles que l'apprentissage par renforcement et l'apprentissage par imitation, visent à apprendre ces objectifs à partir de données. Cependant, ces deux paradigmes rencontrent des difficultés majeures pour réaliser une prise de décision efficace, en raison de l'interaction entre dynamiques non linéaires, modes de contact hybrides et incertitudes physiques inhérentes aux interactions avec contact.

Cette thèse aborde ces défis à travers un paradigme d'approximation structurée, fondé sur l'idée que les fonctions objectif doivent être représentées sous des formes intrinsèquement propices à l'optimisation. En particulier, les réseaux de tenseurs — initialement développés en physique quantique des systèmes à nombreux corps — sont exploités pour construire des représentations structurées de fonctions de haute dimension, permettant un calcul et une optimisation efficaces dans des paysages fortement non convexes. De manière complémentaire, un champ de vitesse des actions est introduit comme représentation géométrique des trajectoires d'action pour l'apprentissage par imitation, permettant un contrôle rapide et réactif à partir de retours tactiles à haute fréquence.

S'appuyant sur cette perspective, la thèse apporte trois contributions principales. Premièrement, le format Tensor Train (TT), une classe particulière de réseaux de tenseurs, est introduit comme un cadre computationnel général pour la planification et le contrôle sous dynamiques de contact non linéaires, permettant une optimisation efficace à grande échelle dans divers scénarios de manipulation. Deuxièmement, afin de traiter la complexité combinatoire induite par les modes de contact hybrides, le TT est combiné au raisonnement symbolique et au contrôle optimal, permettant une planification structurée de séquences de compétences pour des tâches de manipulation multi-étapes. Troisièmement, en présence d'incertitudes de contact, le TT est intégré à

l'apprentissage de politiques robustes, permettant une adaptation motrice rapide sous des distributions probabilistes des paramètres. Par ailleurs, la Tube Diffusion Policy est proposée pour l'apprentissage de politiques visuo-tactiles réactives, où un champ de vitesse des actions contraint par un tube constitue une alternative géométrique aux segments d'action conventionnels, permettant un contrôle réactif face aux variations de contact et aux perturbations externes.
Les méthodes proposées sont validées sur un ensemble de tâches de manipulation riche en contacts, incluant la poussée plane avec changement de face, la manipulation bimanuelle à l'échelle du corps entier, la manipulation séquentielle multi-étapes, ainsi que la manipulation dextrère visuo-tactile. Dans l'ensemble, ces résultats montrent que l'approximation structurée des fonctions objectif constitue une base principielle et extensible pour une optimisation efficace, une planification à long horizon, ainsi qu'un apprentissage de politiques robuste et réactif dans les systèmes robotiques à contacts riches, ouvrant ainsi une voie prometteuse pour concilier complexité physique et efficacité computationnelle.

Mots-clés : Manipulation riche en contacts, réseaux de tenseurs, contrôle optimal, planification de tâches et de mouvements, apprentissage de politiques robustes, apprentissage par imitation visuo-tactile

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Manipulation underlies many of the most demanding capabilities expected of robotic systems, from dexterous tool use and in-hand object reorientation to non-prehensile pushing and bimanual assembly. Unlike collision-free motion planning, which treats contact as a constraint to be avoided, manipulation fundamentally relies on contact as a resource for generating and regulating motion (Mason, 2001b). Within this broad class, we focus on tasks that demand sustained and sequential contact interactions, which motivate the notion of contact-rich manipulation.

Definition 1 (Contact-Rich Manipulation)**.** *Contact-rich manipulation refers to a sequential decision-making process that involves strategic contact selection among multiple alternatives.*

This formulation aligns with the definition discussed in Suh (2025), while placing additional emphasis on the sequential nature of contact selection. The key criterion is the richness of contact options: at each step, the robot faces many possible contact configurations, modes, and transition strategies, and this option space expands exponentially when considered over a sequence of decisions. This distinguishes contact-rich manipulation from simple contact-involving actions, such as instantaneous grasping, where contact is brief or largely predetermined.

A simple yet illustrative example of contact-rich manipulation is the task of retrieving a book from a shelf, as shown in Fig. 1.1. When the clearance around the book is limited, direct grasping is not feasible. Instead, the hand must select, at each stage, from multiple possible ways of interacting with the book: whether to establish sticking contact to pull it outward, slide the fingers to reconfigure the grip, or transition to a

secure grasp. The success of the task depends on making the right contact choices in the right order, which is precisely the sequential decision-making over multiple contact possibilities that defines contact-rich manipulation.

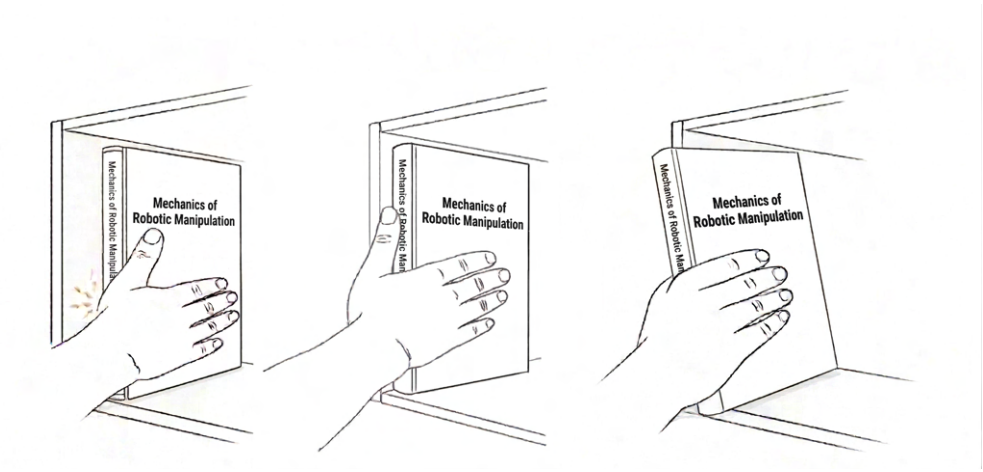


Figure 1.1: A contact-rich manipulation example of retrieving a book from a shelf with limited grasping clearance. The motion involves sequential transitions between sticking, sliding, and grasping, illustrating the challenges of nonlinear and non-smooth dynamics, hybrid contact modes, and sensitivity to physical parameters such as friction. Adapted from (Hogan and Rodriguez, 2020a).

This example reveals some fundamental challenges inherent in contact-rich manipulation. As a sequential decision-making process over multiple possible contact choices, contact-rich manipulation necessarily involves reasoning about nonlinear dynamics, hybrid contact modes, and model uncertainty. In the book-retrieval task, transitions between sticking and sliding give rise to nonlinear and non-smooth dynamics; the successive interaction phases correspond to hybrid contact modes; and the success of the motion depends sensitively on uncertain physical parameters such as friction coefficients. These challenges are pervasive across contact-rich manipulation tasks and fundamentally limit the scalability and robustness of existing approaches. Achieving this level of physical intelligence is essential for deploying robotic systems beyond traditional industrial environments into human-centered settings.

1.1.1 Challenges

Despite decades of advances in motion planning, optimal control, and recent developments in robot learning, contact-rich manipulation nevertheless remains one of the most challenging problems in robotics. The difficulty arises from the three tightly coupled structural properties of contact dynamics, as illustrated in the example above:

Nonlinear and Non-Smooth Contact Dynamics. Contact dynamics are inherently nonlinear and non-smooth due to rigid-body interactions under frictional constraints and complementarity conditions. These characteristics give rise to complex and highly non-convex objective landscapes, making the resulting optimization problems challenging to solve. Consequently, gradient-based trajectory optimization and optimal control methods are highly sensitive to initialization and often converge to poor local optima (Lembono et al., 2020; Moura et al., 2022). While sampling-based methods (Pang et al., 2022) can partially alleviate this issue by exploring the solution space more broadly, they do not scale well as the problem becomes more complex and higher-dimensional.

Hybrid Contact Modes and Combinatorial Structure. Contact-rich tasks exhibit intrinsically hybrid structures. At the dynamics level, systems switch between discrete contact modes (e.g., sticking, sliding, separation) while evolving continuously within each mode. At the task level, complex manipulation requires sequencing multiple primitives or skills, such as the pull–regrasp–grasp sequence in the book retrieval example, each associated with distinct contact configurations. Planning in such settings requires joint reasoning over continuous trajectories and discrete mode transitions (Toussaint, 2015). This hybrid structure induces a combinatorial growth in the number of feasible mode sequences as the task horizon increases. Existing approaches, including sampling-based planners (LaValle, 2006; Kavraki et al., 1996), search-based methods (Hart et al., 1968), and mixed-integer formulations (Burer and Letchford, 2012), struggle to scale due to this combinatorial structure and its tight coupling with nonlinear system dynamics.

Uncertainty in Physical Parameters. Real-world contact interactions are highly sensitive to physical parameters such as friction, compliance, and mass distribution, which are often uncertain, time-varying, or difficult to estimate accurately (Qi et al., 2023). As in the book retrieval task, even small variations in the friction coefficient between the hand and the book can determine whether the object can be successfully pulled or slips. Such sensitivity can lead to significant performance degradation or failure under model mismatch. Learning-based approaches, including imitation learning (Chi et al., 2024; Zhang and Gienger, 2024) and reinforcement learning (Sutton and Barto, 2018), offer a promising way to handle complex contact dynamics. However, they typically require large amounts of data and generalize poorly under physical uncertainty. Furthermore, contact uncertainty demands fast responses to local events, which are often only observable through high-frequency tactile feedback. However, most existing generative policies rely on computationally intensive multi-step denoising (Ho et al.,

2020) or iterative velocity integration (Lipman et al., 2023b), limiting their ability to effectively leverage such feedback.

1.1.2 Structured Approximation for Efficient Optimization

Contact-rich manipulation can be fundamentally viewed as a decision-making process comprising two tightly coupled stages: *objective approximation* (e.g., formulating a tractable surrogate objective by approximating system dynamics or objective functions) and *objective optimization* (e.g., action retrieval or trajectory generation over the approximated objective). In this context, optimization is conceptualized in a broad sense, encompassing gradient-based and derivative-free methods in optimal control, multi-step Langevin sampling in energy-based and diffusion models, and iterative velocity integration in flow-matching frameworks. However, the intrinsic properties of contact, such as non-smoothness, hybridity, and physical uncertainty, pose significant challenges to both accurately formulating these objectives and efficiently solving the resulting optimization problems.

To address these challenges, classical optimal control methods have traditionally focused on approximating non-smooth system dynamics through local linearization or randomized smoothing (Li and Todorov, 2004; Posa et al., 2014; Suh et al., 2025; Pang et al., 2022), thereby enabling the use of efficient numerical solvers. However, since contact dynamics are inherently non-smooth, such approximations only hold locally. This necessitates iterative optimization within restricted trust regions or the use of computationally expensive global search to navigate the resulting rugged and highly non-convex optimization landscapes, leading to significant real-time overhead and poor scalability in high-dimensional systems. In contrast, learning-based approaches (Sutton and Barto, 2018; Chi et al., 2024; Zhang and Gienger, 2024) have shifted the focus toward approximating the objective function (e.g., value functions, energy landscapes, or score functions). Because objective functions effectively aggregate discrete contact events into a continuous scalar field, they are naturally smoother and more amenable to function approximation than the underlying hybrid dynamics (Song et al., 2023b). However, in existing literature, these objectives are typically represented using unstructured, high-capacity models such as standard neural networks. This lack of explicit structure makes the subsequent optimization process a significant bottleneck, whether via gradient descent or iterative sampling, resulting in poor local optima and high inference latency.

In this dissertation, we argue that approximation and optimization should be treated in a coupled manner. Under this view, objective functions should be approximated

in a way that is inherently amenable to subsequent optimization. That is, objectives should be explicitly *structured* to enable efficient optimization, a paradigm we term *structured approximation*. By embedding appropriate mathematical structure during the approximation phase, we can transform otherwise intractable problems into computationally manageable ones. In particular,

- Optimization can be performed in a structured manner without relying on restrictive convex approximations;
- Hybrid mode reasoning becomes more compositional, enabling scalable planning over complex contact sequences;
- Policy learning achieves improved generalization and robustness under physical parameter uncertainty;
- Generative imitation learning enables fast closed-loop reactivity and supports the integration of high-frequency tactile feedback.

Motivated by this perspective, this dissertation develops a structure-aware computational framework centered on tensor networks (also referred to as tensor factorization or tensor decomposition) (Orús, 2019). Tensor networks, originally developed in quantum many-body physics, provide a principled mathematical tool for representing high-dimensional functions through structured factorizations. This representation offers several key advantages for contact-rich manipulation:

- **Efficient Optimization:** Tensor networks enable complex algebraic operations, such as contraction, integration, and optimization, to be performed directly on the factorized components, thereby mitigating the curse of dimensionality.
- **Compositional Hybrid Reasoning:** The discretized structure naturally captures the interplay between continuous variables (e.g., motions) and discrete variables (e.g., contact modes), enabling scalable reasoning over complex sequences.
- **Structure-Aware Modeling:** Unlike purely data-driven black-box approximators, tensor networks support efficient construction techniques such as *cross-approximation* (Oseledets and Tyrtyshnikov, 2010), which explicitly leverage underlying analytical information (e.g., Bellman equations or physical constraints) to construct structured approximations.

By treating approximation and optimization in a coupled manner, this work aims to bridge the gap between high-level objective approximation and low-level action

generation. To this end, this dissertation leverages Tensor Train (TT) (Oseledets, 2011a), a specific class of tensor networks, to exploit objective structure for efficient decision-making. Furthermore, we demonstrate that the principle of structured approximation naturally extends to imitation learning, where velocity fields can be used to capture the geometric structure of action trajectories, enabling fast, closed-loop reactivity and supporting the integration of high-frequency visual–tactile feedback in model-free settings.

In summary, the challenge of contact-rich manipulation lies not only in approximating complex objective functions, but also in uncovering and exploiting their underlying structure to enable efficient optimization. This dissertation shows that tensor networks provide a principled and scalable framework for achieving this, highlighting the role of structured approximation in enabling efficient planning, control, and learning in contact-rich systems. The introduction of velocity fields in generative policy learning further complements this perspective from a data-driven standpoint, extending it to model-free settings.

1.2 Thesis Objectives and Outline

The thesis is organized into four main parts. The first part introduces a representative contact-rich manipulation task that serves as a motivating example throughout the dissertation, highlighting its inherent physical and computational challenges. The subsequent parts demonstrate how Tensor Train and velocity fields serve as structured approximations of objective functions, thereby enabling efficient planning, control, and learning. The overall structure is illustrated in Figure 1.2.

1.2.1 Part I: A Preliminary Study of Contact-Rich Manipulation

The first part of the thesis introduces a representative contact-rich manipulation problem that serves as the motivating example throughout the dissertation. Contact-rich manipulation is characterized by nonlinear dynamics, hybrid contact modes, and uncertainty in physical parameters, which together lead to highly challenging planning and control problems. A concrete example is therefore necessary to illustrate these challenges and to evaluate the methods developed in the later parts of the dissertation.

In Chapter 3, we introduce a planar pushing task with face-switching mechanism. The corresponding contact dynamics are analyzed, and the resulting optimization problem is formulated, highlighting the inherent non-convexity, hybrid mode transitions, and physical uncertainty in contact interactions. To address these challenges, we propose

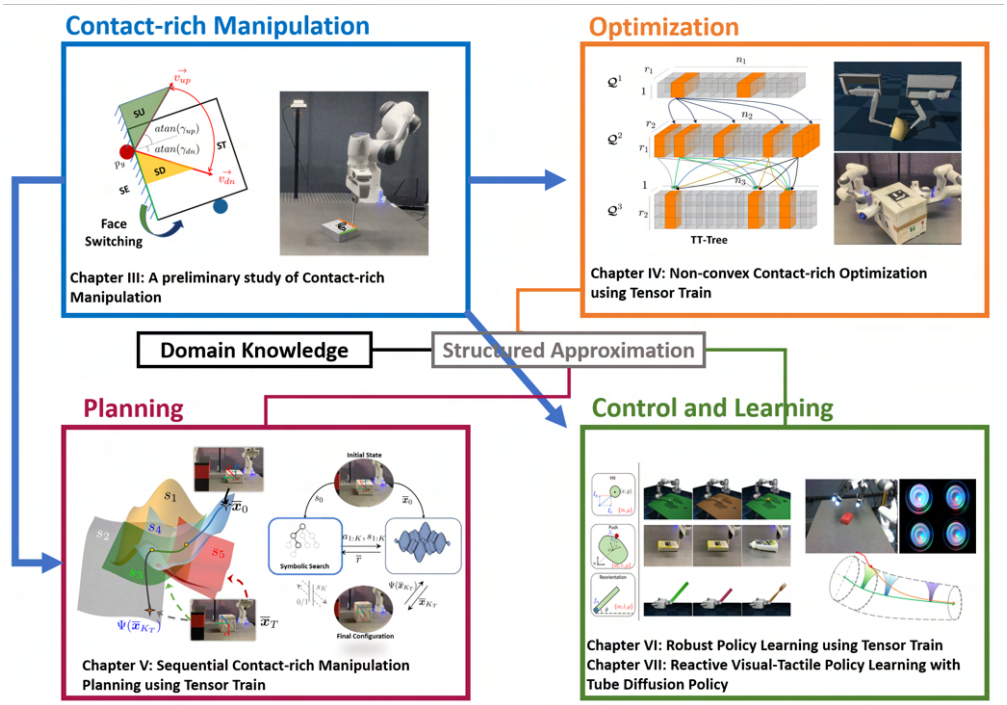


Figure 1.2: Outline of the thesis.

a demonstration-guided optimal control framework, which leverages human demonstrations as warm-start initializations for directly optimizing the task objective. While this approach enables gradient-based trajectory optimization for long-horizon non-prehensile manipulation, its performance remains highly dependent on the quality of the provided demonstrations and the resulting initial guesses. The computational limitations observed in this study motivate the structure-aware approaches developed in the subsequent chapters.

1.2.2 Part II: Tensor Networks for Contact-Rich Optimization

The second part of the thesis focuses on addressing the intrinsic non-convexity arising from contact dynamics. Contact-rich manipulation often leads to highly non-convex optimization landscapes due to nonlinear dynamics, frictional interactions, and hybrid contact modes. These properties make conventional gradient-based optimization highly sensitive to initialization and prone to poor local minima, while gradient-free optimization methods quickly become computationally intractable.

In Chapter 4, we develop a structure-aware computational framework based on tensor

networks that exploits the latent structure of objective functions during approximation. Specifically, we formulate contact-rich optimization as a decision-making process over a high-dimensional decision tree, utilizing Tensor Train as a structured approximation of this tree. This enables efficient computation and structured search in combinatorial spaces without relying on restrictive convex relaxations, which are often difficult to construct due to complex contact dynamics. By developing this framework, the chapter provides the mathematical foundation for the methods presented in subsequent chapters.

1.2.3 Part III: Tensor Networks for Sequential Contact-Rich Manipulation Planning

The third part addresses the hybrid and sequential nature of contact-rich manipulation. Complex manipulation tasks typically require composing multiple primitives across different contact modes, leading to combinatorial growth in planning complexity.

In Chapter 5, we formulate sequential contact-rich manipulation as a hierarchical decision-making problem. We analyze the challenges of long-horizon planning involving coupled discrete contact transitions and continuous motions, and introduce Tensor Train for constructing structured skill libraries with fast and global value function approximations. We further propose *Logic-Skill Programming*, which integrates symbolic search with tensor-structured continuous optimization, enabling joint reasoning over high-level skill skeletons and low-level subgoal sequences for each skill. This approach enables long-horizon contact-rich manipulation in real-world settings under contact uncertainty and external disturbances.

1.2.4 Part IV: Structure-aware Robust and Reactive Contact-rich Policy Learning

The fourth part of the dissertation addresses uncertainty in physical parameters and the challenge of sim-to-real transfer, while emphasizing reactivity in visual–tactile contact-rich settings.

In Chapter 6, we introduce a probabilistic domain transfer framework that unifies and generalizes existing domain randomization and adaptation strategies. Building on this formulation, we develop a robust policy learning approach that leverages Tensor Train to exploit the decoupled structure of diverse physical parameters, enabling efficient parameter-conditioned policy retrieval and rapid motor adaptation across varying contact conditions.

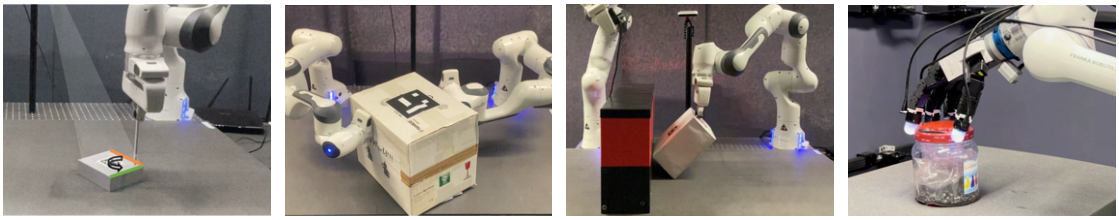


Figure 1.3: A selection of contact-rich manipulation tasks conducted in this thesis, including (a) face-switching planar push, (b) whole-body bimanual manipulation, (c) sequential contact-rich manipulation, and (d) visual-tactile dexterous manipulation.

In Chapter 7, we introduce *Tube Diffusion Policy*, which reformulates the policy from a static sequence (e.g., conventional action chunking) into a continuous dynamical system for reactive control. Specifically, the policy is represented as a tube-bounded velocity field that captures the underlying geometric structure of action trajectories. This structured approximation enables fast, closed-loop behavior, making it particularly well-suited for leveraging high-frequency tactile feedback in complex contact-rich manipulation tasks.

Together, these four parts establish a unified, structure-aware paradigm for contact-rich manipulation, spanning non-convex optimization, sequential planning, robust control, and reactive visual–tactile policy learning. The dissertation demonstrates that structured approximation provides a principled foundation for efficient decision-making in complex contact-rich robotic systems, as illustrated in Fig. 1.3.

1.3 Thesis Statement

This thesis aims to advance the frontiers of contact-rich manipulation by developing computational frameworks that reconcile three fundamental challenges: nonlinear dynamics, hybrid contact modes, and physical uncertainty. The contributions of this thesis enable robots to intelligently establish and break contact for manipulation across diverse hardware morphologies, from parallel-jaw grippers and whole-body surfaces to multi-fingered dexterous hands. Beyond these specific applications, the proposed algorithms and theoretical insights are expected to generalize and inspire research in related domains. In summary, this thesis aims to show that:

Contact-rich manipulation is not merely a problem of black-box objective approximation followed by brute-force optimization, but requires structured objective approximations that make downstream optimization efficient.

This principle reflects a structure-aware view of computation for contact-rich manipulation. Rather than treating objective approximation and objective optimization as two separate stages, this thesis emphasizes that the form of approximation directly shapes the efficiency, scalability, and robustness of the subsequent optimization process. In particular, it shows that task objectives should be approximated in ways that preserve exploitable structure, such as low-rank, compositional, or contact-induced regularities, so that action retrieval and trajectory generation can be performed efficiently. Through this perspective, robots can better bridge the gap between high-level task approximation and the low-level motion dexterity required for real-world physical interaction.

2 Background

This chapter reviews the mathematical foundations underlying tensor networks and presents background on planning, control and learning for contact-rich manipulation. The objective of this chapter is to provide the theoretical foundation necessary for the subsequent developments in this thesis, while also reviewing established methodologies in the field.

2.1 Matrix Factorization and Tensor Networks

2.1.1 Matrix Factorization

Matrix factorization provides the simplest form of function structure exploitation. Consider a bivariate function

$$f(x, y),$$

defined over compact domains $\Omega_x \subset \mathbb{R}^{d_x}$ and $\Omega_y \subset \mathbb{R}^{d_y}$. After discretization over grids $\{x_i\}_{i=1}^n$ and $\{y_j\}_{j=1}^m$, the function can be represented as a matrix

$$\boldsymbol{F}_{ij} = f(x_i, y_j).$$

A rank-r matrix factorization seeks an approximation of the form

$$\boldsymbol{F} \approx \boldsymbol{U}\boldsymbol{V}^\top, \quad \boldsymbol{U} \in \mathbb{R}^{n\times r}, \quad \boldsymbol{V} \in \mathbb{R}^{m\times r},$$

which is equivalent to the functional approximation

$$f(x, y) \approx \sum_{k=1}^{r} \phi_k(x)\psi_k(y),$$

where $\phi_k(x_i) = \boldsymbol{U}_{ik}$ and $\psi_k(y_j) = \boldsymbol{V}_{jk}$.

Computational Advantage.

A dense matrix representation requires storing nm entries. In contrast, a rank-r approximation stores only the factor matrices, requiring

$$r(n + m)$$

parameters. When $r \ll \min(n, m)$, the storage cost is reduced from quadratic to linear scaling in the problem dimension.

Beyond storage reduction, the factorized representation also enables more structured computation and search. Instead of operating on the entire $n \times m$ grid, algorithms can exploit the separable form

$$\boldsymbol{F}_{ij} \approx \sum_{k=1}^{r} \phi_k(x_i)\psi_k(y_j)$$

and perform optimization over the lower-dimensional factors. As a result, operations that would otherwise require examining all nm entries can often be carried out with complexity on the order of $O(r(n + m))$.

When extended to higher-dimensional tensors, this principle becomes crucial for avoiding the exponential scaling that arises from enumerating all combinations of variables.

In summary, matrix factorization embodies the core principles of variable separation and structured approximation, offering both computational efficiency and physical interpretability. Tensor networks generalize this idea to high-dimensional functions.

2.1.2 Tensor and Tensor Networks

A multivariate function $f(x_1, \ldots, x_d)$ defined on a Cartesian product domain $I_1 \times \cdots \times I_d$ can be discretized into a tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ by sampling it on a tensor-product grid:

$$\mathcal{F}_{i_1,\ldots,i_d} = f(x_1^{i_1},\ldots,x_d^{i_d}), \quad i_k \in \{1,\ldots,n_k\}.$$

The storage complexity of $\mathcal{F}$ scales as

$$\mathcal{O}\left(\prod_{k=1}^{d} n_k\right),$$

which grows exponentially with dimension d. Such scaling becomes prohibitive in high-dimensional problems arising in robotics, including trajectory optimization and value function approximation.

Tensor networks (TNs) (Orús, 2019) address this challenge by representing $\mathcal{F}$ in a structured form. Originally developed in quantum physics for modeling many-body systems, they exploit localized correlations among subsystems, allowing the global wavefunction to be factorized into a network of low-rank tensors that capture the dominant local dependencies.

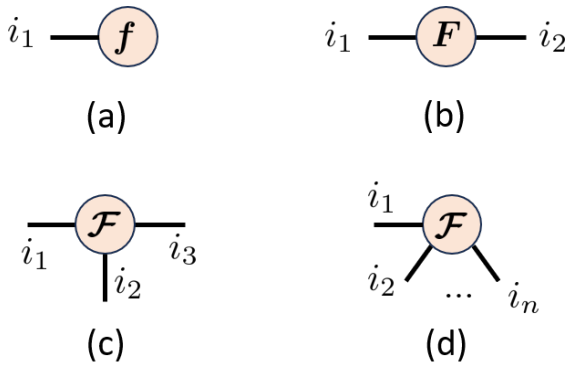
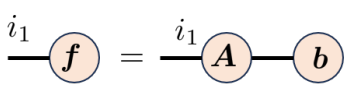


Figure 2.1: Basic diagrammatic notation for tensor networks. Each tensor is represented as a node, with its indices encoded by the connected edges. The number of such edges therefore defines the tensor order. Subfigures (a)–(d) illustrate tensors of various orders, ranging from vectors and matrices to higher-order cases.

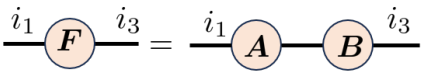
Common TNs include Canonical Polyadic (CP) (Kolda and Bader, 2009), Tucker (Tucker, 1966), Tensor Train (TT) (Oseledets, 2011a), and Hierarchical Tucker (HT) decompositions (Grasedyck, 2010). Tensor diagrams present a convenient way to describe them, as illustrated in Fig. 2.3, with the associated symbols and contraction operations shown in Fig. 2.1 and Fig. 2.2. TNs have recently gained significant attention across diverse disciplines, including machine learning (Rabanser et al., 2017), computer vision (Yang

(a)



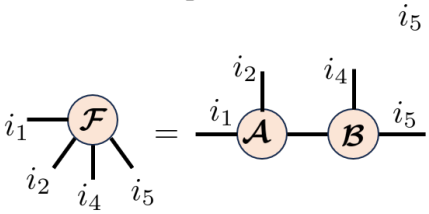
$$\boldsymbol{u}_{i_1} = \sum_{i_2} \boldsymbol{A}_{i_1,i_2} \boldsymbol{b}_{i_2}$$

(b)



$$\boldsymbol{F}_{i_1,i_3} = \sum_{i_2} \boldsymbol{A}_{i_1,i_2} \boldsymbol{B}_{i_2,i_3}$$

(c)



$$\boldsymbol{\mathcal{F}}_{i_1,i_2,i_4,i_5} = \sum_{i_3} \boldsymbol{\mathcal{A}}_{i_1,i_2,i_3} \boldsymbol{\mathcal{B}}_{i_3,i_4,i_5}$$

Figure 2.2: Tensor contraction operations. (a) Matrix–vector multiplication, resulting in a vector. (b) Matrix–matrix multiplication, producing a matrix. (c) Contraction between two third-order tensors, resulting in a fourth-order tensor. In these subfigures, edges connecting nodes represent shared indices over which summation is performed during contraction.

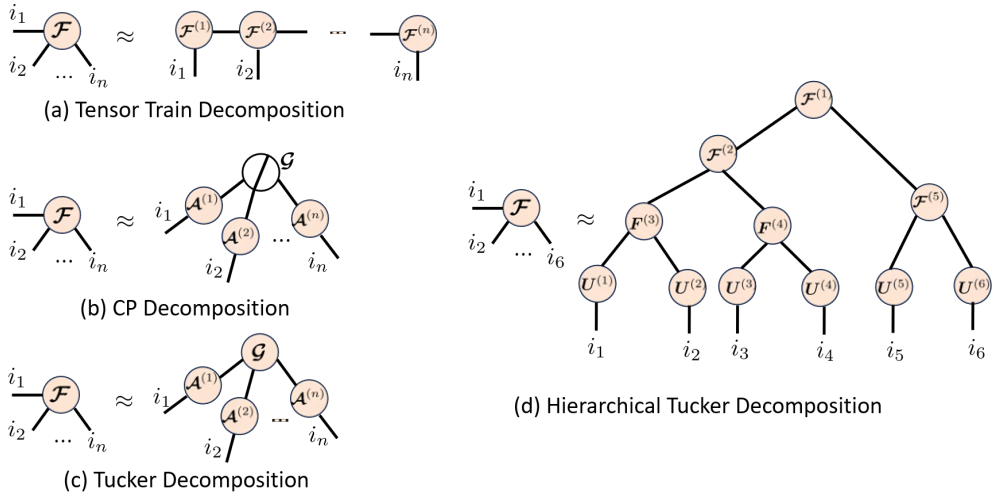


Figure 2.3: Tensor networks. Using tensor diagrams and contraction operations, various tensor networks can be represented, such as (a) Tensor Train (TT), (b) Canonical Polyadic (CP), (c) Tucker, and (d) Hierarchical Tucker (HT) decompositions. Given a high-dimensional tensor, TT represents it as a sequence of small third-order tensor cores, while CP expresses it as a sum of rank-1 components. Tucker decomposition represents it using a core tensor $\mathcal{G}$ together with factor matrices along each mode, and HT organizes it into a tree-structured network. For further background on tensor networks, we refer to (Orús, 2019; Biamonte and Bergholm, 2017; Cichocki et al., 2016).

et al., 2017), and control theory (Horowitz et al., 2014; Gorodetsky et al., 2015), owing to their ability to efficiently represent high-dimensional functions, exploit structured dependencies, and enable scalable computation. For further details, we refer the reader to (Biamonte and Bergholm, 2017; Cichocki et al., 2016).

Among the various types of TNs, Tensor Train (TT) is particularly appealing due to its linear scaling with the problem dimension, strong numerical stability, and its ability to capture separable structures for efficient optimization. Accordingly, this thesis adopts TT as the core computational tool. Specifically, TT represents a high-dimensional tensor $\boldsymbol{\mathcal{F}}$ as a sequence of *small* third-order cores, where each entry is expressed as

$$\boldsymbol{\mathcal{F}}(i_1, \ldots, i_d) = \boldsymbol{\mathcal{F}}^1_{:,i_1,:} \boldsymbol{\mathcal{F}}^2_{:,i_2,:} \cdots \boldsymbol{\mathcal{F}}^d_{:,i_d,:}, \quad i_k \in \{1, \ldots, n_k\},$$

where

$$\boldsymbol{\mathcal{F}}^k_{:,i_k,:} \in \mathbb{R}^{r_{k-1} \times r_k}, \quad r_0 = r_d = 1,$$

and $\{r_k\}_{k=1}^{d-1}$ are called the TT-ranks. Figure 2.4 presents TT decomposition for 3D and 4D tensors.

2.1.3 Function Approximation in Tensor Train Format

Algorithms such as TT-SVD (Oseledets, 2011b) and TT-Cross (Oseledets and Tyrtyshnikov, 2010; Savostyanov and Oseledets, 2011) provide practical frameworks for approximating multivariate functions in TT format.

TT-SVD constructs the decomposition through successive matricizations and truncated singular value decompositions, yielding quasi-optimal low-rank approximations but requiring access to the full tensor.

TT-Cross, in contrast, avoids full tensor evaluation by constructing the TT representation from adaptively selected tensor entries. Based on cross approximation and the maximum-volume principle, TT-Cross identifies informative rows and columns of unfolding matrices and interpolates the remaining entries. This approach is particularly effective when $\boldsymbol{\mathcal{F}}$ is defined by known or analytically derived functions.

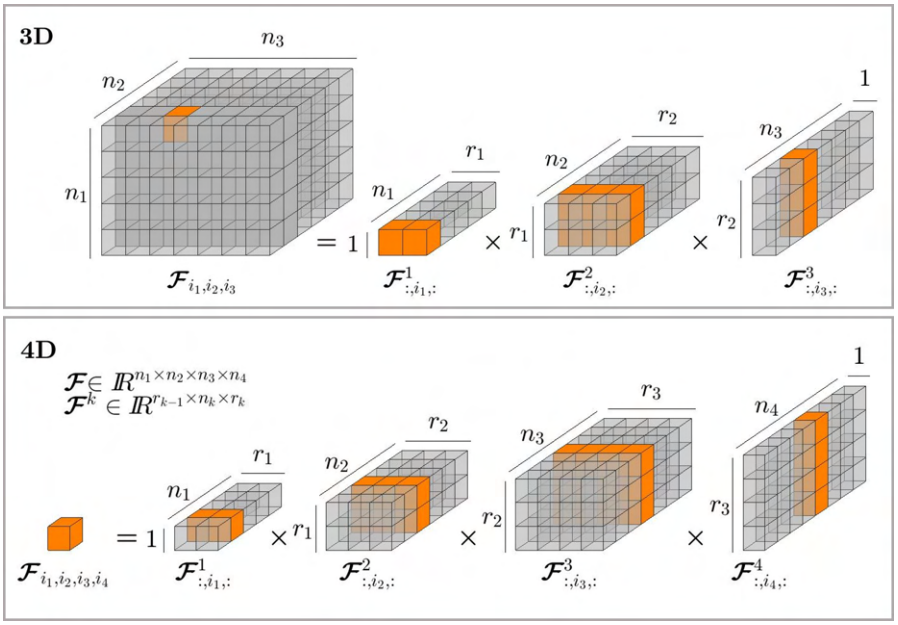


Figure 2.4: Tensor Train (TT) decomposition for 3D and 4D tensors. TT extends matrix decomposition to higher-order tensors, where each tensor entry can be obtained by multiplying the corresponding slices of the TT cores.

2.1.4 Algebraic Operations over Tensor Train

A central advantage of TT is that many algebraic operations can be performed directly on tensor cores without reconstructing the full tensor. This property is essential in high-dimensional optimization and control, where repeated algebraic manipulations are required.

Let $\mathcal{F}$ and $\mathcal{G}$ be two tensors in TT format with cores $\{\mathcal{F}^k\}$ and $\{\mathcal{G}^k\}$ and TT-ranks $\{r_k\}$ and $\{s_k\}$, respectively.

Addition. The sum

$$\mathcal{H} = \mathcal{F} + \mathcal{G}$$

can be represented in TT format by block-diagonal concatenation of cores:

$$\mathcal{H}^k_{:,i_k,:} = \begin{bmatrix} \mathcal{F}^k_{:,i_k,:} & 0 \\ 0 & \mathcal{G}^k_{:,i_k,:} \end{bmatrix}.$$

The resulting TT-ranks satisfy

$$r_k^{(H)} = r_k + s_k.$$

After addition, TT-rounding can be applied to reduce ranks while controlling approximation error.

Hadamard (Element-wise) Product. The element-wise product

$$\boldsymbol{\mathcal{H}}(i_1, \ldots, i_d) = \boldsymbol{\mathcal{F}}(i_1, \ldots, i_d)\boldsymbol{\mathcal{G}}(i_1, \ldots, i_d)$$

admits a TT representation with cores

$$\boldsymbol{\mathcal{H}}^k_{:,i_k,:} = \boldsymbol{\mathcal{F}}^k_{:,i_k,:} \otimes \boldsymbol{\mathcal{G}}^k_{:,i_k,:},$$

where $\otimes$ denotes the Kronecker product.

The TT-ranks multiply:

$$r_k^{(H)} = r_k s_k.$$

The Hadamard product is particularly important in nonlinear cost construction, likelihood weighting, and pointwise Bellman updates.

Inner Product and Norm. For two tensors $\boldsymbol{\mathcal{F}}$ and $\boldsymbol{\mathcal{G}}$ defined on the same discrete domain, the inner product is defined as

$$\langle \boldsymbol{\mathcal{F}}, \boldsymbol{\mathcal{G}} \rangle = \sum_{i_1,\ldots,i_d} \boldsymbol{\mathcal{F}}(i_1, \ldots, i_d)\boldsymbol{\mathcal{G}}(i_1, \ldots, i_d).$$

When the tensors are represented in TT format, this inner product can be computed efficiently by sequentially contracting the corresponding TT cores. Let $\boldsymbol{\mathcal{F}}^k$ and $\boldsymbol{\mathcal{G}}^k$ denote the k-th TT cores with ranks r_k and s_k, respectively. The contraction proceeds along the dimensions and produces intermediate matrices whose sizes depend on the TT ranks. The computational complexity is

$$\mathcal{O}\left(\sum_{k=1}^{d} n_k r_{k-1} r_k s_{k-1} s_k\right),$$

which scales linearly with the tensor order d when the TT ranks remain moderate.

The Frobenius norm of a tensor follows directly from the inner product with itself,

$$\begin{aligned} \|\boldsymbol{\mathcal{F}}\|_F &= \sqrt{\langle \boldsymbol{\mathcal{F}}, \boldsymbol{\mathcal{F}} \rangle} \\ &= \left(\sum_{i_1,\dots,i_d} \boldsymbol{\mathcal{F}}(i_1,\dots,i_d)^2 \right)^{1/2} \\ &= \left(\sum_{i_1,\dots,i_d} \left(\boldsymbol{\mathcal{F}}^1_{:,i_1,:} \cdots \boldsymbol{\mathcal{F}}^d_{:,i_d,:} \right)^2 \right)^{1/2} . \end{aligned} \tag{2.1}$$

Both the inner product and the Frobenius norm can therefore be evaluated through successive contractions of the TT cores without explicitly forming the full tensor.

Tensor Contraction and Marginalization. Summation over selected indices can be performed by contracting the corresponding core:

$$\sum_{i_k} \boldsymbol{\mathcal{F}}^k_{:,i_k,:}.$$

This operation is fundamental in:

- Marginalization of probability tensors,
- Bellman backup operations,
- Integration over uncertainty.

Linear Operator Application. Let $\boldsymbol{\mathcal{A}}$ be a linear operator represented in TT format. The product

$$\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{A}}\boldsymbol{\mathcal{F}}$$

can be computed by contracting corresponding cores, leading to TT-ranks that scale multiplicatively. This enables efficient solution of high-dimensional linear systems and operator equations entirely within TT format.

Rank Growth and TT-Rounding. Most algebraic operations increase TT-ranks. TT-rounding applies successive SVD truncations to ensure

$$\|\boldsymbol{\mathcal{F}} - \boldsymbol{\mathcal{F}}_{\text{rounded}}\| \leq \varepsilon,$$

while restoring compactness.

These algebraic capabilities allow optimization algorithms, dynamic programming updates, and probabilistic inference to be performed directly in compressed form, which is crucial for addressing the challenges of contact-rich manipulation.

2.1.5 Continuous Function Approximation in TT Format

Given a TT representation of the discretized tensor $\boldsymbol{\mathcal{F}}$, one can extend it to approximate the continuous function f through interpolation across tensor cores.

For example, using linear interpolation between adjacent slices,

$$\boldsymbol{F}^k(x_k) = \frac{x_k - x_k^{i_k}}{x_k^{i_k+1} - x_k^{i_k}} \boldsymbol{\mathcal{F}}^k_{:,i_k+1,:} + \frac{x_k^{i_k+1} - x_k}{x_k^{i_k+1} - x_k^{i_k}} \boldsymbol{\mathcal{F}}^k_{:,i_k,:},$$

for $x_k^{i_k} \leq x_k \leq x_k^{i_k+1}$.

The resulting approximation takes the form

$$f(x_1, \ldots, x_d) \approx \boldsymbol{F}^1(x_1)\boldsymbol{F}^2(x_2)\cdots\boldsymbol{F}^d(x_d),$$

allowing efficient evaluation of high-dimensional functions defined on mixed discrete–continuous domains.

2.1.6 Tensor Optimization

When a function defined on a discretized domain is represented in TT format, its values naturally define a probability distribution over the domain, providing a mechanism for exploring large combinatorial spaces without explicit enumeration. The TT structure enables probabilistic operations, such as sampling and conditional inference, to be performed efficiently via tensor contractions.

In this section, we introduce the concept of *TT distribution*, describe how to efficiently generate samples from it, and discuss how conditioning can be performed.

TT Distribution

Consider a tensor $\boldsymbol{\mathcal{F}}$ represented in TT format that approximates a function f defined over a discretized domain $\mathcal{X} \subset \Omega_{\mathbf{x}}$. Such a representation allows the function values to be stored compactly through a sequence of low-rank TT cores.

From this tensor representation, one can naturally define a probability distribution over the discretized domain. In particular, we introduce the *TT distribution* defined as

$$p(\mathbf{x}) = \frac{\boldsymbol{\mathcal{F}}_{\mathbf{x}}^2}{Z}, \qquad \mathbf{x} \in \mathcal{X}, \tag{2.2}$$

where the normalization constant

$$Z = \sum_{\mathbf{x} \in \mathcal{X}} \boldsymbol{\mathcal{F}}_{\mathbf{x}}^2 \tag{2.3}$$

ensures that $p(\mathbf{x})$ forms a valid probability distribution. The squared form guarantees non-negativity of the resulting probabilities.

A key benefit of TT is its separable structure. Because the tensor is factorized into a sequence of TT cores, many operations on the induced probability distribution can be performed efficiently through tensor contractions. In particular, sampling from the TT distribution can be carried out without explicitly computing the normalization constant Z. When required, Z itself can also be evaluated efficiently using TT-based summation operations, as described in (2.1).

Sampling from TT Distribution

Sampling from a TT distribution can be performed sequentially using the factorization of the joint distribution into conditional distributions:

$$p(x_1, \ldots, x_d) = p_1(x_1)\, p_2(x_2 \mid x_1) \cdots p_d(x_d \mid x_1, \ldots, x_{d-1}). \tag{2.4}$$

Each conditional can be expressed in terms of marginals

$$p_k(x_k \mid x_1, \ldots, x_{k-1}) = \frac{m_k(x_1, \ldots, x_k)}{m_{k-1}(x_1, \ldots, x_{k-1})}, \tag{2.5}$$

where

$$m_k(x_1, \ldots, x_k) = \sum_{x_{k+1}} \cdots \sum_{x_d} p(x_1, \ldots, x_d), \qquad m_0 = 1. \tag{2.6}$$

Therefore, samples can be generated recursively as

$$x_k \sim p_k(x_k \mid x_1, \ldots, x_{k-1}), \qquad k = 1, \ldots, d. \tag{2.7}$$

Direct evaluation of the marginals in (2.6) would require summation over exponentially many tensor entries. However, when $p(\mathbf{x})$ is represented in TT format, the marginal can be computed efficiently by contracting the TT cores:

$$\begin{aligned} m_k(x_1, \ldots, x_k) &= \sum_{x_{k+1}} \cdots \sum_{x_d} p(\mathbf{x}) \\ &\approx \sum_{x_{k+1}} \cdots \sum_{x_d} \boldsymbol{\mathcal{F}}_{\mathbf{x}} \\ &= \boldsymbol{\mathcal{F}}^1_{:,x_1,:} \cdots \boldsymbol{\mathcal{F}}^k_{:,x_k,:} \Big(\sum_{x_{k+1}} \boldsymbol{\mathcal{F}}^{k+1}_{:,x_{k+1},:} \Big) \cdots \Big(\sum_{x_d} \boldsymbol{\mathcal{F}}^d_{:,x_d,:} \Big). \end{aligned} \tag{2.8}$$

Thus, the high-dimensional summation reduces to a sequence of low-dimensional contractions over the TT cores. This structure enables sampling complexity that scales linearly with the number of dimensions, rather than exponentially. In practice, algorithms such as Tensor-Train Conditional Distribution (TT-CD) sampling (Dolgov et al., 2020) exploit this property to efficiently generate exact samples from TT distribution.

Conditional TT Distribution

The TT representation also enables efficient conditioning on subsets of variables. Let the variable vector be partitioned as $\mathbf{x} = (\mathbf{y}_1, \mathbf{y}_2)$, where $\mathbf{y}_1$ contains the first d_1 variables and $\mathbf{y}_2$ contains the remaining variables. Given a fixed assignment $\mathbf{y}_1 = \mathbf{y}_t$, a conditioned TT tensor $\boldsymbol{\mathcal{F}}^{\mathbf{y}_t}$ can be obtained by selecting the corresponding slices from the first d_1 TT cores while leaving the remaining cores unchanged:

$$(\boldsymbol{\mathcal{F}}^{\mathbf{y}_t})^k = \begin{cases} \boldsymbol{\mathcal{F}}^k_{:,y_{t_k},:}, & k = 1, \ldots, d_1, \\ \boldsymbol{\mathcal{F}}^k, & k = d_1 + 1, \ldots, d. \end{cases} \tag{2.9}$$

This conditioned tensor defines a distribution over $\mathbf{y}_2$,

$$p(\mathbf{y}_2 \mid \mathbf{y}_1 = \mathbf{y}_t) = \frac{|\boldsymbol{\mathcal{F}}^{\mathbf{y}_t}_{\mathbf{y}_2}|}{Z_1}, \tag{2.10}$$

where Z_1 is the corresponding normalization constant. Hence, conditioning reduces to slicing the relevant TT cores, which makes conditional inference and conditional sampling computationally efficient.

The TT framework introduced here forms the basis for the optimization and sampling strategies developed in the following chapters.

2.2 Task and Motion Planning

Robot manipulation often require reasoning at multiple levels of abstraction. At the high level, a robot must determine a sequence of discrete actions such as grasping, placing, or switching contact modes. At the low level, these actions must be realized through continuous robot motions that satisfy geometric, kinematic, and dynamic constraints. The problem of jointly solving these discrete and continuous decisions is commonly referred to as *Task and Motion Planning* (TAMP).

TAMP problems are inherently challenging because symbolic task decisions and geometric feasibility are tightly coupled. A high-level task plan may be logically valid but geometrically infeasible due to collisions or kinematic limitations. Conversely, feasible motion plans may depend on specific symbolic decisions such as grasp choices or contact modes. As a result, solving TAMP problems typically requires integrating symbolic reasoning with continuous motion planning (Garrett et al., 2021).

Existing approaches to TAMP can be broadly categorized into *sampling-based methods* and *optimization-based methods*. These two paradigms differ primarily in how they explore the combined symbolic and geometric search space.

2.2.1 Sampling-based Task and Motion Planning

Sampling-based approaches address TAMP by interleaving symbolic planning with geometric sampling. In these methods, a symbolic planner generates candidate action sequences while geometric feasibility is evaluated through sampling-based procedures.

Early work such as FFRob (Garrett et al., 2018) integrates symbolic planning with sampling-based motion planning by lazily sampling geometric parameters such as grasps, placements, and robot configurations. More recent frameworks such as PDDLStream (Garrett et al., 2020) generalize this idea by allowing symbolic planners to invoke external generators that produce geometric samples on demand.

These methods are attractive because they can leverage powerful symbolic planners

and probabilistically complete motion planning algorithms. However, they often rely on extensive sampling and repeated feasibility checks, which may become computationally expensive in problems with high-dimensional continuous spaces or complex contact interactions.

2.2.2 Optimization-based Task and Motion Planning

Optimization-based approaches formulate TAMP as a mathematical program that jointly optimizes continuous trajectories and discrete task decisions. In this paradigm, the goal is to directly solve for trajectories that satisfy geometric constraints while achieving task objectives.

One prominent framework in this category is Logic-Geometric Programming (LGP) (Toussaint, 2015), which integrates first-order logic into a mathematical program for addressing TAMP problems. The general formulation is as follows: Given a logic $\mathcal{L}$, a knowledge base $\mathcal{K} \in \mathcal{L}$, and an objective function $l(x)$ over geometric configurations $x \in \mathcal{X}$, the logic-geometric program can be expressed as:

$$\min_{x,\kappa} l(x) \quad \text{s.t. } \kappa \models \mathcal{K},\ g(x,\kappa) \leq 0,\ h(x,\kappa) = 0, \tag{2.11}$$

where $\models$ represents logical implication, indicating that $\mathcal{K}$ is satisfied once the logical statement κ is True. This defines the associated equality and inequality constraints: $g(x,\kappa) \leq 0$ and $h(x,\kappa) = 0$.

Optimization-based methods can exploit gradient information and structured constraints to efficiently solve high-dimensional motion planning problems. Moreover, they naturally support reasoning about contact dynamics, trajectory smoothness, and task objectives within a unified framework. However, these methods often require careful initialization and may suffer from local minima in highly nonconvex optimization problems.

2.3 Optimal Control

Optimal control provides a principled framework for computing control inputs that minimize a cost while satisfying the system dynamics. In robotics, optimal control methods are widely used for motion planning, trajectory optimization, and feedback control.

Consider a discrete-time dynamical system

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.12}$$

where $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$ denotes the system state and $\boldsymbol{u}_k \in \mathbb{R}^{n_u}$ denotes the control input at time step k. Given an initial state $\boldsymbol{x}_0$, the goal is to compute a sequence of control inputs $\{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{T-1}\}$ that minimizes the cumulative cost

$$\min_{\{\boldsymbol{u}_k\}} \sum_{k=0}^{T-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \ell_T(\boldsymbol{x}_T) \tag{2.13}$$

subject to the system dynamics.

The approaches can broadly be categorized into gradient-based methods, which exploit derivative information of the dynamics and cost functions, and gradient-free methods, which explore the control space through sampling or stochastic search.

2.3.1 Gradient-Based Methods

Gradient-based optimal control methods exploit derivative information of the dynamics and cost functions to iteratively improve a candidate trajectory. When the dynamics are smooth and differentiable, these methods typically achieve fast convergence and high solution accuracy. Depending on how the trajectory optimization problem is formulated, gradient-based approaches are commonly categorized into shooting methods and collocation methods.

Shooting Methods

Shooting methods treat the control sequence as the primary optimization variable. Starting from an initial control sequence $\boldsymbol{u} = \{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{T-1}\}$, the state trajectory is obtained by propagating the system dynamics

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.14}$$

which produces the corresponding state trajectory $\boldsymbol{x} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T\}$.

The optimal control problem can therefore be written as an optimization over the control sequence

$$\min_{\boldsymbol{u}} \sum_{k=0}^{T-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \ell_T(\boldsymbol{x}_T), \tag{2.15}$$

where the states are obtained through forward simulation of the dynamics.

Among shooting methods, Differential Dynamic Programming (DDP) (Jacobson and Mayne, 1970) is one of the most widely used algorithms. DDP iteratively improves a nominal trajectory by performing a backward pass followed by a forward rollout.

During the backward pass, the value function is approximated locally using a second-order Taylor expansion around the nominal trajectory

$$V(\boldsymbol{x}_k + \delta\boldsymbol{x}_k) \approx V_k + V_x^\top \delta\boldsymbol{x}_k + \frac{1}{2}\delta\boldsymbol{x}_k^\top V_{xx}\delta\boldsymbol{x}_k. \tag{2.16}$$

Similarly, the local Q-function is approximated as

$$Q(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) = Q_0 + Q_x^\top \delta\boldsymbol{x}_k + Q_u^\top \delta\boldsymbol{u}_k + \frac{1}{2}\begin{bmatrix}\delta\boldsymbol{x}_k \\ \delta\boldsymbol{u}_k\end{bmatrix}^\top \begin{bmatrix}Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{x}_k \\ \delta\boldsymbol{u}_k\end{bmatrix}. \tag{2.17}$$

Minimizing this quadratic model yields the control update

$$\delta\boldsymbol{u}_k = \boldsymbol{k}_k + \boldsymbol{K}_k\delta\boldsymbol{x}_k, \tag{2.18}$$

where $\boldsymbol{k}_k$ is the feedforward term and $\boldsymbol{K}_k$ is the feedback gain.

The forward pass then applies the updated control law

$$\boldsymbol{u}_k \leftarrow \boldsymbol{u}_k + \alpha\boldsymbol{k}_k + \boldsymbol{K}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) \tag{2.19}$$

to generate an improved trajectory.

DDP and its variants, such as the iterative Linear Quadratic Regulator (iLQR) (Li and Todorov, 2004), have become popular in robotics due to their computational efficiency and their ability to handle nonlinear dynamics. These methods have been successfully applied to a wide range of robotic control problems including locomotion and manipulation.

However, gradient-based shooting methods rely on smooth system dynamics and accurate gradient information. In contact-rich manipulation tasks, the presence of hybrid contact modes and nonsmooth dynamics can lead to highly non-convex optimization landscapes, making these methods sensitive to initialization.

Collocation Methods

Direct collocation methods (Hargraves and Paris, 1987) discretize both the system states and control inputs and treat them as optimization variables in a large-scale nonlinear program. The optimal control problem can be written as

$$\min_{\{\boldsymbol{x}_k, \boldsymbol{u}_k\}} \quad \sum_{k=0}^{T-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \ell_T(\boldsymbol{x}_T) \tag{2.20}$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.21}$$

$$\boldsymbol{x}_k \in \mathcal{X}, \quad \boldsymbol{u}_k \in \mathcal{U}. \tag{2.22}$$

In practice, the dynamics constraints are enforced at a set of collocation points using numerical integration schemes such as Euler or trapezoidal integration.

Compared with shooting methods, collocation approaches can provide better numerical stability for long planning horizons because the state trajectory is optimized directly rather than obtained purely through forward simulation, but they typically lead to larger nonlinear optimization problems due to the introduction of additional decision variables and constraints, which can increase computational cost and complicate the treatment of hybrid or nonsmooth dynamics.

2.3.2 Gradient-Free Methods

Gradient-free optimization methods do not require derivatives of the objective or the system dynamics. Instead, they explore the control space using sampling or stochastic search strategies. Such methods are often more robust to non-smooth dynamics and discontinuous cost functions, which commonly arise in contact-rich manipulation.

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a sampling-based algorithm for sequential decision making Browne et al. (2012). It incrementally builds a search tree by balancing exploration and exploitation during node selection.

At each step, the next node is selected according to the upper confidence bound (UCB) criterion

$$i^* = \arg\max_i \left(\frac{w_i}{v_i} + c\sqrt{\frac{\log v_p}{v_i}} \right), \tag{2.23}$$

where w_i denotes the accumulated reward of node i, v_i is the visit count of node i, v_p is the visit count of its parent node, and c is a constant that balances exploration and exploitation.

After node selection, the algorithm expands the search tree, performs a rollout simulation to estimate the outcome of the selected action, and propagates the resulting reward back through the tree.

MCTS has been successfully applied to problems with large discrete action spaces and combinatorial decision structures, including planning problems involving hybrid contact modes.

Cross-Entropy Method

The Cross-Entropy Method (CEM) is a stochastic optimization algorithm that iteratively updates a sampling distribution over candidate solutions (Rubinstein, 1999). At iteration t, a set of candidate solutions $\{\boldsymbol{u}^{(i)}\}_{i=1}^{N}$ is sampled from a parameterized distribution

$$\boldsymbol{u}^{(i)} \sim p(\boldsymbol{u}; \boldsymbol{\theta}_t). \tag{2.24}$$

After evaluating the objective function, the parameters of the distribution are updated using the elite samples with the highest rewards. For a Gaussian distribution $p(\boldsymbol{u}; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the update rule becomes

$$\boldsymbol{\mu}_{t+1} = \frac{1}{K} \sum_{i \in \mathcal{E}} \boldsymbol{u}^{(i)}, \tag{2.25}$$

$$\boldsymbol{\Sigma}_{t+1} = \frac{1}{K} \sum_{i \in \mathcal{E}} \left(\boldsymbol{u}^{(i)} - \boldsymbol{\mu}_{t+1}\right) \left(\boldsymbol{u}^{(i)} - \boldsymbol{\mu}_{t+1}\right)^{\top}, \tag{2.26}$$

where $\mathcal{E}$ denotes the set of elite samples.

CEM has been widely used in trajectory optimization and model predictive control due to its simplicity and robustness to nonsmooth objective landscapes.

Covariance Matrix Adaptation Evolution Strategy

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a population-based evolutionary algorithm that adapts the covariance matrix of a Gaussian sampling distribution (Hansen et al., 2003). At iteration t, candidate solutions are sampled

according to

$$\boldsymbol{u}^{(i)} \sim \mathcal{N}(\boldsymbol{m}_t, \sigma_t^2 \boldsymbol{C}_t), \tag{2.27}$$

where $\boldsymbol{m}_t$ is the mean, σ_t is the step size, and $\boldsymbol{C}_t$ is the covariance matrix.

The distribution parameters are then updated using the best-performing samples

$$\boldsymbol{m}_{t+1} = \sum_{i=1}^{\mu} w_i \boldsymbol{u}^{(i)}, \tag{2.28}$$

while the covariance matrix is adapted to capture correlations between decision variables.

By adapting the covariance matrix, CMA-ES can efficiently explore high-dimensional parameter spaces and has been applied to policy optimization and trajectory optimization problems where gradient information is unavailable or unreliable.

Although gradient-free methods are typically more robust to discontinuities and nonsmooth dynamics, they often require a large number of function evaluations and may become computationally expensive in high-dimensional control problems.

2.4 Robust Policy Learning

Learning-based control has become an increasingly important approach for complex robotic manipulation tasks. In particular, reinforcement learning (RL) (Sutton and Barto, 2018) enables robots to acquire control policies through trial-and-error interactions with the environment. However, obtaining large amounts of real-world data for robot learning is often time-consuming, expensive, and potentially unsafe. As a result, a common paradigm is to train control policies in simulation and transfer them to real robotic systems (Peng et al., 2018).

A major challenge in this paradigm is the *sim-to-real gap*. Simulation models inevitably differ from real-world dynamics due to modeling inaccuracies, unmodeled contacts, sensor noise, and unknown physical parameters such as friction coefficients or object masses. These discrepancies can significantly degrade the performance of policies trained purely in simulation when deployed on real robots. To address this challenge, several approaches have been proposed to improve the robustness and transferability of learned control policies.

2.4.1 Domain Randomization

Domain randomization is one of the most widely used approaches for sim-to-real transfer. Instead of training a policy under a single simulation model, domain randomization introduces random variations in key environment parameters during training, such as masses, friction coefficients, sensor noise, and other physical properties (Tobin et al., 2017). By exposing the policy to a diverse range of simulated conditions, the learned controller is encouraged to develop behaviors that generalize across variations in system dynamics.

While domain randomization can improve generalization, it often requires specifying distributions over environment parameters. In practice, these distributions are typically chosen heuristically, for example using uniform or Gaussian sampling. Such assumptions may lead to overly conservative policies that must perform reasonably well across many possible environments, sometimes resulting in suboptimal or high-variance behaviors.

2.4.2 Domain Adaptation

Domain adaptation seeks to reduce the discrepancy between simulation and the real world by leveraging data from the target environment. When the target domain is known and specific, domain adaptation can be an effective approach for improving transfer performance (Bousmalis et al., 2018; Arndt et al., 2020; Chebotar et al., 2019). These methods typically use real-world observations to adapt policies or system representations learned in simulation, for example through system identification, feature alignment, or policy fine-tuning.

However, domain adaptation often relies on domain-specific data collected from the target environment. As a result, the learned policies may become specialized to a particular environment, limiting their ability to generalize to other scenarios with different dynamics or physical conditions.

2.4.3 Rapid Motor Adaptation

More recently, rapid motor adaptation (RMA) (Yu et al., 2017), also known as the teacher–student policy framework (Lee et al., 2020a), has emerged as an effective approach for learning robust control policies and has demonstrated strong performance in both locomotion (Kumar et al., 2021) and manipulation (Qi et al., 2023) tasks.

In this paradigm, a teacher policy is first trained in simulation under a diverse set of

environment parameters, such as variations in mass, friction, or external disturbances. A student policy is then trained to imitate the teacher's behavior while receiving a compact embedding derived from recent proprioceptive observations. This embedding is produced by an adaptation module that processes a short history of sensor measurements and encodes them into a latent representation. The latent embedding implicitly captures information about the underlying environment dynamics, such as unmodeled physical parameters or external perturbations.

During execution, the adaptation module continuously updates this latent representation from incoming observations, allowing the policy to adjust its actions online as the environment changes. As a result, the learned controller can rapidly adapt to variations in physical parameters without requiring explicit system identification.

Despite its empirical success, rapid motor adaptation typically requires an additional training stage for the student policy, resulting in a more complex training pipeline and potential performance degradation due to imperfect teacher imitation.

2.5 Generative Models for Imitation Learning

Imitation learning aims to learn a policy directly from demonstration data without requiring explicit reward engineering. In this setting, generative models provide a flexible way to represent multimodal action distributions conditioned on observations, which is particularly useful in contact-rich manipulation where multiple action sequences can yield successful outcomes.

2.5.1 Diffusion Models for Policy Learning

Diffusion models learn data distributions through a gradual noising and denoising process (Ho et al., 2020; Song et al., 2021b). Let $\boldsymbol{u}_0 \in \mathbb{R}^{H \times d_u}$ denote a clean action trajectory sampled from expert demonstrations or an offline dataset, where H is the prediction horizon and d_u is the action dimension. A forward diffusion process progressively perturbs $\boldsymbol{u}_0$ by adding Gaussian noise:

$$q(\boldsymbol{u}_k \mid \boldsymbol{u}_{k-1}) = \mathcal{N}\Big(\sqrt{1-\beta_k}\,\boldsymbol{u}_{k-1},\, \beta_k \boldsymbol{I}\Big), \quad k = 1, \ldots, K, \tag{2.29}$$

where $\boldsymbol{u}_k$ denotes the noisy action trajectory at diffusion step k, K is the total number of diffusion steps, and $\{\beta_k\}_{k=1}^{K}$ is a predefined variance schedule. Equivalently, $\boldsymbol{u}_k$ can

be written in closed form as

$$\boldsymbol{u}_k = \alpha_k \boldsymbol{u}_0 + \sigma_k \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{2.30}$$

where

$$\alpha_k = \sqrt{\bar{\alpha}_k}, \qquad \sigma_k = \sqrt{1 - \bar{\alpha}_k}, \qquad \bar{\alpha}_k = \prod_{j=1}^{k} (1 - \beta_j).$$

A neural network parameterized by θ is then trained to approximate the reverse-time denoising process and recover samples from the data distribution.

For conditional policy generation, the denoising model is conditioned on the observation history $\boldsymbol{h}$ and predicts either the injected noise or the clean action trajectory. A common training objective is

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{k, \boldsymbol{u}_0, \boldsymbol{\epsilon}, \boldsymbol{h}} \left[\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\alpha_k \boldsymbol{u}_0 + \sigma_k \boldsymbol{\epsilon}, k | \boldsymbol{h}) \|_2^2 \right], \tag{2.31}$$

where ϵ_θ denotes the neural denoising model with parameters θ. At inference time, action trajectories are generated by initializing from Gaussian noise and iteratively applying the learned denoising model conditioned on $\boldsymbol{h}$.

Diffusion-based imitation policies have shown strong empirical performance in visuomotor imitation learning due to their ability to represent multimodal action distributions and capture complex contact-rich behaviors (Chi et al., 2024; Wang et al., 2022).

2.5.2 Flow Matching for Policy Learning

Flow matching provides an alternative continuous-time formulation for training generative models (Lipman et al., 2023b; Liu et al., 2023). Instead of simulating a stochastic reverse diffusion process, it learns a time-dependent velocity field that transports a simple prior distribution to the target action distribution. Let $\boldsymbol{u}_t$ be a trajectory state at time $t \in [0, 1]$. The generation process is described by an ordinary differential equation (ODE)

$$\frac{d\boldsymbol{u}_t}{dt} = \boldsymbol{v}_\theta(\boldsymbol{u}_t, t | \boldsymbol{h}). \tag{2.32}$$

Using a prescribed probability path, such as linear interpolation between noise $\boldsymbol{u}_0$ and data sample $\boldsymbol{u}_1$,

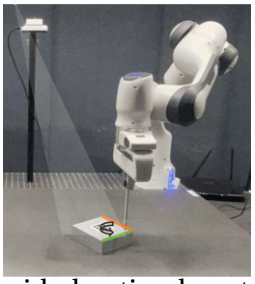
$$\boldsymbol{u}_t = (1 - t)\boldsymbol{u}_0 + t\boldsymbol{u}_1, \quad \dot{\boldsymbol{u}}_t = \boldsymbol{u}_1 - \boldsymbol{u}_0, \tag{2.33}$$

the model is trained by regressing the target velocity field:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t,\boldsymbol{u}_0,\boldsymbol{u}_1,\boldsymbol{h}} \left[\| \boldsymbol{v}_\theta(\boldsymbol{u}_t, t|\boldsymbol{h}) - (\boldsymbol{u}_1 - \boldsymbol{u}_0) \|_2^2 \right]. \tag{2.34}$$

Compared with diffusion sampling, flow matching often enables faster generation because trajectories can be produced with a small number of ODE solver steps while maintaining high sample quality. This advantage is attractive for receding-horizon control, where action generation must run in real time. Recent work has begun applying flow-matching policies to robot manipulation, demonstrating promising performance for contact-rich tasks (Zhang and Gienger, 2024; Black et al., 2024).

3 A Preliminary Study of Contact-rich Manipulation



This chapter presents planar pushing with face-switching as a representative task for studying contact-rich manipulation. The resulting dynamics are highly nonlinear, involving multiple contact modes and uncertain physical parameters, making the task a compact yet expressive testbed for investigating the fundamental complexities of contact-rich interactions. To address these challenges, we develop a demonstration-guided optimal control framework that leverages human demonstrations to guide trajectory optimization and mitigate local minima. The proposed approach is validated through both simulation and real-world experiments on a pusher–slider system using a Franka Emika robot. Beyond demonstrating effective manipulation behaviors, this study highlights inherent computational challenges, thereby motivating the methodologies developed in the subsequent chapters of this dissertation.

Publication Note

The material presented in this chapter is adapted from the following publication.

- Xue, T., Girgin, H., Lembono, T. S., and Calinon, S. (2023b). Demonstration-guided optimal control for long-term non-prehensile planar manipulation. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 4999–5005.

Supplementary materials related to this chapter are available at: https://sites.google.com/view/dg-oc/.

3.1 Introduction

Contact-rich manipulation, which refers to manipulating objects through rich contact interactions with the robot or environment, has attracted significant attention. Such tasks require robots to continuously identify and adapt to contact conditions during interaction. Given the challenges of contact-rich manipulation as described in Chapter 1, it is important to first study a representative contact-rich manipulation task that captures the essential properties of hybrid contact interactions while remaining sufficiently structured for systematic analysis. Non-prehensile manipulation, especially planar pushing, has long been regarded as a canonical benchmark for studying contact-rich manipulation (Mason, 1986, 2001a), as it exhibits underactuated dynamics, hybrid contact modes, and uncertainty of frictional parameters. In this chapter, we introduce a planar pushing task with face-switching mechanism, which allows separation between the pusher and the slider as well as transitions between different contact faces, thereby extending prior settings (Hogan and Rodriguez, 2020a; De Moura et al., 2022) to more complex hybrid dynamics, characterized by the following key properties:

- **Underactuated dynamics**. Due to the unilateral nature of contact and the friction cone constraint, the pusher can only exert a limited range of forces on the slider. This restriction makes it impossible to command arbitrary accelerations, leading to underactuated motion.

- **Hybrid contact modes**. The face-switching pusher–slider system involves multiple interaction modes, including separation, sticking, sliding up, and sliding down, as well as transitions between different contact faces (left, bottom, right, and top). As a result, solving the task requires reasoning over both discrete variables (contact modes and faces) and continuous variables (contact locations).

- **Contact uncertainty**. Frictional interactions between the pusher, the object, and the supporting surface are difficult to model accurately, making it necessary to incorporate feedback mechanisms that enable online adaptation during execution.

The existence of both continuous and discrete variables characterizes the hybrid nature of contact, making it challenging for gradient-based optimization algorithms (De Moura et al., 2022). On the other hand, gradient-free methods like tree search (Doshi et al., 2020) have the ability to perform long-term global planning, but can struggle with the vast number of potential contact modes. Human decision-making, conversely, excels at discrete decisions. Integrating human demonstration into the

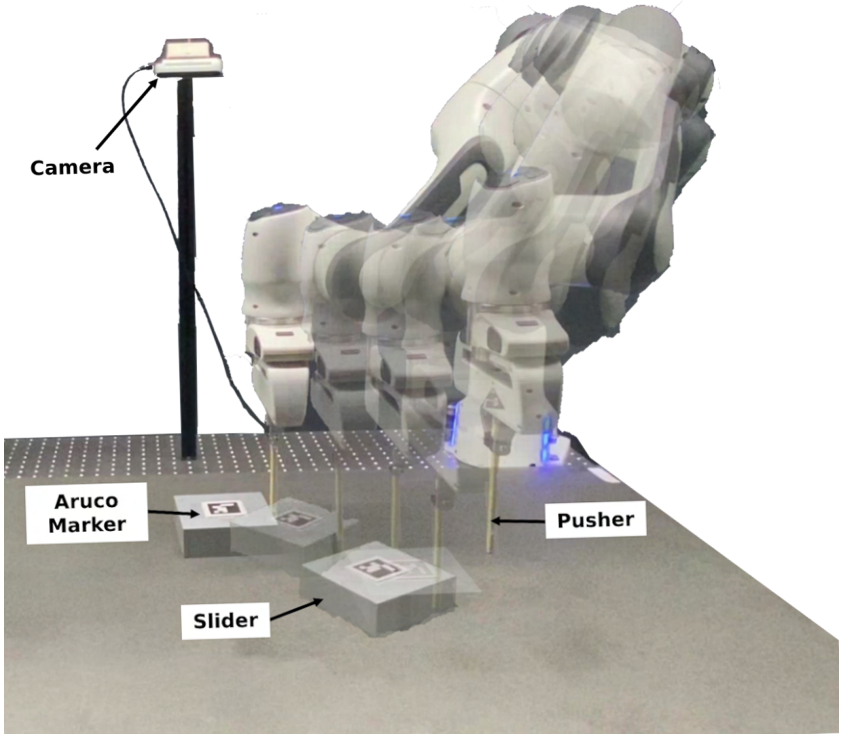


Figure 3.1: System setup where the robot pushes an object with changes of contact points.

hybrid optimization framework could greatly enhance its convergence towards near-optimal solutions.

In this chapter, we propose a hybrid framework that integrates optimization with learning from demonstrations to mitigate the issue of poor local optima, a common challenge in many state-of-the-art solvers. Specifically, we first develop an interface [1] that enables a human to guide the robot in pushing a cube toward the target, where discrete decision variables are implicitly encoded through the continuous motion of the robot end-effector. We then use the collected demonstrations to guide optimization, facilitating efficient exploration of the solution space. In addition, we introduce a real-time feedback controller to account for model mismatch and contact uncertainty during physical interaction.

[1] https://sites.google.com/view/dg-oc/

3.2 Related Work

In this section, we discuss some related work on dynamics modeling, multi-stage motion planning, and robot learning from demonstration.

Non-prehensile manipulation has long been studied as a challenging benchmark for planning and control (Mason, 1999), with the pusher–slider system being a representative example. Under the quasi-static assumption (Goyal et al., 1989) and the limit surface model (Lee and Cutkosky, 1991), early research predominantly focused on offline open-loop planning. Building on friction-cone analysis for sticking and sliding behaviors (Mason, 1986), Lynch *et al.* (Lynch et al., 1992) derived analytical methods for computing object velocities via the motion cone, which was later extended into a geometric path planner to sequence discrete transitions between contact faces (Lynch and Mason, 1996). This framework successfully enables the synthesis of open-loop stable pushing paths with face switching. More recently, Hogan *et al.* shifted the focus to online reactive control within the quasi-static regime, leveraging piecewise affine dynamics to actively control both sticking and sliding behaviors (Hogan and Rodriguez, 2020a; Hogan et al., 2018). While robust, their framework restricts the pusher to a single pre-designated face, losing the multi-face mobility explored in earlier geometric planners. In this chapter, we bridge this gap by extending Hogan *et al.*'s framework into a unified quasi-static optimal control formulation. We model face-switching and intermittent contact as an integrated hybrid system, allowing the robot to reason about sliding control and multi-face transitions simultaneously.

Deploying such a hybrid formulation for long-horizon planar pushing requires simultaneously reasoning over discrete interaction modes and geometric variables, while seamlessly integrating planning with feedback control. For example, Hogan *et al.* (Hogan and Rodriguez, 2020a) incorporated interaction mode selection into a model predictive control (MPC) framework using mixed-integer programming (MIP). While this formulation jointly optimizes discrete and continuous variables, the non-convexity introduced by integer variables leads to high computational cost, making it difficult to deploy in high-frequency feedback control. To mitigate this issue, an offline classifier has been proposed to predict mode sequences (Hogan et al., 2018), though the required training data grows rapidly as the number of interaction modes and contact faces increases. Moreover, these approaches rely on short-horizon MPC, limiting their ability to reason over long-horizon manipulation tasks involving separation modes or face switching. Alternatively, the problem can be formulated as a mathematical program with complementarity constraints (MPCC) (De Moura et al., 2022), which improves convergence for planning and disturbance recovery but remains susceptible to poor local optima due to smoothing and relaxation.

Another line of work addresses the combinatorial structure of hybrid manipulation through search-based planning, in which symbolic action sequences are explored while geometric variables are optimized (Simeonov et al., 2021; Toussaint, 2015; Migimatsu and Bohg, 2020). In the context of non-prehensile manipulation, a fixed-depth search tree has been used to determine hybrid switches, together with a hybrid Differential Dynamic Programming (DDP) solver for continuous optimization (Doshi et al., 2020). Similarly, Cheng et al. (2023) used Monte Carlo Tree Search to construct a hierarchical contact exploration framework that decomposes planning into multiple abstraction levels, enabling more global reasoning over the contact space without enumerating all mode sequences upfront. However, such search-based approaches are vulnerable to the exponential growth of contact modes, leading to substantial computational complexity and limiting their applicability to real-time feedback control.

In contrast to purely optimization-based approaches, humans often rely on experience to efficiently select promising strategies, particularly for discrete decisions. Learning from Demonstration (LfD) has therefore been widely studied as a way to leverage human expertise for robot motion generation (Billard et al., 2016; Calinon and Lee, 2019). LfD methods aim to extract motion patterns from demonstrations and generalize them to new situations. Various techniques have been proposed to encode demonstrations, including Dynamic Movement Primitives (DMP) (Ijspeert et al., 2013), Gaussian Mixture Regression (GMR) (Calinon and Lee, 2019), and task-parameterized probabilistic models (Calinon, 2018). Among these methods, the K-nearest neighbor (k-NN) algorithm (Peterson, 2009) provides a simple yet effective approach for discrete decision classification (Cunningham and Delany, 2021).

3.3 Dynamical System

In this section, we describe the hybrid dynamics of the pusher–slider system. We first introduce the kinematic model (Sec. 3.3.1), and then extend the motion cone (Sec. 3.3.2) and motion equations (Sec. 3.3.3) to incorporate separation mode and face-switching mechanism.

3.3.1 Kinematics

Fig. 3.2 illustrates the kinematics of the pusher–slider system. The red and blue circles denote the pusher at the initial contact face and after face switching, respectively. The pose of the slider is defined as $\boldsymbol{q}_s = [x\ y\ \theta]^\top$ with respect to the global frame $\mathbb{F}_g$, where x and y represent the location of the center of mass, and θ denotes the rotation about the vertical axis. The contact position between the pusher and the slider is represented

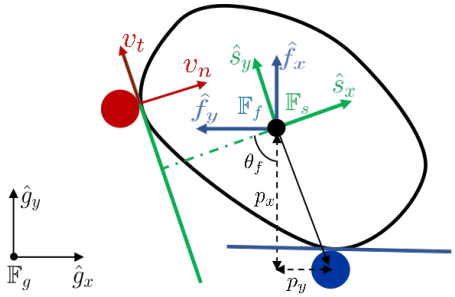


Figure 3.2: Kinematics of pusher-slider system allowing face switching.

as $\boldsymbol{q}_{p_f} = [p_x\ p_y]^\top$ in the current slider frame $\mathbb{F}_f$ after face switching. Its expression in the initial slider frame $\mathbb{F}_s$ is given by $\boldsymbol{q}_{p_s} = \boldsymbol{R}_{\theta_f}\boldsymbol{q}_{p_f}$, where θ_f denotes the rotation from $\mathbb{F}_s$ to $\mathbb{F}_f$. The system input is the pusher velocity $\boldsymbol{u} = [v_n\ v_t]^\top$ defined in $\mathbb{F}_s$. The corresponding velocity in frame $\mathbb{F}_f$ is given by $\boldsymbol{u}_f = \boldsymbol{R}_{\theta_f}^\top \boldsymbol{u} = [v_{nf}\ v_{tf}]^\top$.

3.3.2 Generalized Motion Cone

Based on the quasi-static approximation and ellipsoidal limit surface assumption (Hogan and Rodriguez, 2020a; Goyal et al., 1989; Lee and Cutkosky, 1991; Mason, 1986), a motion cone is introduced to determine the contact mode given by the current pusher velocity and contact position. The two boundaries of the motion cone are given as $\boldsymbol{v}_{up}^{\mathcal{MC}} = 1\boldsymbol{f_x} + \gamma_{up}\boldsymbol{f_y}$ and $\boldsymbol{v}_{dn}^{\mathcal{MC}} = 1\boldsymbol{f_x} + \gamma_{dn}\boldsymbol{f_y}$, resolved in the current slider frame $\mathbb{F}_f$, with

$$\gamma_{up} = \frac{\mu_p c^2 - p_x p_y + \mu_p {p_x}^2}{c^2 + {p_y}^2 - \mu_p p_x p_y}, \tag{3.1}$$

$$\gamma_{dn} = \frac{-\mu_p c^2 - p_x p_y - \mu_p {p_x}^2}{c^2 + {p_y}^2 + \mu_p p_x p_y}, \tag{3.2}$$

where μ_p is the friction coefficient between the pusher and the slider, and c is the parameter connecting applied force and the resulting velocity. More details are introduced in (Hogan and Rodriguez, 2020a).

In the previous work (Hogan and Rodriguez, 2020a; Lynch et al., 1992), the pusher is assumed to remain on one fixed contact face during the execution. We would like to enable more complex pushing strategies by involving face switching. For that, we introduce another interaction mode, which we call separation mode. Table 3.1 lists the relationship between state constraints and interaction modes, where Ω is the set within the boundaries of the motion cone, ϕ is the space where the pusher goes towards the slider, and ψ is the set where the pusher keeps touching with the slider. r_s and r_p are the half length of the slider and the radius of the pusher, respectively. Figure 3.3 illustrates

Table 3.1: Constraints of different interaction modes.

Sticking	**Sliding Up**	**Sliding Down**	**Separation**
$\boldsymbol{u}_f \in \Omega,\ \boldsymbol{q}_{p_f} \in \psi$	$\boldsymbol{u}_f \in \phi \setminus \Omega,\ \boldsymbol{q}_{p_f} \in \psi$	$\boldsymbol{u}_f \in \phi \setminus \Omega,\ \boldsymbol{q}_{p_f} \in \psi$	$\boldsymbol{u}_f \notin \phi \ \vee\ \boldsymbol{q}_{p_f} \notin \psi$
$\gamma_{dn} \leq \frac{v_{tf}}{v_{nf}} \leq \gamma_{up}$, $v_{nf} > 0$, $\lvert p_x \rvert = r_s + r_p$, $\lvert p_y \rvert \leq r_s$	$\frac{v_{tf}}{v_{nf}} > \gamma_{up}$, $v_{nf} > 0$, $\lvert p_x \rvert = r_s + r_p$, $\lvert p_y \rvert \leq r_s$	$\frac{v_{tf}}{v_{nf}} < \gamma_{dn}$, $v_{nf} > 0$, $\lvert p_x \rvert = r_s + r_p$, $\lvert p_y \rvert \leq r_s$	$v_{nf} < 0 \vee$ $\lvert p_x \rvert > r_s + r_p \vee$ $\lvert p_y \rvert > r_s$

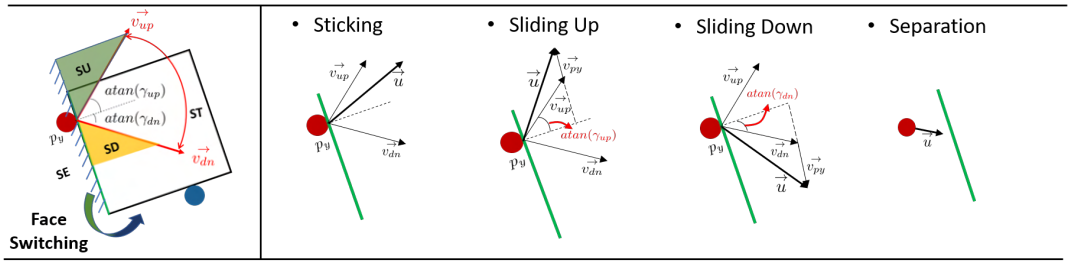


Figure 3.3: Hybrid contact modes in planar pushing. The task involves multiple contact modes (sticking, sliding, and separation) and transitions between different contact faces.

the multiple contact modes involved in the planar pushing task with face switching.

3.3.3 Hybrid Contact Dynamics

We build the generalized motion equation on the basis of (Lynch et al., 1992) and (Hogan and Rodriguez, 2020a), by including separation mode and contact face switching, namely

$$\dot{\boldsymbol{x}} = \begin{cases} \boldsymbol{g}_1(\boldsymbol{x}, \boldsymbol{u}), & \text{if } \textbf{Sticking}, \\ \boldsymbol{g}_2(\boldsymbol{x}, \boldsymbol{u}), & \text{if } \textbf{Sliding Up}, \\ \boldsymbol{g}_3(\boldsymbol{x}, \boldsymbol{u}), & \text{if } \textbf{Sliding Down}, \\ \boldsymbol{g}_4(\boldsymbol{x}, \boldsymbol{u}), & \text{if } \textbf{Separation}, \end{cases} \tag{3.3}$$

where $\boldsymbol{x} = [\boldsymbol{q}_s^\top\ \boldsymbol{q}_{p_s}^\top]^\top$, and

$$\boldsymbol{g}_j(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \boldsymbol{R}_{\theta_f} \boldsymbol{R}_\theta \boldsymbol{Q} \boldsymbol{P}_j \\ \boldsymbol{b}_j \\ \boldsymbol{R}_{\theta_f} [\boldsymbol{d}_j \quad \boldsymbol{c}_j]^\top \end{bmatrix} \boldsymbol{R}_{\theta_f}^\top \boldsymbol{u}, \quad \boldsymbol{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

$$\boldsymbol{Q} = \frac{1}{c^2 + p_x^2 + p_y^2} \begin{bmatrix} c^2 + {p_x}^2 & p_x p_y \\ p_x p_y & c^2 + p_y^2 \end{bmatrix},$$

$$\boldsymbol{b}_1 = \begin{bmatrix} \frac{-p_y}{c^2+p_x^2+p_y^2} & \frac{p_x}{c^2+p_x^2+p_y^2} \end{bmatrix}, \boldsymbol{b}_2 = \begin{bmatrix} \frac{-p_y+\gamma_{up}p_x}{c^2+p_x^2+p_y^2} & 0 \end{bmatrix},$$

$$\boldsymbol{b}_3 = \begin{bmatrix} \frac{-p_y+\gamma_{dn}p_x}{c^2+p_x^2+p_y^2} & 0 \end{bmatrix}, \quad \boldsymbol{b}_4 = [0 \quad 0],$$

$$\boldsymbol{c}_1 = [0 \quad 0], \quad \boldsymbol{c}_2 = [-\gamma_{up} \quad 1], \boldsymbol{c}_3 = [-\gamma_{dn} \quad 1], \boldsymbol{c}_4 = [0 \quad 1],$$

$$\boldsymbol{d}_1 = [0 \quad 0], \quad \boldsymbol{d}_2 = [0 \quad 0], \quad \boldsymbol{d}_3 = [0 \quad 0], \quad \boldsymbol{d}_4 = [1 \quad 0],$$

$$\boldsymbol{P}_1 = \boldsymbol{I}_{2\times 2}, \quad \boldsymbol{P}_2 = \begin{bmatrix} 1 & 0 \\ \gamma_{up} & 0 \end{bmatrix}, \boldsymbol{P}_3 = \begin{bmatrix} 1 & 0 \\ \gamma_{dn} & 0 \end{bmatrix}, \quad \boldsymbol{P}_4 = \boldsymbol{0}_{2\times 2},$$

where $j = 1, 2, 3, 4$ corresponds to sticking, sliding up, sliding down, and separation mode, respectively.

3.4 Methodology

3.4.1 Problem Formulation

The planar pushing task can be formulated as an optimal control problem (OCP), namely

$$\begin{aligned} \min_{\boldsymbol{u}_t} \quad & c_T(\boldsymbol{x}_T) + \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \\ \text{s.t.} \quad & \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t), \end{aligned} \tag{3.4}$$

where f denotes the discrete-time dynamics derived from Eq. (3.3), and $\boldsymbol{x}_t, \boldsymbol{u}_t$ denote the state and control input at time step t, respectively.

We employ Differential Dynamic Programming (DDP) (Mayne, 1966) to solve this problem, resulting in a local stabilizing controller of the form

$$\boldsymbol{u}_t = \hat{\boldsymbol{u}}_t + \boldsymbol{K}_t (\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t), \tag{3.5}$$

where $\boldsymbol{K}_t$ is the feedback gain, and $\hat{\boldsymbol{u}}_t$ is the feedforward term.

Due to the non-convex nature of Eq. (3.4), DDP employs an iterative optimization scheme centered around the current candidate solution. However, its convergence is highly sensitive to the initial guess, making the algorithm susceptible to poor local optima when initialized far from the global minimum.

3.4.2 Demonstration-guided DDP

To avoid falling into poor local optima, we use human demonstrations to warm-start the solver. The demonstrations are represented as continuous variables, which implicitly encode discrete decisions, rather than explicitly specifying mode sequences as in prior work (Hogan and Rodriguez, 2020a; De Moura et al., 2022; Hogan et al., 2018). In this way, the joint logical and geometric optimization problem can be reformulated as a continuous optimization problem that can be solved more efficiently.

The collected human demonstrations are denoted as $[\widetilde{\boldsymbol{q}}_s, \widetilde{\boldsymbol{q}}_{p_s}, \widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{u}}]$, with $\widetilde{\boldsymbol{q}}_s = [\widetilde{\boldsymbol{q}}_{\boldsymbol{s_0}}, \widetilde{\boldsymbol{q}}_{\boldsymbol{s_1}}, \cdots, \widetilde{\boldsymbol{q}}_{\boldsymbol{s_T}}]$, $\widetilde{\boldsymbol{q}}_{p_s} = [\widetilde{\boldsymbol{q}}_{\boldsymbol{p_{s_0}}}, \widetilde{\boldsymbol{q}}_{\boldsymbol{p_{s_1}}}, \cdots, \widetilde{\boldsymbol{q}}_{\boldsymbol{p_{s_T}}}]$, where $\widetilde{\boldsymbol{q}}_{s_t} \in \mathbb{R}^3$ and $\widetilde{\boldsymbol{q}}_{p_{s_t}} \in \mathbb{R}^2$ represent the state of the slider and the pusher at timestep t, respectively. $\widetilde{\boldsymbol{u}} = [\widetilde{\boldsymbol{u}}_0, \widetilde{\boldsymbol{u}}_1, \cdots, \widetilde{\boldsymbol{u}}_{T-1}] \in \mathbb{R}^2$ denotes the velocity at each timestep.

Given a target $\boldsymbol{q}_s^*$, we use k-NN to select the index of j^* with the closest demonstration $\widetilde{\boldsymbol{q}}_{s_T}$ to the target in the task space by evaluating

$$j^* = \underset{j \in \mathbb{S}}{\arg\min} \quad \text{dist}(\widetilde{\boldsymbol{q}}_{s_T}^j, \boldsymbol{q}_s^*), \tag{3.6}$$

where $\mathbb{S} = \{j : j \in \{0, 1, \cdots, n_d - 1\}\}$, n_d is the number of demonstrations, and

$$\text{dist}(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{r=1}^{d} |\boldsymbol{x}_r - \boldsymbol{y}_r|^2 \right)^{1/2}, \tag{3.7}$$

where d is the dimension of slider state.

The cost function is defined as

$$c_1 = c_{re} + c_{rg} + c_{bd}, \tag{3.8}$$

with

$$c_{re} = (\boldsymbol{\mu}_T - \boldsymbol{x}_T)^\top \boldsymbol{Q}_T (\boldsymbol{\mu}_T - \boldsymbol{x}_T), \quad c_{rg} = \sum_{t=0}^{T-1} \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t,$$
$$c_{bd} = \sum_{t=0}^{T-1} \boldsymbol{f}^{\text{cut}}(\boldsymbol{u}_t, \boldsymbol{u}_l)^\top \boldsymbol{Q}_f \boldsymbol{f}^{\text{cut}}(\boldsymbol{u}_t, \boldsymbol{u}_l),$$

where c_{re} is the reaching cost, and c_{rg}, c_{bd} are the regularizer and boundary penalizer of control commands. $\boldsymbol{u}_l$ is the predefined bounding box of $\boldsymbol{u}$. $\boldsymbol{f}^{\text{cut}}$ is a soft-thresholding function. The initial guess $\boldsymbol{u}^0 = \widetilde{\boldsymbol{u}}$ is drawn from human demonstrations directly.

We name this method Demonstration-started DDP (DS-DDP). Although it appears to

be an effective warm-starting approach, its performance is limited by the number of available demonstrations. To alleviate this limitation, we propose to use demonstrations as soft constraints, achieved by designing the cost function as

$$c_2 = c_{re} + c_{rg} + c_{bd} + c_{sw} + c_{ve}, \tag{3.9}$$

with

$$c_{sw} = \sum_{n=t_0}^{t_{N-1}} (\boldsymbol{\mu}_n - \boldsymbol{x}_n)^\top \boldsymbol{Q}_n (\boldsymbol{\mu}_n - \boldsymbol{x}_n),$$
$$c_{ve} = \sum_{t=0}^{T-1} (\widetilde{\boldsymbol{u}}_t - \boldsymbol{u}_t)^\top \boldsymbol{R}_{dv} (\widetilde{\boldsymbol{u}}_t - \boldsymbol{u}_t),$$

where c_{sw} and c_{ve} are designed to follow the demonstrated face switching strategy and pusher velocity. $n = [t_0, \ldots, t_{N-1}]$ is the timestep when the contact face switches, $\boldsymbol{\mu} = [\widetilde{\boldsymbol{q}}_s^\top \ \widetilde{\boldsymbol{q}}_{p_s}^\top]^\top$ is the state of the selected demonstration, and $\widetilde{\boldsymbol{u}}_t$ is the demonstrated velocity at timestep t.

We refer to this approach as Demonstration-penalized DDP (DP-DDP), which shows good performance and can avoid poor local optima, but in order to further improve its convergence properties, we propose a hierarchical optimization framework, where the solution of DP-DDP is used to initialize another DDP problem that we call Warm-starting DDP (WS-DDP).

The cost function of WS-DDP is the same as that of DS-DDP (3.8), allowing it to explore more freely toward the final target. The initial guess $\boldsymbol{u}^0$ is given by the solution of the previous DP-DDP, namely $\boldsymbol{u}^0 = \boldsymbol{u}^*_{DP}$.

3.4.3 Adaptation to Disturbance

Solving the demonstration-guided DDP yields a feedback controller (3.5), which generates control commands at each time step based on the current state to stabilize the motion along the nominal trajectory. However, the friction and other unmodeled nonlinearities might cause undesired behaviors, especially when the error between the current state and the planned solution $\|\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t\|^2$ is too big. To alleviate this issue, inspired by (Girgin et al., 2022), we propose to use an error filtering method based on a trust region defined as a ball of radius r as $\mathcal{B}_t(r) = \{\boldsymbol{x} \in \mathbb{R}^n | \|\boldsymbol{x} - \hat{\boldsymbol{x}}_t\| < r\}$. By denoting the actual robot state as $\boldsymbol{x}_t^0$, we filter the $\boldsymbol{x}_t$ in (3.5) as follows: if $\boldsymbol{x}_t^0 \in \mathcal{B}_t(r)$, then we take $\boldsymbol{x}_t = \hat{\boldsymbol{x}}_t$, else we take $\boldsymbol{x}_t = \boldsymbol{x}_t^0$. This means that if the error between the actual robot state and the planned state is small, we can use the feedforward terms of the controller directly, and if it is big, then we regenerate the trajectory using the

feedback controller gains. This allows us to fully exploit the feedback and feedforward gains computed offline during the online execution of the task avoiding expensive recomputations at each timestep.

3.5 Experiments

We evaluate in this section the proposed offline programming method (Sec. 3.5.1) and the online tracking controller (Sec. 3.5.2).

3.5.1 Offline Planning

In this work, the task space is the horizontal plane on the table, defined as

$$\{(x, y, \theta) \mid x \in [-25\,\text{cm},\ 25\,\text{cm}], y \in [-25\,\text{cm},\ 25\,\text{cm}], \theta \in [-\pi, \pi]\}.$$

We collected three representative demonstrations

$$[15\,\text{cm}, -10\,\text{cm}, -\pi/2], \quad [0, -20\,\text{cm}, \pi/2], \quad [15\,\text{cm}, -15\,\text{cm}, \pi/2],$$

corresponding to $N_s = 0$, $N_s = 1$, and $N_s = 2$, respectively, where N_s denotes the number of face switches during the demonstration.

The initial state is defined as $[0,\ 0,\ 0,\ \alpha p_x,\ 0,\ 0,\ 0]^\top$, where $\alpha = 1.3$, corresponding to the separation mode in Sec. 3.3.2, allowing the pusher to select the contact point at the beginning. The cost function gains are set to $\boldsymbol{Q}_T = 10^6 \times \text{diag}\{1, 1, 1, 10^{6p-5}, 10^{1-6p}, 10^{-3}, 10^{-3}\}$, $\boldsymbol{Q}_n = 10^6 \times \text{diag}\{10^{-3}, 10^{-3}, 10^{-3}, p, (1-p), 0, 0\}$, where $p = 1$ when $\theta_f = 0$ or $\theta_f = \pi$, otherwise $p = 0$. $\boldsymbol{R} = \boldsymbol{K} = \text{diag}\{1, 1\}$, $\boldsymbol{R}_{du} = \boldsymbol{R}_{dv} = \text{diag}\{100, 100\}$.

We first randomly selected 100 targets from the task space to test our proposed framework, along with two benchmarks: Zero-started DDP (ZS-DDP), and Demonstration-started DDP (DS-DDP). Successful offline programming is defined as achieving $\{x_{\text{err}} < 1\text{cm},\ y_{\text{err}} < 1\text{cm},\ \theta_{\text{err}} < 5^\circ\}$, where x_{err}, y_{err}, and θ_{err} are the differences between the final and target states. We list the statistical results in Table 3.2. The success rate of ZS-DDP is about 40%, indicating many poor local minima that trap ZS-DDP in bad regions. Initializing with demonstrations increases the success rate by nearly 20%, showing the benefits of using human demonstrations to avoid poor local minima. Our proposed demonstration-guided hierarchical framework, composed of DP-DDP and WS-DDP, achieves a significantly higher success rate (85%) by using demonstrations as soft constraints and warm-starting a new DDP. We consider using demonstration as

Table 3.2: Performance of ZS-DDP, DS-DDP, DP-DDP, and WS-DDP for offline programming.

Method	x_{err} (cm)	y_{err} (cm)	θ_{err} (rad)	succ_rate
ZS-DDP	3.18 ± 8.30	4.78 ± 8.60	0.04 ± 0.39	40%
DS-DDP	1.70 ± 9.07	1.62 ± 9.66	0.21 ± 1.82	58%
DP-DDP	0.14 ± 1.54	0.57 ± 2.77	0.22 ± 0.12	77%
WS-DDP	$\mathbf{0.07 \pm 1.31}$	$\mathbf{0.19 \pm 2.90}$	$\mathbf{0.01 \pm 0.08}$	**85%**

soft constraints, rather than initialization, as the key feature of our proposed method. This provides the nonlinear optimization process with a few but valuable human guidance, leading to a more efficient exploration process. WS-DDP is then warm-started by DP-DDP, yielding a much smoother and more precise optimal solution.

To demonstrate the strengths of our method, we compared it with the Mixed-Integer Programming (MIP) formulation, which was commonly used for planar pushing tasks previously (Hogan and Rodriguez, 2020a; De Moura et al., 2022; Hogan et al., 2018). To ensure a fair comparison, we used demonstrations as initialization for MIP as well, which we refer to as DS-MIP. We used CasADi (Andersson et al., 2019) with Bonmin solver (Bonami et al., 2012) and Crocoddyl (Mastalli et al., 2020) with FDDP solver to solve DS-MIP and DS-DDP, respectively. Convergence and computation time of 10 randomly selected target configurations are shown in Fig. 3.4 and Fig. 3.5. The vertical axis of Fig. 3.4 represents the reaching error, in meters for x_e and y_e and in radians for θ_e, between the final state $\boldsymbol{x}_T$ and the target $\boldsymbol{\mu}_T$. DS-MIP achieves performance comparable to DS-DDP (Fig. 3.4), but requires approximately ten times longer computation time (Fig. 3.5). Furthermore, our experiments show that the proposed DP-DDP and WS-DDP algorithms converge to better solutions than DS-DDP, albeit with moderately increased computation time that remains within acceptable limits. In contrast, DS-DDP tends to converge to poor local optima, and increasing the number of iterations does not improve its performance. These results suggest that relying solely on demonstrations for initialization is insufficient; instead, incorporating demonstrations as soft constraints to guide the search can yield improved solutions.

3.5.2 Online Tracking

For online tracking, we conducted both numerical simulations and real-world robot experiments. In simulation, a constant state disturbance $\boldsymbol{x} = \overline{\boldsymbol{x}} + \boldsymbol{\epsilon}$ was introduced, where the components of $\boldsymbol{\epsilon} = [\epsilon_x, \epsilon_y, \epsilon_\theta]^\top$ were drawn from uniform distributions $\epsilon_x \sim \mathbb{U}(-x_M, x_M)$, $\epsilon_y \sim \mathbb{U}(-y_M, y_M)$, and $\epsilon_\theta \sim \mathbb{U}(-\theta_M, \theta_M)$. Fig. 3.6 illustrates the

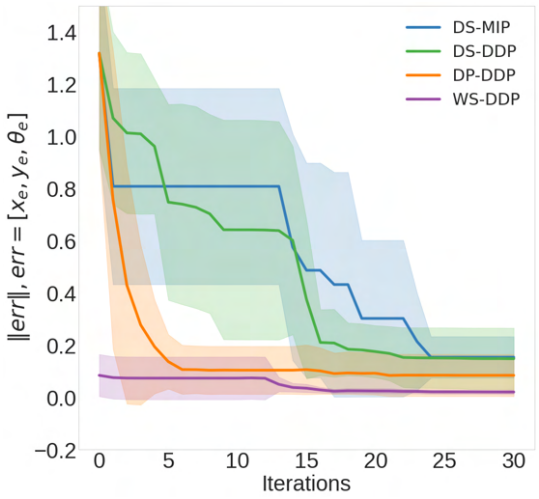


Figure 3.4: Optimization convergence

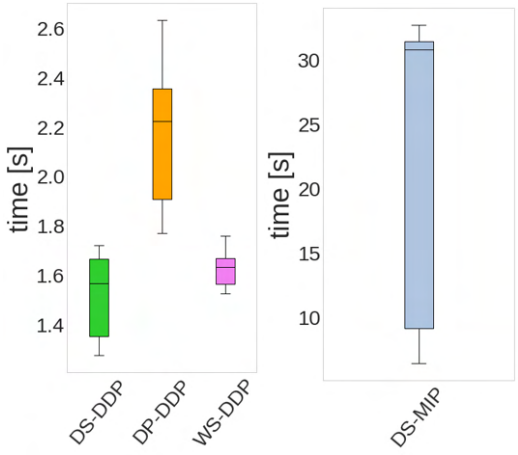


Figure 3.5: Computation time

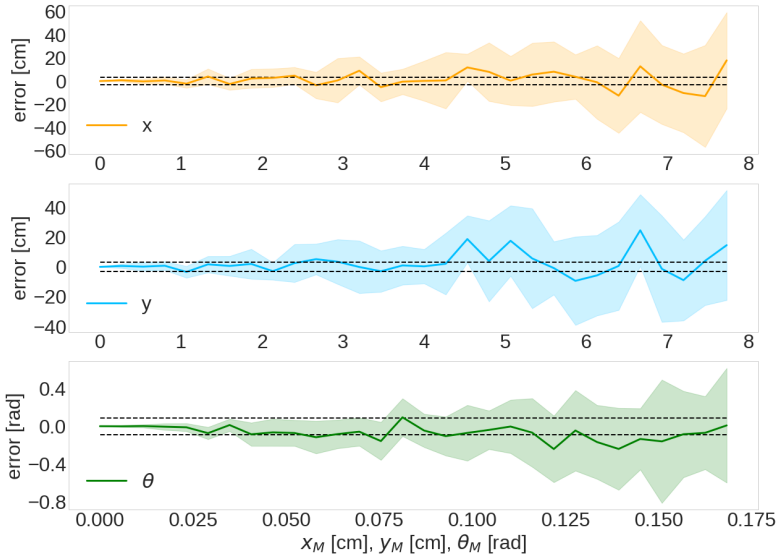
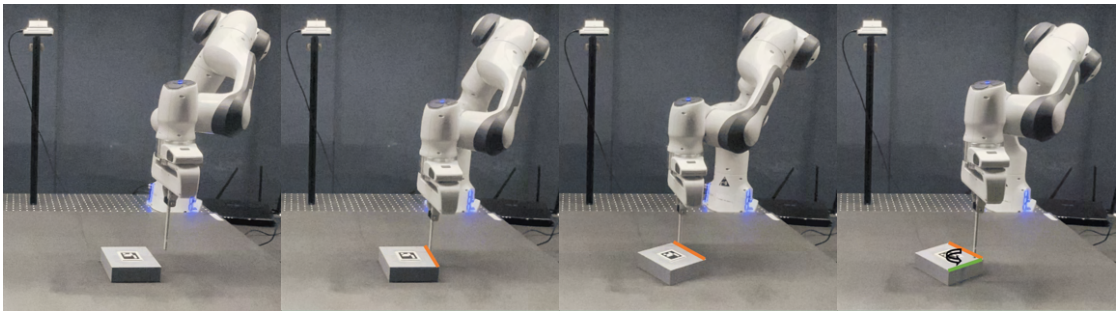


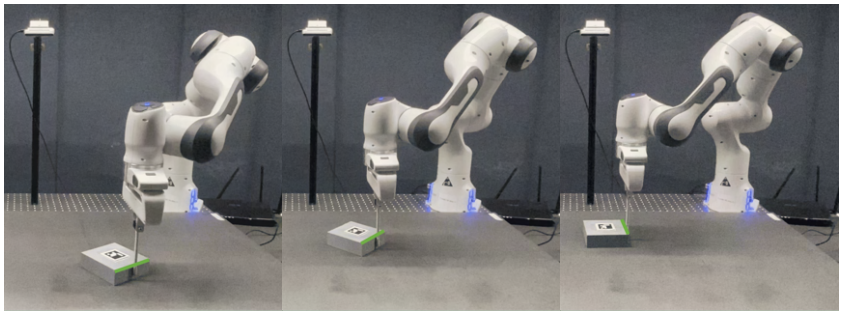
Figure 3.6: Tracking performance under disturbances drawn from uniform distributions, where x_M, y_M and θ_M are the boundaries. The dashed lines present the tolerance.

evolution of the final errors in x, y, and θ as the disturbance magnitudes x_M, y_M, and θ_M increase. The success criteria for online tracking were defined as $|x_{\text{err}}| < 3$ cm, $|y_{\text{err}}| < 3$ cm, and $|\theta_{\text{err}}| < 5^\circ$ (0.087 rad). The controller successfully maintained stability under perturbations of up to 4 cm in x and y, and 0.117 rad in θ.

Then, we tested the proposed method on the real robot setup (Fig. 3.1), using a 7-axis Franka Emika robot and a RealSense D435 camera. The prismatic slider, a 12×12 cm block made of 3D-printed PLA and weighing 110 g, was positioned on a flat plywood

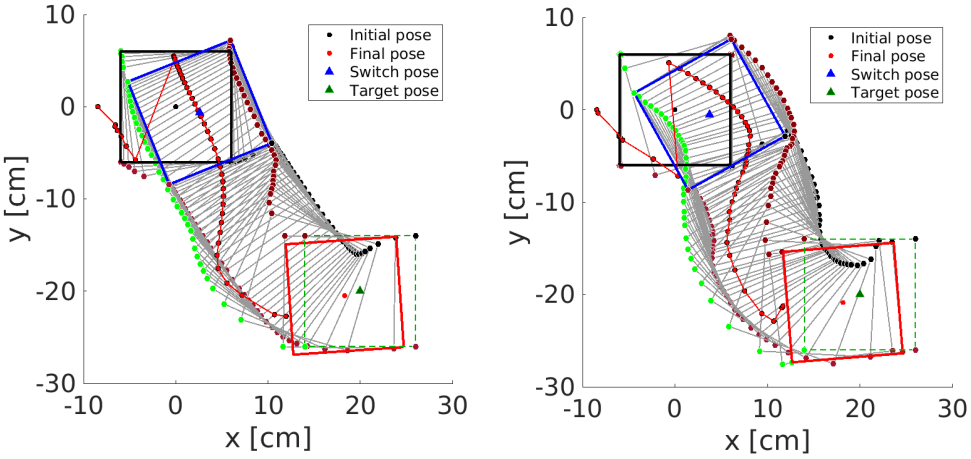


(a) Initialization (b) Contact (c) Pushing (d) Face switching



(e) Contact (f) Pushing (g) Reaching

Figure 3.7: Keyframes of real-world planar pushing task with face switching. The manipulator starts from (a) and selects an optimal face (orange) and contact point (b) for pushing, until reaching the planned face-switching point (c). Next, the manipulator changes to another face (d) and touches the object again at the selected point (e), followed by the next phase of pushing (f), until reaching the final target (g). This example is for $N_s = 1$. (d)$\sim$(f) should be repeated if $N_s > 1$. The colored lines are used to express the current active faces, and the black arrow in (d) represents the face-switching process.



a) Simulation results b) Experimental results

Figure 3.8: Trajectories of the pusher and slider in simulation and real-world experiments. Both successfully reach their targets within tolerance.

surface with an ArUco marker attached to its top face. The friction coefficients between the pusher and the slider, and between the plywood surface and the slider, are 0.3 and 0.35, respectively. The robot was equipped with a wooden pusher that had a 0.5 cm radius and was used to move the slider. The camera captured the motion of the slider at 30 Hz, while the feedback controller ran at 100 Hz. The low-level Cartesian impedance controller, responsible for actuating the robot, ran at 1000 Hz.

As reported in (Doshi et al., 2020), most target configurations can be achieved within one to two face switches. We therefore validated our approach on an externally disturbed line-tracking task and two face-switching tasks. The trajectories in Fig. 3.8 correspond to a task requiring a single face switch to push the object from $[0, 0, 0]$ to $[20\text{ cm}, -20\text{ cm}, \pi/2]$. The resulting control strategy is intuitive: the robot pushes the left face to adjust the pose before transitioning to the top face for the subsequent phase. Fig. 3.8(a) illustrates the simulation trajectory generated using PyBullet (Coumans and Bai, 2016), while Fig. 3.8(b) shows the corresponding real-world execution. Although these trajectories differ significantly due to discrepancies in friction parameters between the simulated and real environments, both successfully overcome these uncertainties to reach the target. Despite unstructured elements—such as visual system noise, robot controller latency, modeling assumptions, and unmeasurable friction—the feedback controller effectively compensates for these model mismatches to track the reference trajectory. Fig. 3.7 presents keyframes of the task, showing that seven steps are required under the face-switching strategy. An additional case with a target of $[5\text{ cm}, -18\text{ cm}, \pi/5]$, requiring two face switches, is demonstrated in the accompanying

video.

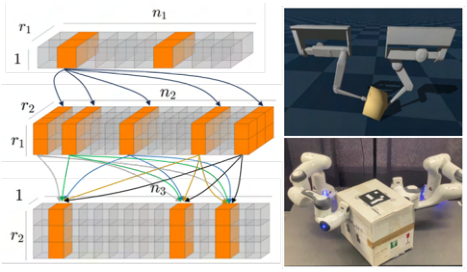
3.6 Discussion

In this chapter, we introduced an extended planar pushing task with face-switching mechanism as a representative example of contact-rich manipulation. This example highlights the key challenges arising from hybrid interaction modes, underactuated dynamics, and joint reasoning over discrete and continuous decisions.

To address these challenges, we proposed a demonstration-guided optimal control framework that incorporates human demonstrations to guide the optimization process. By leveraging a small number of demonstrations, the originally multi-stage motion planning problem can be reformulated as a continuous, warm-started optimization problem, significantly improving computational efficiency. Experimental results in both simulation and real-robot experiments demonstrate that the proposed framework can generate feasible manipulation strategies and maintain robust performance under contact uncertainty.

However, the effectiveness of the demonstration-guided optimization still depends heavily on the quality of the provided demonstrations and the resulting initial guesses. When demonstrations are suboptimal or incomplete, the optimizer may still converge to undesirable solutions. This limitation highlights the inherent difficulty of solving contact-rich manipulation problems using generic optimization alone and motivates the development of approaches that exploit the underlying structure of objective functions for efficient decision making, as explored in the subsequent chapters.

4 Non-convex Contact-rich Optimization using Tensor Train



Many contact-rich manipulation tasks can be naturally formulated as optimization problems, but solving them requires navigating non-smooth contact dynamics and hybrid contact modes, resulting in highly non-convex, multi-modal objective landscapes. While Monte Carlo Tree Search (MCTS) provides a powerful gradient-free strategy for exploring such challenging solution spaces, its scalability to robotics is severely limited by the curse of dimensionality. To address this issue, this chapter introduces Tensor Train Tree Search (TTTS), which represents decision trees in Tensor Train format to obtain a compact approximation with linear complexity. This enables efficient optimization over non-convex landscapes while substantially reducing storage requirements. We validate TTTS on diverse robotic tasks, showcasing its effectiveness in solving highly non-convex optimization problems arising in contact-rich manipulation.

Publication Note

The material presented in this chapter is adapted from:

- Xue, T., Zhang, Y., Razmjoo, A., and Calinon, S. (2025c). Monte carlo tree search with tensor factorization for robot optimization. *under review, available as arXiv:2507.04949*.

Supplementary materials including videos and source code related to this chapter are available at: https://sites.google.com/view/tt-ts/.

4.1 Introduction

Optimization plays a critical role in robotics and serves as the foundation for a wide range of tasks, including inverse kinematics (Goldenberg et al., 2003), obstacle avoidance (Marcucci et al., 2023), multi-stage motion planning (Toussaint, 2015), and contact-rich manipulation (Mason, 1986, 1999), see Figure 4.1. Among these problems, contact-rich manipulation represents a particularly challenging and representative setting, as it simultaneously embodies several core difficulties in robot optimization. It involves nonlinear dynamics, hybrid contact modes, and uncertainty in physical parameters, all of which are tightly coupled and must be addressed jointly, making it significantly more challenging than addressing each aspect in isolation. Specifically, the nonlinearity renders many of these problems non-convex, making gradient-based methods prone to getting trapped in poor local optima (Lembono et al., 2020). Similarly, the need to jointly optimize over discrete modes and continuous motion introduces significant combinatorial complexity, substantially increasing computational costs and slowing convergence for both sampling-based and search-based approaches. Another key challenge is multimodal solution discovery, where multiple distinct feasible solutions may exist due to task redundancies or environmental symmetries. Identifying and reasoning over such diverse solutions is essential for both robustness and global optimization.

To cope with these challenges, existing approaches typically adopt a decomposed strategy, in which different aspects of the problem are addressed using specialized methods. Nonlinear dynamics are often handled through local convexification in trajectory optimization (Wang and Grant, 2017; Malyuta et al., 2022), hybrid discrete–continuous decisions are managed via sampling- or search-based methods for mode exploration (LaValle, 2006; Kavraki et al., 1996; Hart et al., 1968), and uncertainty in contact interactions is commonly addressed using receding-horizon control schemes such as model predictive control (MPC) (Morari and Lee, 1999; Jones and Morari, 2010; Zeilinger et al., 2014), which enable reactive replanning under model mismatch and disturbances. These approaches have proven effective in addressing their respective challenges.

Efforts have also been made toward more integrated formulations, such as task and motion planning (TAMP) (Garrett et al., 2021), which aim to couple discrete task planning with continuous motion generation. However, existing approaches often struggle to scale to more complex contact-rich scenarios involving intricate interactions. And they typically lack the ability to perform fast replanning, which is essential for reliable execution in real-world settings under significant physical uncertainty.

In this chapter, we develop a general computational framework for robot optimization

that addresses these challenges in a unified manner. Rather than relying on manually designed, problem-specific simplifications, the proposed framework systematically exploits the underlying structure of a given task to construct more favorable optimization objectives. To realize this idea, we propose *Tensor Train Tree Search* (TTTS), a framework that builds upon Monte Carlo Tree Search (MCTS). MCTS provides a flexible foundation for handling non-convexity, hybrid decision spaces, and multi-modal solution discovery, as it does not rely on gradient information and instead performs strategic search over the decision space. However, when applied directly to robot optimization, it suffers from prohibitively large decision trees due to the discretization of continuous domains. To address this limitation, TTTS approximates the decision tree in Tensor Train (TT) format, enabling a compact representation that captures structured correlations across decision variables. This formulation transforms the exponential complexity of the original problem into a tractable form with near-linear scaling. As a result, TTTS enables efficient exploration of large decision spaces while preserving the strategic search capabilities of MCTS.

Our main contributions are as follows:

1. We propose a general formulation for decision making in robotics that can handle nonlinear dynamical systems, non-convex constraints, hybrid state-action spaces, and multi-modal solution discovery.
2. We introduce tensor train to exploit the inherent structure of decision trees, enabling a compact representation with linear complexity and significantly reducing both computational and memory requirements for large-scale search.
3. We evaluate our approach against state-of-the-art baselines and demonstrate its effectiveness on a diverse set of robotic problems, including inverse kinematics, obstacle-aware motion planning, multi-stage motion planning, legged manipulation, and real-world bimanual whole-body manipulation tasks.

4.2 Related Work

A comprehensive review of optimization methods for contact-rich manipulation has been presented in Chapter 2. In this section, we focus specifically on positioning the proposed TTTS framework with respect to the most relevant lines of prior work.

TTTS combines the advantages of existing optimization paradigms while addressing several of their key limitations in contact-rich manipulation. In particular, it integrates tensor train (TT) decomposition with strategic tree search to enable efficient global

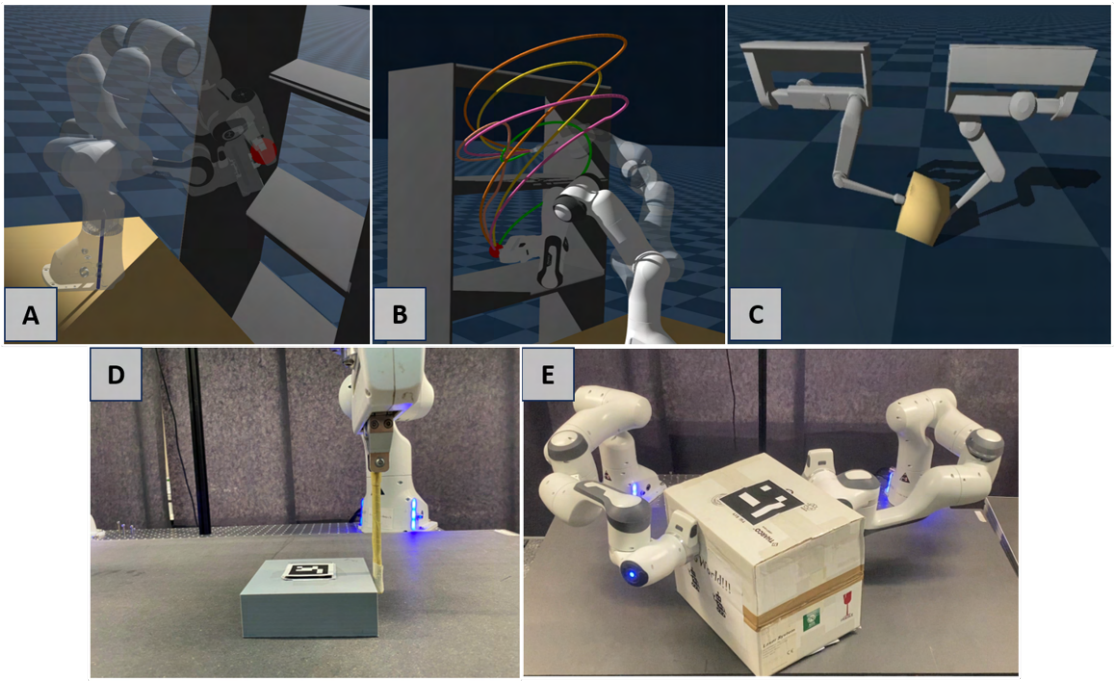


Figure 4.1: Overview of diverse applicable domains. We demonstrate that TTTS is widely applicable in many robot optimization tasks, such as (A) Inverse Kinematics, (B) Motion Planning, (C) Legged Robot Manipulation, (D) Multi-stage Motion Planning and (E) Bimanual Whole-body Manipulation.

optimization over nonlinear, non-convex, and mixed-integer decision spaces. The TT representation provides a compact and separable approximation of the underlying objective function, thereby reducing the combinatorial complexity of the search space. Meanwhile, the tree search component maintains strategic exploration and supports global optimization over the resulting decision tree.

More broadly, TTTS connects to several lines of work that exploit problem structure to improve the scalability of search and decision making. For example, Kim *et al.* (Kim et al., 2020) extend MCTS to continuous action spaces using Voronoi optimistic optimization and provide regret bounds for continuous-space search, whereas TTTS focuses on compressing the full decision tree using tensor train representation. These two perspectives are complementary: the former improves exploration in continuous action spaces directly, while the latter exploits low-rank structure in the induced global objective landscape.

A related motivation also appears in stochastic dynamic programming with factored representations (Boutilier et al., 2000), where compact factored models are used to

avoid explicitly enumerating large state spaces. TTTS shares the same general goal of exploiting problem structure for scalable decision making, but differs in both representation and problem setting. Specifically, TTTS factorizes black-box objective functions using tensor train decomposition and targets deterministic finite-horizon optimization, rather than stochastic MDP value iteration.

This tensorized representation also makes TTTS particularly suitable for hybrid robot optimization. Unlike conventional hybrid approaches that separate the problem into discrete search and continuous optimization stages, TTTS represents discrete choices and discretized continuous variables within a unified decision tree. Gradient-based or sampling-based methods, such as CMA-ES, can then be incorporated as refinement steps, bridging the gap between the discretized tree representation and the original continuous control space.

4.3 Problem Formulation

We consider a general robot optimization formulation that can address diverse problems in robotics:

$$\max_{\boldsymbol{x}\in\Omega_x} J(\boldsymbol{x}),\ \text{s.t. } \phi(\boldsymbol{x}) \leq 0,\ \psi(\boldsymbol{x}) = 0, \tag{4.1}$$

where $\boldsymbol{x}$ is the *decision variable*, Ω_x is the *decision domain*, $J : \Omega_x \to \mathbb{R}$ is the objective function, and $\phi(\boldsymbol{x}) \leq 0$, $\psi(\boldsymbol{x}) = 0$ are inequality and equality constraints encoding system dynamics and other physical limits. We assume that J is *evaluable* and *deterministic*: given the same $\boldsymbol{x}$, $J(\boldsymbol{x})$ always returns the same value, with no assumption on gradient, convexity, or closed-form structure. Any black-box function that can be queried suffices, including those evaluated through black-box physical simulators. The decision domain Ω_x is flexible and may take different forms:

- **Discrete:** $\boldsymbol{x}$ is drawn from a finite set (e.g., combinatorial sequences, integer choices), such as hybrid contact modes (i.e., push, pivot) in contact-rich manipulation tasks.
- **Continuous:** $\boldsymbol{x} \in \mathbb{R}^n$ represents the continuous decision variables to be optimized (e.g., in trajectory optimization). The coarse grid solution can subsequently be refined by a local continuous optimizer.
- **Hybrid:** $\boldsymbol{x} = (\boldsymbol{x}_d, \boldsymbol{x}_c)$ combines discrete choices and continuous variables, which corresponds to the multi-stage manipulation problems that require joint reasoning over discrete modes and continuous motion trajectories.

To handle all three cases in a unified manner, we keep the discrete domain unchanged and discretize the continuous domain onto a finite grid of N_j candidate values, thereby reducing all domain types to the same discrete structure. The resulting index set $\mathcal{I} \subseteq \{1,\dots,N_1\} \times \cdots \times \{1,\dots,N_d\}$ defines a d-layer decision tree with branching factor N_j at layer j: each root-to-leaf path encodes a candidate solution $\boldsymbol{x}$ with associated value $J(\boldsymbol{x})$. The optimization problem thus reduces to finding the maximum-value branch, a tree search problem over $\mathcal{O}(N^d)$ leaves.

MCTS is a natural and powerful approach for solving this problem. It selectively searches branches in a strategic manner, but suffers from the combinatorial complexity of the underlying tree: both memory and search time scale as $\mathcal{O}(N^d)$, becoming intractable as depth d grows.

4.4 Tensor Train Tree Search

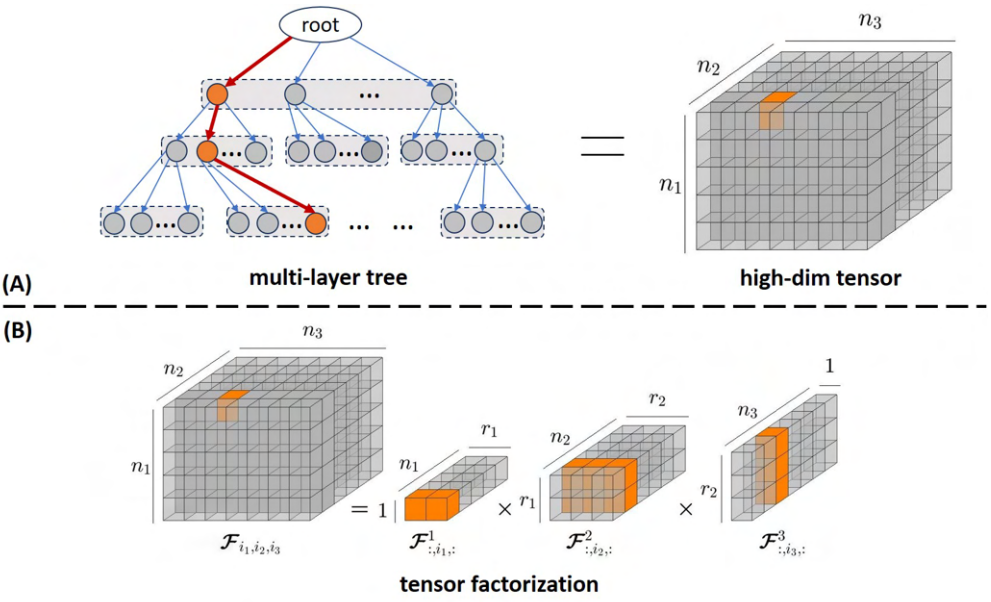


Figure 4.2: **Tree-Tensor-TT transformation.** **(A)** A multi-layer decision tree can be equivalently represented as a high-dimensional tensor, where each tensor element corresponds to the value at the terminal node of a branch. Every branch, including all $N_1 \times \cdots \times N_d$ leaves, is encoded in the tensor, not only a sampled subset. **(B)** Tensor decomposition in TT format. The TT cores provide a compact, global surrogate for this entire tensor: any leaf value can be approximated by contracting the corresponding core slices, enabling efficient UCB computation over the full tree without explicitly storing all N^d entries.

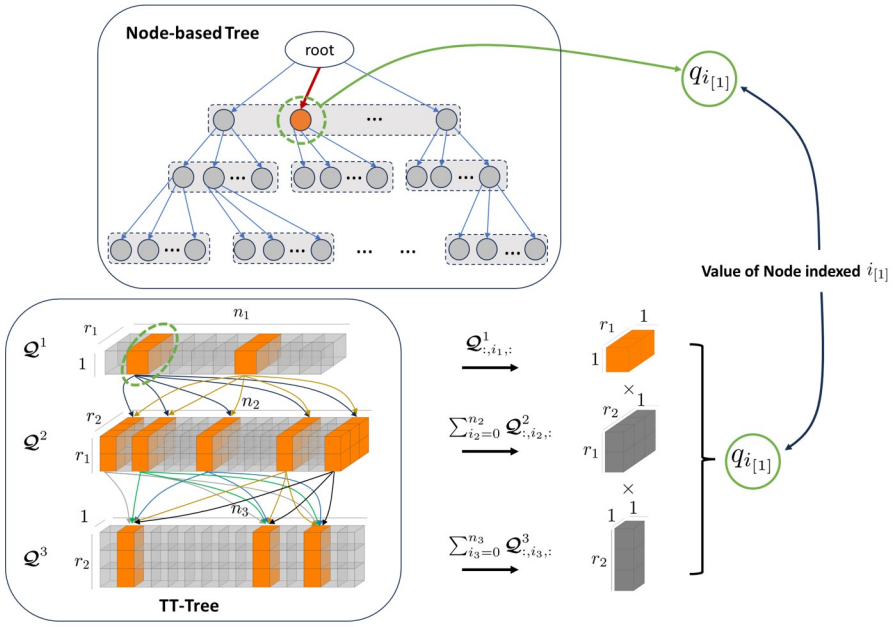


Figure 4.3: **Node value computation for a tree represented in TT format.** This example illustrates how the value of a first-layer node is computed using TT cores. The value of node i_1 is obtained by fixing the slice $\mathcal{Q}^1_{:,i_1,:}$ from the first TT core and summing over all indices of the remaining cores ($\mathcal{Q}^2$ through $\mathcal{Q}^3$) to aggregate all leaf completions, as in Eq. (4.3).

To reduce this combinatorial complexity, our objective is to express the decision tree with linear complexity $\mathcal{O}(\lambda N d)$, where λ is a scaling factor. This is equivalent to representing a high-dimensional tensor of size $\mathcal{O}(N^d)$ using a compact, low-rank decomposition, which is a well-studied problem in numerical mathematics (Cichocki et al., 2016). Specifically, we employ tensor train (TT) to exploit the separable structure and reduce inter-layer dependencies. As a result, the original high-dimensional tensor can be efficiently represented using significantly fewer parameters through a sequence of third-order TT cores, namely:

$$\widehat{\boldsymbol{\mathcal{T}}}_{(i_1,\ldots,i_d)} \approx \boldsymbol{\mathcal{T}}^1_{:,i_1,:} \boldsymbol{\mathcal{T}}^2_{:,i_2,:} \cdots \boldsymbol{\mathcal{T}}^d_{:,i_d,:}, \tag{4.2}$$

where $\widehat{\boldsymbol{\mathcal{T}}}$ denotes the original high-dimensional tensor, and $\boldsymbol{\mathcal{T}}^j$ is the core corresponding to the j-th dimension (i.e., the j-th layer of the tree). Figure 4.2 (B) presents an illustrative example of a three-dimensional tensor.

Algorithm 1 presents the pseudocode of the proposed method. As indicated in (2.23), a key component of MCTS is its ability to perform strategic search by leveraging the value and visit count information stored at each node in the tree, denoted by $\widehat{\boldsymbol{\mathcal{Q}}}$ and $\widehat{\boldsymbol{\mathcal{V}}}$, respectively. However, $\widehat{\boldsymbol{\mathcal{Q}}}$ and $\widehat{\boldsymbol{\mathcal{V}}}$ are high-dimensional tensors, which are impractical to store due to the combinatorial complexity. In this work, we utilize tensor factorization to approximate them in TT format, denoted as $\boldsymbol{\mathcal{Q}}$ and $\boldsymbol{\mathcal{V}}$. Crucially, the full tensors never need to be explicitly instantiated or stored. Instead, TT-Cross uses an evaluation function to query individual elements of $\widehat{\boldsymbol{\mathcal{Q}}}$ and $\widehat{\boldsymbol{\mathcal{V}}}$ on the fly.

To avoid performing TT approximation repeatedly for every individual scenario within a task, we augment the decision tree with task variables $\boldsymbol{z}$ (e.g., task parameters or initial states). We then employ TT-Cross to approximate the objective function J in TT format, yielding an augmented tensor model $\boldsymbol{\mathcal{Q}}^+$. During the online search phase, given any specific task variable $\boldsymbol{z}$, the corresponding task-specific tree representation is instantaneously recovered via partial evaluation, denoted as $\boldsymbol{\mathcal{Q}} = \boldsymbol{\mathcal{Q}}^+(\boldsymbol{z})$. Owing to the *maximum volume* principle of TT-Cross, the resulting model inherently captures the most informative branches of the decision tree, enabling the immediate identification of promising solutions even during the first iteration. The visit count tensor $\boldsymbol{\mathcal{V}}$ is initialized to zero.

Due to dependence on the parent nodes, the value and visitation statistics of a node are path-dependent, introducing non-Markovian behavior. Querying the value of node $n_{i_{[j]}}$ requires summing over all completions of the branch from layer $j+1$ to depth d. Given the full tree approximated in TT format, the value of a node at level j, denoted

as $q_{i_{[j]}}$, can be efficiently computed via tensor contractions as

$$\begin{aligned} q_{i_{[j]}} &= \sum_{i_{j+1}=0}^{N_{j+1}} \cdots \sum_{i_d=0}^{N_d} \widehat{\mathcal{Q}}_{(i_1,\ldots,i_j,i_{j+1},\ldots,i_d)} \\ &\approx \mathcal{Q}^1_{:,i_1,:} \cdots \mathcal{Q}^j_{:,i_j,:} \sum_{i_{j+1}=0}^{N_{j+1}} \cdots \sum_{i_d=0}^{N_d} \mathcal{Q}^{j+1}_{:,i_{j+1},:} \cdots \mathcal{Q}^d_{:,i_d,:}, \end{aligned} \tag{4.3}$$

where each TT core $\mathcal{Q}^l$ encodes the factorized structure of the tree at layer l. To compute the node value at depth j, we extract the corresponding slices from the first j TT cores (representing the visited layers), and perform summation over indices in the remaining cores from $j+1$ to d. The final node value is obtained by multiplying the resulting matrices across all layers. Figure 4.3 illustrates how to compute the value of a node in the first layer of a three-layer tree using TT cores. The multiple arrows between TT cores indicate the ability to perform parallel selection in the TT-Tree.

Similarly, the visit count $v_{i_{[j]}}$ is computed as:

$$\begin{aligned} v_{i_{[j]}} &= \sum_{i_{j+1}=0}^{N_{j+1}} \cdots \sum_{i_d=0}^{N_d} \widehat{\mathcal{V}}_{(i_1,\ldots,i_j,i_{j+1},\ldots,i_d)} \\ &\approx \mathcal{V}^1_{:,i_1,:} \cdots \mathcal{V}^j_{:,i_j,:} \sum_{i_{j+1}=0}^{N_{j+1}} \cdots \sum_{i_d=0}^{N_d} \mathcal{V}^{j+1}_{:,i_{j+1},:} \cdots \mathcal{V}^d_{:,i_d,:}. \end{aligned} \tag{4.4}$$

With $q_{i_{[j]}}$ and $v_{i_{[j]}}$ efficiently available via TT contractions (Eqs. (4.3)–(4.4)), the algorithm follows the standard MCTS pipeline: selection, expansion, simulation, and backpropagation, with an additional top-τ solutions maintenance step. We now describe each step in detail.

Selection and Expansion. Given $q_{i_{[j]}}$, $v_{i_{[j]}}$, and $v_{i_{[j-1]}}$, node expansion follows the UCB rule from (2.23). The TT-based representation enables parallel selection, which is typically non-trivial in traditional node-based or table-based implementations (Chaslot et al., 2008). If no further expansion is possible, the current branch terminates. Otherwise, the branch is extended by one layer and followed by a simulation to the leaf.

Simulation. Rather than performing a random rollout, we leverage the TT value model $\mathcal{Q}$ as a global approximation of the decision tree to guide the simulation strategy via stochastic sampling, treating TT values as unnormalized probabilities. This enables

parallel simulation and yields a top-τ set of candidate solutions.

Backpropagation. Following the simulation phase, the TT models associated with the traversed branches are updated during the MCTS backpropagation phase. Specifically, the visit count tensor $\mathcal{V}$ is updated incrementally via a residual strategy. Let $\Delta\widehat{\boldsymbol{V}}_{\gamma}^{\ell}$ denote the binary indicator tensor that tracks node visitations at layer γ during iteration ℓ, where an element is 1 if the corresponding node is visited and 0 otherwise. Leveraging the fact that the newly visited indices are precisely known a priori, we tailor the standard TT-Cross algorithm into an efficient, guided variant. By restricting evaluation exclusively to the active indices (where the indicator is 1), this guided variant recovers the TT approximation of $\Delta\widehat{\boldsymbol{V}}_{\gamma}^{\ell}$ in a single iteration. The value model $\mathcal{Q}$ can also be updated in a similar manner to enhance approximation accuracy, but we omit this step in practice to reduce runtime.

Top-τ Update and Refinement. We maintain a candidate set $\mathcal{S}$ to dynamically track the top-τ elite solutions, which is updated iteratively with newly discovered solutions. For continuous decision variables, the coarse solutions obtained from TT-Tree Search are subsequently optimized using CMA-ES as a local refiner.

Remark (Parallelism in TTTS). Standard MCTS faces two fundamental challenges when parallelizing tree search: (1) all threads tend to follow the same UCB-greedy path from the root, causing redundant exploration; and (2) concurrent updates to shared per-node statistics require expensive locking mechanisms (Chaslot et al., 2008). TTTS addresses both challenges through the separable structure of its TT representation, which decouples the tree across layers and enables global, layer-wise operations. For challenge (1): in standard MCTS, selection walks down the tree one node at a time along a single root-to-leaf path, so parallel threads all tend to visit the same nodes. In TTTS, the separable structure allows UCB scores for *all* nodes at a given layer to be computed simultaneously via TT contractions (Eq. (4.3)–(4.4)), and the top-τ nodes are selected in one shot. This layer-wise parallel selection naturally diversifies exploration across τ branches at once. For challenge (2): in standard MCTS, both visit counts and action-value estimates are stored as per-node counters, so threads updating these counters simultaneously must contend for locks. In TTTS, both statistics are encoded as TT models ($\mathcal{V}$ for visit counts and $\mathcal{Q}$ for node values) and updated via TT-Cross in a single bulk operation, eliminating the need for per-node synchronization. The simulation phase likewise draws multiple complete trajectories in parallel by treating $\mathcal{Q}$ as an unnormalized probability model, yielding τ candidate solutions per iteration without thread contention.

Algorithm 1 Tensor Train Tree Search (TTTS)

function TENSORTRAINTREESEARCH($\boldsymbol{x}_0$, $\boldsymbol{z}$, J, Ω_x, L, $\mathcal{I}$)
  Input: Initial state $\boldsymbol{x}_0$, Condition var. $\boldsymbol{z}$,
   Maximum iter. L, Objective function J,
   Search domain $\Omega_x = \{\boldsymbol{x}^{(i_1,\ldots,i_d)} : i_j \in \{1,\ldots,N_j\}\}$,
   Index set $\mathcal{I} \subseteq \{1,\ldots,N_1\} \times \cdots \times \{1,\ldots,N_d\}$,
  Hyperparameters: Number of solutions τ,
   Exploration param. c # default: $\tau = 10$, $c = 3$
  Output: top-τ solutions: solution set $\mathcal{S}$
  // ===== TT-Tree Initialization =====
  $\boldsymbol{\mathcal{Q}}^+ \leftarrow$ TT-Cross(J, Ω_x), $\boldsymbol{\mathcal{Q}} \leftarrow \boldsymbol{\mathcal{Q}}^+(\boldsymbol{z})$
  // ===== TT-Tree Search =====
  $i_0 \leftarrow$ Node$(\boldsymbol{x}_0)$, $\boldsymbol{\mathcal{V}_0} \leftarrow \boldsymbol{0}$, $\mathcal{S} = [\,]$
  for $\ell = 1, 2, \ldots,$ L **do**
    for $j = 1, \ldots, d$ **do**
      $q_{i_{[j]}} \leftarrow$ Eq. (4.3), $v_{i_{[j]}}, v_{i_{[j-1]}} \leftarrow$ Eq. (4.4)
      $i_{[j]} \leftarrow \arg\max^{\tau}_{i_{[j]} \in \mathcal{I}_{[j]}} \left(\frac{q_{i_{[j]}}}{v_{i_{[j]}}} + c \cdot \sqrt{\frac{\log v_{i_{[j-1]}}}{v_{i_{[j]}}}} \right)$
      if i_j is not expanded **then break**
    end for
    for $s = j+1, \ldots, d$ **do**
      $q_{i_{[s]}} \leftarrow$ Eq. (4.3)
      $i_{[s]} \leftarrow \arg\max^{\tau}_{i \in \mathcal{I}_{[s]}} q_{i_{[s]}}$
    end for
    $\boldsymbol{x}^\star \leftarrow \arg\max^{\tau}_{\boldsymbol{x} \in \Omega_x(i_{[d]})} J(\boldsymbol{x})$
    for $\gamma = 1, \ldots, j$ **do**
      $\boldsymbol{\mathcal{V}}^{\ell}_{\gamma} = \boldsymbol{\mathcal{V}}^{\ell-1}_{\gamma} +$ Guided TT-Cross$(\Delta\widehat{\boldsymbol{\mathcal{V}}}^{\ell}_{\gamma}, i_{[\gamma]})$
    end for
    $\mathcal{S}^+ \leftarrow$ append$(\mathcal{S}, \boldsymbol{x}^\star)$
    $\mathcal{S} \leftarrow \arg\max^{\tau}_{\boldsymbol{x} \in \mathcal{S}^+} J(\boldsymbol{x})$
  end for
  // ===== TT-Tree Refinement =====
  $\mathcal{S} \leftarrow$ Refine$(\mathcal{S})$ # e.g., CMA-ES on continuous dims
  return $\mathcal{S}$
end function

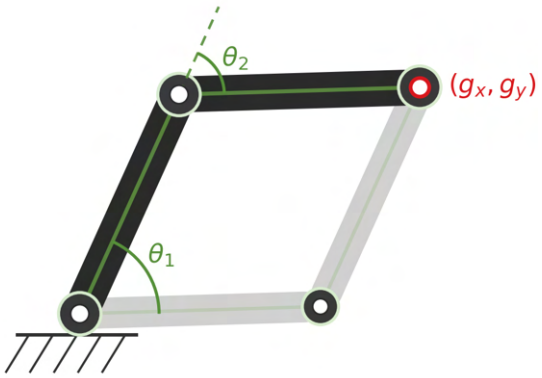


Figure 4.4: **Two-joint inverse kinematics with multi-modal solutions.** Two distinct arm configurations, shown in dark and light gray, reach the same end-effector goal (g_x, g_y) due to the nonlinear kinematics, illustrating the multi-modal and non-convex nature of the problem.

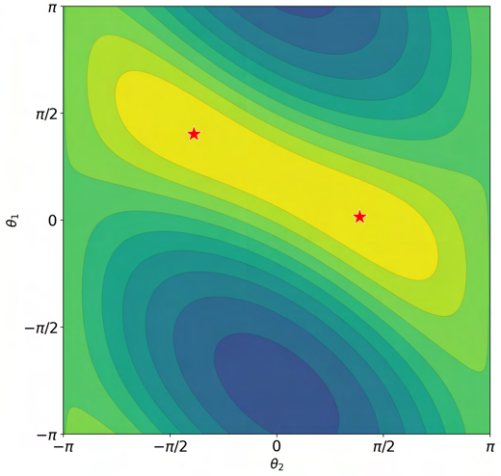
4.4.1 Algorithm Walkthrough via an Illustrative Example

We walk through Algorithm 1 step-by-step using a 2-DOF inverse kinematics (IK) problem. As shown in Fig. 4.4, the task is to find joint angles $(\theta_1, \theta_2) \in [-\pi, \pi]^2$ such that the robot end-effector position, denoted by (r_x, r_y), reaches the target position (g_x, g_y). Because of the nonlinear kinematics, the same goal position can be reached by two distinct arm configurations, shown in dark and light gray in Fig. 4.4. This yields a representative nonlinear problem with a multi-modal solution set, well suited for illustrating the key properties of TTTS.

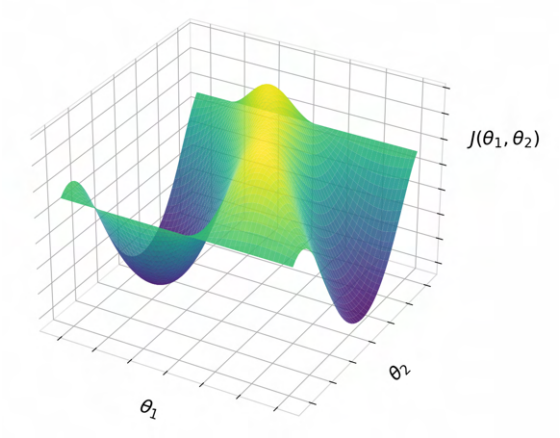
Objective function and discretization. We define the cost function as $c(\theta_1, \theta_2) = (r_x - g_x)^2 + (r_y - g_y)^2$, which represents the squared end-effector positioning error, and convert it into a maximization objective:

$$J(\theta_1, \theta_2) = e^{-c(\theta_1, \theta_2)}. \tag{4.5}$$

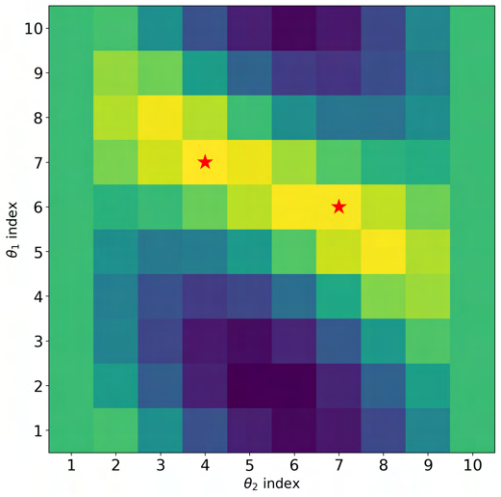
Figs. 4.5a and 4.5b show the landscape of J from top-down and perspective views, respectively. The 3D surface in Fig. 4.5b clearly reveals the non-convex, multi-modal structure of J, with two distinct peaks corresponding to the two kinematic solutions, a landscape in which gradient-based methods are prone to converging to a single optimum. Discretizing (θ_1, θ_2) onto a 10×10 grid yields the objective matrix in Fig. 4.5c, where the two red-star entries mark the optimal grid configurations.



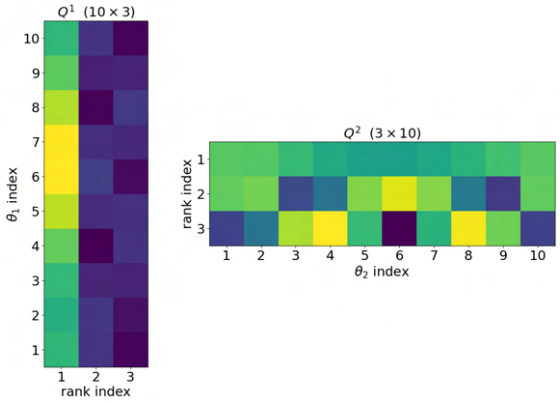
(a) Top-down view of $J(\theta_1, \theta_2)$ over the range $[-\pi, \pi]$ for each joint. Lighter colors (yellow) indicate higher objective values.



(b) 3D surface of $J(\theta_1, \theta_2)$, revealing its non-convex, multi-modal structure with two distinct peaks.



(c) Discretized 10×10 value matrix. Red-star markers indicate the two optimal configurations.



(d) TT factorization of the value matrix into $\boldsymbol{Q}^1 \in \mathbb{R}^{10\times 3}$ and $\boldsymbol{Q}^2 \in \mathbb{R}^{3\times 10}$ at rank $r = 3$. TT factorization corresponds to matrix factorization in this low-dimensional illustrative example.

Figure 4.5: **Objective function, discretization, and TT approximation for 2-DOF IK example. (a)** Top-down heatmap of the value function $J(\theta_1, \theta_2)$ over $[-\pi, \pi]^2$. **(b)** 3D surface view of J, clearly showing its nonconvex, multimodal landscape with two distinct peaks, each corresponding to one kinematic solution. **(c)** Discretizing onto a 10×10 grid yields a matrix representation; red-star markers indicate the two optimal configurations. **(d)** TT-Cross decomposes the value matrix into two low-rank factors $\boldsymbol{Q}^1 \in \mathbb{R}^{10\times 3}$ and $\boldsymbol{Q}^2 \in \mathbb{R}^{3\times 10}$ at rank $r = 3$, capturing the dominant structure with significantly fewer parameters.

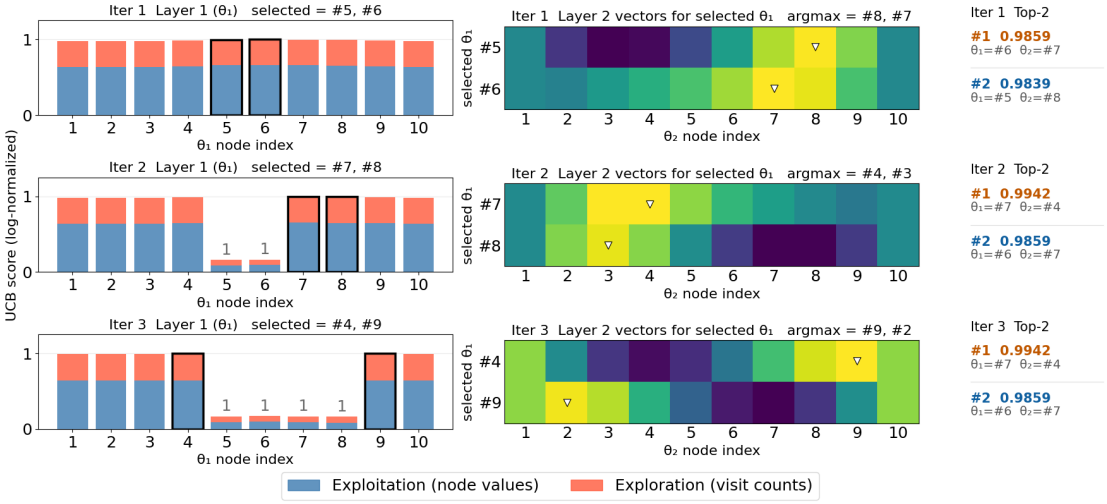


Figure 4.6: **TTTS applied to the 2-DOF IK problem across three iterations. Left:** UCB bar charts for Layer-1 (θ_1) node selection, decomposed into exploitation (blue) and exploration (red) components; nodes with black borders are selected, and gray labels denote accumulated visit counts. **Middle:** Layer-2 (θ_2) value heatmaps conditioned on each selected first-layer node; triangular markers indicate the top-τ selected second-layer nodes. **Right:** maintained top-τ candidate solutions after each iteration. As visit counts accumulate across iterations, the exploration bonus progressively shifts selection toward unvisited regions, enabling systematic coverage of the full solution space.

Initialization. We start TTTS by calling TT-Cross to construct a compact TT approximation of the objective function, with the maximum rank budget set to 3. This produces two low-rank factor matrices $\boldsymbol{Q}^1 \in \mathbb{R}^{10\times 3}$ and $\boldsymbol{Q}^2 \in \mathbb{R}^{3\times 10}$ with TT rank $r = 3$, as illustrated in Fig. 4.5d. Despite using far fewer parameters than the full 10×10 matrix, this factorization captures the dominant structure of the value landscape and concentrates probability mass near high-value regions, providing an informative global prior before any tree traversal begins.

Selection. At each iteration ℓ, TTTS selects the top-τ nodes in Layer 1 (θ_1) by maximizing the UCB score. The node value for the first layer is computed from the TT cores as:

$$q_{i_{[1]}} = \boldsymbol{Q}^1_{i_1,:} \sum_{i_2=1}^{N_2} \boldsymbol{Q}^2_{:,i_2}, \tag{4.6}$$

which marginalizes over all second-layer completions via $\boldsymbol{Q}^2$, serving as the exploitation term. The visit count v_{i_1} provides the exploration bonus. Given the selected first-layer node i_1^*, the second-layer node values are:

$$q_{i_{[2]}} = \boldsymbol{Q}^1_{i_1^*,:} \boldsymbol{Q}^2_{:,i_2}, \tag{4.7}$$

and the top-τ nodes in θ_2 are selected accordingly.

Fig. 4.6 traces this process across three iterations (left: Layer-1 UCB bar charts; middle: Layer-2 value heatmaps; right: maintained top-τ solutions). In Iteration 1, all visit counts are zero, so the UCB score is dominated entirely by the exploitation term (blue bars). Nodes $i_1 = 5$ and $i_1 = 6$ achieve the highest q_{i_1} values and are selected. Conditioned on these selections, the Layer-2 heatmaps identify $i_2 = 8$ for $i_1 = 5$ and $i_2 = 7$ for $i_1 = 6$. The TT model successfully guides search toward high-value regions from the very first iteration. However, due to the low-rank approximation, not all optima may be captured precisely in a single pass. The top-τ solution list is maintained and updated across iterations as the UCB exploration bonus progressively steers search toward other promising regions.

Backpropagation. After each iteration, the visit count model $\boldsymbol{\mathcal{V}}$ is updated via Guided TT-Cross to record the explored nodes. In Iteration 2, nodes $i_1 = 5$ and $i_1 = 6$ carry visit count 1 (indicated by the "1" labels in Fig. 4.6), which reduces their exploration bonus and raises the UCB scores of their neighbors $i_1 = 7$ and $i_1 = 8$. By Iteration 3, nodes $i_1 = 4$ and $i_1 = 9$ are selected, revealing a systematic outward expansion: TTTS uses the exploitation term to focus on high-value regions and the exploration term to

spread coverage as nodes are visited, together driving strategic search across the full solution space.

Refinement. The top-τ discrete candidate solutions from TT-Tree search are passed to CMA-ES for continuous refinement. This corrects for grid discretization errors and yields the two precise IK configurations shown in Fig. 4.4.

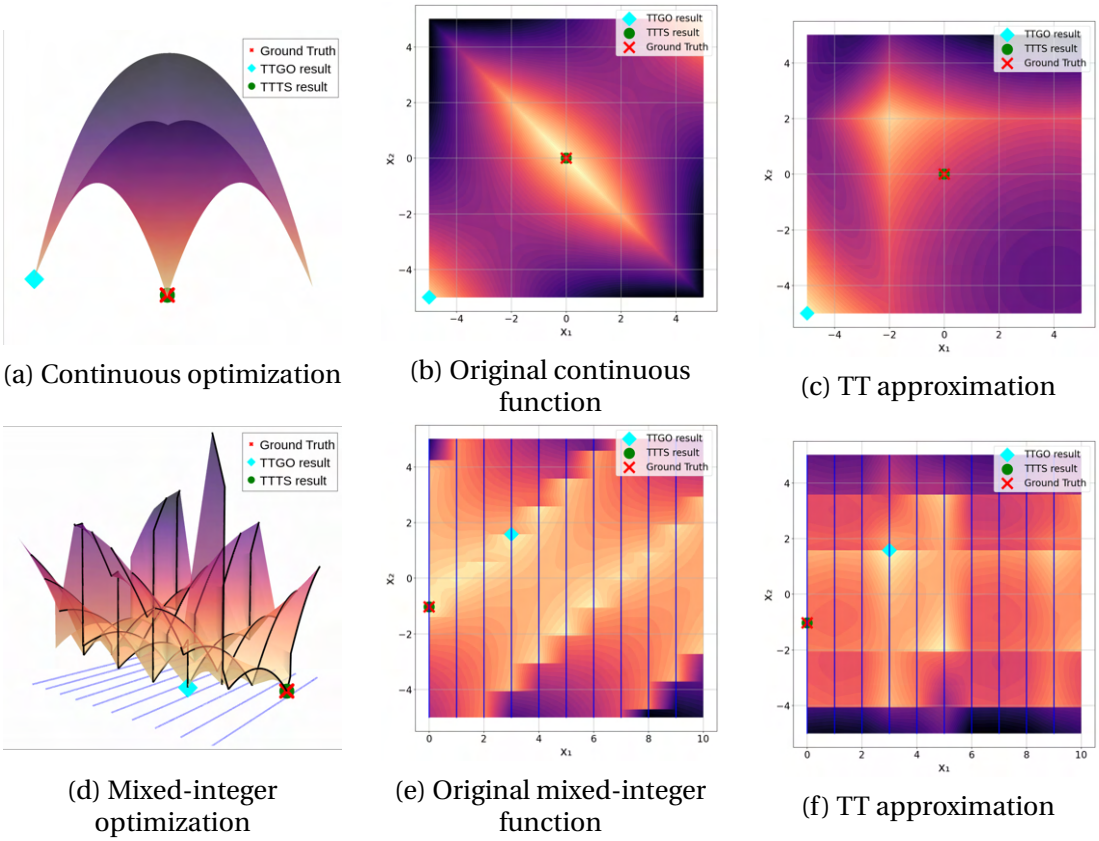
This IK example could of course be solved differently; it is used here to didactically illustrate how TTTS can combine a compact TT representation as a global prior with the UCB-driven exploration–exploitation mechanism of MCTS, simultaneously discovering all optimal modes through linear-complexity tensor operations.

4.4.2 Toy Examples on Function Optimization

Continuous non-convex optimization. Many optimization problems in robotics are non-convex due to obstacles and nonlinear system dynamics. To illustrate both the strengths and limitations of low-rank TT approximation, we consider a full-rank non-convex continuous function (A.1) and approximate it using a low-rank TT model with rank $r_{max} = 2$ via TT-Cross. The discrete matrix analogue of this function, shown in Figure 4.7b, clearly exhibits full-rank characteristics. Note that the general objective J in (4.1) is defined as $J(\boldsymbol{x}) = e^{-c(\boldsymbol{x})}$, where c is the cost function. For better visual clarity, we directly illustrate the landscape of c in Figure 4.7b. The detailed definitions are provided in Appendix A.1.2.

The resulting approximation is shown in Figure 4.7c, which shows that, despite using a much lower rank, TT approximation effectively captures the structure of the function and enables rapid localization of regions likely to contain optimal solutions. This is the strategy adopted by TTGO (Shetty et al., 2024b), which directly samples candidate solutions from the TT model. However, relying solely on the TT model cannot guarantee convergence to the global optimum: approximation errors may cause the model to underestimate the quality of regions not well captured at low rank. TTTS overcomes this limitation by coupling TT approximation with UCB-driven tree search. The TT model guides early exploration toward high-value regions, while UCB ensures every node is eventually visited, achieving asymptotic global convergence (Figure 4.7a).

Mixed-integer programming. We further demonstrate the effectiveness of our approach in tackling a more challenging problem: mixed-integer nonlinear programming (MINLP). Such problems include both non-convexity due to nonlinear constraints and



(a) Continuous optimization

(b) Original continuous function

(c) TT approximation

(d) Mixed-integer optimization

(e) Original mixed-integer function

(f) TT approximation

Figure 4.7: **Toy examples of function optimization with low-rank tensor-train (TT) approximations.** (a–c) Continuous non-convex optimization: (a) landscape of a multi-modal non-convex function $f_1(x_1, x_2)$; (b) its contour map; (c) TT-Cross reconstruction with maximal TT rank $r_{\max} = 2$, which captures the dominant structure but smooths out fine details. (d–f) Mixed-integer non-convex optimization: (d) landscape of $f_2(x_1, x_2)$ with both discrete (x_1) and continuous (x_2) variables; (e) its grid-aligned contour plot; (f) TT approximation, which highlights promising regions but may miss the true global optimum. Markers indicate solutions obtained by TTGO and TTTS compared with the ground-truth optimum.

the combinatorial complexity imposed by integer variables. This setting is particularly difficult because integer variables are discrete and lack gradient information, rendering standard nonlinear programming (NLP) solvers ineffective.

Figure 4.7d illustrates a simple MINLP example, where x_1 is an integer variable ranging from 0 to 10, and x_2 is a continuous variable. The cost function (A.2) is nonlinear and exhibits multiple local optima. Figure 4.7e presents the discrete matrix analogue of the continuous cost function. We approximate the function in TT format with rank $r_{\max} = 2$, as shown in Figure 4.7f. The results indicate that even with a low TT rank, TT-Cross can identify high-quality local optima, demonstrating the strong modeling and optimization capabilities of TT. However, the results obtained by TTGO do not correspond to the global optimum, highlighting that a low-rank TT approximation may fail to fully capture the complexity of the original function. In such cases, TTTS leverages strategic search to efficiently converge to the global optimal solution.

Remark on the low-rank approximation. The low-rank assumption governs the *speed* of convergence, not convergence itself, and two complementary regimes arise in practice. When the decision tree is genuinely low-rank, TT approximation captures the global landscape accurately from a small number of function evaluations, and TTTS identifies near-optimal solutions from the very first iterations. When the task is not perfectly low-rank, the TT approximation introduces nonzero ϵ but still captures the dominant features of the objective landscape, providing an informative warm-start that concentrates early MCTS exploration near promising regions. This is illustrated in Figure 4.7(c), where even a rank-2 approximation of a high-rank continuous function effectively highlights high-value regions. By combining this TT-guided initialization with UCB-driven tree search, TTTS achieves both efficient exploration and global convergence. In practice, many robotics tasks exhibit low-rank structure due to factored objective functions and constrained kinematics and dynamics, making low-rank TT approximations both effective and broadly applicable.

4.5 TTTS for Generalized Robot Optimization

We now instantiate (4.1) for generalized robot optimization, such as inverse kinematics, motion planning, multi-stage motion planning, and model predictive control. These problems correspond to a large set of mathematical programs, including nonlinear programming (NLP), large-scale (a.k.a. high-dim) NLP and mixed-integer nonlinear programming (MINLP). In particular, we leverage basis functions to reduce the dimensionality in large-scale NLP formulations, and the choice of basis functions remains flexible.

We describe the notation and variables:

- K the number of discrete modes (a.k.a. stages).
- $m_k \in \mathcal{M}$ the discrete *mode* at stage k, chosen from a finite (or countable) set $\mathcal{M}$.
- $a_k \in \mathcal{A}$ the discrete action at stage k, chosen from the finite action set $\mathcal{A}$.
- $\boldsymbol{x}_k^t \in \Omega_{\boldsymbol{x}} \subseteq \mathbb{R}^n$ a continuous state (or configuration) at stage k at time t.
- $\boldsymbol{u}_k^t \in \Omega_{\boldsymbol{u}} \subseteq \mathbb{R}^p$ a continuous control input at stage k at time t.
- T trajectory length for each stage.
- $B = B_1 + \cdots + B_K$, the total number of basis functions, where B_k denotes the number of basis functions at stage k.
- $\boldsymbol{\Psi}_k \in \mathbb{R}^{T \times B_k}$ a chosen set of basis functions for stage k, used to reconstruct the continuous control from weights.
- $\boldsymbol{w}_k \in \Omega_{\boldsymbol{w}} \subseteq \mathbb{R}^{B_k}$ the vector of basis *weights* at stage k. We let $\boldsymbol{u}_k^{[T]} = \boldsymbol{\Psi}_k \boldsymbol{w}_k$, encoding the continuous trajectory in basis form.
- $c(m_k, a_k, x_k^t, u_k^t)$ stage cost (e.g., energy, distance, penalty) at time t in stage k.
- $c_{\text{terminal}}(x_K^T)$ terminal cost, e.g. capturing the final configuration error.
- $\phi(\cdot) \leq 0$, $\psi(\cdot) = 0$ general inequality/equality constraints (kinematic limits, collision avoidance, dynamics, boundary conditions).

The decision variable is $\boldsymbol{x} = (a_1, \ldots, a_K, , \boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$, collecting both the discrete action modes and the continuous basis weights. We use a bracket subscript for sequences (e.g., $\boldsymbol{x}_k^{[T]}$ represents the full trajectory at stage k, and $\boldsymbol{u}_{[K]}^{[T]}$ represents the control variables across all stages and time steps). The robot optimization problem is:

$$\min_{a_{[K]}, \boldsymbol{u}_{[K]}^{[T]}} \quad \sum_{k=0}^{K} \int_0^T c\Big(m_k,\ a_k,\ \boldsymbol{x}_k^t,\ \boldsymbol{u}_k^t\Big) dt \ +\ c_{\text{terminal}}\Big(\boldsymbol{x}_K^T\Big) \tag{4.8}$$

$$\text{s.t.} \quad \boldsymbol{u}_k^{[T]} = \sum_{b=0}^{B} \boldsymbol{\Psi}_k^b \, \boldsymbol{w}_k^b, \tag{4.9}$$

$$\forall_{k=0}^{K} \quad m_{k+1} \ \in\ \text{succ}\Big(m_k,\ a_k,\ \boldsymbol{x}_k^{[T]},\ \boldsymbol{u}_k^{[T]}\Big), \tag{4.10}$$

$$\forall_{k=0}^{K} \quad \phi\Big(m_k,\ a_k,\ \boldsymbol{x}_k^{[T]},\ \boldsymbol{u}_k^{[T]}\Big) \ \leq\ 0, \tag{4.11}$$

$$\forall_{k=0}^{K} \quad \psi\Big(m_k,\ a_k,\ \boldsymbol{x}_k^{[T]},\ \boldsymbol{u}_k^{[T]}\Big) \ =\ 0, \tag{4.12}$$

$$(m_0,\ \boldsymbol{x}_0^0) = (m_{\text{init}},\ \boldsymbol{x}_{\text{init}}), \tag{4.13}$$

where:

- (4.9) encodes the full trajectory of decision variables using basis functions.
- (4.10) enforces the allowed transition to m_{k+1} in the discrete mode set $\mathcal{M}$, given the current mode m_k, discrete action a_k, and the continuous trajectories $\boldsymbol{x}_k^{[T]}$, $\boldsymbol{u}_k^{[T]}$.
- (4.11) and (4.12) represent the system dynamics, consistency of different modes, and other physical constraints.
- m_{init} and x_{init} denote the initial mode and state.

This formulation covers a wide range of robotic tasks:

Inverse Kinematics (IK) Set $T = 1$, $K = 1$. The decision variable is $\boldsymbol{w}_1$ with $\boldsymbol{u}_1 = \boldsymbol{\Psi}_1 \boldsymbol{w}_1$, and the cost measures end-effector positioning error subject to joint-limit and collision constraints.

Motion Planning (MP) Set $K = 1$ with trajectory length T. The decision variable is $\boldsymbol{w}$ encoding the full joint trajectory via basis functions.

Multi-stage Motion Planning (MsMP) Use K stages, where at each stage k the discrete action $a_k \in \mathcal{A}$, mode $m_k \in \mathcal{M}$, and continuous trajectory $\boldsymbol{u}_k^{[T]} = \boldsymbol{\Psi}_k \boldsymbol{w}_k$ must satisfy collision-free motion, piecewise dynamics, and valid contact transitions.

To apply TTTS to (4.8), we construct the search space Ω_x as a $d = K + B$ layer decision tree: the first K layers index the discrete action choices (each with $|\mathcal{A}|$ options), and the remaining B layers index the discretized basis weights (each with N grid points). TTTS then optimizes over this index space as described in Section 4.4, recovering the optimal $\boldsymbol{x}^* = (a^*_{[K]}, \boldsymbol{w}^*)$, followed by CMA-ES refinement of the continuous weights.

4.6 Experimental Results on Generalized Robot Optimization

The toy examples in Section 4.4.2 establish the algorithm's core properties on simple function optimization. In this section, we further evaluate TTTS on a diverse set of robotic tasks to demonstrate its broad applicability. We conduct ablation studies to assess the contribution of each component and compare against state-of-the-art baselines. Hyperparameters and cost functions for each task are detailed in the appendix.

4.6.1 Inverse Kinematics

Building on the 2-DOF IK example in Section 4.4.1, we here evaluate TTTS on 3-DOF and 7-DOF manipulators (denoted *IK1* and *IK2*) with collision avoidance. This corresponds to $T = 1$, $K = 1$ in (4.1), with continuous decision variables $\boldsymbol{u}$ subject to joint limits, collision avoidance, and reachability constraints.

We randomly generated five targets in the ablation study to analyze the necessity of each component of TTTS, which consists of three main components: TT-Tree Initialization, TT-Tree Search, and TT-Tree Refinement. TT-Tree Initialization typically requires some computation time due to TT-Cross approximation (particularly for high-dimensional systems), but this process is performed offline. At this step, the state space is augmented with task variables, enabling rapid task-conditioned retrieval for TT-Tree Search. In our experiments, we set $r_{\max} = 21$ for both the 3-joint and 7-joint manipulators (which we refer to as *IK1* and *IK2* in Figure 4.11), resulting in a coarse approximation of the full decision tree. Notably, obstacle avoidance tends to introduce high-rank behavior, making low-rank TT approximations less accurate. As shown in Figure 4.8 (A), TT-Tree Initialization alone does not yield the global optimum, highlighting the limitations of using TT approximation by itself. However, after performing TT-Tree Search and Refinement, the final task-space error is zero, indicating that the globally optimal solution was successfully found.

We further compared TTTS against state-of-the-art baselines: TTGO, MCTS, and CMA-ES. TTGO and MCTS can each be viewed as special cases of TTTS. TTGO relies on direct TT sampling instead of UCB-driven tree search, while MCTS operates without

A **Ablation Study for Inverse Kinematics**

	TT-Tree Init. (Offline)		TT-Tree Search		TT-Tree Refine.	
	Error	Time (s)	Error	Time (s)	Error	Time (s)
3-joint	0.02 ± 0.01	0.28	0.02 ± 0.01	0.05 ± 0.00	0.00 ± 0.00	0.08 ± 0.00
7-joint	0.21 ± 0.10	0.65	0.10 ± 0.06	0.22 ± 0.02	0.00 ± 0.00	0.31 ± 0.00

B **Ablation Study for Motion Planning**

	TT-Tree Init. (Offline)			TT-Tree Search			TT-Tree Refine.		
	Error	Total Cost	Time (s)	Error	Total Cost	Time (s)	Error	Total Cost	Time (s)
3-joint	0.13 ± 0.06	7.42 ± 3.97	5.50	0.13 ± 0.06	7.41 ± 3.98	0.08 ± 0.05	0.00 ± 0.00	0.41 ± 0.49	0.31 ± 0.00
7-joint	0.06 ± 0.03	3.64 ±1.54	1.27	0.05 ± 0.03	3.08 ± 1.43	0.10 ± 0.03	0.00 ± 0.00	0.11 ± 0.04	0.34 ± 0.00

C **Ablation Study for Face-switching Planar Pushing**

TT-Tree Init. (Offline)			TT-Tree Search			TT-Tree Refine.		
Error	Total Cost	Time (s)	Error	Total Cost	Time (s)	Error	Total Cost	Time (s)
0.19 ± 0.05	0.16 ± 0.02	4.85	0.08 ± 0.03	0.11 ± 0.03	0.09 ± 0.03	0.00 ± 0.00	0.06 ± 0.00	0.03 ± 0.00

Figure 4.8: **Ablation studies for diverse robotics tasks.** *Error* denotes the ℓ_2 norm between the final and target configurations. *Total Cost* refers to the value of the cost function for the obtained solution. *Time* indicates the net algorithmic runtime required to find the solution.

TT guidance. CMA-ES is a gradient-free, sampling-based method well-suited to robotic problems with nonlinear objectives and obstacle avoidance constraints. As shown in Figure 4.11, TTTS, MCTS, and CMA-ES all recover the global optimum, whereas TTGO does not, as it lacks UCB-driven exploration and therefore does not achieve asymptotic convergence. Among the three successful methods, TTTS is the fastest, confirming that TT guidance substantially accelerates the search.

4.6.2 Motion Planning Around Obstacles

We further applied our framework to a motion planning problem in which a robot must generate a smooth, collision-free trajectory from a given start to a goal configuration. The problem is formulated over T time steps, where the continuous trajectory is represented using basis functions as $\boldsymbol{u}^{[T]} = \sum_{b=1}^{B} \boldsymbol{\Psi}^b \boldsymbol{w}^b$, and the optimization variables are the corresponding weights $\{\boldsymbol{w}^b\}_{b=1}^{B}$. The objective is to minimize a cost function that encourages smooth motion (e.g., penalizing velocity or acceleration), while ensuring accurate goal reaching. The constraints include joint limits and collision avoidance with static obstacles.

Figure 4.9 (A) presents our first test scenario: a 3-joint manipulator reaching task,

which we refer to as *MP1*. This task is particularly challenging because the target lies on the opposite side of the robot, with two circular obstacles obstructing the direct path. The trajectories found by our approach exhibit multiple solution modalities under the same initial configuration and target. Notably, the first two trajectories shown in the figure require the manipulator to initially move away from the target and then pass through a narrow passage between the obstacles, which is not easy to find and requires long-horizon anticipation. This setting involves numerous local optima and a narrow feasible solution space, necessitating long-horizon planning rather than short-horizon control. The results illustrate our framework's ability to overcome local optima and support long-term decision-making. We further applied our approach to a 7-joint manipulator reaching task, which we refer to as *MP2*. The robot arm must move its end-effector from one level of the shelf to another while avoiding obstacles, such as the shelf frame, across its entire body surface. This setup is representative of daily tasks such as pick-and-place or bookshelf arrangement. Figure 4.1 (B) shows the resulting trajectories, where end-effector paths are visualized as curves. Different colors indicate distinct solutions discovered by the algorithm.

Figure 4.8 (B) presents the ablation study of TTTS applied to motion planning (MP) problems. As the three stages of TTTS are applied, both the final state error and the total trajectory cost consistently decrease, demonstrating the effectiveness of each component. The final errors for both tasks are zero, indicating that a collision-free joint trajectory was successfully found to reach the target. An interesting observation is that TT-Tree Initialization takes longer for task *MP1* than for *MP2*, as *MP1* is a more challenging problem with more local optima. Accordingly, we set $r_{\max} = 41$ for *MP1* and $r_{\max} = 21$ for *MP2*. The TT approximation in *MP1* achieves high accuracy, and further iterations of TT-MCTS provide limited improvement. This suggests that, with sufficient storage capacity, we can use higher TT ranks for more accurate tree approximation, accelerating online inference. In contrast, the results of *MP2* highlight the effectiveness of TTTS under limited storage conditions, where low-rank approximations still capture informative representations of complex decision trees. The integration of compact low-rank approximation with UCB-driven exploration enables TTTS to effectively search complex, multimodal solution space and reliably converge to high-quality solutions.

Figure 4.11 compares TTTS with other optimization methods in terms of *Final Error*, *Total Cost*, and *Runtime*. *Final Error* denotes the ℓ_2 norm between the system's final and target configurations. *Total Cost* refers to the value of the cost functions evaluated at the obtained solution, with the cost functions described in Section A.1.2. *Runtime* indicates the net algorithmic runtime required to find the solution, excluding forward rollout evaluations (i.e., physics simulator calls), as rollout speed is task-dependent and does not reflect the computational efficiency of the algorithm itself. From the

Final Error and *Total Cost*, we observe that TTTS achieves results comparable to MCTS when the latter is given sufficient time, highlighting TTTS's ability to find similar global solutions. The *Runtime* further shows that TTTS requires significantly less computation time. TTGO performs well for *MP2*, but it struggles in *MP1* due to the complex cost landscape, which shows TT approximation alone is insufficient to capture all necessary features. In contrast, TTTS leverages strategic tree search to explore the decision space more effectively, achieving global solutions almost surely. CMA-ES also fails in *MP1* because the narrow passage between obstacles challenges its single-modality evolutionary strategy, causing it to be trapped in local optima. These experiments highlight both the computational efficiency of TTTS, enabled by the TT approximation of the decision tree, and its global solution-finding capability, enabled by TT contractions and the strategic search adopted from MCTS.

In addition, we compared TTTS with two widely-used approaches for obstacle-avoidance motion planning: VP-STO (Jankowski et al., 2023) and the Probabilistic Roadmap combined with trajectory optimization (PRM+TO) (Gasparetto et al., 2015). To evaluate performance, we randomly generated five targets and applied the approaches to compute the joint trajectories required to reach them. The comparison results, including *reaching error*, *total cost*, and *computation time*, are reported in Figure 4.10. We also visualize the manipulator trajectories for one of the targets, where TTTS and PRM+TO both find a solution, while VP-STO struggles with the narrow passage. From the table, we observe that TTTS achieves performance comparable to PRM+TO while requiring less computation time, thereby demonstrating its computational efficiency. In contrast, VP-STO results in higher error due to its reliance on a good initial guess and its inability to handle multimodal solution spaces.

4.6.3 Legged Robot Manipulation

In addition to motion planning around obstacles, we also evaluated our approach on a contact-rich manipulation task using a legged robot (Zhu et al., 2023). As illustrated in Figure 4.1(C), a solo robot manipulates a cube of size 10×10 cm under a joint impedance controller in the Genesis simulator (Zhou et al., 2024). The cube, weighing 0.05kg, is required to pivot about the y-axis by a certain angle. This task is challenging because the robot must coordinate contact interactions with the object while maintaining stability, handle the hybrid dynamics arising from intermittent contacts, and plan a smooth motion under impedance control. Small errors in trajectory generation can cause the cube to slip, fail to pivot, or destabilize the supporting leg. To solve this task, we define the cost function as a weighted sum of the terminal pose error and the average cube velocity, with details provided in the appendix. During planning,

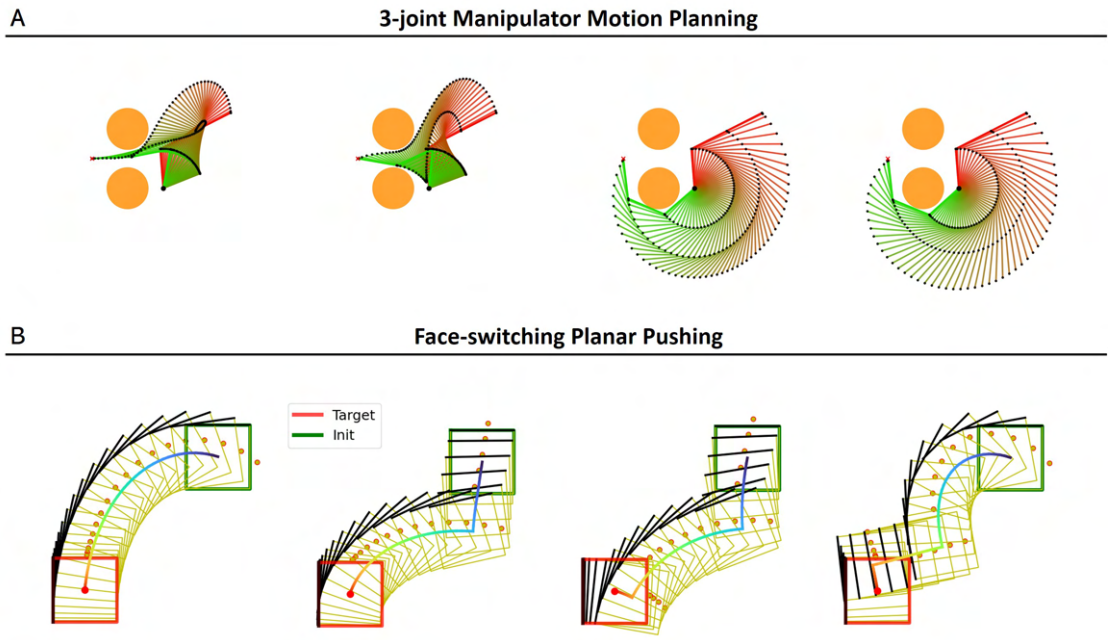


Figure 4.9: **Multi-modal solutions for motion planning around obstacles and face-switching planar pushing.** (A) 3-joint manipulator motion planning with multimodal solutions. The trajectory is visualized using a red-to-green color spectrum to indicate temporal evolution. (B) Face-switching planar pushing task with multimodal solutions. Given the same initial configuration $[0.25, 0.25, -\frac{\pi}{2}]$ and target $[0.25, 0.25, -\frac{\pi}{2}]$, our algorithm can find diverse solutions to accomplish the task, by jointly optimizing over discrete contact faces and continuous motion variables. The black edge of the rectangle indicates the cube's orientation. The number of face switches varies from 0 to 2.

we optimize the Cartesian-space position trajectory for each leg tip. Both legs are then controlled via impedance control with default parameters of $K_p = 100$ N/m and $K_v = 10$ N·s/m. To enable stable pivoting against gravity, we increase the stiffness of the left leg to $K_p = 2000$ N/m along the y- and z-axes.

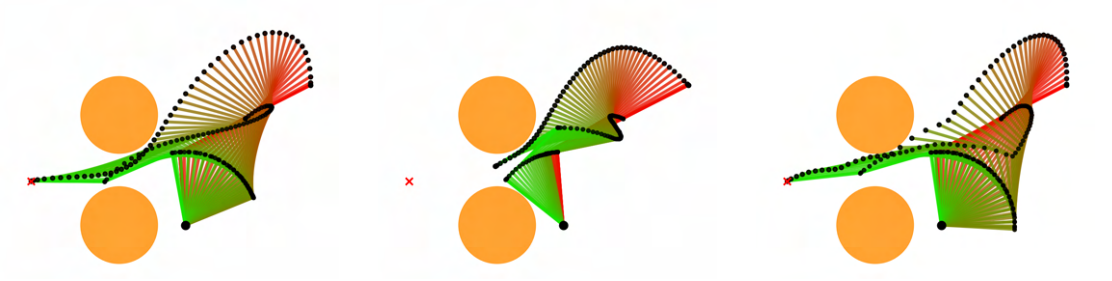
Figure 4.11 reports the comparison of TTTS with other methods on this task. TTTS and MCTS achieve similar performance in terms of reaching error and total cost, highlighting TTTS's asymptotic completeness comparable to MCTS, while TTTS requires significantly less computation time owing to TT approximation of the decision tree. In contrast, TTGO and CMA-ES perform worse. This further underscores the importance of TT factorization and strategic search for effectively solving this challenging task.

4.6.4 Multi-stage Motion Planning

Multi-stage motion planning encompasses a broad class of realistic robotic problems, such as multi-stage forceful manipulation (Holladay et al., 2024) and multi-primitive sequencing (Xue et al., 2024a). The objective is to generate a trajectory that enables a robot to interact intelligently with its environment, typically involving contact mode switches (e.g., sticking, sliding, or transitioning between different manipulation primitives). In this article, the trajectory is parameterized by a sequence of basis weights $\boldsymbol{w}_k$, such that the continuous state at each stage is given by $\boldsymbol{u}_k = \boldsymbol{\Psi}_k \boldsymbol{w}_k$. Discrete modes m_k represent different contact or task-specific phases. The optimization aims to minimize a total cost consisting of smoothness penalties, control effort, and task-specific objectives, while satisfying constraints such as collision avoidance, dynamics, and mode transitions.

We use face-switching planar pushing (Doshi et al., 2020; Xue et al., 2023b) as a representative hybrid manipulation task. In this problem, a robot needs to push a cube from an initial configuration to a target configuration under underactuated dynamics, while deciding both the continuous end-effector velocity trajectory and the discrete contact-face sequence. This task is difficult because the nonlinear dynamics and hybrid discrete-continuous decision structure pose significant challenges to both gradient-based and sampling-based optimization methods. The purpose of this experiment is to evaluate whether our approach can efficiently solve hybrid decision-making problems involving nonlinear dynamics. We formulate this task as an optimization problem with 4 discrete contact-face variables, each taking values in $\{0, 1, 2, 3\}$, and 4 continuous B-spline control weights that parameterize the continuous motion using basis functions. After discretizing each continuous dimension into 30 grid points and allowing up to 4 contact-face selection stages, each with 4 possible face choices, the resulting MINLP has a search space of $4^4 \times 30^4$ grid points. As shown in Fig. 4.9(B), our method discovers multiple cube trajectories for the same initial and target configurations, highlighting the multimodal structure of the solution space. These solutions exhibit different numbers of face switches, ranging from 0 to 2, together with diverse and smooth continuous trajectories.

Figure 4.8 (C) presents the ablation study for this task. The TT approximation of the decision tree is obtained with a rank setting of $r_{\max} = 41$. The results show that the TT approximation alone is not sufficiently accurate to capture the full landscape of the objective function. However, the subsequent tree search significantly improves the solution quality, and the final refinement step enables convergence to the global optimum. An interesting observation is that, compared with the policy learning formulation (i.e., infinite-horizon dynamic programming) proposed in (Xue et al., 2024b,



(a) TTTS (b) VP-STO (c) PRM+TO

TTTS			VP-STO			PRM+TO		
Error	Total Cost	Comp. Time (s)	Error	Total Cost	Comp. Time (s)	Error	Total Cost	Comp. Time (s)
0.00 ± 0.00	0.56 ± 0.74	1.00 ± 0.03	0.07 ± 0.13	3.67 ± 6.85	2.52 ± 2.43	0.00 ± 0.00	0.41 ± 0.51	1.87 ± 0.34

Figure 4.10: Comparison of TTTS, VP-STO, and PRM+TO for motion planning around obstacles. The red cross indicates the reaching target. TTTS and PRM+TO successfully generate optimal manipulator trajectories, while VP-STO fails in the narrow passage. The table reports reaching error, control cost, and computation time, highlighting TTTS's superior efficiency.

2025b), this finite-horizon planning formulation requires a significantly higher TT rank to accurately represent the objective function. This is because trajectory-level planning uses decision variables (i.e., basis weights) that influence the entire trajectory, where small changes can induce large, structured variations, necessitating higher representational capacity to capture complex temporal dependencies. This further motivates the necessity of combining TT approximation with tree search, rather than relying on TT alone.

Figure 4.11 compares TTTS with other approaches. CMA-ES struggles with this problem (indicated by diagonal hatching in the bar chart) because it involves both discrete and continuous decision variables. Both TTTS and MCTS successfully reach the final target, but TTTS requires less computation time owing to the TT factorization of the decision tree. TTGO is highly efficient in terms of runtime, but its solutions are less accurate due to the absence of the strategic search incorporated in TTTS. Overall, these experiments highlight both the computational efficiency and the global solution-finding capability of TTTS.

4.6.5 Comparison with Neural-Guided MCTS and MINLP Solvers

We benchmark TTTS against two competitive baselines: neural-guided MCTS (Silver et al., 2017) and Differential Evolution (DE) for MINLP (Storn and Price, 1997; Virtanen

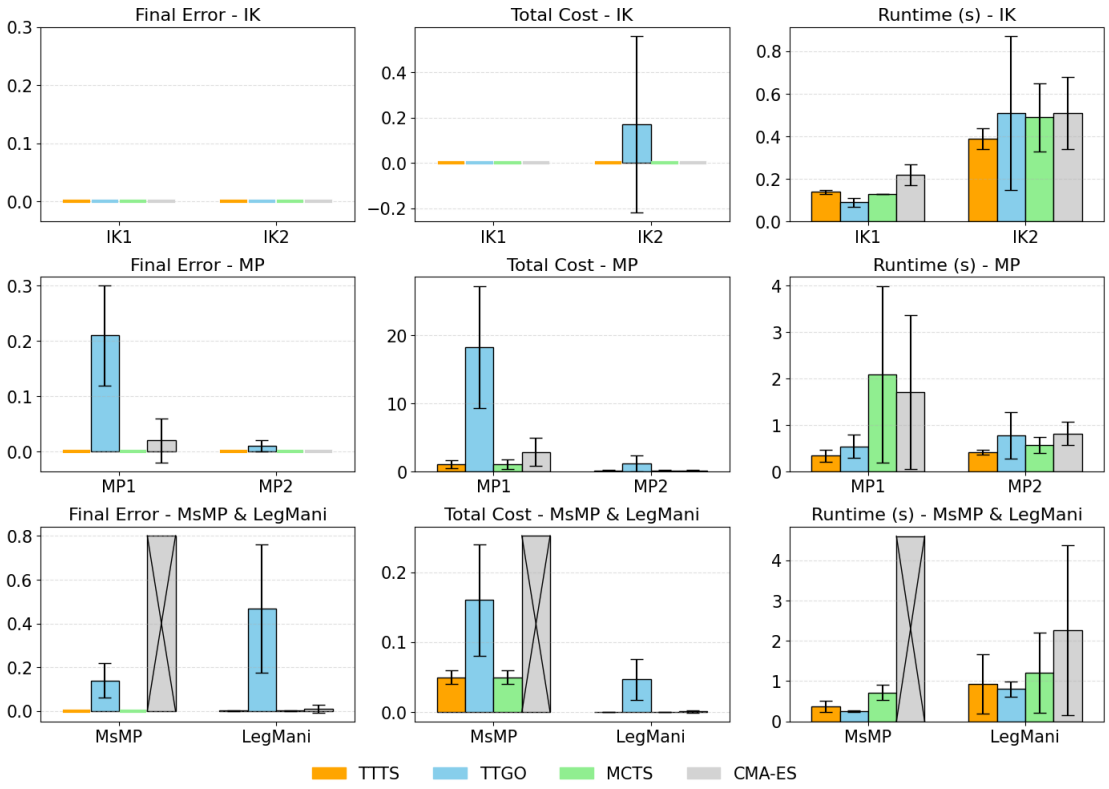
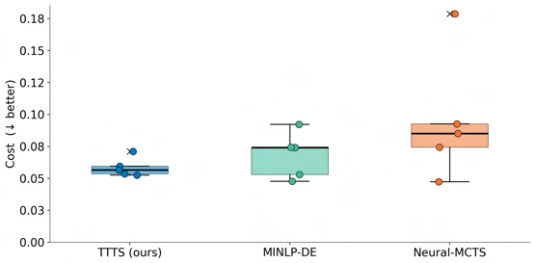


Figure 4.11: Comparison of TTTS, TTGO, CMA-ES, and MCTS with respect to final reaching error, total control cost, and runtime. *IK1* and *IK2* represent the 3-joint and 7-joint inverse kinematics tasks, respectively. *MP1* and *MP2* correspond to the 3-joint and 7-joint manipulator reaching tasks with obstacle avoidance. *MsMP* denotes the planar pushing task with a face-switching mechanism, and *LegMani* refers to the legged robot manipulation (cube pivoting) task. Diagonal patterns in the bar charts indicate that CMA-ES is incompatible with *MsMP* task.

et al., 2020). Evaluations are conducted on a planar pushing task with five varying initial slider poses.

We note that AlphaZero-style methods are powerful in highly challenging settings with large decision spaces, such as Go, but require substantial training investment. The robot optimization problems considered in this work are individually of moderate difficulty, yet span a broad and heterogeneous range of task types, making it impractical to invest substantial training effort for each task. TTTS is well suited to this setting, as it replaces sample-inefficient RL training with TT approximation, which can effectively exploit the inherent structure of the objective function and enable much more efficient pretraining.



(a) Box plots of final trajectory cost across five test scenarios. TTTS achieves consistently lower and more stable costs than both Neural-MCTS and MINLP-DE.

Method	Pretraining	Online Search
Neural-MCTS	4.77 h	26.1 ± 0.1 s
MINLP-DE	0	693.4 ± 7.0 s
TTTS (ours)	**10.5 s**	$\mathbf{3.1 \pm 0.0}$ **s**

(b) Comparison of computational time, including pretraining time and per-scenario online search time. TTTS achieves substantially faster pretraining than Neural-MCTS and faster online search than MINLP-DE.

Figure 4.12: **Comparison of solution quality and computational efficiency on the planar pushing task,** including TTTS (ours), Neural-MCTS, and MINLP-DE. (a) Cost distributions show that TTTS achieves higher-quality and more consistent solutions than the baselines. (b) Timing comparisons show that TTTS requires substantially less offline preparation and online computation; in contrast, Neural-MCTS incurs heavy offline training, while MINLP-DE requires prolonged online search.

Figure 4.12 summarizes the results. In terms of solution quality, Neural-MCTS yields higher variance than TTTS. It excels in four scenarios but fails on one, yielding a cost more than $4\times$ worse. This indicates that networks pretrained for over 4.77 hours still struggle to generalize across all task instances (though performance may improve with further pretraining). In contrast, TTTS requires only 10.5 seconds of pretraining, yet can already approximate the full decision tree well, facilitating downstream online search. MINLP-DE has no pretraining stage and achieves moderate solution quality after prolonged online search. The results highlight the core advantage of TTTS: it actively exploits correlations among decision variables across the entire search space to efficiently construct a compact and global TT surrogate. This surrogate then guides tree search toward low-cost regions with far fewer objective evaluations than MINLP-DE requires.

4.6.6 Analysis of Offline-Online Computation Budget

Fig. 4.13 illustrates the convergence behavior of MCTS under four offline TT-Cross pretraining budgets, measured by final pose error plotted against online MCTS iterations. The results reveal a clear trade-off: heavier offline pretraining (larger TT rank r) yields a substantially better warm-start for online search, enabling faster convergence and fewer iterations to reach the success threshold (dashed line). Specifically, the Heavy budget (r=50) produces an initial solution close to the threshold at iteration 1 and

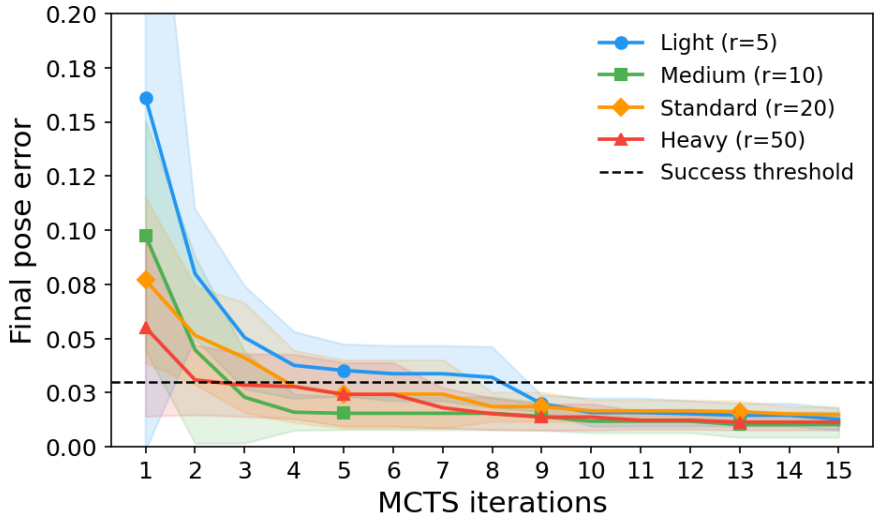


Figure 4.13: **Trade-off between offline TT-Cross pretraining and online MCTS search for planar pushing task.** Final pose error is plotted against online MCTS iterations for four pretraining budgets (TT rank $r \in \{5, 10, 20, 50\}$). Heavier pretraining (larger r) yields a better warm-start and faster convergence to the success threshold (dashed line); all budgets ultimately achieve comparable accuracy below this threshold.

crosses it within one additional iteration, while the Light budget (r=5) requires approximately 9–10 iterations to cross the same threshold. Importantly, all budgets ultimately converge to comparable low-error solutions (below 0.02 m) given sufficient online iterations, demonstrating that the method remains applicable even under constrained hardware resources, at the cost of a larger online search budget. This flexibility allows practitioners to shift computation between the offline pretraining phase and the online computation phase according to available resources, making the approach suitable for a wide range of deployment scenarios. Furthermore, since TT approximation is inherently compact and requires only linear memory to approximate the full exponential tree, TTTS is naturally suited for memory-limited hardware.

4.6.7 Model Predictive Control for Bimanual Whole-body Manipulation

We evaluate our approach using a Model Predictive Control (MPC) formulation applied to a bimanual whole-body manipulation task. This task is particularly challenging due to complex contact dynamics between objects, the whole-body geometry of the robot, and interactions with the environment, all of which make accurate modeling difficult. Physical simulators such as MuJoCo (Todorov et al., 2012) and IsaacGym (Liang et al., 2018) can help address these challenges. However, the resulting sim-to-

real gap necessitates the use of real-time MPC. Given the simulator as a black-box forward dynamics model, sampling-based MPC becomes a promising approach, as it does not rely on explicit gradient information. Nevertheless, such methods often suffer from high sample complexity and slow convergence, limiting their practicality for real-world deployment. In this experiment, we aim to demonstrate that TTTS can quickly find high-quality solutions to support real-time, sampling-based MPC. Specifically, we use Genesis (Zhou et al., 2024) as the simulator due to its parallel simulation capabilities, and the number of environments is set to 500.

Figure 4.14 illustrates the performance differences between TTTS, TTGO, MCTS, and CMA-ES. The computation time is limited to 1 second. A task is considered successful if the angular error at the final timestep satisfies $|\theta - \theta^*| < 3°$, where θ is the object's final z-axis orientation and θ^* is the desired target angle. We evaluate five randomly generated configurations and report success rate, final state error, and total trajectory cost. TTGO achieves a decent success rate with a low TT rank ($r_{\max} = 10$), but its limited accuracy leads to occasional failures. MCTS offers asymptotic completeness guarantees but requires more time, making it impractical for real-time MPC. CMA-ES also performs poorly due to slow convergence. In contrast, TTTS first approximates the decision tree in TT format, which accelerates convergence toward promising regions. It then performs a TT-based tree search, enabling efficient exploitation and strategic exploration. The results, including final error and total cost, confirm the effectiveness of TTTS in supporting real-time MPC for contact-rich manipulation.

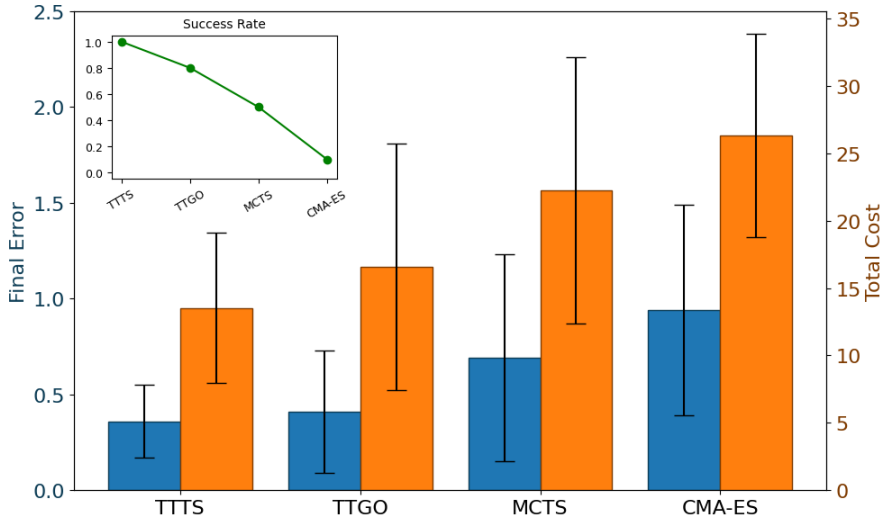
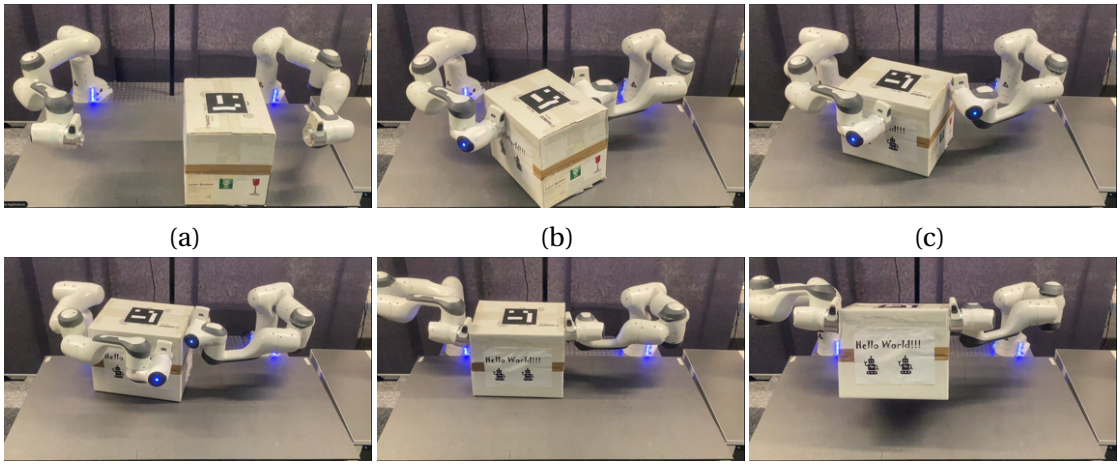


Figure 4.14: **Comparison for bimanual whole-body manipulation.** The blue bars represent the final state error achieved by different methods, while the orange bars correspond to the total cost.



(a) (b) (c)

(d) (e) (f)

Figure 4.15: **Keyframes of bimanual whole-body manipulation.** The system is initialized as (a), and the objective is to rotate the box $90°$ and then lift it as (f). We can observe that the robots can actively exploit whole-body geometry to make and break contacts with the object, completing the overall task successfully.

4.6.8 Real-world Experiments

We validated our approach in the real world through the whole-body bimanual manipulation task. Two 7-DoF Franka robots and a RealSense D435 camera were used. The manipulated object was a large box with the size of $36\text{cm} \times 26\text{cm} \times 34\text{cm}$, which is representative of objects commonly found in warehouse applications. The task was to rotate the box to a target orientation and then lift it. It involved complex contact interactions among the robots, the object, and the table, as well as the full-body surface geometry of the two robot arms, making the system particularly difficult to model.

To address this, Genesis (Zhou et al., 2024) is utilized to predict the box trajectory given a sequence of control commands, thereby eliminating the need for manual modeling of the complex system dynamics. Initially, a low-rank TT approximation augmented with the object pose is obtained via TT-Cross. During execution, conditioned on the current object pose, a tree search is performed for 3 iterations. This typically yields effective results due to the guidance provided by the TT approximation.

Following a model predictive control (MPC) paradigm to bridge the sim-to-real gap, a 9-second trajectory is generated at each 1-second time step. Figure 6.12 presents keyframes from a representative task of rotating the box by $90°$. Notably, the robot's geometry is actively utilized to establish contacts with the object, enabling whole-body bimanual manipulation. Furthermore, a comparative evaluation between TTTS

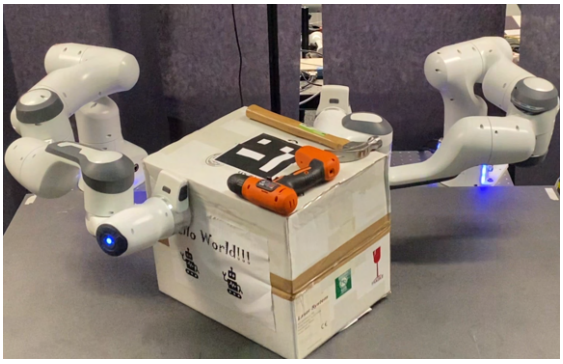


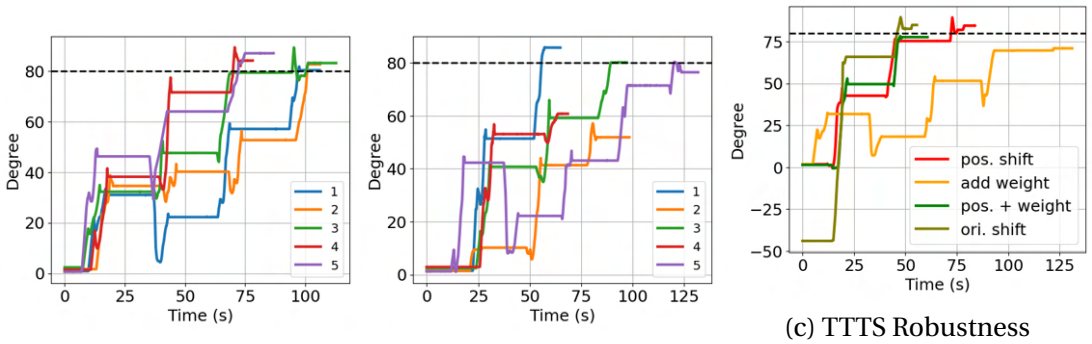
Figure 4.16: **Experimental setup with mass perturbation.** A 1.2 kg external mass is introduced to evaluate the robustness of TTTS against model mismatch between the real world and the Genesis simulation.

and CMA-ES was conducted across five trials each, with the results summarized in Figure 4.17a and Figure 4.17b. Given a tolerance of $10°$, TTTS achieves a 100% success rate, whereas CMA-ES achieves only 40% under similar computational budgets. This highlights the superior sampling efficiency of TTTS, which stems directly from the tensor factorization.

To evaluate the robustness of the proposed method, we further tested it under several variations, including changes in the initial position, adding weight to the box (Figure 4.16), combining both variations, and altering the initial orientation. The corresponding box trajectories are presented in Figure 4.17c. We observe that changes in the initial position and orientation have little effect on TTTS performance, thanks to the feedback mechanism embedded in MPC. However, adding weight results in slightly worse performance because of the sim-to-real gap. Nevertheless, even with this gap, the box can still be reoriented by more than $70°$, which demonstrates the robustness of our approach under model uncertainty.

4.7 Discussion

In this chapter, *Tensor Train Tree Search* (TTTS) is presented as a structure-aware framework for robot optimization. The key idea is to approximate the decision tree in TT format, enabling efficient optimization over large combinatorial decision spaces while maintaining linear complexity. Its effectiveness is demonstrated across a range of domains, including inverse kinematics, obstacle-avoidance motion planning, planar pushing with face switching, and legged robot manipulation. Furthermore, a real-world bimanual whole-body manipulation experiment highlights the practical efficiency of



(a) TTTS Results (b) CMA-ES Results (c) TTTS Robustness Evaluation

Figure 4.17: **Statistical analysis of real-world bimanual whole-body manipulation.** The objective is to rotate the box by $90°$. A trial is considered successful if the box is rotated beyond $80°$ (indicated by the dashed line in the plots). (a) and (b) present the box orientation trajectories produced by TTTS and CMA-ES, respectively, showing that TTTS achieves a significantly higher success rate. To further assess TTTS robustness, additional experiments were conducted by varying initial position, adding weight on the box, combining both variations, and altering the initial orientation, as illustrated in (c).

TTTS for fast sampling-based model predictive control.

Despite these promising results, several challenges remain in further improving the scalability of the proposed approach. TTTS is currently formulated under a deterministic action execution assumption, where each action sequence yields a consistent system output, which is a natural assumption for standard model-based planning and control. However, this assumption may not hold in the presence of real-world stochasticity. Looking ahead, extending TTTS to stochastic settings (e.g., via chance-constraint formulations, robust cost objectives, or belief-space tree search) represents a promising direction for further broadening its applicability.

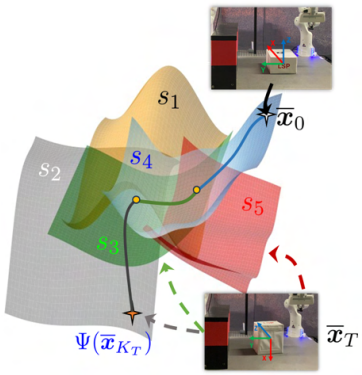
In this work, tensor networks in the standard Tensor Train (TT) format is used to capture the structure in the objective function, enabling efficient optimization and compact representation. However, more expressive representations may be explored to capture richer dependencies in complex scenarios. Hierarchical Tensor Networks (Grasedyck, 2010; Guo et al., 2026) provide a promising extension by introducing a multi-resolution representation, in which high-dimensional functions are decomposed across multiple scales with a tree-structured hierarchy. Such hierarchical formulations align naturally with the layered structure of decision processes and may enable coarse-to-fine reasoning, as well as more efficient branch-and-bound strategies for global optimization.

Another limitation arises from the reliance on computationally expensive system rollouts during planning and control. In the current implementation, the Genesis simulator is used to provide accurate physics and modeling flexibility, but rollout speed remains a bottleneck for tasks requiring extensive sampling. Integrating learned dynamics models as lightweight surrogates offers a promising direction, potentially enabling significantly faster rollouts and improving the overall scalability of the framework.

Overall, this chapter establishes tensor networks as a structured approximation framework for efficient robot optimization, thereby laying a principled foundation for the planning, control, and learning methods developed in the subsequent chapters of this dissertation.

5 Sequential Contact-rich Manipulation Planning using Tensor Train

In the previous chapter, we introduced Tensor Train (TT) as a structured approximation for efficient optimization in classical motion planning and optimal control. Here, we extend this framework to Task and Motion Planning (TAMP) for sequential contact-rich manipulation with hybrid contact modes. We formulate the problem as sequential skill planning, using TT to approximate the value function of each skill in a task-agnostic skill library. An interleaved search-and-optimization framework is then proposed to jointly identify the skill skeleton and subgoal sequence. By casting TAMP as sequential skill planning, the proposed method improves computational tractability and enables reliable execution under contact uncertainty and external disturbances.



Publication Note

The material presented in this chapter is adapted from:

- Xue, T., Razmjoo, A., Shetty, S., and Calinon, S. (2024a). Logic-skill programming: An optimization-based approach to sequential skill planning.
- Shetty, S., Xue, T., and Calinon, S. (2024c). Generalized policy iteration using tensor approximation for hybrid control. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Supplementary materials related to this chapter are available at: https://sites.google.com/view/lsp4plan.

5.1 Introduction

Consider the following task:

> A large box is positioned on the table, next to a wall. The objective is to reorient the box to a new 6D pose using a single robot manipulator with minimal control efforts.

A potential solution involves pushing the box until it reaches the wall, then pivoting against the wall, followed by pulling it to the target. It is noteworthy that the objective is only given in terms of the evaluation of the final geometric configuration, and potential control costs. Figure 5.1 illustrates the planning landscape of the sequential manipulation problem.

Such tasks are quite common in robot manipulation scenarios, typically involving the sequencing of multiple manipulation primitives, such as `push`, `pivot`, and `pull`, to achieve a long-horizon target with sparse rewards. Solving these tasks requires reasoning about the appropriate sequence of primitives and corresponding motion trajectories. This hybrid structure results in combinatorial complexity, making it expensive to find a solution. To address this challenge, Mixed-Integer Programming (MIP) (Marcucci and Tedrake, 2019) is an intuitive approach that does not require a careful design of the system model. While it works well with convex optimization formulations, such as graph of convex sets (Marcucci et al., 2023), manipulation tasks involving interactions with the surroundings usually violate this assumption. Another approach is to implicitly model the hybrid system with complementary constraints (Posa et al., 2014; Moura et al., 2022). By uncovering the internal structures of different manipulation primitives, this method eliminates the need for integer variables in problem formulation, allowing the use of continuous optimization techniques. However, it often leads to poor local optima.

A more general approach involves using logic as a combinatorial expression of possible manipulation primitives (Toussaint et al., 2018). This approach follows the principles of MIP but extends the formulation to the first-order logic level, allowing the utilization of powerful classical AI techniques. This idea is widely used in Task and Motion Planning (TAMP) (Garrett et al., 2021), where the objective is to find a feasible or optimal plan for a complicated task given full knowledge of the system and environments. Various approaches have been proposed to solve TAMP problems, including PDDLStream (Garrett et al., 2020) and Logic-Geometric Programming (Toussaint, 2015), demonstrating high performance in diverse scenarios, such as construction assembly (Zimmermann et al., 2020; Hartmann et al., 2022), table rearrangement (Quintero-Pena et al., 2023), and mo-

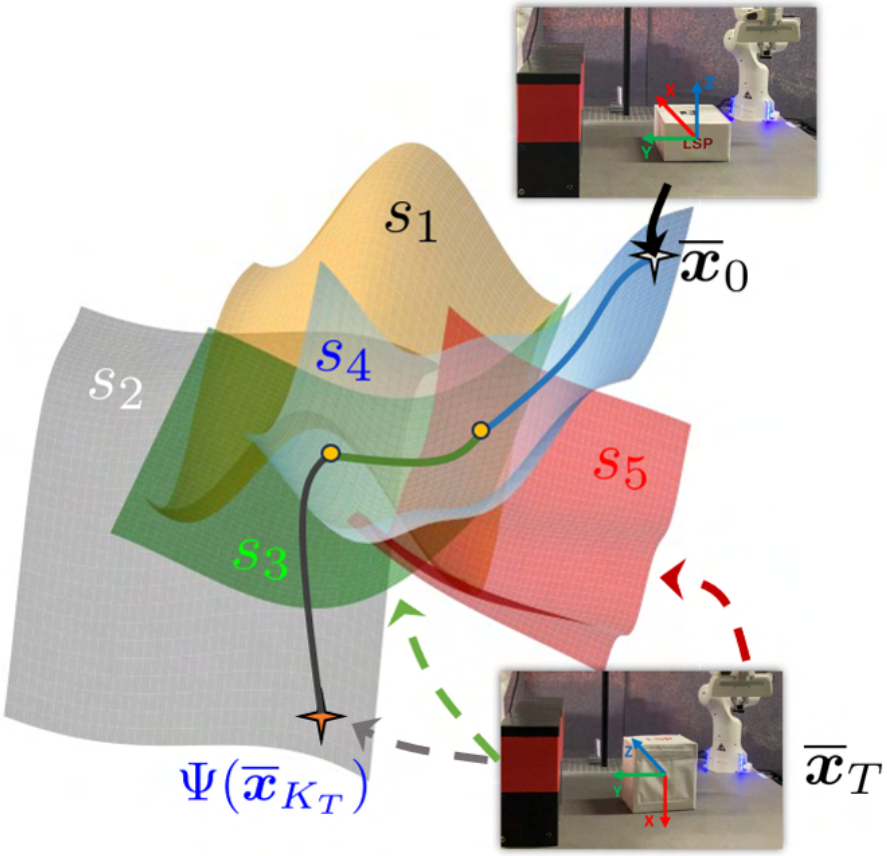


Figure 5.1: Illustration of a sequential manipulation planning landscape. The task is to sequence multiple manipulation skills to reach the target configuration $\overline{x}_T$, resulting in a highly nonlinear landscape composed of multiple manifolds (S_1–S_5). Each manifold corresponds to a different interaction mode or symbolic state. An optimal solution requires identifying a trajectory that transitions across multiple manifolds (e.g., pushing, pivoting, and pulling) to reach the target configuration. The insets show the initial and target configurations in a real robotic setup.

bile manipulation (Yang et al., 2023b). However, they often exhibit poor performance in many realistic scenarios involving model uncertainty and external disturbances. For instance, considering the task introduced at the beginning, although we can derive an offline feasible/optimal trajectory using predefined contact parameters like friction coefficients, this trajectory can be totally wrong because we cannot know exactly the contact model between the robot, objects, and the environment.

Thanks to the advances in learning-based techniques, it is now possible to learn a powerful skill policy for each specific manipulation primitive. The policy can be seen as a short-horizon model predictive controller (MPC) with a good terminal cost function, showing robust and reactive behaviors in the face of model uncertainty and external disturbances. When compared with methods that require online planning (Xue et al., 2023b; Hogan and Rodriguez, 2020b), policy-based methods (Sutton and Barto, 2018; Chi et al., 2024) usually exhibit good performance in physical manipulation tasks that need to cope with contact uncertainty.

We can construct a skill library composed of multiple task-agnostic policies trained independently. This allows for the iterative augmentation of the skill library in a lifelong manner. However, sequencing such task-agnostic policies remains an open problem. Two questions should be considered:

1) Within the library, multiple skills are available. To tackle an unseen, long-horizon manipulation task, the questions arise: which skills should be employed, and what is the correct order?

2) The acquired skills are task-agnostic. Each skill depends on a specific subgoal to generate the appropriate action given the current state. How to determine the subgoal sequence given a skill skeleton?

There has been prior work aimed at addressing these questions (Agia et al., 2023; Huang et al., 2019; Wu et al., 2021). However, most existing approaches necessitate a well-defined task planning problem with an explicit symbolic goal description, followed by sampling-based methods to check whether the symbol-defined constraints can be satisfied. This limitation restricts the applicability of such methods to tasks that do not specify a symbolic goal and require an optimal path. These tasks are, in fact, optimization problems rather than constraint satisfaction problems.

To address these issues, we propose Logic-Skill Programming (LSP), an optimization-based approach designed to eliminate the need for explicit symbolic goal descriptions for sequential skill planning. This work draws inspiration from Logic-Geometric Programming (Toussaint, 2015) and shares the same philosophy, with a key distinction that

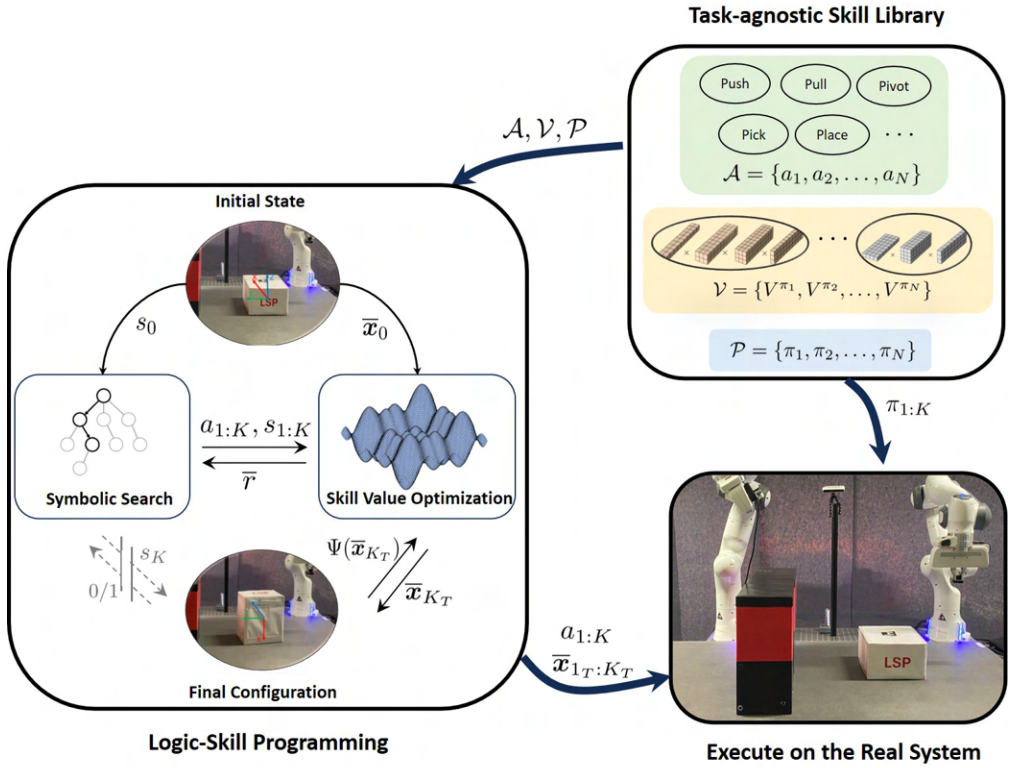


Figure 5.2: Overview of Logic-Skill Programming. Given the evaluation function Ψ of the final configuration, along with the initial symbolic state s_0 and geometric state $\overline{\boldsymbol{x}}_0$, the objective of LSP is to find a solution that can accomplish the task with minimal control costs. A task-agnostic skill library is pretrained, consisting of N skill operators $\mathcal{A} = \{a_{1:N}\}$, along with corresponding value functions $\mathcal{V} = \{V^{\pi_{1:N}}\}$ and policies $\mathcal{P} = \{\pi_{1:N}\}$ in Tensor Train format. LSP solves this problem by alternating between symbolic search and skill value optimization for joint logic-geometric reasoning. Symbols $s_{1:K}$ are used as constraints for skill optimization, while skill optimization is used to check skeleton feasibility and final configuration performance, with a feedback reward $\overline{r}$ informing the symbolic search. This results in the appropriate skill skeleton $a_{1:K}$ and subgoal sequence $\overline{\boldsymbol{x}}_{1_T:K_T}$, which are then combined with the skill policies $\pi_{1:K}$ existing in the skill library to actuate the real robot. Notably, the gray channel with symbolic final state s_K is interrupted because our framework eliminates the need for a symbolic target goal s_T, while such information is typically required in existing sampling-based sequential skill planning methods.

we concentrate on skill policy planning, rather than motion planning in a fully-known environment.

Fig. 6.2 is an overview of LSP. Given the initial state s_0 and $\overline{x}_0$, along with the evaluation function Ψ for the final geometric configuration, LSP can generate the solutions to sequence multiple skills from the task-agnostic skill library. This problem is formulated as an extended first-order mathematical program, where first-order logic is introduced to constrain the optimization problem. The objective is to optimize both the overall cumulative reward and the performance of the final geometric configuration. The overall cumulative reward is expressed as the sum of all value functions within the plan. While Reinforcement Learning (RL) algorithms have achieved remarkable success on complex tasks, the resulting value function approximations are typically learned from sampled data, which can lead to limited global accuracy and unreliable estimates outside the explored regions. This limitation is particularly critical in sequential skill planning, where accurate value estimates are required to determine the optimal sequence of subgoals that maximizes the cumulative reward. To address this issue, we propose to approximate value functions in Tensor Train (TT) format, enabling efficient and global approximation over high-dimensional spaces. The appropriate skill skeleton and optimal subgoal sequence are then obtained by alternating between symbolic search and numerical optimization over the associated skill value functions.

We summarize the contributions of this work as follows:

- We propose to formulate the sequential skill planning problem as an extended first-order mathematical program, eliminating the need for symbolic goal description and enabling to find the optimal solutions rather than only feasible ones.

- We propose Logic-Skill Programming to address sequential skill planning tasks by alternating between symbolic search and skill value optimization.

- We propose to use TT to approximate the value functions, aiding in finding the optimal subgoal sequence with maximum cumulative reward respecting system dynamics.

5.2 Related Work

5.2.1 Skill Learning

Methods such as behavior cloning (BC) (Florence et al., 2022) employ deep neural networks to model state–action distributions from offline datasets. This class of methods heavily relies on the quality and coverage of the provided data, and often lacks effective exploration in complex or unseen state spaces. An alternative paradigm is reinforcement learning (RL), which, through the exploration–exploitation mechanism, enables agents to actively explore the state space and discover feasible solutions. However, actions in RL are typically generated either by sampling from the current policy or by locally optimizing value-related functions such as the Q-function or advantage function. In both cases, the resulting exploration is strongly biased by the local structure induced by the current policy or value landscape, leading to predominantly local exploration in the action space.

In this chapter, we aim to achieve global optimality over the entire (logic and geometric) path. To this end, we require a global approximation of the value function over the full state space. Notably, both BC and RL methods fall short in providing this information. This assertion aligns with the motivation of Approximate Dynamic Programming (ADP), where the goal is to approximate the optimal value function for the entire state space. However, this objective is challenging due to the expensive storage and computation involved in solving the dynamic programming algorithms such as value iteration. We argue that this challenge explains why existing literature primarily focuses on feasibility rather than pursuing optimality in sequential skill planning. In this chapter, we address this issue by leveraging Tensor Train (TT) to approximate the state-value function and the advantage function within the ADP framework.

5.2.2 Hybrid Long-horizon Planning

Hybrid long-horizon planning in contact-rich manipulation has been discussed in Chapter 2. Here, we briefly position our approach with respect to existing methods.

Classical approaches such as Mixed-Integer Programming (MIP) and MPCC (Marcucci and Tedrake, 2019; Posa et al., 2014) handle hybrid dynamics through discrete variables or complementarity constraints, but struggle with non-convexity and scalability in complex contact scenarios. Task and Motion Planning (TAMP) (Garrett et al., 2021) extends this paradigm by integrating symbolic reasoning with geometric optimization, including both sampling-based (Srivastava et al., 2014; Garrett et al., 2020) and optimization-based (Zimmermann et al., 2020; Envall et al., 2023) methods. In partic-

ular, optimization-based approaches such as Logic-Geometric Programming (LGP) (Toussaint, 2015) formulate sequential decision making as a first-order extension of a mathematical program, enabling joint logic–geometric reasoning.

Our work is closely related to this line of research. Similar to LGP, we formulate long-horizon manipulation as an optimization problem. However, while LGP primarily focuses on planning under fully known models, our approach explicitly targets the practical execution of plans in contact-rich settings with potential model mismatch. By leveraging skill value functions and symbolic search, it enables the generation of long-horizon plans that are executable under complex contact dynamics and uncertainties.

5.2.3 Sequential Skill Planning

Prior works that focus on sequential skill planning typically adopt the options framework (Sutton et al., 1999), where a high-level policy is trained to sequence low-level skills toward the final goals. For instance, Xu et al. proposed learning a skill proposal network as the high-level policy and utilizes learned skill-centric affordances (value functions) to assess the feasibility of the proposed skill skeleton (Xu et al., 2021). Similarly, Shah et al. suggested using the value functions of low-level skills as the state space to represent the symbolic skill affordance (Shah et al., 2021). This representation is then utilized to train an upper-level RL policy toward long-horizon goals. Although such methods have demonstrated the ability to solve long-horizon tasks, the resulting policies are typically task-specific and exhibit poor generalization on unseen tasks. Moreover, we believe the capabilities of value function has not been fully explored by (Shah et al., 2021; Xu et al., 2021). Instead of using the value functions symbolically, we regard it as an abstraction of cumulative reward, taking into account the system dynamics in geometric level. Subsequently, we leverage it to identify the optimal subgoal sequence that results in the maximum cumulative reward of the full path.

To enhance generalization, several recent works rely on symbolic planning to sequence task-agnostic skills. In (Agia et al., 2023), the product of Q-functions is maximized to ensure the joint success of all skills in a sequence, with the skill skeleton provided by a high-level task planner. Similarly, (Huang et al., 2019) employs a symbolic planner to sequence imitation learning policies, leveraging continuous relaxation to improve symbolic grounding. A key principle in these methods is to formulate a constraint satisfaction problem where the effects of a parameterized skill primitive satisfy the preconditions of the subsequent skill in the skeleton. Such skeletons are either predefined (Mishra et al., 2023) or generated via PDDL planners (Agia et al., 2023), both of which require an explicit symbolic goal description. Consequently, these methods do

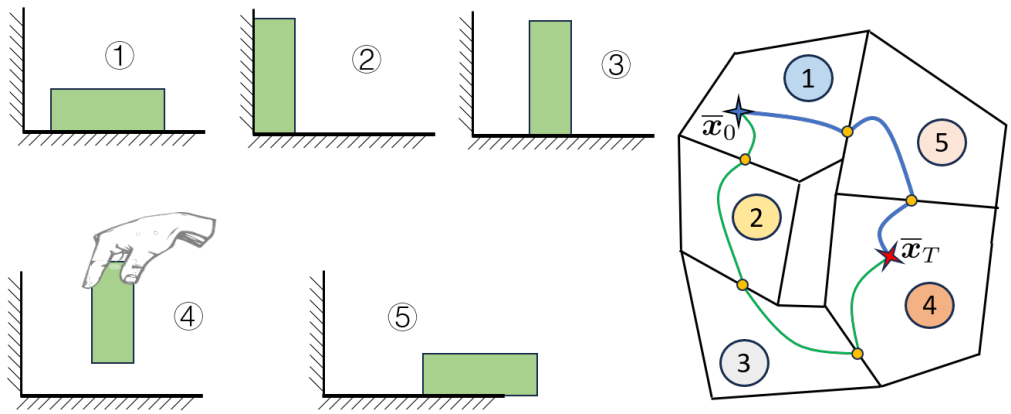


Figure 5.3: An illustrative example of Logic-Skill Programming for sequential manipulation. Left: The symbolic states of a rectangular object that can only be grasped from the side. Right: The graph structure with the discovered optimal logic-geometric paths that connect $\overline{x}_0$ and $\overline{x}_T$. Both kinematic and dynamic constraints are considered to be satisfied.

not optimize over a final configuration defined solely by an objective function, which limits their applicability to the problem addressed in this work.

Therefore, we are motivated to propose a method that eliminates the need for a symbolic goal description while ensuring the maximum cumulative reward of the obtained skill skeleton and motion trajectories. This work aligns with LGP (Toussaint, 2015), except that we plan over skills rather than motions, as we believe skills are more applicable in realistic environments.

5.3 Problem Formulation

Our objective is to address long-horizon manipulation tasks by sequentially executing a series of skills included in a skill library $\mathcal{P} = \{\pi_1, \pi_2, \ldots, \pi_N\}$. To ensure that the learned skills can be sequenced in an arbitrary manner and generalized to any long-horizon tasks, the skills should be task-agnostic, as also described in (Agia et al., 2023). This implies that the learned policy is trained independently, with its own skill-centric parameters. Each skill domain can be modeled by a Markov Decision Process (MDP)

$$\mathcal{M}_n = (\mathcal{X}_n, \mathcal{U}_n, \mathcal{T}_n, \mathcal{R}_n), \tag{5.1}$$

where $\mathcal{X}_n$ is the state space, $\mathcal{U}_n$ is the action space, $\mathcal{T}_n(x_n'|x_n, u_n)$ is the transition model, $\mathcal{R}_n(x_n, u_n)$ is the reward function given current state and action.

The full K-length long-horizon domain is the union of $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_K\}$, where each time segment corresponds to a single skill policy. It is specified as

$$\overline{\mathcal{M}} = (\mathcal{M}_{1:K}, \overline{\mathcal{X}}, \Phi_{1:K}, \Gamma_{1:K}), \tag{5.2}$$

where $\overline{\mathcal{X}}$ is the full state space of the long-horizon domain, $\Phi_{1:K} : \mathcal{X}_k \to \overline{\mathcal{X}}$ is a function that maps the policy state space to the full state space, and $\Gamma_{1:K} : \overline{\mathcal{X}} \to \mathcal{X}_k$ is a function that extracts the policy state from the full state. Note that the state spaces of different skills usually have different dimensions and structures. Taking pushing and pivoting as examples, they are planar manipulation primitives defined in horizontal and vertical planes, respectively, affecting different elements of the object pose after execution. The long-horizon full state space serves as a bridge, transmitting the local state changes from the previous skill to the subsequent one.

After acquiring these task-agnostic policies, our goal is to determine the optimal skill skeleton and subgoal sequence, given the evaluation function Ψ of the final configuration. Each skill $\pi_k(\boldsymbol{u}|\boldsymbol{x})$ is trained to maximize the cumulative reward given the current state $\boldsymbol{x}$. Therefore, finding the optimal skill sequence aims to maximize the overall cumulative reward for the complete skill sequence $a_{1:K}$ and the evaluation of the final configuration $\Psi(\overline{\boldsymbol{x}}_{K_T})$:

$$\max_{\boldsymbol{x}, a_{1:K}} \sum_{k=1}^{K} \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{\infty} \gamma^t R_{k_t} \right] + \Psi(\overline{\boldsymbol{x}}_{K_T}). \tag{5.3}$$

Here, R_{k_t} denotes the reward of skill a_k at time t. However, directly optimizing over the infinite full horizon is impossible. Considering that the value function is defined to approximate the cumulative reward in dynamic programming:

$$V^{\pi_k}(\boldsymbol{x}) = \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{\infty} \gamma^t R_{k_t}(\boldsymbol{x}_{k_t}, \pi_k(\boldsymbol{x}_{k_t})) \mid \boldsymbol{x}_{k_0} = \boldsymbol{x} \right], \tag{5.4}$$

and the objective of each policy is to find an action from the action space that can maximize the cumulative reward from current state:

$$\begin{aligned} Q^{\pi_k}(\boldsymbol{x}, \boldsymbol{u}) &= \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{\infty} \gamma^t R_{k_t}(\boldsymbol{x}_{k_t}, \boldsymbol{u}_{k_t}) \mid \boldsymbol{x}_{k_0} = \boldsymbol{x}, \boldsymbol{u}_{k_0} = \boldsymbol{u} \right], \\ \pi_k(\boldsymbol{x}) &= \arg\max_{\boldsymbol{u}} Q^{\pi_k}(\boldsymbol{x}, \boldsymbol{u}). \end{aligned} \tag{5.5}$$

The sequential skill planning problem can be formulated as maximizing the sum

of value functions, along with the evaluation function Ψ of the final configuration, incorporating first-order logic as constraints:

$$\begin{aligned}
\max_{\overline{\boldsymbol{x}}, a_{1:K}, s_{1:K}} \quad & \sum_{k=1}^{K} V^{\pi_k}(\boldsymbol{x}_{k_0}) + \Psi(\overline{\boldsymbol{x}}_{K_T}) \\
\text{s.t.} \quad & s_k \in succ(s_{k-1}, a_k), \overline{\boldsymbol{x}}_{1_0} = \overline{\boldsymbol{x}}_0, \\
& h_{\text{path}}(\boldsymbol{x}_{k_t}, \pi_k(\boldsymbol{x}_{k_t})|s_k) = 0, \\
& g_{\text{path}}(\boldsymbol{x}_{k_t}, \pi_k(\boldsymbol{x}_{k_t})|s_k) \leq 0, \\
& h_{\text{switch}}(\overline{\boldsymbol{x}}_{k_T}|a_{k+1}, s_k) = 0, \\
& g_{\text{switch}}(\overline{\boldsymbol{x}}_{k_T}|a_{k+1}, s_k) \leq 0, \\
& \overline{\boldsymbol{x}}_{k_t} = \Phi_k(\boldsymbol{x}_{k_t}|a_k), \\
& \boldsymbol{x}_{k_0} = \Gamma_k(\overline{\boldsymbol{x}}_{k_0}, \overline{\boldsymbol{x}}_{k_T}|a_k),
\end{aligned} \tag{5.6}$$

where $\overline{\boldsymbol{x}}_{k_0}$ and $\overline{\boldsymbol{x}}_{k_T}$ are the initial and final configuration of skill operator a_k in the long-horizon domain $\overline{\mathcal{X}}$. The goal configuration of a_k is identical to the initial configuration of a_{k+1}, namely $\overline{\boldsymbol{x}}_{(k+1)_0} = \overline{\boldsymbol{x}}_{k_T}$. The initial state of a_k within the skill domain $\mathcal{X}_k$, denoted as $\boldsymbol{x}_{k_0}$, is computed using $\overline{\boldsymbol{x}}_{k_0}$ and $\overline{\boldsymbol{x}}_{k_T}$. $V^{\pi_k}(\boldsymbol{x}_{k_0})$ therefore represents the cumulative reward from $\overline{\boldsymbol{x}}_{k_0}$ to $\overline{\boldsymbol{x}}_{k_T}$ in $\mathcal{X}_k$. Φ_k and Γ_k are the mapping functions between long-horizon domain $\overline{\mathcal{X}}$ and skill domain $\mathcal{X}_k$. Thanks to Γ_k, given new initializations and targets, there is no need to retrain new value functions. The definition of Γ_k is outlined in Sec. 5.6.1. s_0 and $\overline{\boldsymbol{x}}_0$ denote the initial symbolic state and geometric configuration, respectively. h_{path} and g_{path} indicate the constraints on the path $\boldsymbol{x}_{k_t}$ given current symbolic state s_k. Such constraints are addressed by the skill policies π_k in each skill domain. h_{switch} and g_{switch} express the transition consistency of configuration $\overline{\boldsymbol{x}}_{k_T}$ with the following skill operator a_{k+1}. For example, to `pivot` an object against the wall, the object should be well-positioned in contact with the wall after the previous action. These constraints specify the degrees of freedom in the system configurations that can be actuated under the symbolic state s_k, thereby effectively reducing the number of decision variables in $\overline{\boldsymbol{x}}_{k_T}$ and streamlining the optimization process. For instance, an object marked as `non-graspable` and `onTable` can only be manipulated through `push` or `pull`, with actuation in the dimensions of `x`, `y` and `yall`. Conversely, if the object is marked as `atWall`, it can be actuated by `pivot`, with changes in `roll` or `pitch`.

The value functions of all skills $a_{1:K}$ collectively constitute a value function space, offering insights into optimal control for sequentially executed skills, while considering system dynamics and kinematics. By optimizing over it, we can identify the subgoal sequence that maximizes the cumulative reward while satisfying system constraints.

For instance, as depicted in Fig. 5.7, the value function space can inform the planner which configuration along the table edge is more dynamically optimal, especially considering the under-actuated pushing dynamics.

We assume the initial geometric configuration $\overline{\boldsymbol{x}}_0$ is given, along with an initial symbolic state $s_0 \in \mathcal{L}$. The symbolic states can be transited through the skill operator a_k as $s_k = succ(a_k, s_{k-1})$. The existing skill planning frameworks typically require an explicit symbolic goal target $s_T \models \mathcal{G}$, while our method eliminates this requirement by considering only the evaluation Ψ of the final geometric configuration. Figure 5.3 provides an illustration of this setting. Each symbolic state represents a feasible manipulation mode of the object, while edges correspond to executable manipulation skills that satisfy both kinematic and dynamic constraints. The planning problem can therefore be interpreted as finding an optimal logic–geometric path connecting $\overline{\boldsymbol{x}}_0$ and $\overline{\boldsymbol{x}}_T$.

5.4 Skill Value Function Approximation

To solve (5.6), the first step is to obtain value functions that accurately approximate the cumulative reward within each skill domain. From a control perspective, the value function quantifies the expected return of reaching a state while accounting for both kinematic and dynamic constraints. However, approximating value functions in high-dimensional state–action spaces is challenging due to the curse of dimensionality.

To address this challenge, we propose to leverage tensor train (TT) to approximate both state-value and advantage functions. The state-value function V^π corresponding to a policy π with discount factor γ is defined as

$$V^\pi(\boldsymbol{x}_0) = \sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)), \qquad \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)). \tag{5.7}$$

Given a value function $V : \Omega_{\boldsymbol{x}} \to \mathbb{R}$, the Bellman operator $\mathcal{B}^\pi$ is defined as

$$\mathcal{B}^\pi V(\boldsymbol{x}) = R(\boldsymbol{x}, \pi(\boldsymbol{x})) + \gamma V\Big(f(\boldsymbol{x}, \pi(\boldsymbol{x}))\Big), \qquad \forall \boldsymbol{x} \in \Omega_{\boldsymbol{x}}. \tag{5.8}$$

The corresponding advantage function A_V is

$$A_V(\boldsymbol{x}, \boldsymbol{u}) = R(\boldsymbol{x}, \boldsymbol{u}) + \gamma\Big(V(f(\boldsymbol{x}, \boldsymbol{u})) - V(\boldsymbol{x})\Big), \qquad (\boldsymbol{x}, \boldsymbol{u}) \in \Omega_{\boldsymbol{x}} \times \Omega_{\boldsymbol{u}}. \tag{5.9}$$

The resulting algorithm, referred to as *Generalized Policy Iteration using Tensor Train*

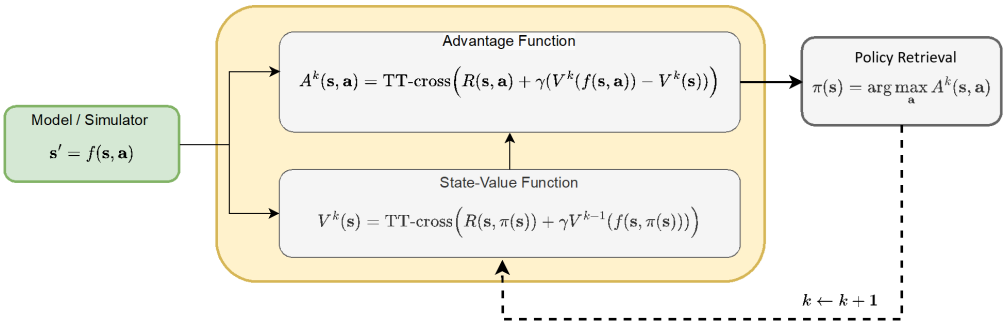


Figure 5.4: Overview of TTPI algorithm. In each iteration, the state-value function and the advantage function are represented in TT format and computed via TT-Cross. This process utilizes the preceding estimates of the value function and the policy, with the updated policy subsequently retrieved from the new advantage function.

(TTPI), is summarized in Algorithm 2, with a pictorial overview illustrated in Figure 5.4.

The value function is initialized to zero and updated iteratively via value iteration. At iteration i, the update proceeds as

$$\begin{aligned} A^i(\boldsymbol{x},\boldsymbol{u}) &= R(\boldsymbol{x},\boldsymbol{u}) + \gamma\Big(V^i(f(\boldsymbol{x},\boldsymbol{u})) - V^i(\boldsymbol{x})\Big), \qquad \forall \boldsymbol{x}, \\ \pi(\boldsymbol{x}) &= \arg\max_{\boldsymbol{u}} A^i(\boldsymbol{x},\boldsymbol{u}), \\ \mathcal{B}^\pi V^i(\boldsymbol{x}) &= R(\boldsymbol{x},\pi(\boldsymbol{x})) + \gamma V^i(f(\boldsymbol{x},\pi(\boldsymbol{x}))), \\ V^{i+1} &= \text{TT-Cross}(\mathcal{B}^\pi V^i, \epsilon), \end{aligned} \tag{5.10}$$

where $A^i(\boldsymbol{x},\boldsymbol{u})$ is the advantage function, $f(\boldsymbol{x},\boldsymbol{u})$ denotes the system dynamics, $R(\boldsymbol{x},\boldsymbol{u})$ is the reward function, and ϵ is the accuracy threshold for TT-Cross approximation.

5.5 Logic-Skill Programming

Given the approximated value functions, we solve Eq. (5.6) by alternating between high-level symbolic search and low-level skill value optimization. The symbolic search defines the task structure and associated constraints, while the skill value optimization evaluates the feasibility of each skeleton and the resulting final configuration.

Level 1: Symbolic Search

We use Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998) to describe the task domain and then utilize Monte Carlo Tree Search (MCTS) to

Algorithm 2 TTPI: Generalized Policy Iteration using Tensor Train

Input:
  n_v: Number of value update steps
  ϵ: Accuracy of TT representation
  $r_{\max}$: Maximum TT-rank
  $\delta_{\max}$: Convergence tolerance
  $r(\boldsymbol{x}, \boldsymbol{u})$: Reward function
  Δt: Time Discretization
  $f(\boldsymbol{x}, \boldsymbol{u})$: Forward simulation
  $\hat{\Omega}_{\boldsymbol{x}}$: Discretization of state space
  $\hat{\Omega}$: Discretization of state-action space ($\hat{\Omega}_{\boldsymbol{x}} \times \hat{\Omega}_{\boldsymbol{u}}$)
  N: Number of candidate samples.
Output: Policy π^*
Initialize:
  Initialize in TT-format: $V^0 = 0$
  Initialize Advantage model: $A_{V^0} = \text{TT-Cross}(R(\boldsymbol{x}, \boldsymbol{u}),\ \hat{\Omega},\ r_{\max},\ \epsilon)$
  (alternatively, initialize arbitrarily)
  Set $k = 0$
while $\delta \leq \delta_{\max}$ **do**
    $k \leftarrow k + 1$
    $\pi^k(\boldsymbol{x}) := \underset{a}{\operatorname{argmax}}\, A_{V^{k-1}}(\boldsymbol{x}, \boldsymbol{u})$
    $V_0^k = V^{k-1}$
    for $j \leftarrow 1$ to n_v **do**
      $V_j^k(\boldsymbol{x}) = \text{TT-Cross}(\mathcal{B}^{\pi^k} V_{j-1}^k,\ \hat{\Omega}_{\boldsymbol{x}},\ r_{\max},\ \epsilon)$
    end for
    $V^k = \text{TT-round}(V_{n_v}^k,\ \epsilon)$
    $A^k(\boldsymbol{x}, \boldsymbol{u}) = R(\boldsymbol{x}, \boldsymbol{u}) + \gamma(V^k(f(\boldsymbol{x}, \boldsymbol{u})) - V^k(\boldsymbol{x}))$
    $A_{V^k} = \text{TT-Cross}(A^k,\ \hat{\Omega},\ r_{\max},\ \epsilon)$
    $\delta = \frac{\|V^k - V^{k-1}\|_2}{\|V^{k-1}\|2}$
end while
Set $V^* = V^k$
$\pi^*(\boldsymbol{x}) = \underset{\boldsymbol{u}}{\operatorname{argmax}} A_{V^*}(\boldsymbol{x}, \boldsymbol{u})$

search for the appropriate skill sequence $a_{1:K}$ from the skill library. The preconditions and effects of each skill form a rule-based representation of symbolic transitions $s_k = succ(s_{k-1}, a_k)$, serving as the forward model for symbolic search. In each iteration, MCTS relies on the Upper Confidence Bound (UCB1) (Williams, 2016) to select the node:

$$UCB1(s_k) = \frac{w_{s_k}}{v_{s_k}} + C_E \sqrt{\frac{2\ln\left(v_{s_k^p}\right)}{v_{s_k}}}, \tag{5.11}$$

where v_{s_k} represents the number of visits on symbolic state s_k, and w_{s_k} indicates the total accumulated reward obtained through state s_k. $v_{s_k^p}$ denotes the number of visits on the parent node of s_k. C_E is a constant to balance exploitation and exploration.

If no child node is found in the current branch of the search tree, the branch will expand and simulate until it either reaches the target or surpasses the maximum length. Since our formulation does not have a symbolic goal, we rely on the skill value optimization process (Sec. 5.5) to evaluate the performance of the final geometric configuration. A reward $\overline{r}$ will be backpropagated through the branch, depending on the simulation result. This iterative process refines the search tree, focusing on promising branches and ultimately converging towards an optimal decision. It is important to note that the sequence length (K in Eq. (5.6)) is purely unknown before symbolic search, allowing diverse sequence lengths for the same target configuration. By applying a higher C_E in the UCB1 formula, MCTS can return multiple solutions. For example, as shown in Fig. 5.5b, if we want to grasp the cube that is only graspable from the lateral side, multiple symbolic sequences can be found by symbolic search. One can be `push-pick`, pushing the cube to the edge and then grasping it from the table side, while another solution can be `push-pivot-pull-pick`. Given multiple solutions, we can either choose the most robust one before execution using domain knowledge, or switch between solutions during execution.

Level 2: Skill Value Optimization

The symbolic skill skeleton encodes feasibility only at the symbolic level. Therefore, it must be validated through Eq. (5.6) to ensure that the resulting trajectory satisfies the constraints and reaches a valid final configuration. This step is particularly important in our framework because the objective depends solely on the evaluation function Ψ of the final configuration. Moreover, since the sequenced skills are task-agnostic, appropriate subgoals must be determined for each skill so that the corresponding skill policy can infer where to move from the current state. In our formulation, these subgoals are obtained by solving Eq. (5.6) conditioned on the symbolic skill skeleton.

Given the value functions obtained in Sec. 5.4, we solve Eq. (5.6) for the skill skeleton produced by the symbolic search layer. The resulting subgoals $\overline{\boldsymbol{x}}_{1_T:K_T}$ must satisfy both path constraints and switch constraints. The switching constraints h_{switch} and g_{switch} require that $\overline{\boldsymbol{x}}_{k_T}$ lies in the intersection of two adjacent skill domains, $\mathcal{X}_k$ and $\mathcal{X}_{k+1}$, ensuring configuration consistency between consecutive skills. The path constraints h_{path} and g_{path} are enforced by the corresponding skill policies within each domain. For example, to verify whether a subgoal $\overline{\boldsymbol{x}}_{k_T}$ is reachable while satisfying path constraints, the skill policy π_k is executed in domain $\mathcal{X}_k$ starting from the initial state $\boldsymbol{x}_{k_0}$.

We assume that the discrepancy between the model parameters used during skill policy learning and those in the real world is moderate. With the receding-horizon mechanism during policy execution, the skill policy can reach the target despite modeling errors and disturbances, thereby naturally satisfying h_{path} and g_{path}. Our focus is therefore on sequencing multiple skills to accomplish more complex tasks. In Chapter 6, we further investigate how to learn robust skill policies that can handle the sim-to-real gap.

To design an appropriate optimization technique, it is essential to note that both discrete and continuous variables may be involved in this problem. For instance, in the task mentioned at the beginning, we have to rely on pivoting against the wall to change the `roll` or `pitch` angle of the box. This requires one face of the box to be parallel to the wall. In other words, the `yall` angle of the box has to be in $[-\pi, -\pi/2, 0, \pi/2]$ after pushing. Moreover, the objective function can be in any form, either convex or non-convex, depending on the value functions of selected skills. All of these factors pose significant challenges for optimization techniques. In this work, we employ the Cross-Entropy Method (CEM) (Rubinstein, 1999; De Boer et al., 2005) with mixed distribution, namely CEM-MD, as the optimization technique. It can handle mixed-integer programming by using Gaussian distribution and categorical distribution for continuous and discrete variables, respectively. The distributions are iteratively updated towards the fraction of the population with higher objective scores until converging to the best solution. The pseudocode of CEM-MD is shown in Alg. 3. Note that the C samples in each iteration are generated in batch (lines 5-10) for fast computation.

Overall, our proposed approach is to alternate between symbolic search and skill value optimization. The symbolic search is achieved by MCTS, with the obtained skill skeleton as the constraints for skill value optimization. The skill value optimization is achieved by CEM-MD, with its results to inform symbolic search. The values of visited nodes in the search tree will be updated, initiating a new iteration. Given an evaluation function Ψ of the final configuration, this framework can return multiple solutions.

Algorithm 3 CEM-MD: Cross-Entropy Method with Mixed Distribution

Input: Initial mean $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$ for continuous variables, initial probability vector $\mathbf{p}$ for discrete variables, population size C, elite fraction p, maximum iterations H, number of categories K_c, skill sequence $a_{1:K}$, Initial configuration $\overline{\boldsymbol{x}}_0$, evaluation function Ψ
Output: Subgoal sequence $\boldsymbol{x}^*$, cumulative reward: J^*
Initialize $h \leftarrow 0$
while $h < H$ **do**
  for $i = 1$ to C **do**
    Sample continuous variables: $\boldsymbol{x}_{\mathrm{c}_i} \sim \mathcal{N}(\mu, \Sigma)$
    Sample discrete variables: $\boldsymbol{x}_{\mathrm{d}_i} \sim \text{Categorical}(\mathbf{p})$
    Combine: $\boldsymbol{x}_i = (\boldsymbol{x}_{\mathrm{c}_i}, \boldsymbol{x}_{\mathrm{d}_i})$
    Evaluate the samples: $J_i \leftarrow l(\boldsymbol{x}_i, a_{1:K}, \overline{\boldsymbol{x}}_0, \Psi)$ ▷ objective function in Eq. (5.6)
  end for
  Select the top p solutions (elite set): $\{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_{pC}\}$
  Update the mean and covariance for continuous variables:

$$\mu \leftarrow \frac{1}{pC} \sum_{i=1}^{pC} \hat{\boldsymbol{x}}_{c,i}$$

  Update the probability vector for discrete variables:
  for $k_c = 1$ to K_c **do**

$$\mathbf{p}_{\text{elite}_{k_c}} = \frac{\text{count}(\hat{\boldsymbol{x}}_{d,k_c})}{pC}$$

  end for

$$\mathbf{p} \leftarrow \mathbf{p}_{\text{elite}}$$

  $h \leftarrow h + 1$
end while
$\boldsymbol{x}_{\mathrm{c}}^* \leftarrow \mathcal{N}(\mu, \Sigma)$ and $\boldsymbol{x}_{\mathrm{d}}^* \leftarrow \text{Categorical}(\mathbf{p})$
$\boldsymbol{x}^* = (\boldsymbol{x}_{\mathrm{c}}^*, \boldsymbol{x}_{\mathrm{d}}^*)$, $J^* \leftarrow l(\boldsymbol{x}^*, a_{1:K}, \overline{\boldsymbol{x}}_0, \Psi)$
Output $\boldsymbol{x}^*$ and J^*

The pseudocode of our Logic-Skill Programming method is shown in Alg. 4.

Algorithm 4 LSP: Logic-Skill Programming

Input: Initial configuration $\overline{\boldsymbol{x}}_0$, evaluation function Ψ, initial symbolic state s_0, maximum iterations $\widetilde{H}$, maximum number of solutions $\widetilde{N}_s$
Output: Skill skeleton $a_{1:K}$, Subgoal sequence $\overline{\boldsymbol{x}}_{1_T:K_T}$
Initialize $h \leftarrow 0$, $N_s \leftarrow 0$, Value Set: $\mathcal{O}_v \leftarrow \emptyset$, Solution Set: $\mathcal{O}_s \leftarrow \emptyset$
while $h < \widetilde{H}$ **do**
    Symbolic search: $a_{1:K} = \text{MCTS}(s_0)$
    Skill value optimization:
        $\overline{\boldsymbol{x}}_{1_T:K_T}, J = \text{CEM-MD}(a_{1:K}, \overline{\boldsymbol{x}}_0, \Psi)$
    if solved **then**
        $N_s \leftarrow N_s + 1$
        $\mathcal{O}_v \leftarrow \mathcal{O}_v \cup (J)$
        $\mathcal{O}_s \leftarrow \mathcal{O}_s \cup (a_{1:K}, \overline{\boldsymbol{x}}_{1_T:K_T})$
    end if
    if $N_s \geq \widetilde{N}_s$ **then**
        $break$
    end if
    $h \leftarrow h + 1$
end while
Output $\mathcal{O}_s$

5.6 Experiments

In this section, we compare our method with state-of-the-art baselines, in terms of policy learning, subgoal optimization, and sequential skill planning.

5.6.1 Evaluation on Skill Policy Learning

To evaluate the skill policies and their corresponding value functions, we initially construct a skill library using different skill learning methods. These methods include TTPI and two state-of-the-art RL methods: Soft Actor-Critic (SAC) (Haarnoja et al., 2018) and Proximal Policy Optimization (PPO) (Schulman et al., 2017).

The skill library encompasses five manipulation skills: `Push`, `Pivot`, `Pull`, `Pick` and `Place`, each characterized by unique state and action spaces. Specifically, `Push`, `Pivot`, `Pull` are planar manipulation primitives.

1. `Push`: The state is characterized by $(\boldsymbol{p}_o, \theta_o, \boldsymbol{p}_r, f_c)$, while the action is denoted by

$(\boldsymbol{v}_r, f_n)$. Here, $(\boldsymbol{p}_o, \theta_o) \in$ SE(2) denotes the object's pose in the world frame. $\boldsymbol{p}_r$ and $\boldsymbol{v}_r$ represent the position and velocity of the robot end-effector in the object frame. We assume that the object has a rectangular shape (common in industry), where $f_c \in 0, 1, 2, 3$ represents the current contact surface and f_n denotes the next contact surface. Thus, the system encompasses a total of 6 states and 3 control variables, comprising both continuous and discrete variables.

2. `Pivot`: The pivoting domain features a goal-augmented state $(\beta, \tilde{\beta})$, where β is the current rotation angle of the object in the gravity plane, and $\tilde{\beta}$ is the desired angle. The control input is $\dot{\beta}$, which denotes the angular velocity of the robot end-effector..

3. `Pull`: The state is defined as $(\boldsymbol{p}_o, \theta_o) \in SE(2)$, representing the object's position and orientation in a planar plane. The control input is $(\dot{\boldsymbol{p}}_o, \dot{\theta}_o)$, denoting the translational and angular velocities of the robot end-effector.

4. `Pick`: In this domain, the state is defined as $(x, y, z, \alpha, \beta, \theta) \in SE(3)$, representing the pose of the robot end-effector, while the control input is the Cartesian velocity of the end-effector, $(\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\theta}) \in SE(3)$. Without loss of generality, we assume that the object to be picked is located at $(0, 0, 0, 0, 0, 0)$.

5. `Place`: This domain shares the same state and action spaces as the `Pick` domain.

We define the general reward for skill learning as:

$$r = -(c_p + \rho c_o + 0.01 c_a + 0.1 c_f), \tag{5.12}$$

with

$$\begin{aligned} &c_p = \|\boldsymbol{x_p} - \boldsymbol{x_p^{\text{des}}}\|/l_p, \quad c_o = \|\boldsymbol{x_o} - \boldsymbol{x_o^{\text{des}}}\|/l_o, \\ &c_a = \|\boldsymbol{u}\|, \quad c_f = 1 - \delta(f_c - f_n), \end{aligned} \tag{5.13}$$

where $\boldsymbol{x}_p$ and $\boldsymbol{x}_o$ represent the current position and orientation of the system in each skill domain, while $\boldsymbol{x}_p^{\text{des}}$ and $\boldsymbol{x}_o^{\text{des}}$ denote the target position and orientation. We set the state space to be within $[-0.5m, 0.5m]$ for position elements, and $[-\pi, \pi]$ for orientation elements. l_p and l_o are therefore set to 0.5 and π respectively to normalize the position error and orientation error. ρ is a hyperparameter used to balance position and orientation errors. Due to the underactuated nature of pushing dynamics, we set $\rho = 0.5$ in practice. For other skills, ρ is set to 1. $\boldsymbol{u}$ represents the control inputs of each skill domain. c_f is a specialized term unique to the pushing domain, used to penalize face switching during pushing. Here, $\delta(f_c - f_n)$ returns 1 when $f_c = f_n$ (indicating no face switching), otherwise, it returns 0.

Notably, the skill policies learned for `push`, `pull`, `pick` and `place` are in a regulated manner, with the target state $\boldsymbol{x}^{\text{des}}$ set as $\mathbf{0}$. In terms of `pivot`, the reward function does not include a positional term. In the orientation term, $\boldsymbol{x}_o$ is defined as β, and $\boldsymbol{x}_{\boldsymbol{o}}^{\text{des}}$ is defined as $\tilde{\beta}$. Given new initial and target states in the long-horizon domain, along with the skill operator a_k, the skill-specific state can be computed as $\boldsymbol{x}_{k_0} = \Gamma_k(\overline{\boldsymbol{x}}_{k_0}, \overline{\boldsymbol{x}}_{k_T} | a_k)$ (as shown in Eq. (5.6)). For `push`, `pull`, `pick` and `place`, the domain mapping function Γ_k is defined as

$$\Gamma_k(\overline{\boldsymbol{x}}_{k_0}, \overline{\boldsymbol{x}}_{k_T}) = \varphi_k(\overline{\boldsymbol{x}}_{k_0}) - \varphi_k(\overline{\boldsymbol{x}}_{k_T}), \tag{5.14}$$

where φ_k is a dimension reduction function that selects skill-specific dimensions from the long-horizon state. In the case of pivot, the skill is acquired through goal-augmented policy learning, achieved by augmenting the state as $(\beta, \tilde{\beta})$. This augmented state is the concatenation of the current rotation angle $\varphi_{\text{pivot}}(\overline{\boldsymbol{x}}_{k_0})$ with the desired angle $\varphi_{\text{pivot}}(\overline{\boldsymbol{x}}_{k_T})$. Γ_{pivot} is therefore defined as

$$\Gamma_{\text{pivot}}(\overline{\boldsymbol{x}}_{k_0}, \overline{\boldsymbol{x}}_{k_T}) = \text{Concat}\Big(\varphi_{\text{pivot}}(\overline{\boldsymbol{x}}_{k_0}), \varphi_{\text{pivot}}(\overline{\boldsymbol{x}}_{k_T})\Big). \tag{5.15}$$

The obtained $\boldsymbol{x}_{k_0}$ can then be directly fed into V^{π_k} without the need to retrain new skill policies and value functions.

As for another domain mapping function $\Phi_k : \mathcal{X}_k \to \overline{\mathcal{X}}$ which maps skill-specific state to long-horizon state, it is defined as a function that updates the value of skill-specific dimensions in the long-horizon state based on the skill-specific state.

Notably, since SAC is not naturally suited for handling hybrid action spaces, we exclude f_c and f_n for a fair comparison of skill learning performance in this subsection. For both PPO and SAC, our primary implementation relies on Stable-Baselines3 (Raffin et al., 2021), utilizing a Multilayer Perceptron (MLP) architecture with dimensions of 32×32 as the policy network. We set the discount factor to 0.99 and the learning rate to 0.001, while configuring the task horizon to 10^4. In TTPI, we establish the accuracy threshold for TT-cross as $\epsilon = 10^{-3}$ and set the maximum rank $r_{\max}$ to 10^2.

The obtained policies are then evaluated by comparing the success rates across 1000 initial states in skill domains. We define a successful task as achieving a position error of less than 0.03cm and an orientation error of less than 15°. Table 5.1 reveals that all the policies can nearly reach the final targets, demonstrating the proficiency of the learned policies in accomplishing individual manipulation tasks. However, rather than focusing solely on having skills for each specific task, we are also concerned with finding the optimal trajectory with the maximum cumulative reward for the sequenced skills. To achieve this, an accurate value function is required to inform

Table 5.1: Comparison of skill policy performance and value function precision.

	TTPI			SAC			PPO		
	success rate	value prediction	time (min)	success rate	value prediction	time (min)	success rate	value prediction	time (min)
pushing	**1.0**	**0.85**	**5.6**	0.83	0.63	36.3	0.95	0.61	44.8
pivoting	**1.0**	**0.94**	**0.9**	1.0	0.51	16.0	1.0	0.68	8.7
pulling	**1.0**	**0.97**	**1.1**	1.0	0.84	16.5	1.0	0.72	10.7
pick/place	**1.0**	**0.93**	**1.67**	0.98	0.64	33.6	1.0	0.66	24.6

Table 5.2: Comparison of computation error and time for skill value optimization.

	TTGO			Shooting		CEM-MD	
	error	approximation time (s)	inference time (s)	error	time (s)	error	time (s)
NPM	0.03 ± 0.00	0.48 ± 0.21	0.003 ± 0.00	0.35 ± 0.01	0.01 ± 0.00	0.02 ± 0.01	0.06 ± 0.01
PPM	0.12 ± 0.01	3.90 ± 0.22	0.004 ± 0.00	0.59 ± 0.01	0.02 ± 0.00	0.05 ± 0.01	0.09 ± 0.01
PM	0.12 ± 0.01	20.28 ± 5.23	0.01 ± 0.01	0.75 ± 0.27	0.01 ± 0.00	0.03 ± 0.01	0.10 ± 0.01

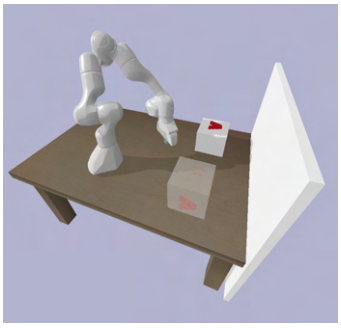
which state in the intersection space between two adjacent skills leads to the maximum cumulative reward. Therefore, we examine whether the value functions can offer the same guidance as the cumulative reward given two states, shown as *value prediction* in Table 5.1. The metric is to compare whether $V^{\pi_k}(\boldsymbol{x}_1) - V^{\pi_k}(\boldsymbol{x}_2)$ has the same direction as $\mathcal{R}_c^{\pi_k}(\boldsymbol{x}_1) - \mathcal{R}_c^{\pi_k}(\boldsymbol{x}_2)$, where $\mathcal{R}_c^{\pi_k}(\boldsymbol{x})$ is the cumulative reward, defined as

$$\mathcal{R}_c^{\pi_k}(\boldsymbol{x}) = \sum_{t=0}^{\infty} \gamma^t R_{k_t}\Big(\boldsymbol{x}_{k_t}, \pi_k(\boldsymbol{x}_{k_t})\Big), \text{ s.t.} \quad \boldsymbol{x}_{k_0} = \boldsymbol{x}. \tag{5.16}$$

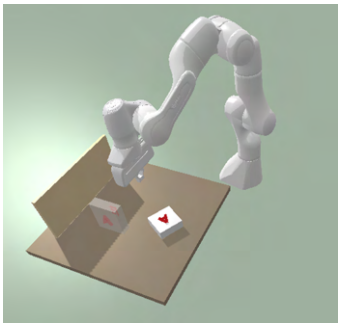
We randomly selected 1000 state pairs in the state domain. The value function of TTPI demonstrates superior prediction performance compared to SAC and PPO. This indicates that value functions in TT format offer better accuracy in approximating the cumulative reward, which can inform which state in the state domain is more dynamically optimal given current policy. This observation aligns with our motivation, emphasizing that typical RL methods approximate the value function primarily within a local space and the policy retrieval is often sub-optimal due to gradient-based optimization. In contrast, TTPI, as an ADP method, aims to approximate the value function across the entire state space. This feature is advantageous in our sequential skill planning framework for identifying the subgoal for each skill, which can be anywhere within the skill-specific state space.

5.6.2 Evaluation on Skill Value Optimization

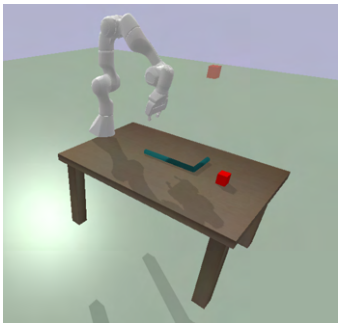
Table 5.1 illustrates that the acquired skills are successful in accomplishing each specific manipulation task. However, to achieve a long-horizon task where only the evaluation function Ψ of final configuration is given, finding the subgoal for each skill is important. This necessitates a good optimization technique capable of finding the



(a) Non-Prehensile Manipulation domain

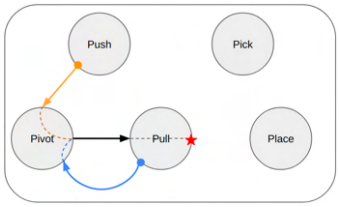


(b) Partly-Prehensile Manipulation domain

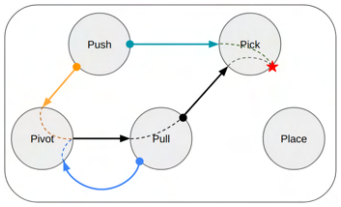


(c) Prehensile Manipulation domain

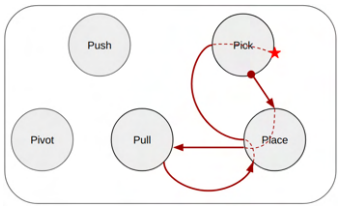
Figure 5.5: Three sequential manipulation domains, including both prehensile and non-prehensile manipulation primitives. The transparent object represents the final target configuration in each domain.



(a) Non-Prehensile Manipulation



(b) Partly-Prehensile Manipulation



(c) Prehensile Manipulation

Figure 5.6: Action skeletons obtained by LSP for three domains. The dot point denotes the start of the skill sequence. Each color represents one solution, with black lines indicating the common shared tunnel. The red star illustrates the end of the skill skeleton.

optimal solution based on the given objective function.

Three different long-horizon domains are used to validate our method, involving both non-prehensile and prehensile manipulation primitives, as shown in Fig. 5.5. The simulation environments are built using PyBullet (Coumans and Bai, 2019) and the AIRobot library (Chen et al., 2019).

a) Non-Prehensile Manipulation (NPM): As depicted in Fig. 5.5a, this domain involves objects that cannot be grasped. The objective is to manipulate the box within the 3D world by leveraging multiple non-prehensile planar manipulation primitives and establishing contacts with the surroundings to achieve the final 6D pose. This task can also be seen as a special case of in-hand manipulation, where the robot and the wall act as active and passive "fingers", respectively.

b) Partly-Prehensile Manipulation (PPM): As shown in Fig. 5.5b, this domain involves objects that can only be grasped in specific directions. The goal in this domain is to manipulate the 6D pose of the cube, including the z direction. Therefore, the robot must strategically decide how to pick up the object in the end. This domain requires the robot to reason about physical contact and object geometry to unify both prehensile and non-prehensile primitives.

c) Prehensile Manipulation (PM): As illustrated in Fig. 5.5c, this domain involves objects that can be directly grasped. The goal is to alter the 6D pose of a block placed on the table, beyond the robot's reachability. To achieve this, the robot must pick up the block in the end. This requires the robot to figure out using a hook to extend the kinematic chain and pull the block back into the reachability region, and then grasp it. However, the way in which the robot picks up the hook will affect whether the block can be reached, and where to pull will also affect the entire trajectory and the total energy cost. In this task, we want to show that our method can find an optimal trajectory which has the minimum energy cost, while successfully finishing the task.

We therefore define the evaluation function Ψ of the final configuration $\overline{\boldsymbol{x}}_{K_T}$ as

$$\Psi(\overline{\boldsymbol{x}}_{K_T}) = \lambda \|\overline{\boldsymbol{x}}_{K_T} - \overline{\boldsymbol{x}}_T\|, \tag{5.17}$$

where $\overline{\boldsymbol{x}}_T$ is the target configuration in each domain, and $\lambda = 10^2$. The initial configuration $\overline{\boldsymbol{x}}_0$ and target configuration $\overline{\boldsymbol{x}}_T$ are randomly sampled from the long-horizon domain $\overline{\mathcal{X}}$, which is the configuration space of the entire environment. In NPM and PPM, $\overline{\mathcal{X}} = \mathcal{S}_R \times \mathcal{S}_O$, while in PM, $\overline{\mathcal{X}} = \mathcal{S}_R \times \mathcal{S}_O \times \mathcal{S}_T$. Here, $\mathcal{S}_O = SE(3)$ and $\mathcal{S}_R = SE(3)$ denote the pose of the object and robot end-effector, respectively, while $\mathcal{S}_T = SE(2)$ represents the pose of a tool positioned on the table.

It is worth noting that both PPM and NPM tasks involve optimizing both discrete and continuous variables, akin to mixed-integer programming. This complexity presents significant challenges in optimization, especially considering that the objective functions are arbitrary without any structure, depending on the value functions of skills. To this end, we compare CEM-MD with some other techniques capable of handling mixed-integer variables, including TTGO (Shetty et al., 2024a) and the random shooting method. TTGO is an optimization technique specifically designed for functions in tensor train format, enabling it to find the optimal solution extremely fast once the TT model is provided. Random shooting is a naive technique that randomly samples variables from the domain and returns the one with the highest objective score. We apply these three optimization techniques across the three domains, conducting 10 random initializations for each domain. The maximum number of iterations for

Table 5.3: Comparison of computation time and solution quality for sequential skill planning.

	Computation Time (s) [w/ sym. goal]		Computation Time (s) [w/o sym. goal]		Cumulative Reward		Sequence Length
	LSP	STAP	LSP	STAP	LSP	STAP	
NPM	0.14 ± 0.23	0.06 ± 0.03	0.27 ± 0.15	NA	3.0 ± 0	2.4 ± 1.57	3.0 ± 0
PPM	0.17 ± 0.1	0.08 ± 0.02	0.22 ± 0.13	NA	2.9 ± 0.7	1.88 ± 1.01	2.9 ± 0.7
PM	0.25 ± 0.15	0.21 ± 0.02	0.41 ± 0.02	NA	5.0 ± 0	3.28 ± 0.62	5.0 ± 0

CEM-MD is set to $H = 300$, with an early stopping criterion of 10^{-3} to terminate the while loop if the objective value no longer decreases. The population size is set to $C = 1000$, and the elite fraction is set to $p = 0.3$. The number of categories K_c is set to 4. The initial parameters $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and $\mathbf{p}$ are computed using samples collected from a uniform distribution in the first iteration. The results are shown in Table 5.2, with the computation error defined as the L2 norm between the final state $\overline{\boldsymbol{x}}_{K_T}$ and the target configuration $\overline{\boldsymbol{x}}_T$.

Given an arbitrary function, TTGO needs to approximate it first using TT-cross and then optimize over the approximated TT model, corresponding to the approximation stage and inference stage, separately. Table 5.2 shows that the approximation stage usually takes a longer time, but once the TT model has been obtained, the inference stage will be very fast. This suits problems with static objective functions quite well (Shetty et al., 2024a). However, in this work, if the high-level skill skeleton $a_{1:K}$ changes, the resulting objective function will change as well. This requires a new TT approximation, leading to expensive computation. Meanwhile, the shooting method is much faster but can obtain poor solutions because of the random distribution. CEM-MD takes a bit longer time compared to random shooting but obtains good solutions by updating the mixed continuous and discrete distribution. The total computation time for these three domains is less than 0.1s, which is still fast enough in terms of long-horizon planning.

5.6.3 Evaluation on Overall Performance of LSP

We then run the complete LSP framework for these three domains to assess the overall performance. In MCTS, the exploration parameter C_E is set to 3, allowing for efficient tree search while exploring different skeleton solutions. The feedback reward $\overline{r}$ is defined as a binary variable $0/1$, determined by whether the geometric target is achieved. The maximum iteration is set to $\widetilde{H} = 100$, with an early stopping criterion that terminates the while loop upon finding enough $\widetilde{N}_s$ solutions. We set $\widetilde{N}_s = 5$ to explore multiple solutions. Note that the $\widetilde{N}_s$ solutions can be identical, depending on MCTS and the total number of feasible solutions. We derive 13 skill operators from the

Table 5.4: The specification of skill operators, preconditions and effects. In the effects column, only the states that are changed after skill execution are listed. The remaining symbolic states in the preconditions will be directly inherited by the effects.

Skill Operator	Preconditions	Effects
push_wall	(¬ (AtEdge o)) ∧ (¬ (AtWall o)) ∧ (¬ (AfterFlip o)) ∧ (onTable o)	(AtWall o)
pivot	(AtWall o)	(AtWall o) ∧ (AfterFlip o)
pull_wall	(¬ (AtEdge o)) ∧ (¬ (AtWall o)) ∧ (¬ (AfterFlip o)) ∧ (onTable o)	(AtWall o)
pull_center	(AtWall o) ∧ (AfterFlip o) ∧ (onTable o)	(¬ (AtWall o))
pull_edge	(¬ (AtEdge o)) ∧ (¬ (AtWall o)) ∧ (¬ (AfterFlip o)) ∧ (onTable o)	(AtEdge o)
pick_edge	(AtEdge o) ∧ (PartGraspable o) ∧ (HandEmpty r)	(InHand o) ∧ (¬ (HandEmpty r))
pick_center	(¬(AtWall o)) ∧ (AfterFlip o) ∧ (PartGraspable o) ∧ (HandEmpty r)	(InHand o) ∧ (¬ (HandEmpty o))
push_edge	(¬ (AtEdge o)) ∧ (¬ (AtWall o)) ∧ (¬ (AfterFlip o)) ∧ (onTable o)	(AtEdge o)
pick_tool	(¬ (ReadyPull t o)) ∧ (Graspable o) ∧ (Reachable t) ∧ (HandEmpty r)	(¬ (HandEmpty r))
pull_tool	(ReadyPull t o) ∧ (¬ (Reachable o)) ∧ (¬ (HandEmpty r)) ∧ (onTable o)	(Reachable o)
place_toolmove	(¬ (HandEmpty r)) ∧ (¬ (Reachable o))	(ReadyPull t o)
place_tool	(¬ (HandEmpty r))	(HandEmpty r)
pick_object	(Reachable o) ∧ (Graspable o) ∧ (HandEmpty r)	(InHand o)

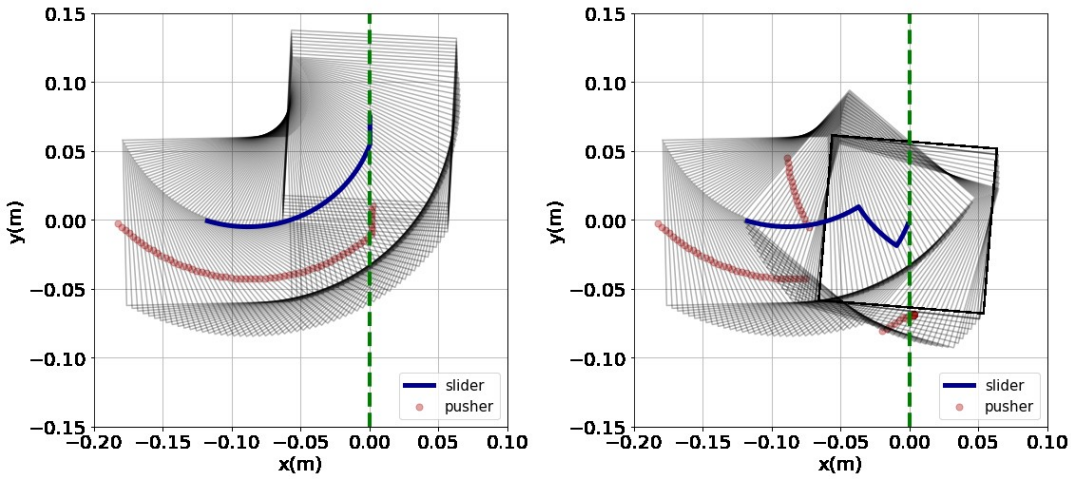
Table 5.5: Initial symbolic state and symbolic goal of each domain. Note that LSP does not need symbolic goals. Such goals are needed by the sampling-based sequential skill planning methods for comparison.

Domain	Initial symbolic state	Symbolic goal
NPM	(¬ (AtEdge o)) ∧ (¬ (AtWall o)) ∧ (¬ (AfterFlip o)) ∧ (onTable o)	(AfterFlip o) ∧ (¬ (AtWall o))
PPM	(¬ (AtEdge o)) ∧ (¬ (AtWall o)) ∧ (¬ (AfterFlip o)) ∧ (HandEmpty r) ∧ (PartGraspable o) ∧ (onTable o)	(Inhand o)
PM	(¬ (Reachable o)) ∧ (Graspable o) ∧ (Reachable t) ∧ (HandEmpty r) ∧ (onTable o)	(Inhand o)

five skill policies existing in the skill library. The logic defining the preconditions and effects of these operators is presented in Table 5.4. Each operator's name, preceding the underscore, denotes the skill policy it utilizes. Elements highlighted in yellow are common across both NPM and PPM domains, those in gray are specific to the PPM domain, and those in green are specific to the PM domain. Throughout the table, we use the following symbols: o for an object, t for a tool, and r for a robot arm. Table 5.5 displays the initial symbolic state s_0 used in each domain, along with the symbolic goals used for comparison.

Given the same initial state s_0 and $\overline{x}_0$ in each domain, we randomly sample 10 different target configurations $\overline{x}_T$ that require multi-step manipulation. Fig. 5.6 shows that LSP can actively find multiple solutions. For the Non-Prehensile domain, two solutions found by LSP are `push-pivot-pull` and `pull-pivot-pull`. For the Partly-Prehensile domain, four different skeletons are found: `push-pick`, `pull-pick`, `push-pivot-pull-pick` and `pull-pivot-pull-pick`. For the Prehensile domain, the solved skill skeleton is `pick-place-pull-place-pick`.

We then compare our method with another state-of-the-art sampling-based sequential



(a) LSP (b) STAP

Figure 5.7: The pushing subtask obtained for the PPM domain. Both 5.7a and 5.7b have the same initialization configuration. The objective is to employ the robot end effector (pusher) to move the slider to the green line ($x = 0$), representing the edge of the table. LSP provides a solution with the highest value in the space, requiring less control effort, while STAP outputs one that involves multiple face switches.

skill planning method called STAP (Agia et al., 2023). STAP focuses on constraints satisfaction. It requires an explicit symbolic goal for high-level task planning, followed by feasible solution sampling. To ensure a fair comparison, we use MCTS with an explicit symbolic goal as the task planner in STAP and then employ CEM-MD for feasibility checking given the skill skeleton. Table 5.3 illustrates the time required to find one solution and the solution quality between LSP and STAP. We can observe that LSP does not require a symbolic target goal, whereas STAP relies on it. STAP can find the solution faster than LSP. The reason is that STAP focuses only on finding the feasible solution, while LSP aims to provide (global) optimality over the full logic-geometric path. This aligns with the comparison of cumulative rewards. Note that the cumulative reward of each skill in the sequence is normalized by the highest value in the corresponding value function. We can observe that the trajectory found by LSP leads to a higher cumulative reward compared with STAP, indicating better optimality. Moreover, in the PPM domain, the skill sequence varies with different lengths, showing that multiple solutions are found, as depicted in Fig. 5.6b.

Fig. 5.7 illustrates a toy example of the solutions found by LSP and STAP for the PPM domain, where the robot needs to push the block (slider) to the edge of the table (depicted as the green line) for grasping from the lateral side. LSP determines the subgoal with the highest cumulative reward by utilizing the value function, which

represents the optimal state considering the system dynamics. In contrast, STAP outputs a feasible state, which can be any configuration on the edge theoretically. Here, we pick up the one closest to the initialization in Euclidean space. However, to reach this target, the pusher exerts more effort, involving two face switches (visible in the discontinuous pusher trajectory in Fig. 5.7). This also underscores the utility of the value function space for finding the optimal path considering system dynamics, compared with Euclidean state space.

5.6.4 Real-Robot Experiments

This section first evaluates the performance of the TTPI algorithm on the face-switching planar pushing task, with a specific focus on its execution under contact uncertainty and external disturbances. We then demonstrate the efficacy of LSP for sequential manipulation planning on the large box manipulation task introduced at the beginning of this chapter.

Planar Pushing Task

The goal is to push a block while allowing both contact mode transitions and switching between different contact faces. To model this problem, we adopt the motion equations introduced in Chapter 3 and explicitly include the contact face as part of the state and control variables. The system state is defined as $[\boldsymbol{q}_s^\top \ \boldsymbol{q}_p^\top \ c_c]^\top$, where $\boldsymbol{q}_s = [s_x \ s_y \ s_\theta]^\top$ represents the position and orientation of the block, $\boldsymbol{q}_p = [p_x \ p_y]^\top$ denotes the end-effector position, and $c_c \in \{0, 1, 2, 3\}$ indicates the current contact face. The control input is given by $[\boldsymbol{v}^\top \ c_n]^\top$, where $\boldsymbol{v} = [v_n \ v_t]^\top$ denotes the end-effector velocity and $c_n \in \{0, 1, 2, 3\}$ specifies the next contact face. The resulting system therefore has $n = 6$ state variables and $m = 3$ control inputs, comprising both continuous and discrete components.

The control policy was first trained in simulation, with continuous state and action variables discretized into 200 and 50 points, respectively, while discrete variables were kept unchanged. The domain is set in the range from $[-0.5m, -0.5m, -\pi]$ to $[0.5m, 0.5m, \pi]$, with maximum velocity defined as 0.1 m/s. The accuracy of TT-cross is defined as 10^{-3}. The rank of the final value function was found to be 4 and the rank of the advantage function was 40. Each iteration of the VI procedure took about 10 seconds on average. To test the generalization of the policy, we randomly selected 1000 initializations in the domain. The success rate is 100%. Fig. 5.8 shows part of the simulation results. We reached a robust policy generating a 100% success rate in about 40 iterations while the convergence of the whole algorithm took about 500 iterations.

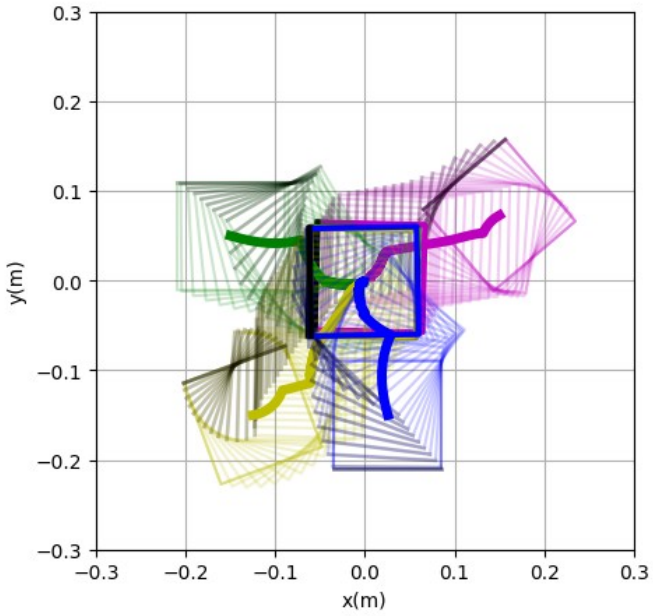


Figure 5.8: Simulated trajectories of the block given four different initial states. The colored trajectories represent the motion of the block to the target ($\boldsymbol{q}_s = [0\ 0\ 0]^\top$).

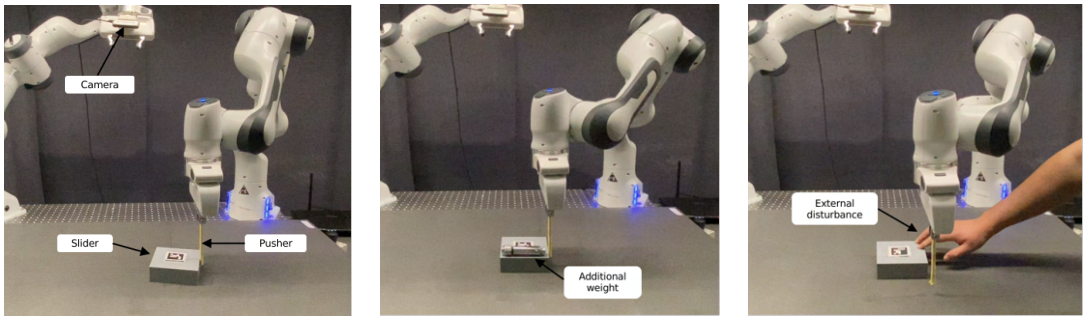
The reward function is defined as:

$$R(s,a) = -\|\boldsymbol{q}_s\| \times (1 + \gamma_1 \times (1 - \delta(c_c - c_n)) + \gamma_2 \times \|\boldsymbol{v}\|), \tag{5.18}$$

where $\boldsymbol{q}_s$ represents the block pose, $\delta(c_c - c_n)$ will return 1 if $c_c = c_n$, otherwise, 0. γ_1 and γ_2 are set as 0.1 and 0.001, respectively. Note that the flexibility offered by our method allows us to utilize such reward functions.

We then tested the trained policy on the real robot setup (Fig. 5.9), using a 7-axis Franka Emika robot and a RealSense D435 camera. The slider ($r_s = 6$ cm) is a 3D-printed prismatic object with PLA, lying on a flat plywood surface, with an Aruco Marker on the top face. A wooden pusher ($r_p = 0.5$cm) is attached to the robot to move the object. The motion of the object is tracked by the camera at 30 HZ, and the policy is called at 100 HZ, with a low-level Cartesian velocity controller (1000 HZ) actuating the robot.

Three experiments were conducted to assess the robustness of our policy: (a) Reaching task: The robot pushes the slider from $\boldsymbol{q}_{s_0} = [0.05\text{m}\ 0.16\text{m}\ 0]^\top$ to origin (Fig. 5.9a); (b) Reaching with additional weight: The robot pushes the block from the same initialization as before, but with an additional weight, 3 times heavier than the block (Fig. 5.9b);



(a) Reaching (b) Reaching with additional weight (c) Reaching with external disturbance

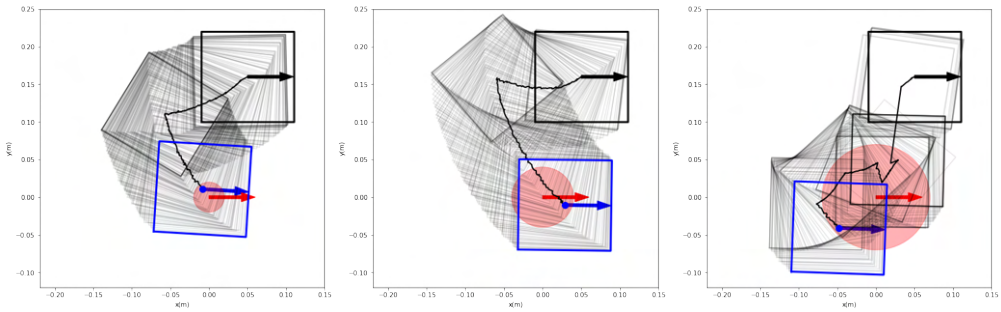
Figure 5.9: Pusher-slider system where the robot pushes an object by contact switching. Three experiments were performed: a) Reaching: The block is pushed by the end-effector from initial state $\boldsymbol{q}_{s_0}$ to the origin. b): Reaching with additional weight: The block is pushed by the end-effector from $\boldsymbol{q}_{s_0}$ to the origin, but with an additional weight on the block, leading to nonuniform friction distribution. c) Reaching with external disturbance: The block is pushed by the end-effector from $\boldsymbol{q}_{s_0}$ to the origin, disturbed by moving around $\boldsymbol{q}_{dist} = [0.1\text{m}\ 0.03\text{m}\ 90°]^\top$.

Table 5.6: Performance of three real-world experiments

Experiments	x_{err}/cm	y_{err}/cm	θ_{err}/rad
Reaching	-0.83	1.07	-0.06
Reaching, w. weight	2.89	-1.04	-0.01
Reaching, w. disturbance	-4.78	-4.10	-0.04

(c) Reaching with external disturbance: The same initialization like before, but with a significant external disturbance of $\boldsymbol{q}_{dist} = [0.1\text{m}\ 0.03\text{m}\ 90°]^\top$ exerted by a human (Fig. 5.9c).

The results of these experiments are displayed in Fig. 5.10 and Table 5.6, showing that in all experiments, the policy successfully reaches the final target in terms of both position and orientation. The error increases with the disturbance, while orientation errors remaining less than $4°$ and position errors staying less than 5cm even under a significant disturbance. Experiment (c) demonstrates that the policy is able to dynamically select the contact face based on the current state, as evidenced by the change in contact face after a $90°$ rotation. This highlights the global optimality of TTPI and its ability to handle both continuous and discrete variables in hybrid systems.



(a) Reaching (b) Reaching with additional weight (c) Reaching with external disturbance

Figure 5.10: Real-world block trajectories across three physical experiments. The black square and arrow indicate the initial configuration. The blue square and arrow denote the final pose of the block. The red arrow represents the target orientation, while the red circle indicates the position error. Quantitative results are reported in Table 5.6.

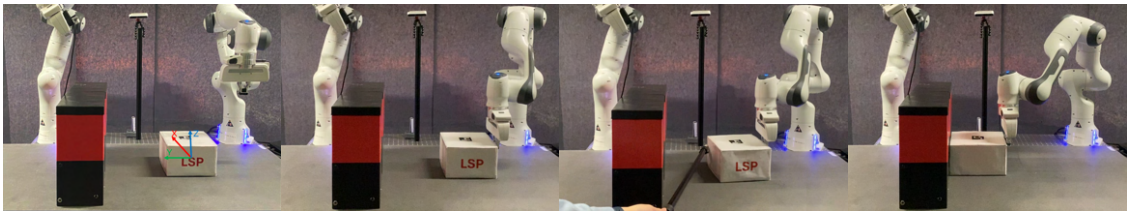
Sequential Manipulation Task

We further conducted real-robot experiments for sequential non-prehensile manipulation, employing a 7-axis Franka Emika robot and a RealSense D435 camera. A large box (21cm x 21cm x 16cm) was positioned on a flat plywood surface. The camera tracked the object motion at 30 Hz with ArUco markers, and the policy for each skill was updated at 100 Hz, with low-level controllers (1000 HZ) actuating the robot. We employed different controllers depending on the skill operator. Specifically, for pushing, we utilized the Cartesian velocity controller. For other skills, we utilized Cartesian impedance controller.

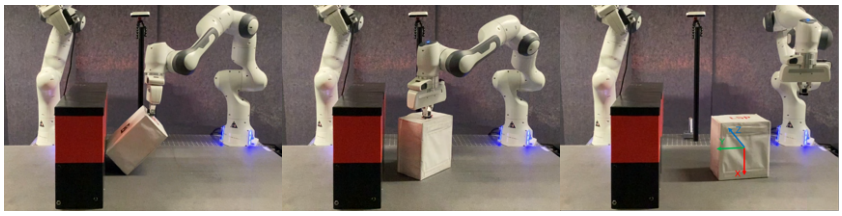
Fig. 5.11 displays the keyframes of the robot experiments. Given the initial and final configurations, the robot successfully manipulated the box by utilizing three planar manipulation primitives, establishing and breaking contact with the surroundings. It is worth noting that long-horizon contact-rich manipulation in the real world is quite challenging due to friction uncertainty and external disturbances. This highlights the importance of introducing skills for online real-time control. Through skill sequencing, we demonstrate that the robot can actively engage with the physical world, accomplishing a much more complicated long-horizon task.

5.7 Discussion

In this chapter, we introduced an optimization-based framework for sequential skill planning, referred to as Logic-Skill Programming (LSP). The problem is formulated as



(a) Initialization (b) Pushing (c) Disturbance (d) Push-Pivot Switch



(e) Pivoting (f) Pulling (g) Target Reaching

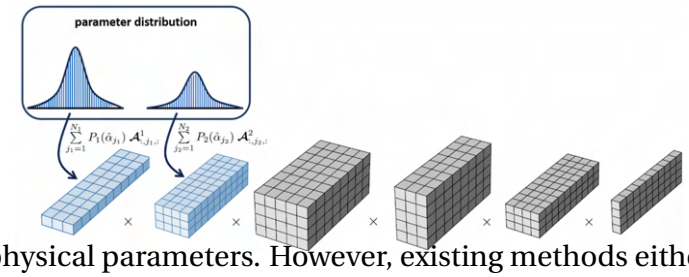
Figure 5.11: Sequential non-prehensile manipulation in the real world. The system is initialized as (a), and the objective is to manipulate the box to achieve the target configuration as (g). The first stage involves pushing the box towards the wall with a 90° rotation. Additionally, we apply an external disturbance to test the skill policy (c). After the pushing stage, the robot switches to the pivoting skill (d, e), followed by pulling (f), until reaching the final geometric configuration.

a first-order extension of a mathematical program that jointly optimizes cumulative reward and the quality of the final configuration. The cumulative reward is represented as the sum of skill value functions along the plan. To solve this problem efficiently, we employ Tensor Train (TT) for value function approximation and interleave symbolic search with skill value optimization to identify the full logic-geometric path.

Experimental results show that value functions represented in TT format provide a more global approximation compared to state-of-the-art reinforcement learning methods. Moreover, the LSP framework can generate multiple skill skeletons and their corresponding subgoal sequences given only an evaluation function of the final geometric configuration. The approach was validated across three manipulation domains, demonstrating its effectiveness in sequencing both prehensile and non-prehensile manipulation primitives. Real-world experiments further show that LSP enables reliable execution of long-horizon contact-rich manipulation under external disturbances.

Despite these promising results, the robustness of the learned skills is limited to a narrow range of contact uncertainty. Significant changes in object properties or friction parameters may lead to failure of the manipulation policies. This limitation motivates the next chapter, which focuses on learning robust control policies for contact-rich manipulation.

6 Robust Contact-rich Policy Learning using Tensor Train



Physical uncertainty remains another key challenge in contact-rich manipulation. A common strategy is to learn policies that are robust to variations in physical parameters. However, existing methods either rely on predefined training distributions or require additional policy training. In this chapter, we introduce Implicit Motor Adaptation (IMA), which enables rapid motor adaptation under arbitrary parameter distributions. Specifically, the parameter-augmented policy is implicitly represented in Tensor Train format, where the separable structure of the tensor cores allows efficient parameter-conditioned policy retrieval.

Publication Note

The material presented in this chapter is adapted from:

- Xue, T., Razmjoo, A., Shetty, S., and Calinon, S. (2025b). Robust contact-rich manipulation through implicit motor adaptation. *International Journal of Robotics Research (IJRR)*.
- Xue, T., Razmjoo, A., Shetty, S., and Calinon, S. (2024b). Robust manipulation primitive learning via domain contraction. In *Proc. Conference on Robot Learning (CoRL)*.

Supplementary materials related to this chapter are available at: https://sites.google.com/view/implicit-ma.

6.1 Introduction

Chapter 5 demonstrates that long-horizon contact-rich manipulation can be decomposed into several simple contact-rich manipulation primitives and then sequenced using PDDL planners (Ghallab et al., 1998; Cheng and Xu, 2023) or Large Language Models (Driess et al., 2023). However, since manipulation primitives are typically sequenced by naive symbolic planners that lack geometric or motion-level information, it is crucial to develop primitives that are robust to diverse instances with varying physical parameters, such as shape, mass, and friction coefficient. For example, once a `push` primitive is scheduled by a high-level symbolic planner, it should be capable of pushing objects with different physical properties from any initial configuration to their targets.

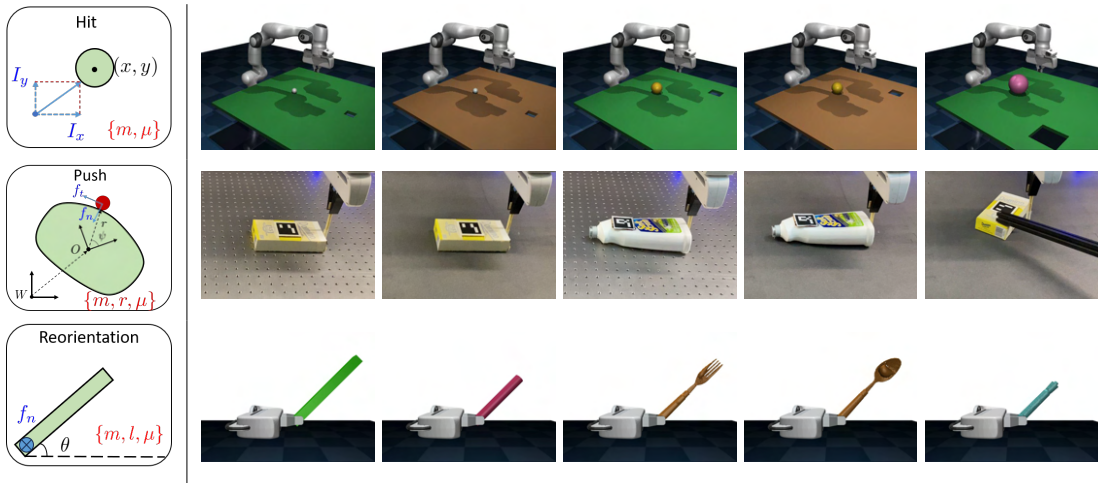


Figure 6.1: Implicit motor adaptation for robust contact-rich manipulation. The left images illustrate the state and action spaces for each primitive (`Hit`, `Push`, `Reorientation`) in black and blue, respectively, along with the parameter spaces represented by the red variables: m for mass, μ for the friction coefficient, r for the radius, and l for the length. A single policy is trained for each primitive and deployed directly across a wide range of objects with varying shapes, weights, and friction parameters, while preserving instance-specific optimal behaviors.

To enable robust policy learning, commonly adopted methods include domain randomization (DR) (Tobin et al., 2017; Muratore et al., 2018) and domain adaptation (DA) (Bousmalis et al., 2018; Arndt et al., 2020; Chebotar et al., 2019). DR assumes that environment parameters follow specific distributions (e.g., normal or uniform), and the goal is to maximize (or minimize) the expected reward (or cost) under randomly sampled parameters during training. While this method provides good generalization to diverse instances, it often results in suboptimal and high-variance behaviors due to

restrictive assumptions about the parameter distributions. In contrast, DA incorporates more instance-specific data during policy training, leading to optimal behaviors in the specific target domain but sacrificing generalization to other instances without additional fine-tuning.

Combining DR and DA to harness the benefits of both is a sensible strategy (Muratore et al., 2022; Qi et al., 2023), but how to effectively balance them is still an open question. It typically involves online system identification to capture domain-specific information and multi-domain learning to account for diverse instances. One promising approach to address this trade-off is to learn a base policy that uses diverse privileged environmental information as input and then relies on proprioceptive history to estimate the system parameters or a latent low-dimensional embedding online. For simplicity, we refer to both as the system parameters in this work, which are then used as input to the base policy to generate domain-specific optimal behaviors. This framework, referred to as Teacher-Student Policy in (Lee et al., 2020a) and Rapid Motor Adaptation in (Kumar et al., 2021) and (Qi et al., 2023), is more broadly termed *explicit motor adaptation (EMA)* in this work, as the policies are typically represented as explicit feedforward functions that directly output actions given observations and estimated parameters. It can be viewed as a variant of DA that enhances generalization by including online parameter estimation.

This approach has demonstrated impressive performance in quadrupedal locomotion across diverse terrains (Lee et al., 2020a; Kumar et al., 2021) and robust in-hand manipulation (Qi et al., 2023). However, it either relies on high-quality system identification (Qi et al., 2023) or additional training of a student policy to mimic the teacher policy (Lee et al., 2020a), leading to performance degradation and increased training effort.

To address these limitations, we introduce *domain contraction* in this chapter, a mechanism for retrieving policies conditioned on a probabilistic parameter distribution. Building on this concept, we develop a complete sim-to-real transfer framework termed *Implicit Motor Adaptation* (IMA). The framework incorporates an online system identification module that probabilistically estimates environmental parameters from proprioceptive history and directly retrieves the parameter-conditioned policy from the base policy augmented with these parameters. IMA can be seen as an elaborated version of DR, leveraging instance-aware distributions instead of being parameter-blind, while also retaining the advantage of DR in not relying heavily on good-quality system identification. It addresses sim-to-real transfer as a stochastic problem, pushing the limits for finding an optimal policy under sim-to-real uncertainty without requiring policy retraining for new instances. The policy is implicitly represented as

$\arg\max$ of the parameter-conditioned advantage function

$$\boldsymbol{u}^* = \arg\max_{\boldsymbol{u}\in\mathcal{U}} A(\boldsymbol{h}, \boldsymbol{x}, \boldsymbol{u}) \quad \text{instead of} \quad \boldsymbol{u}^* = \pi_\theta(\boldsymbol{h}, \boldsymbol{x}), \tag{6.1}$$

where $\boldsymbol{h}$ is the proprioceptive history. EMA estimates the system parameter from $\boldsymbol{h}$ and directly inputs it into the base policy π_θ to output the action. In contrast, thanks to the implicit policy representation, IMA can retrieve the parameter-conditioned policy using a probabilistic estimate based on $\boldsymbol{h}$. This makes it particularly effective for contact-rich manipulation tasks, in which contact parameter identification is often challenging. Although implicit representations have been explored in imitation learning (Florence et al., 2022) and as components of reinforcement learning (Haarnoja et al., 2017; Wang et al., 2022), to the best of our knowledge, our work is the first to investigate this idea in the context of robust policy learning.

However, computing the parameter-conditioned advantage function and retrieving the policy can be computationally expensive, making it difficult to apply in real-world settings. To address this issue, we employ Tensor Train (TT) for function approximation, enabling fast computation of the advantage function and efficient online policy retrieval.

In summary, the main contributions of this chapter are as follows:

1) We propose *domain contraction*, a unification of domain adaptation and domain randomization, which results in optimal behavior while maintaining generalization.

2) We introduce *implicit motor adaptation*(IMA), for robust contact-rich manipulation. The framework highlights the potential of implicit representations for robust policy learning, in contrast to the widely used *explicit motor adaptation* (EMA) paradigm.

3) We develop a probabilistic system adaptation module that enables robots to adapt their behavior in a probabilistic manner without requiring high-accuracy system identification and additional student policy training.

4) We provide theoretical analysis and numerical experiments demonstrating the effectiveness of IMA compared with EMA.

6.2 Related Work

6.2.1 Learning for Contact-rich Manipulation

Many approaches have been proposed to learn manipulation policies for contact-rich manipulation, including Behavior Cloning (BC) (Florence et al., 2022; Torabi et al., 2018), Deep Reinforcement Learning (DRL) (Pertsch et al., 2021; Qi et al., 2023; Lee et al., 2020a), and Approximate Dynamic Programming (ADP) (Powell, 2007; Werbos, 1992). In our work, we employ an ADP approach called Tensor Train Policy Iteration (TTPI) (Shetty et al., 2024c) for policy learning, where the state space is augmented with system parameters to enable subsequent parameter-conditioned policy retrieval given different instances. However, our proposed approach is flexible and can be integrated with any policy learning technique, provided the policy can be implicitly represented by advantage functions or energy-based functions.

6.2.2 Implicit Policy Representation

Instead of being expressed as an explicit function, the action policy can be described implicitly through function optimization. Implicit representations have been widely used in various approaches, including energy-based models (EBMs) (LeCun et al., 2006; Du and Mordatch, 2019), diffusion models (Yang et al., 2023a; Ho et al., 2020), and tensor networks (Orús, 2019; Stoudenmire and Schwab, 2016). In imitation learning, behavior cloning can be formulated using EBMs (Florence et al., 2022), highlighting several advantages such as multimodal representation and long-horizon visuomotor policy learning. In methods based on dynamic programming, the control policy is naturally obtained by optimizing objective functions, making it more intuitive to represent the policy implicitly. For example, reactive policy learning can be reformulated as solving sequence-of-constraints model predictive control (MPC) (Toussaint et al., 2022), demonstrating strong generalization and improved compositionality using implicit functions. Similarly, implicit models can be utilized as policy representations in reinforcement learning, emphasizing their expressiveness in action generation (Haarnoja et al., 2017; Wang et al., 2022). For high-dimensional action spaces, Tensor Train (TT) can be employed as a low-rank representation for state-value and advantage functions, enabling implicit retrieval of control actions (Shetty et al., 2024c; Tal et al., 2018). Our work is inspired by the performance of implicit representation in control policy learning and aims to explore its potential for robust policy learning, which has not been investigated so far.

6.2.3 Sim-to-real Transfer

Obtaining large amounts of real-world data for primitive learning is challenging. Therefore, robot learning in simulation and transferring to the real world is a promising idea (Peng et al., 2018). However, the reality gap between simulation and the real world poses a significant challenge to such idea. If the target domain is known and specific, Domain Adaptation (DA) (Bousmalis et al., 2018; Arndt et al., 2020; Chebotar et al., 2019) can be an effective method, but its reliance on domain-specific data limits the generalization to other scenarios. Alternatively, Domain Randomization (DR) (Tobin et al., 2017) seeks to develop a robust policy by introducing random variations into the simulation parameters. While this method offers good generalization, it often results in suboptimal and high-variance behaviors due to the restrictive assumptions about environment parameter distribution (e.g., normal or uniform).

Combining DA and DR can be a promising strategy to balance generalization and optimal performance. The basic idea is to adapt the parameter distribution by leveraging differences between simulated and reference environment (Mehta et al., 2020; Muratore et al., 2022; Ramos et al., 2019). However, policies developed through such methods are often tailored to the reference environment or target domain. Recently, explicit motor adaptation (EMA) (Yu et al., 2017) has demonstrated remarkable success in learning robust locomotion (Lee et al., 2020a; Kumar et al., 2021) and manipulation primitives (Qi et al., 2023). In this approach, a teacher policy is trained in simulation with various domain parameters, and a student policy learns to replicate this behavior, with a low-dimensional embedding of proprioceptive history as input. Our work closely follows this paradigm but introduces a more efficient way for learning the parameter-augmented teacher policy using tensor approximation. Additionally, our method eliminates the need for high-quality system identification and subsequent student policy training, as the parameter-conditioned robust policy can be directly derived from the teacher policy given a rough parameter distribution.

6.3 Problem Formulation

Let us consider a discrete-time dynamical system:

- **State** $\boldsymbol{x} \in \Omega_{\boldsymbol{x}} \subseteq \mathbb{R}^m$: The system's state space.
- **Action** $\boldsymbol{u} \in \Omega_{\boldsymbol{u}} \subseteq \mathbb{R}^n$: The system's action space.
- **Parameters** $\boldsymbol{\alpha} \in \Omega_{\boldsymbol{\alpha}} \subseteq \mathbb{R}^d$: The unknown or uncertain physical parameters of the system, such as mass, friction coefficient, etc.

- **Dynamics Model** $f : \Omega_{\boldsymbol{\alpha}} \times \Omega_{\boldsymbol{x}} \times \Omega_{\boldsymbol{u}} \to \Omega_{\boldsymbol{x}}$: The system's next state depends on current state $\boldsymbol{x}_t$, action $\boldsymbol{u}_t$ and the system parameters $\boldsymbol{\alpha}$.
- **Reward function** $R : \Omega_{\boldsymbol{x}} \times \Omega_{\boldsymbol{u}} \to \mathbb{R}$: The immediate reward, which depends on the current state $\boldsymbol{x}_t$, action $\boldsymbol{u}_t$.
- **Discount factor** $\gamma \in [0, 1]$.

This system can be described by a Markov Decision Process (MDP) $\mathcal{M} = \{\Omega_{\boldsymbol{x}}, \Omega_{\boldsymbol{u}}, \Omega_{\boldsymbol{\alpha}}, f, R, \gamma\}$. The objective is to find a policy π that maximizes the expected cumulative reward

$$\mathbb{E}_{\pi}\left(\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \pi(\boldsymbol{\alpha}, \boldsymbol{x}_t)) \mid \boldsymbol{x}_0 = \boldsymbol{x}\right). \tag{6.2}$$

However, knowing the exact value of $\boldsymbol{\alpha}$ in the real world is intractable. This information can, however, be estimated using multimodal sensors (Lee et al., 2020b) or proprioceptive history (Bianchini et al., 2023; Lee et al., 2020a). Given the sensor noise and model inaccuracy, it is better to estimate the system parameters as a probabilistic distribution, denoted as $\hat{\boldsymbol{\alpha}} \sim P(\hat{\boldsymbol{\alpha}})$. Therefore, our approach aims to find a policy π that maximizes

$$\mathbb{E}_{\hat{\boldsymbol{\alpha}} \sim P(\hat{\boldsymbol{\alpha}})}\left[\mathbb{E}_{\pi}\left(\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \pi(\hat{\boldsymbol{\alpha}}, \boldsymbol{x}_t)) \mid \boldsymbol{x}_0 = \boldsymbol{x}\right)\right]. \tag{6.3}$$

Solving (6.3) involves knowing the probability distribution $P(\hat{\boldsymbol{\alpha}})$ and then finding the policy π that maximizes the expected cumulative reward over this distribution. One approach to address this problem is to learn control policies separately for each instance with its specific parameter distribution, akin to performing domain adaptation within each target domain. However, a single manipulation primitive typically involves numerous instances, making this approach too time-consuming and impractical. An alternative is to learn an augmented base policy that encompasses multiple instances and retrieve instance-specific policies using estimated instance information, similarly to the *explicit motor adaptation (EMA)* strategy. However, EMA cannot solve (6.3) as it focuses only on one specific parameter instance rather than a distribution.

In this work, we present *implicit motor adaptation (IMA)*, which addresses the problem of probabilistic system identification and, more importantly, the retrieval of a robust policy conditioned on the distribution of parameters. Additionally, to reduce the computational burden of online control, we employ tensor train (TT) as an implicit representation of the base policy, enabling efficient parameter-conditioned policy retrieval through tensor core products.

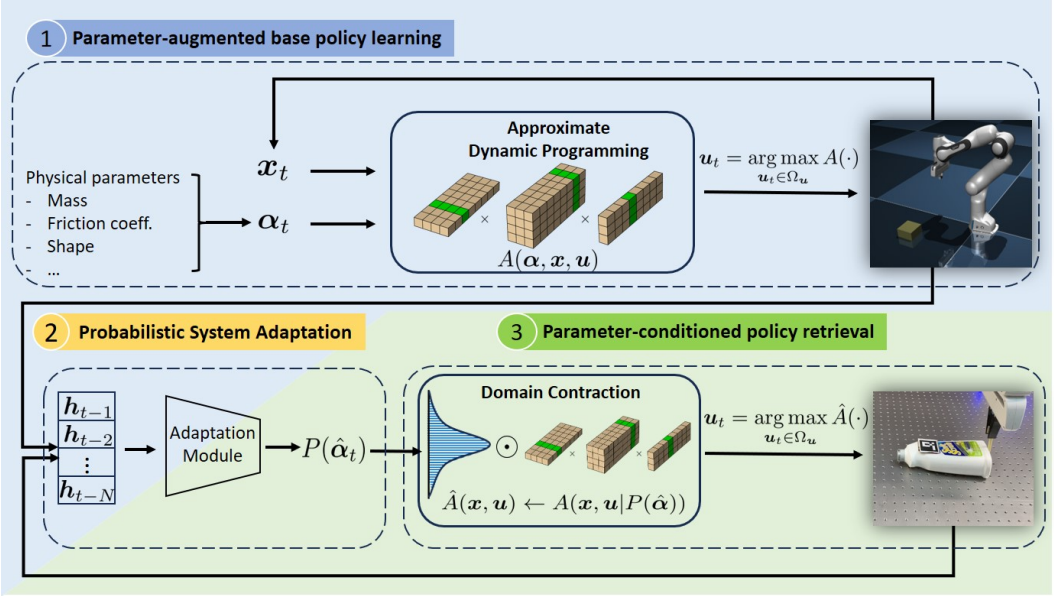


Figure 6.2: Pipeline of Implicit Motor Adaptation, including (1) parameter-augmented base policy learning, (2) probabilistic system adaptation with proprioceptive history, and (3) parameter-conditioned policy retrieval. The base policy and parameter-conditioned policy are implicitly represented by the corresponding advantage functions $A(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u})$ and $\hat{A}(\boldsymbol{x}, \boldsymbol{u})$, respectively. Blue-shaded modules are trained in simulation, and green-shaded ones are used in deployment, with the probabilistic system adaptation module bridging both stages.

6.4 Implicit Motor Adaptation

In this section, we provide a detailed introduction to *implicit motor adaptation (IMA)*, focusing on its three key components: 1) parameter-augmented base policy learning, 2) probabilistic system adaptation conditioned on proprioceptive history, and 3) parameter-conditioned policy retrieval via domain contraction. The full pipeline is shown in Fig. 6.2.

6.4.1 Parameter-augmented Base Policy Learning

Similarly to the multi-goal setting in DRL (Liu et al., 2022), we first train a parameter-augmented policy by augmenting the state space with parameters, treating both the state $\boldsymbol{x}$ and parameter $\boldsymbol{\alpha}$ as inputs and the action $\boldsymbol{u}$ as the output. This policy serves as the base policy for parameter-conditioned policy retrieval in Section 6.4.3, and is also used for data collection to train the probabilistic adaptation module discussed in

Section 6.4.2. The resulting parameter-augmented bellman equation is

$$\begin{aligned} V(\boldsymbol{\alpha}, \boldsymbol{x}) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \boldsymbol{u}_t) \mid \boldsymbol{x}_0 = \boldsymbol{x}\right], \\ A(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u}) &= R(\boldsymbol{x}, \boldsymbol{u}) + \gamma(V(f(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u})) - V(\boldsymbol{\alpha}, \boldsymbol{x})), \\ \pi(\boldsymbol{\alpha}, \boldsymbol{x}) &= \arg\max_{\boldsymbol{u} \in \Omega_{\boldsymbol{u}}} A(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u}), \end{aligned} \tag{6.4}$$

where $V(\boldsymbol{\alpha}, \boldsymbol{x})$ is the parameter-augmented state-value function, and $A(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u})$ is the parameter-augmented advantage function.

Solving (6.4) is a challenging approximate dynamic programming (ADP) problem due to the curse of dimensionality. In this work, we address this challenge by building on our previous method, TTPI (Chapter 5), which employs TT for structured function approximation. The Value Iteration algorithm (Pashenkova et al., 1996) is then used to compute the optimal value function. Upon convergence, both the optimal value function and the advantage function are obtained in Tensor Train (TT) format:

$$\begin{aligned} V(\boldsymbol{\alpha}, \boldsymbol{x}) &\approx \boldsymbol{\mathcal{V}}(\alpha_{1:d}, \boldsymbol{x}_{1:m}) \\ &= \boldsymbol{\mathcal{V}}^1_{:,i_1,:} \cdots \boldsymbol{\mathcal{V}}^d_{:,i_d,:}\, \boldsymbol{\mathcal{V}}^{d+1}_{:,i_{d+1},:} \cdots \boldsymbol{\mathcal{V}}^{d+m}_{:,i_{d+m},:}, \end{aligned} \tag{6.5}$$

$$\begin{aligned} A(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u}) &\approx \boldsymbol{\mathcal{A}}(\alpha_{1:d}, \boldsymbol{x}_{1:m}, \boldsymbol{u}_{1:l}) \\ &= \boldsymbol{\mathcal{A}}^1_{:,i_1,:} \cdots \boldsymbol{\mathcal{A}}^d_{:,i_d,:}\, \boldsymbol{\mathcal{A}}^{d+1}_{:,i_{d+1},:} \cdots \boldsymbol{\mathcal{A}}^{d+m}_{:,i_{d+m},:} \\ &\quad\, \boldsymbol{\mathcal{A}}^{d+m+1}_{:,i_{d+m+1},:} \cdots \boldsymbol{\mathcal{A}}^{d+m+l}_{:,i_{d+m+l},:}. \end{aligned} \tag{6.6}$$

6.4.2 Probabilistic System Adaptation

The system parameters $\boldsymbol{\alpha}$ (also called privileged environment information) is typically not accessible during execution in the real world. However, we can probabilistically estimate it as $\hat{\boldsymbol{\alpha}}$, drawn from a probability distribution $P(\hat{\boldsymbol{\alpha}})$. This distribution can be inferred from the proprioceptive history, namely

$$P(\hat{\boldsymbol{\alpha}}_t) = \phi(\boldsymbol{x}_{t-k:t-1}, \boldsymbol{u}_{t-k,t-1}). \tag{6.7}$$

We name this process as probabilistic system adaptation. To learn the mapping between the proprioceptive history and $P(\hat{\boldsymbol{\alpha}})$, a dataset is required containing the state-action history $\boldsymbol{h} = \{\boldsymbol{x}_{t-k:t-1}, \boldsymbol{u}_{t-k:t-1}\}$ and the corresponding system parameter $\boldsymbol{\alpha}_t$, which is accessible in simulation. Various approaches, such as energy-based models (EBMs) (LeCun et al., 2006) and diffusion models (Ho et al., 2020), can be employed

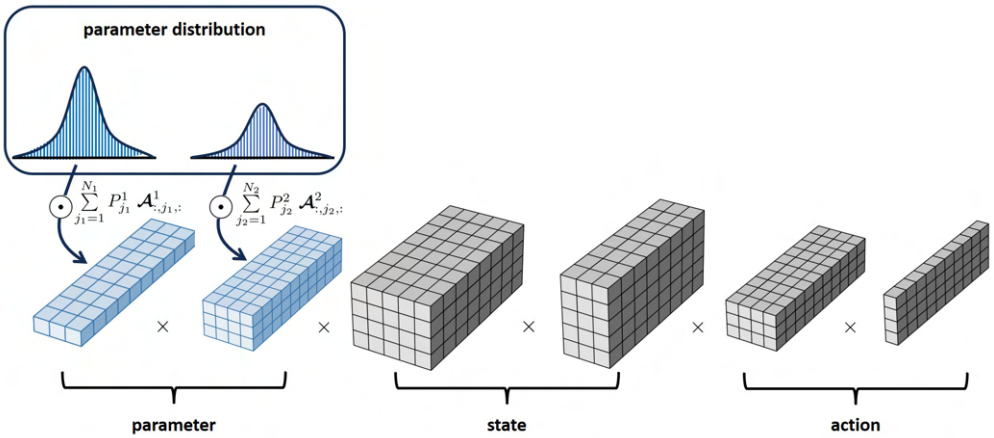


Figure 6.3: Domain contraction in TT format. The parameter-augmented advantage function in TT format typically includes separate 3rd-order cores for different dimensionality, such as parameter, state and action. Given a probabilistic parameter distribution, we can retrieve the parameter-conditioned policy by making product of parameter distributions and corresponding TT cores.

to learn this distribution. In this work, the subsequent probabilistic policy retrieval (Section 6.4.3) reduces the strong reliance on precise parameter estimation. We use a straightforward multilayer perceptron (MLP) to approximate ϕ, while noting that more sophisticated models could be explored in future extensions.

In practice, the loss function is defined as MSE $(\boldsymbol{\nu}_t, \boldsymbol{\alpha}_t) = \|\boldsymbol{\nu}_t - \boldsymbol{\alpha}_t\|^2$, where $\boldsymbol{\nu}_t$ is the output of MLP. We assume that $P(\hat{\boldsymbol{\alpha}}_t)$ follows a uniform distribution $\mathbb{U}(\boldsymbol{\nu}_t - \boldsymbol{w}/2, \boldsymbol{\nu}_t + \boldsymbol{w}/2)$, where $\boldsymbol{w}$ is a hyperparameter representing the bandwidth of $\mathbb{U}$. Note that our framework is flexible and can incorporate other distributions as well.

6.4.3 Parameter-conditioned Policy Retrieval

Without loss of generality, we assume the contact-rich manipulation task involves d types of different parameters, such as mass, size and friction coefficient. The parameter space Ω_α is therefore composed of d subspaces, namely $\Omega_\alpha = \Omega_{\alpha_1} \times \cdots \times \Omega_{\alpha_d}$. We assume each subspace is discretized by N_i points. Let $\boldsymbol{\alpha}_j = (\alpha_{j_1}, \cdots, \alpha_{j_d})$ represents one instance of domain parameter, at the discretization index j across all d dimensions. Similarly, the parameter distribution $P(\hat{\boldsymbol{\alpha}})$ is defined within Ω_P, also composed of d subspaces, namely $\Omega_P = \Omega_{P_1} \times \cdots \times \Omega_{P_d}$, and each subspace is discretized by N_i points. Let $P_j = P_1(\hat{\alpha}_{j_1}) P_2(\hat{\alpha}_{j_2}) \cdots P_d(\hat{\alpha}_{j_d})$ represents the probability of the estimated value $\hat{\boldsymbol{\alpha}}_j$, where P_i is the probability distribution at dimension $i \in \{1, \cdots, d\}$.

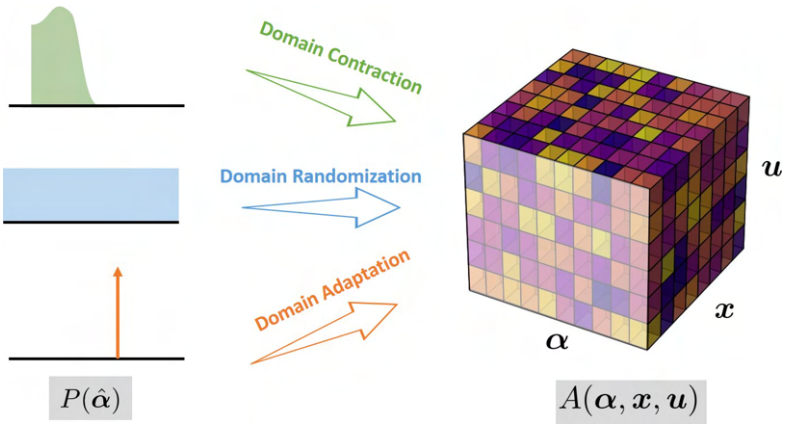


Figure 6.4: Relationship between domain contraction, domain randomization, and domain adaptation. Domain contraction unifies domain randomization and domain adaptation by adopting different parameter distributions.

Given the estimated parameter distribution $P(\hat{\boldsymbol{\alpha}})$, obtaining the corresponding parameter-conditioned policy $\pi(\boldsymbol{x}|P(\hat{\boldsymbol{\alpha}}))$ is not free. It is neither simply averaging the parameter estimates $\sum_{j=1}^{N} P_j \hat{\boldsymbol{\alpha}}_j$ and directly feeding it into $\pi(\boldsymbol{\alpha}, \boldsymbol{x})$, nor merely taking the weighted sum of parameter-specific policies $\sum_{j=1}^{N} P_j \pi(\hat{\boldsymbol{\alpha}}_j, \boldsymbol{x})$. Instead, it involves finding the $\arg\max$ of the weighted sum of parameter-specific advantage functions, namely

$$A(\boldsymbol{x}, \boldsymbol{u}|P(\hat{\boldsymbol{\alpha}})) = \sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j A_{\hat{\boldsymbol{\alpha}}_j}(\boldsymbol{x}, \boldsymbol{u}). \tag{6.8}$$

This process is termed *domain contraction*, as illustrated in Fig. 6.3. Intuitively, domain contraction randomizes over a contracted domain based on instance-specific parameter distribution, instead of randomizing over the full domain by blindly ignoring all instance-specific information (as in domain randomization). We further provide the theoretical proof below:

Proposition 1. *Given the estimated parameter distribution $P(\hat{\boldsymbol{\alpha}})$, the parameter-conditioned policy can be retrieved from the weighted sum of parameter-specific advantage functions.*

Proof. Obtaining the parameter-conditioned policy involves optimizing the parameter-augmented function

$$\begin{aligned} A(\boldsymbol{x}, \boldsymbol{u}|P(\hat{\boldsymbol{\alpha}})) =& R(\boldsymbol{x}, \boldsymbol{u}) + \gamma\Big(V\Big(f(\boldsymbol{x}, \boldsymbol{u}|P(\hat{\boldsymbol{\alpha}}))\Big) - V(\boldsymbol{x}|P(\hat{\boldsymbol{\alpha}}))\Big), \\ & \forall (P, \boldsymbol{x}, \boldsymbol{u}) \in \Omega_P \times \Omega_{\boldsymbol{x}} \times \Omega_{\boldsymbol{u}}, \end{aligned} \tag{6.9}$$

with

$$V(\boldsymbol{x}|P(\hat{\boldsymbol{\alpha}})) = \sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j \; V(\boldsymbol{x}|\hat{\alpha}_{j_1}, \cdots, \hat{\alpha}_{j_d}),$$
$$\sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j = 1. \tag{6.10}$$

For simplicity, we use $\hat{\boldsymbol{\alpha}}_j$ to represent $(\hat{\alpha}_{j_1}, \cdots, \hat{\alpha}_{j_d})$. Given that each subspace Ω_{α_i} of the parameter space Ω_α is discretized by N_i points, the parameter-conditioned advantage function can be written as

$$A(\boldsymbol{x}, \boldsymbol{u}|P(\hat{\boldsymbol{\alpha}})) = R(\boldsymbol{x}, \boldsymbol{u}) + \sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j \; \gamma\Big(V\Big(f(\boldsymbol{x}, \boldsymbol{u}|\hat{\boldsymbol{\alpha}}_j)\Big) - V(\boldsymbol{x}|\hat{\boldsymbol{\alpha}}_j)\Big). \tag{6.11}$$

The right side of (6.11) can be further rewritten as

$$\sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j \Big(R(\boldsymbol{x}, \boldsymbol{u}) + \gamma\Big(V\Big(f(\boldsymbol{x}, \boldsymbol{u}|\hat{\boldsymbol{\alpha}}_j)\Big) - V(\boldsymbol{x}|\hat{\boldsymbol{\alpha}}_j)\Big)\Big), \tag{6.12}$$

and the parameter-specific advantage function is defined as

$$A_{\hat{\boldsymbol{\alpha}}_j}(\boldsymbol{x}, \boldsymbol{u}) = R(\boldsymbol{x}, \boldsymbol{u}) + \gamma\Big(V\Big(f(\boldsymbol{x}, \boldsymbol{u}|\hat{\boldsymbol{\alpha}}_j)\Big) - V(\boldsymbol{x}|\hat{\boldsymbol{\alpha}}_j)\Big). \tag{6.13}$$

From (6.11), (6.12), and (6.13), we can derive that the parameter-conditioned advantage function is the weighted sum of parameter-specific advantage functions, as shown in (6.8), and the parameter-conditioned primitive policy can then be computed by

$$\pi(\boldsymbol{x}|P(\hat{\boldsymbol{\alpha}})) = \arg\max_{\boldsymbol{u} \in \Omega_{\boldsymbol{u}}} A(\boldsymbol{x}, \boldsymbol{u}|P(\hat{\boldsymbol{\alpha}})). \tag{6.14}$$

Note that the parameter-conditioned policy cannot be directly computed as the weighted sum of parameter-specific policies, since the $\arg\max$ operation does not have an associative property with respect to addition. □

Computing (6.8) and then retrieve the parameter-conditioned policy can be computationally expensive due to the combinatorial complexity and $\arg\max$ over an arbitray function. We address this issue by leveraging the separable structure of TT format for efficient algebraic operation, and its advantage of finding optimal solutions for functions in TT format. In Sec. 6.4.1, we have approximated the parameter-augmented

advantage functions in TT format. We define the tensor cores related to $\boldsymbol{x}$ and $\boldsymbol{u}$ in (6.6) as

$$\boldsymbol{\mathcal{A}}(\boldsymbol{x}_{1:m}, \boldsymbol{u}_{1:l}) = \boldsymbol{\mathcal{A}}^{d+1}_{:,i_{d+1},:} \cdots \boldsymbol{\mathcal{A}}^{d+m}_{:,i_{d+m},:} \cdots \boldsymbol{\mathcal{A}}^{d+m+l}_{:,i_{d+m+l},:}, \tag{6.15}$$

thus the parameter-specific advantage functions can be extracted as

$$\begin{aligned} A_{\hat{\boldsymbol{\alpha}}_j}(\boldsymbol{x}, \boldsymbol{u}) &\approx \boldsymbol{\mathcal{A}}(\boldsymbol{x}_{1:m}, \boldsymbol{u}_{1:l} | \hat{\boldsymbol{\alpha}}_j) \\ &= \boldsymbol{\mathcal{A}}^{1}_{:,j_1,:} \cdots \boldsymbol{\mathcal{A}}^{d}_{:,j_d,:} \, \boldsymbol{\mathcal{A}}(\boldsymbol{x}_{1:m}, \boldsymbol{u}_{1:l}), \end{aligned} \tag{6.16}$$

The parameter probability P_j for parameter instance $\hat{\boldsymbol{\alpha}}_j$ is given by

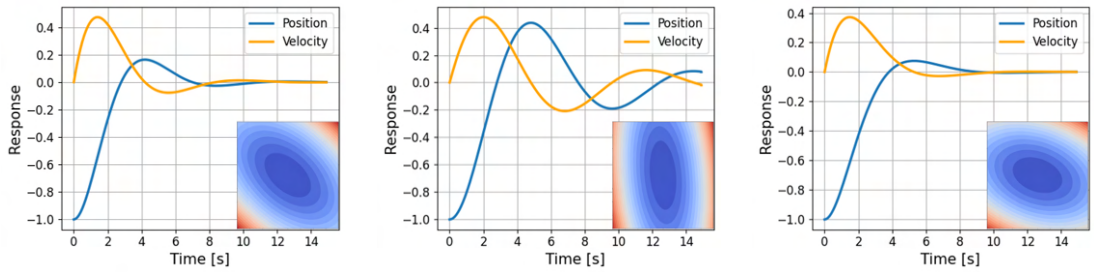
$$P_j = P_1(\hat{\alpha}_{j_1}) P_2(\hat{\alpha}_{j_2}) \cdots P_d(\hat{\alpha}_{j_d}). \tag{6.17}$$

For simplicity, we denote $P_d(\hat{\alpha}_{j_d})$ as $P^d_{j_d}$. The parameter-conditioned advantage function in (6.8) can then be efficiently computed as the weighted sum of parameter-specific advantage functions for each dimension, utilizing the separable structure of the TT model, namely

$$\begin{aligned} A(\boldsymbol{x}, \boldsymbol{u} | P(\hat{\boldsymbol{\alpha}})) &\approx \sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j \boldsymbol{\mathcal{A}}(\boldsymbol{x}_{1:m}, \boldsymbol{u}_{1:l} | \hat{\boldsymbol{\alpha}}_j) \\ &= \sum_{j_1=1}^{N_1} P^1_{j_1} \, \boldsymbol{\mathcal{A}}^{1}_{:,j_1,:} \cdots \sum_{j_d=1}^{N_d} P^d_{j_d} \, \boldsymbol{\mathcal{A}}^{d}_{:,j_d,:} \, \boldsymbol{\mathcal{A}}(\boldsymbol{x}_{1:m}, \boldsymbol{u}_{1:l}). \end{aligned} \tag{6.18}$$

The computation of weighted sum is at the tensor core level, which is much more efficient than in the function level. After obtaining the parameter-conditioned advantage function, we can retrieve the parameter-conditioned policy using tensor optimization, as described in chapter 4.

To retrieve such parameter-conditioned policies, knowing the parameter distribution $P(\hat{\boldsymbol{\alpha}})$ is crucial. Domain randomization (DR) and domain adaptation (DA) rely on assumed distributions during training, while domain contraction (DC) offers a better way to leverage domain knowledge during execution, enabling optimal behaviors while preserving generalization capability. We further demonstrate that DR and DA are two special cases of DC, as shown in Figure 6.4.



(a) Ground truth (b) EMA result (c) IMA result

Figure 6.5: Position and velocity responses of a spring-damper system, along with the approximated value functions (figure insets as colormaps), for the policies derived from the ground truth, EMA, and IMA.

6.4.4 Theoretical Analysis

EMA is a commonly used approach for robust policy learning. Given that precise system identification is intractable in real-world scenarios, we present a theoretical proof demonstrating that the control policy obtained by IMA is theoretically superior to that of EMA for practical applications.

Proposition 2. *Given an estimate of domain parameter $\boldsymbol{\alpha}$, the parameter-conditioned policy derived through implicit motor adaptation achieves higher performance than that obtained through explicit motor adaptation.*

Proof. Knowing the exact parameter $\boldsymbol{\alpha}$ for the real-world domain is intractable, but we can represent it probabilistically. Let $P(\hat{\boldsymbol{\alpha}})$ denote the probabilistic estimation obtained using any off-the-shelf approach. Therefore, the objective of robust policy learning is to find a policy π that maximizes

$$J_\pi(\boldsymbol{x}, P(\hat{\boldsymbol{\alpha}})) = \mathbb{E}_{\hat{\boldsymbol{\alpha}} \sim P(\hat{\boldsymbol{\alpha}})} \left[\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \boldsymbol{u}_t) \middle| \boldsymbol{x}_0 = \boldsymbol{x} \right] \right], \tag{6.19}$$
$$\text{s.t.} \quad \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t, \hat{\boldsymbol{\alpha}}).$$

The key difference between IMA and EMA lies in the method used to retrieve the parameter-conditioned policy. EMA determines the domain parameter $\boldsymbol{\alpha}$ with a single estimate, defined as $\hat{\boldsymbol{\alpha}} = f_\theta(\boldsymbol{h})$. The obtained $\hat{\boldsymbol{\alpha}}$ is then used as input to the parameter-augmented base policy. The resulting parameter-conditioned policy π_e

aims to maximize

$$V(\boldsymbol{x}|\hat{\boldsymbol{\alpha}}) = V(\hat{\alpha}_{j_1}, \cdots, \hat{\alpha}_{j_d}, \boldsymbol{x}). \tag{6.20}$$

IMA estimates the true parameter $\boldsymbol{\alpha}$ using a probability distribution $\hat{\boldsymbol{\alpha}} \sim P(\hat{\boldsymbol{\alpha}})$, and then retrieves the parameter-conditioned policy π_i through domain contraction, with the objective of maximizing

$$V(\boldsymbol{x}|P(\hat{\boldsymbol{\alpha}})) = \sum_{j_1=1}^{N_1} \cdots \sum_{j_d=1}^{N_d} P_j V(\hat{\alpha}_{j_1}, \cdots, \hat{\alpha}_{j_d}, \boldsymbol{x}), \tag{6.21}$$

where $\hat{\boldsymbol{\alpha}} \sim P(\hat{\boldsymbol{\alpha}})$.

Given the same dataset used in EMA and IMA adaptation module training, $\hat{\boldsymbol{\alpha}}$ can be considered a single sample from $P(\hat{\boldsymbol{\alpha}})$. Therefore, we have

$$J_{\pi_i}(\boldsymbol{x}, P(\hat{\boldsymbol{\alpha}})) \geq J_{\pi_e}(\boldsymbol{x}, P(\hat{\boldsymbol{\alpha}})). \tag{6.22}$$

This indicates that the policy obtained by EMA typically demonstrates poorer performance compared to that obtained by IMA. □

Example 1. *Spring-damper System*

To numerically illustrate the difference, we provide a spring-damper system with parameters $\boldsymbol{\alpha} = (m, k, b)$ as a toy example, where m represents the mass, k the spring stiffness, and b the damping coefficient. The system is described by the following state-space equations

$$\dot{\boldsymbol{x}}_t = \mathbf{A}\boldsymbol{x}_t + \mathbf{B}u_t,$$

where $\boldsymbol{x}_t$ includes the displacement and velocity, and u_t is the control input. The system matrices are expressed as

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}.$$

The optimal control law and value function can be obtained by solving the Algebraic Riccati Equation (ARE) (Lancaster, 1995), and they are strongly influenced by the parameter values.

We assume that the real system parameters $\boldsymbol{\alpha} = [m, k, b]$ are unknown but can be estimated as a probability distribution $\hat{\boldsymbol{\alpha}} \sim P(\hat{\boldsymbol{\alpha}})$. Fig. 6.5 compares the results of EMA and IMA with the ground-truth optimal policy. The value function obtained by IMA closely

aligns with the ground truth, effectively stabilizing the mass at the desired equilibrium position. In contrast, the EMA policy shows significantly poorer performance. This toy example highlights the effectiveness of probabilistic system adaptation and implicit policy retrieval in a dynamical system with unknown parameters.

6.5 Experimental Results

We validate the effectiveness of the proposed method on three contact-rich manipulation tasks: `Hit`, `Push`, and `Reorientation`, as shown in Fig. 6.1. All the primitives are highly dependent on the physical parameters between the object, the robot, and the surroundings.

6.5.1 Parameter-augmented Base Policy Learning

We first learn the parameter-augmented base policy over a range of system parameters using tensor approximation. Since the physical dynamics of `Hit` are fully known, the control policy can be derived analytically. For `Push` and `Reorientation`, TTPI (Shetty et al., 2024c) is used to compute the parameter-augmented value functions and advantage functions.

Hit: Hitting involves manipulating objects through impact, which is a representative one-shot manipulation task. This means the object can only be hit once, with no additional actions allowed. Determining an appropriate impact is therefore crucial and varies significantly depending on instance-specific domain parameters. In this work, we focus on the planar hitting primitive. The state is the object position, denoted as $\boldsymbol{x} = [x, y]$, and the control input is the applied impact $\boldsymbol{u} = [I_x, I_y]$. The physical parameters $\boldsymbol{\alpha}$ include the object mass m and the friction coefficient μ.

Given the mass m, the friction coefficient μ, the initial state $\boldsymbol{x}_0$, and the target $\boldsymbol{x}^{\text{des}}$, we define the following single-step reward:

$$r(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u}) = -\Big(\|\boldsymbol{x} - \boldsymbol{x}^{\text{des}}\|^2 \; + \; 0.01\,\|\boldsymbol{u}\|^2\Big), \tag{6.23}$$

where

$$\boldsymbol{x} = \boldsymbol{x}_0 + \frac{\boldsymbol{u}}{m}\,t - 0.5\,\frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}\,\mu\,g\,t^2.$$

Since the hitting task terminates after a single action, this immediate reward function r effectively serves the same role that an "advantage function" would in a multi-step setting, differing only by a constant. We use TT-cross to approximate $r(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{u})$ across

diverse parameter instances in TT format.

Push: Pushing is widely recognized as a challenging task in robot planning and control, primarily due to its hybrid and under-actuated nature (Mason, 1986; Lynch and Mason, 1996; Mason, 1999). In this chapter, we further complicate the problem by considering diverse parameters and aiming to find the instance-aware optimal policy for each. The state of the planar pushing task is characterized by $[s_x, s_y, s_\theta, \psi, \phi]$, and the action is represented as $[f_x, f_y, \dot{\psi}, \dot{\phi}]$. Here, $[s_x, s_y, s_\theta] \in SE(2)$ denotes the position and orientation of the object in the world frame. ψ is the relative angle of the contact point in the object frame, and ϕ represents the distance between the contact point and the object surface. The forces exerted on the object are denoted by $\boldsymbol{f} = [f_x, f_y]^\top$, while $\boldsymbol{v}_p = [\dot{\psi}, \dot{\phi}]^\top$ represents the angular and translational velocities of the robot's end-effector. The physical parameters include the object mass m, radius r, and the friction coefficient μ between the object and the table.

The robot dynamics is defined based on the quasi-static approximation and the limit surface, resulting in a similar expression as (Hogan and Rodriguez, 2020b). To learn the parameter-augmented advantage function, the reward function is defined as

$$r = -(\rho c_p + c_o + 0.01 c_f + 0.01 c_v), \tag{6.24}$$

with

$$\begin{aligned} c_p &= \|\boldsymbol{x}_p - \boldsymbol{x}_p^{\text{des}}\|/l_p, \quad c_o = \|x_o - x_o^{\text{des}}\|/l_o, \\ c_f &= \|\boldsymbol{f}\|, \quad c_v = \|\boldsymbol{v}_p\|, \end{aligned} \tag{6.25}$$

where $\boldsymbol{x}_p = [s_x, s_y]$ and $x_o = \theta$ denote the object's position and orientation. Without loss of generality, we set $\boldsymbol{x}^{\text{des}} = \boldsymbol{0}$ as the target configuration. $\boldsymbol{f}$ is the control force, while $\boldsymbol{v}_p$ is the velocity of the robot's end-effector. l_p and l_o are set to 0.005 and 0.01π, respectively.

Reorientation: In this task, we aim to enable the robot to reorient an object using parallel fingers. The state is the orientation angle θ, and the control input is the normal force f_n between the gripper and object. The physical parameters are the object mass m, length l and torsional friction coefficient μ. The gravitational torque and normal force are used as braking mechanisms to slow down the object motion. We build the dynamics model of the reorientation primitive based on (Viña Barrientos et al., 2016) as

$$\begin{aligned} I\ddot{\theta} &= \tau_g + 2\tau_f, \\ \dot{\theta} &= \dot{\theta}_0 - \ddot{\theta}\Delta t, \end{aligned} \tag{6.26}$$

where $\tau_f = \mu_t f_n^{1+\xi}$ is the torsional sliding friction between robot gripper and the object. In this work, we set $\xi = 0$. μ_t is the torsional friction coefficient, which is related to

the materials and normal force distribution. $\tau_g = mgl\sin(\theta)$ is the gravity torque. We therefore include μ_t, object mass m and length l as the model parameters. The task is to rotate any object from its initial configuration to a vertically upward configuration. To achieve this, the object is given an initial angular velocity $\dot{\theta}_0$ by swinging the robot arm.

The reward function for `Reorientation` primitive learning is defined as

$$r = -(\beta c_g + c_f), \tag{6.27}$$

with $c_g = \|x_o - x_o^{\text{des}}\|$, $c_f = \|f_n\|$. x_o is the orientation angle θ, and f_n is the normal force between the gripper and object. x_o^{des} is set to π as the reorientation goal, and β is set to 10^4.

In our experiments, we employed an NVIDIA GeForce RTX 3090 GPU with 24GB of memory. The accuracy parameter for cross approximation was set to $\epsilon = 10^{-3}$ for TT-Cross approximation. The maximum rank $r_{\max}$ and discount factor were set to 100 and 0.99, respectively. The continuous variables of state, action and parameter domains were discretized as 50 to 500 points using uniform discretization.

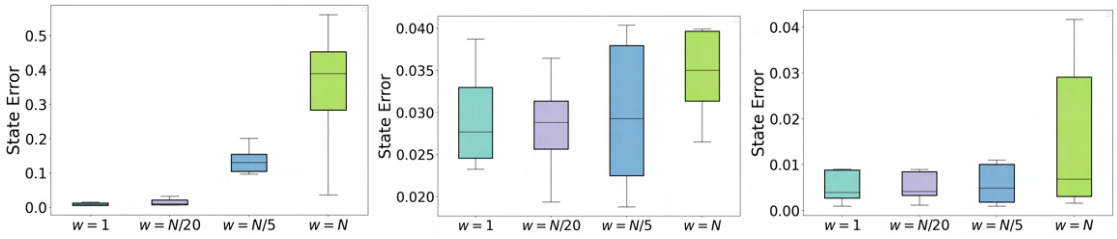
6.5.2 Results of Domain Contraction for Policy Retrieval

Given the parameter-augmented advantage functions learned in Section 6.5.1, we can now retrieve the parameter-conditioned policy for each specific instance through domain contraction (DC). This section aims to demonstrate the unification and flexibility of DC compared to domain randomization (DR) and domain adaptation (DA). To illustrate this, we make a simplifying assumption, without loss of generality, that the system adaptation module provides a uniform distribution as the probabilistic representation of parameters. This distribution spans a range defined by the discretization indices of each dimension (denoted as w), as shown in Table 6.1 and Fig. 6.6. Specifically, $w = 1$ indicates that the exact value of the physical parameter is known, corresponding to DA. $w = N$ reflects no prior knowledge about the model parameters, corresponding to DR. These two variants can be seen as extreme cases of DC, where w determines how finely the distribution is utilized for policy retrieval. Nonetheless, our framework is general and flexible, and can accommodate arbitrary parameter distributions, such as those produced by diffusion models (Ho et al., 2020).

Table 6.1 and Fig. 6.6 demonstrate the comparisons of cumulative reward and final state error, respectively. The state error is quantified as the ℓ_2 norm of the difference between the final state and the target state. The cumulative reward is normalized using

Table 6.1: Cumulative reward of three manipulation tasks

	$w = 1$	$w = N/20$	$w = N/5$	$w = N$
Hit	1.0	0.65 ± 0.21	0.01 ± 0.01	0.02 ± 0.05
Push	1.0	0.99 ± 0.01	0.99 ± 0.03	0.93 ± 0.11
Reori.	1.0	0.99 ± 0.04	0.99 ± 0.07	0.85 ± 0.19

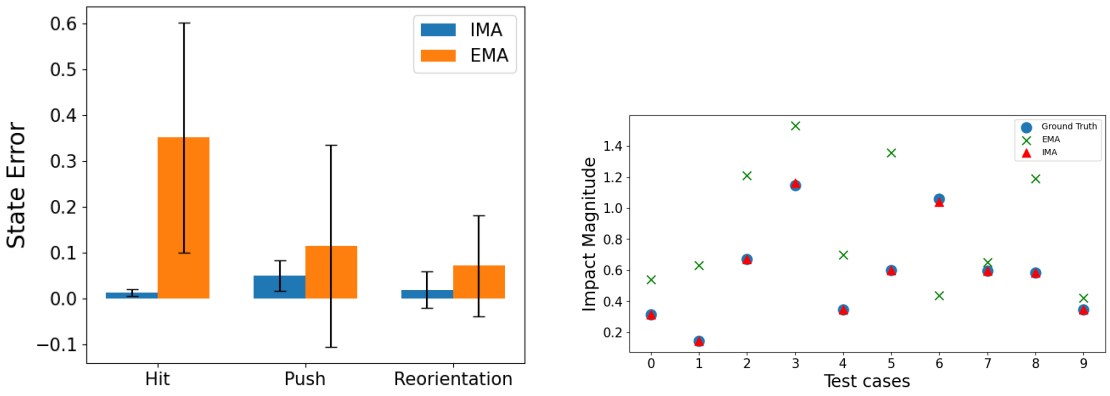


(a) Hit (b) Push (c) Reorientation

Figure 6.6: Ablation study on parameter distribution. We compare the final state errors under different estimated parameter distributions.

the value obtained through DA. We can observe that DA ($w = 1$) typically performs the best in all these three tasks, in particular for the `Hit` primitive, as it resembles an open-loop control. Once the impact is given from the robot to the object, there is no way to adjust control inputs to influence the object movements further. However, knowing the exactly correct domain parameter is intractable (as shown in Fig. 6.8). Instead, domain contraction does not require one specific value and accommodate probabilistic estimation. For example, results show that a rough range ($w = N/20$) is sufficient to achieve the target for the `Hit` primitive. Furthermore, `Push` and `Reorientation` can be considered more akin to closed-loop control, which allows for much more rough parameter estimation. As depicted in Table 6.1 and Fig. 6.6, a rough distribution with $w = N/5$ is adequate.

Moreover, based on Table 6.1, we observe that $w = N$ results in the lowest cumulative reward. This is consistent with our assertion that DR typically leads to conservative behaviors. Although DA($w = 1$) yields the highest cumulative reward, obtaining precise parameter values is often challenging in the real world. DC bridges the gap between domain adaptation and DR, offering greater flexibility to generate optimal behaviors while leveraging instance-specific rough parameter distributions. This is much practical for real-world contact-rich manipulation tasks.



(a) (b)

Figure 6.7: Comparison of IMA and EMA. (a) Final state errors for three manipulation primitives under the same base policy are compared between IMA and EMA. (b) The resulting hitting impacts of policies derived from the ground truth, EMA, and IMA are shown.

6.5.3 Comparison of Implicit and Explicit Motor Adaptation

In this section, we compare the performance of IMA and EMA on each task with 10 different instances, by randomizing system parameters and initial conditions while using the same base policy. As shown in Fig. 6.7a, IMA generally outperforms EMA on the three contact-rich manipulation tasks, particularly in the open-loop hitting task. In this work, we employ a simple MLP with a structure of $512 \times 256 \times 128 \times 64$ for parameter estimation. EMA directly uses the output (denoted as $\hat{\boldsymbol{\alpha}}_t$) to retrieve the parameter-conditioned policy from the base policy. In contrast, IMA assumes $\hat{\boldsymbol{\alpha}}_t$ as the mean $\boldsymbol{\nu}_t$ of a uniform distribution $\mathbb{U}(\boldsymbol{\nu}_t - \boldsymbol{w}/2, \boldsymbol{\nu}_t + \boldsymbol{w}/2)$, where the range w is set to $N/20$ for `Hit` and $N/5$ for `Push` and `Reorientation`.

Fig. 6.7b illustrates the computed impacts for 10 hitting instances with different physical parameters, where the ground truth values are represented by blue dots. By utilizing a probabilistic representation of the estimated parameters and domain contraction for policy retrieval, IMA typically produces values closely aligned with the ground truth, whereas EMA often yields suboptimal results. We further analyze the performance of EMA and IMA under varying levels of parameter estimation accuracy, as shown in Fig. 6.8, and study the impact of different choices for w in DC. When parameters are precisely estimated, EMA achieves the lowest final state error, demonstrating optimal performance. However, as parameter estimation becomes less accurate, greater randomization (a larger w) is required to obtain a better policy. The dotted line and gray shaded area in the figure indicate the estimation accuracy achieved in our work, with

IMA ($w = N/3$) providing the best results. This analysis highlights the effectiveness of IMA in real-world scenarios where system parameters are unknown and can only be roughly estimated. It also demonstrates that domain randomization ($w = N$) is not always good when some domain knowledge is available, even if imperfect. We believe IMA offers a promising approach to find control policies when we have limited access to the domain knowledge.

Figures 6.9 and 6.10 further illustrate the performance of IMA and EMA in the pushing and reorientation tasks, respectively. In Fig. 6.9, the same base policy is used to push a sugar box and a mustard bottle toward their target configurations, including both position and orientation. While both approaches eventually reach the target positions, IMA outperforms EMA in terms of orientation. This is primarily due to the underactuated dynamics of the planar pushing task, where system parameters such as the friction coefficient and object mass play a critical role. IMA estimates these parameters probabilistically and retrieves parameter-conditioned policies implicitly. This allows for increased tolerance of typical system identification errors, enabling robust contact-rich manipulation with unknown parameters. Similarly, Fig. 6.10 demonstrates the superior performance of IMA over EMA in reorienting an arbitrary object vertically from the bottom to the top.

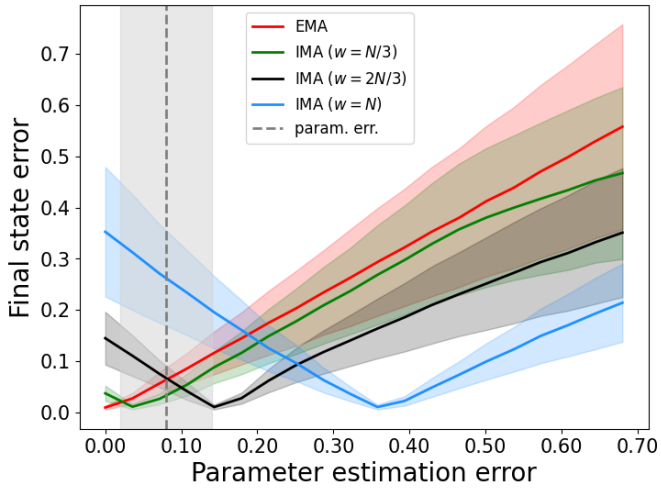


Figure 6.8: Motor adaptation under varying estimation accuracy. The resulting final state errors of different motor adaptation strategies are compared under varying levels of parameter estimation accuracy. The dotted line and gray shaded area indicate the mean and standard deviation of the parameter estimation error obtained using the system estimator in this work.

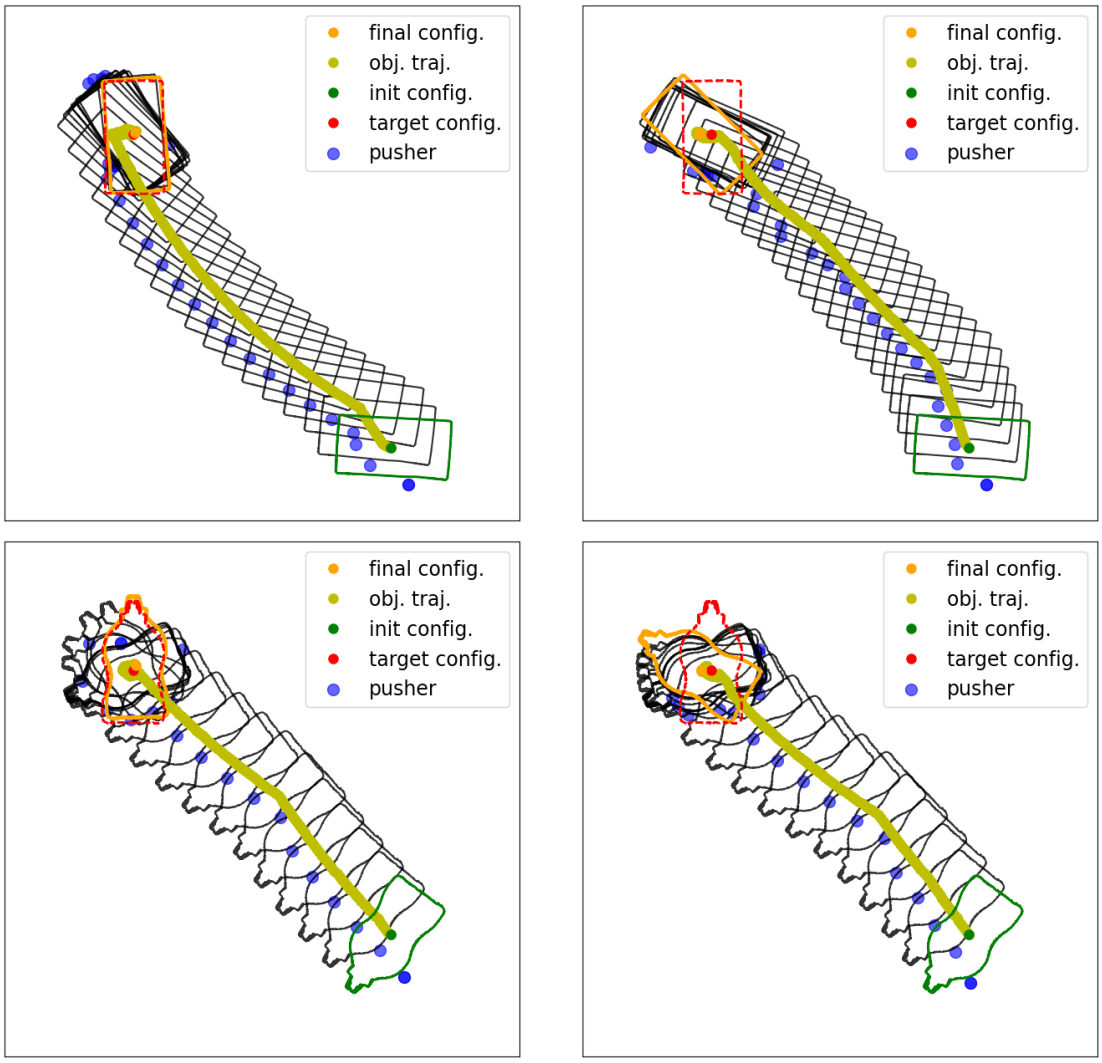


Figure 6.9: Planar pushing tasks with different objects. The top row shows results with a sugar box, while the bottom row shows results with a mustard bottle. **Left**: Object trajectories produced by IMA; **Right**: Object trajectories produced by EMA.

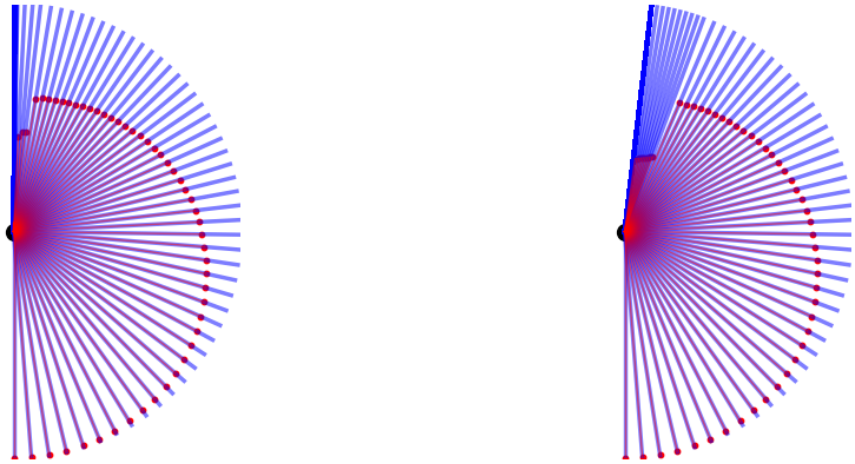


Figure 6.10: Object reorientation trajectories. Trajectories produced by IMA (**left**) and EMA (**right**) for the reorientation task are shown. The blue curves depict the trajectory of an object, with the goal of reorienting it from an initial bottom configuration to a vertical upright configuration. The lengths of the red segments indicate the magnitude of angular velocity, which is indirectly regulated through gravity and friction.

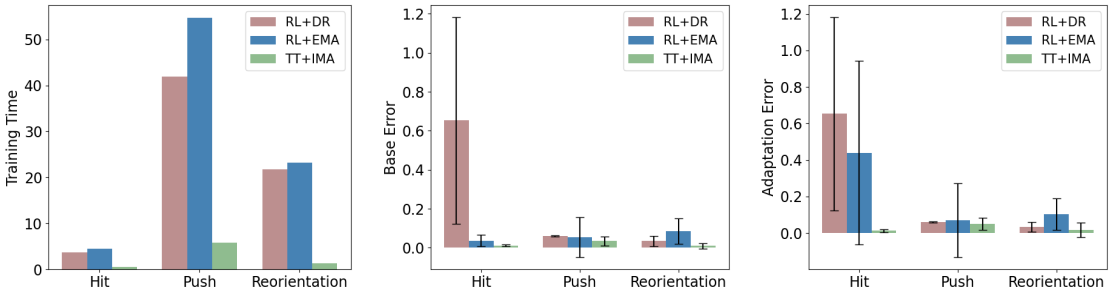
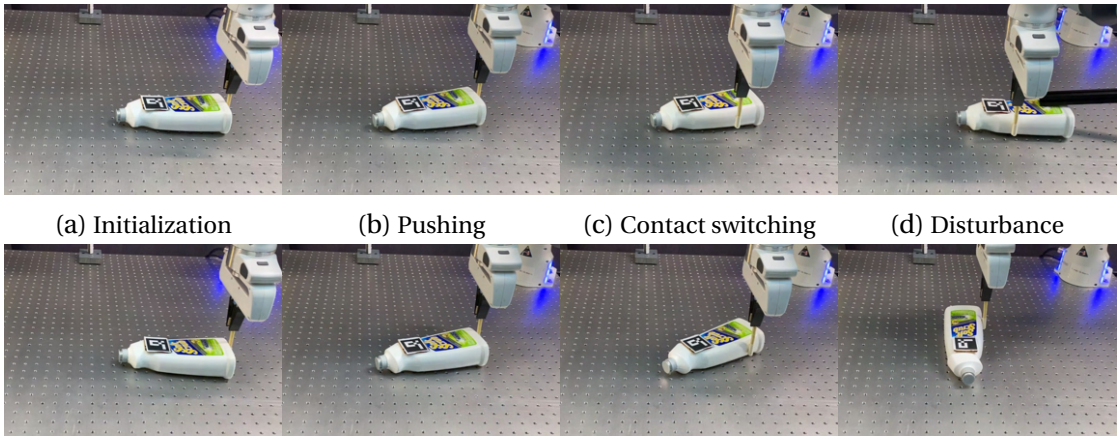


Figure 6.11: Comparison of robust primitive learning methods. Training time and state errors of base and adapted policies are compared across different approaches. Training time is reported in seconds for `Hit` and in minutes for `Push` and `Reorientation`. Errors are computed as the ℓ_2 distance between the final and target configurations.



(a) Initialization (b) Pushing (c) Contact switching (d) Disturbance

(e) Contact switching (f) Pushing (g) Contact switching (h) Target Reaching

Figure 6.12: Keyframes of planar pushing under disturbance. An example with a bleach bottle on a metal surface. The system starts from an initial configuration (a) and aims to reach the target configuration (h). The robot first establishes contact and pushes the bottle slightly (b), followed by a contact switch (c). When an external disturbance is introduced by a human (d), the robot adapts by adjusting the contact point (e). It then continues pushing (f), performs another contact switch (g), and ultimately reaches the target configuration (h).

6.5.4 Results of Robust Policy Learning

We also compared our full pipeline with two widely used robust policy learning approaches: reinforcement learning with domain randomization (RL+DR) (Tobin et al., 2017) and reinforcement learning with explicit motor adaptation (RL+EMA) (Yu et al., 2017; Qi et al., 2023). We utilize Soft Actor-Critic (SAC) (Haarnoja et al., 2018) for policy learning in both RL+DR and RL+EMA, as SAC has been shown to be the state-of-the-art RL algorithm for such contact-rich manipulation tasks, especially planar pushing, as demonstrated in (Potters, 2022; Cong et al., 2022). Our implementation is based on Stable-Baselines3 (Raffin et al., 2021), using a Multilayer Perceptron (MLP) with a $64 \times 64 \times 64$ architecture as the policy network. The discount factor is set to 0.99, and the learning rate to 0.001.

The quantitative results are presented in Fig. 6.11, including the time required for policy training and the state error of the retrieved policy. The *base_error* and *adaptation_error* refer to the final state error of the base policy with the true parameter and the parameter-conditioned policy after motor adaptation, respectively. For RL+DR, the *base_error* and *adaptation_error* are the same since no motor adaptation is performed.

Notably, although the `Hit` task is a one-shot manipulation task in which proprioceptive history is unavailable, the motor adaptation module works by observing the trajectories

Table 6.2: Time comparison for computing parameter-conditioned advantage function between core-level and function-level operations.

	core-level	function-level
Hit	$0.016\text{s} \pm 0.002\text{s}$	$0.720\text{s} \pm 0.236\text{s}$
Push	$0.075\text{s} \pm 0.003\text{s}$	$18.56\text{s} \pm 4.748\text{s}$
Reorientation	$0.018\text{s} \pm 0.003\text{s}$	$8.494\text{s} \pm 0.561\text{s}$

of past rollouts, similarly to how humans perform such tasks. The results show that our method requires significantly less time for base policy training compared to the other methods, while achieving the lowest *base_error*, highlighting the advantages of leveraging TT for learning control policies in contact-rich manipulation tasks.

Furthermore, our method (TT+IMA) achieves the lowest final state error among the three approaches, demonstrating the overall effectiveness of our complete approach for robust contact-rich manipulation tasks, with implicit action representation playing a crucial role in retrieving robust policies. An additional observation is that RL+EMA often exhibits the highest standard deviation, indicating that its control policies perform extremely poorly in some instances, whereas IMA has a better statistical performance over the diverse instances.

6.5.5 Sensitivity Analysis and Ablation Study

Our framework involves several hyperparameters, such as the maximum TT-rank, the discretization granularity, and the bandwidth of the uniform distribution. To analyze their impact on policy performance, we conduct a sensitivity analysis across a range of values for each hyperparameter. Figure 6.13 shows how policy performance, measured by the ℓ_2 error between the final and target states, varies with the maximum TT-rank and discretization granularity. For comparability across tasks, the errors are normalized by the worst-case performance in each task.

From Figure 6.13, we observe that setting the maximum rank $r_{\max}$ above 20 leads to consistently low errors across all tasks. For `Push` and `Hit`, even lower ranks (e.g., $r_{\max} = 10$ or 5) are sufficient, indicating that the underlying policies lie in a low-rank functional space. A relatively low tensor rank is sufficient to capture the essential structure, and policy performance remains stable when the rank is increased further. Empirically, we observe that the piecewise-smooth dynamics (Toussaint et al., 2020) commonly encountered in contact-rich manipulation tasks induce structured, low-dimensional variations in the state-action space. The corresponding advantage functions often exhibit a low-rank structure, which can be efficiently captured using

Table 6.3: TT vs. NN for parameter-conditioned policy retrieval. Error refers to the task execution error using the retrieved policy, while Opt. Time indicates the time required to perform the $\arg\max$ operation.

	TT Optimization		TT Refinement		NN Optimization	
	Task Error	Opt. Time (s)	Task Error	Opt. Time (s)	Task Error	Opt. Time (s)
Hit	0.010 ± 0.008	0.003 ± 0.001	0.002 ± 0.001	0.015 ± 0.001	0.002 ± 0.004	0.346 ± 0.021
Push	0.034 ± 0.024	0.013 ± 0.001	0.033 ± 0.024	0.146 ± 0.001	1.469 ± 0.574	2.020 ± 0.002
Reorientation	0.061 ± 0.104	0.002 ± 0.001	0.061 ± 0.104	0.062 ± 0.001	0.751 ± 0.680	1.514 ± 0.092

the TT representation.

Similarly, the policy is robust to the choice of discretization granularity. As shown in Figure 6.13, when the number of discretization intervals exceeds 8, performance does not change much with further refinement, indicating insensitivity to this hyperparameter. This robustness can be attributed to the fact that low-rank TT models provide smooth function approximations, which support effective interpolation between discretized points (as discussed in Section 2.1.5). While extremely coarse discretization may degrade performance, in practice, setting the number of intervals to 10 or 20 generally yields stable and accurate results.

For the influence of the bandwidth for the uniform distribution used in probabilistic domain adaptation, we refer the readers to Section 6.5.2 for details. In summary, a coarse bandwidth ($w = N/5$, where N is the number of discretization points) is sufficient for closed-loop tasks such as `Push` and `Reorientation`. In contrast, `Hit`, as a one-shot (open-loop) task, benefits from a more precise distribution, with $w = N/20$ being adequate. In general, the bandwidth parameter for other contact-rich tasks can be determined similarly based on whether the task is open-loop or closed-loop, without requiring significant tuning effort.

To further demonstrate the efficiency of the TT representation, we compared it against non-TT function approximation methods, particularly Neural Networks (NN). In our framework, both TT and NN can be used to learn parameter-augmented base policies and to approximate parameter-augmented advantage functions. The key question is which method offers greater efficiency in the subsequent steps: computing the parameter-conditioned advantage function and retrieving the corresponding policy via an $\arg\max$ operation.

Table 6.2 highlights the significant computational advantage of TT in computing the advantage function. Specifically, TT models support efficient *core-level* operations, which avoid combinatorial computation across the multi-dimensional parameter space, leading to a complexity of $\mathcal{O}(Ndr^2)$, where N is the number of discretization

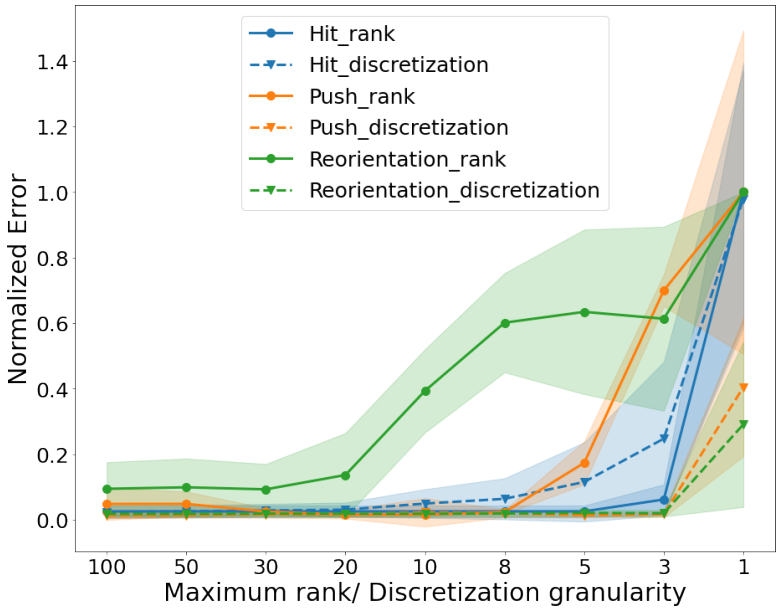


Figure 6.13: Sensitivity to TT-rank and discretization. Policy performance is evaluated with respect to the maximum TT-rank (solid lines) and discretization granularity (dashed lines). The y-axis shows the normalized ℓ_2 error between the final and target states, scaled by the worst-case performance in each task. The x-axis denotes the maximum TT-rank and the number of discretization intervals (non-uniformly spaced to improve readability in low-rank and coarse-discretization regimes). The results indicate that policy performance is relatively insensitive to these hyperparameters.

points per dimension, d is the dimensionality of the parameter space, and r is the TT rank, which typically remains small in practice, as demonstrated in our experiments. In contrast, NN-based methods (or any non-TT models) rely on *function-level* operations, resulting in a complexity of $\mathcal{O}(N^d)$, which makes the computation substantially slower.

After computing the parameter-conditioned advantage function, the next step is policy retrieval via an $\arg\max$ operation. As shown in Table 6.3, TT again demonstrates superior efficiency. Direct TT-based optimization (TT Optimization) yields low-error policies with minimal computation time, achieving near-global solutions within milliseconds. In contrast, NN-based approaches (NN Optimization) typically rely on gradient-based solvers. In this work, we specifically adopt the Adam optimizer (Kingma and Ba, 2015) as a representative method for comparison. The results indicate that this approach is slower and more susceptible to suboptimal convergence, particularly in complex tasks such as `Push` and `Reorientation`.

To further improve accuracy, we also evaluated a hybrid approach (TT Refinement), which uses TT optimization for initialization, followed by a gradient-based Newton-type method. While this refinement slightly improves performance, the gain is often marginal compared to the additional computational cost. In the tasks considered in this work, TT optimization alone is sufficient to achieve high-quality results. These findings highlight the strong potential of TT representation for efficient and robust policy learning in complex, contact-rich manipulation tasks.

6.5.6 Real-robot Experiments: Planar Push

We validated the proposed method for the planar pushing task using a 7-axis Franka robot and a RealSense D435 camera. The manipulated objects included a sugar box and a bleach cleanser from the YCB dataset (Calli et al., 2015), each with different shapes and masses, as shown in Fig. 6.1. The friction coefficients between the objects and the table were varied by using a metal surface and plywood, respectively. Note that it is easier to control the robot kinematically rather than using force control. We therefore leveraged the ellipsoidal limit surface to convert the applied force to velocity, resulting in the motion equations shown in (Hogan and Rodriguez, 2020a; Xue et al., 2023b). We trained a parameter-augmented policy in simulation and then applied it in the real world through domain contraction. The experiments demonstrate the effectiveness of the obtained parameter-conditioned policies in manipulating instances with diverse parameters. Additionally, external disturbances were introduced by humans, showcasing the reactivity of the retrieved policy. Figure 6.12 presents keyframes of the learned planar pushing primitive applied to a bleach bottle on a metal

surface, exemplifying a specific case within a variety of diverse pushing scenarios.

6.6 Discussion

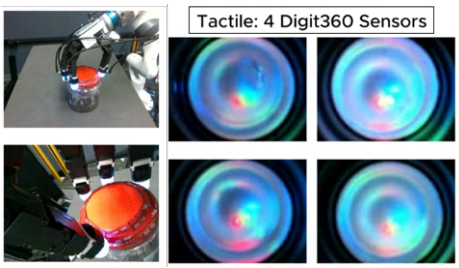
In this chapter, we propose *implicit motor adaptation* for robust manipulation primitive learning, which enables parameter-conditioned policy retrieval given a probabilistic parameter distribution rather than a single estimate. We prove that, to achieve this, the policy must be represented implicitly as the $\arg\max$ of the advantage function, rather than treated as an explicit feed-forward function. The state-value and advantage functions are represented in TT format, facilitating efficient policy retrieval through operations at the tensor core level instead of at the function level. Our approach differs from the widely used *explicit motor adaptation* framework, which requires training an additional student policy to imitate a latent parameter embedding and the corresponding control actions. Learning to accurately reproduce a single value is often difficult and prone to errors. Instead, our method estimates a probabilistic distribution over environmental parameters and retrieves the appropriate parameter-conditioned policy from this distribution. Theoretical analysis and numerical results demonstrate the superior performance of *implicit motor adaptation* over *explicit motor adaptation*. Simulation and real-world experiments further validate the effectiveness of our approach in contact-rich manipulation tasks under parameter uncertainty and external disturbances.

This work paves the way for robust sim-to-real transfer. It can be easily integrated with more general policy learning approaches, such as reinforcement learning methods and classical approaches (e.g., LQR), provided the base policy can be implicitly represented with a state-action value function. The key insight is to represent the base policy implicitly and retrieve the parameter-conditioned policy probabilistically. This concept also holds promise for achieving robust behavior cloning, where the policy can be represented through energy functions (as in implicit behavior cloning (Florence et al., 2022)) or denoising functions (as in diffusion policy (Chi et al., 2024)).

Despite these promising results, the current implementation does not scale well to visual observations, mainly due to the reliance on TT-Cross approximation. This limitation could be mitigated by integrating Tensor Train with neural networks in a data-driven manner (Dolgov et al., 2023; Shetty, 2024). In addition, the current system identification module employs a simple MLP. More expressive models, such as generative models (Ho et al., 2020; Lipman et al., 2023b), could be adopted to improve parameter inference. Large Visual-Language Models (Gao et al., 2024) could also be explored to leverage their commonsense understanding of domain knowledge from

scenario images.

7 Reactive Visual-Tactile Policy Learning with Tube Diffusion Policy



Contact-rich manipulation requires continuous adaptation to contact uncertainty and external disturbances through multimodal perception, particularly vision and tactile feedback. Although imitation learning has shown strong potential for complex manipulation, most existing methods rely on action chunking, which limits their ability to react to unforeseen observations during execution. This limitation is especially critical in contact-rich settings, where physical uncertainty and high-frequency tactile feedback demand rapid, reactive control. To address this, we propose Tube Diffusion Policy (TDP), which exploits the geometric structure of action trajectories by modeling conventional action chunking as a bounded velocity field, termed an action tube, enabling rapid reactions during execution under contact uncertainty and external disturbances.

Publication Note

The material presented in this chapter is adapted from:

- Xue, T., Rigo, A., Huang, B., Shen, J., Xu, Z., Colonnese, N., and Memar, A. (2026). Tube diffusion policy: reactive visual-tactile policy learning for contact-rich manipulation. *under review*.

This work was conducted during my internship at Meta Reality Labs Research. Supplementary materials related to this chapter are available at: https://schortenger.github.io/tube-diffusion-policy/.

7.1 Introduction

Contact-rich manipulation presents significant challenges for control due to uncertainty in contact dynamics and the need for rapid adaptation. Variations in object geometry, inertial properties, and frictional characteristics result in highly nonlinear and difficult-to-predict system behaviors. At the same time, multi-modal perception—particularly vision–tactile sensing—introduces heterogeneous observations, where tactile signals provide high-frequency contact information that requires fast and responsive control (Xue et al., 2025a). Together, these factors make reactive control policies essential for reliable performance in contact-rich manipulation.

Recently, imitation learning based on generative models has emerged as a dominant paradigm for learning robot control policies, such as Diffusion Policy (Chi et al., 2024) and Flow Matching Policy (Zhang and Gienger, 2024). These approaches excel at capturing the multimodal distribution of expert demonstrations and have achieved strong performance across a wide range of manipulation tasks. However, a fundamental limitation of these methods lies in their slow inference, which makes it impractical to generate actions at every timestep. As a result, many approaches adopt action chunking (Zhao et al., 2023), where a sequence of future actions is generated conditioned on the current observation and then executed without intermediate feedback. While this design enables temporally coherent behavior generation and reduces the frequency of expensive inference, the policy operates in an open-loop manner over the chunk horizon. From a control perspective, such generative imitation learning policies can be interpreted as a form of receding-horizon control, analogous to model predictive control (MPC) (Kouvaritakis and Cannon, 2016), in which a nominal action trajectory is repeatedly replanned based on the latest observation. However, unlike classical MPC—which executes only the first control input and replans at high frequency—these methods execute multiple actions between updates due to inference latency, limiting their ability to incorporate high-frequency feedback. This limitation is particularly problematic in contact-rich manipulation tasks, where unmodeled interactions and rapid tactile feedback are critical for success. The lack of reactivity can lead to degraded performance under disturbances and model mismatch. Moreover, executing multi-step actions implicitly assumes accurate prediction of system evolution over the entire horizon, an assumption that often breaks down in contact-rich settings, resulting in brittle behaviors.

In the control literature, robustness to uncertainty has been widely studied (Zhou and Doyle, 1998; Green and Limebeer, 2012; Franco et al., 2006; Shtessel et al., 2014). One representative approach is tube model predictive control (tube MPC) (Langson et al., 2004; Mayne et al., 2011), which decomposes control into a nominal feedforward

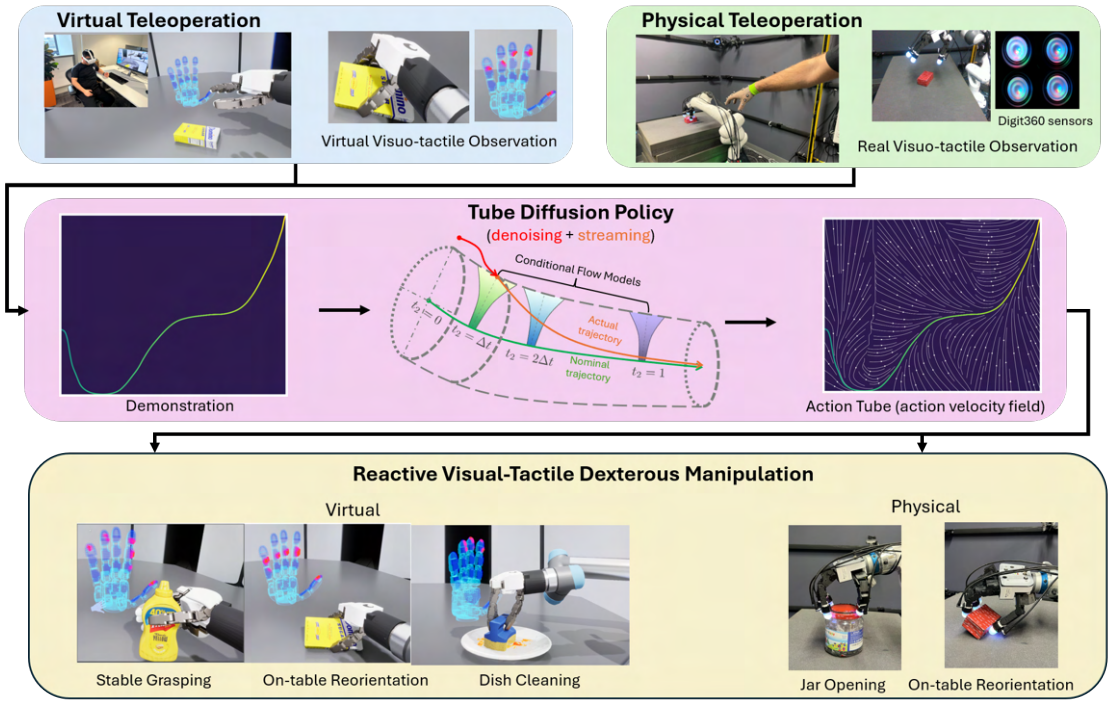


Figure 7.1: Overview of reactive visual-tactile policy learning. We first collect visual-tactile human demonstrations via teleoperation in both simulation and on a physical robot, using a robotic arm equipped with a dexterous hand. Based on these demonstrations, Tube Diffusion Policy (TDP) learns a hybrid action velocity field that combines diffusion-based action generation with streaming conditional feedback flows, forming an action tube that constrains the generated trajectory around the demonstration manifold. This design enables reactive control at every timestep using fresh observations, allowing rapid adaptation to contact uncertainty and external disturbances. Moreover, stepwise correction reduces the required denoising steps and significantly lowers inference latency. We evaluate TDP on three virtual and two physical visual–tactile dexterous manipulation tasks, demonstrating strong performance and consistent gains over state-of-the-art baselines.

trajectory and a stabilizing feedback controller that keeps the actual system state within a bounded tube around the nominal trajectory. This feedback mechanism enables fast local corrections in response to disturbances and modeling errors. However, classical tube MPC typically relies on linear or locally linear dynamics, requires explicit system models, and incurs substantial computational overhead, making it difficult to scale to complex, nonlinear, contact-rich manipulation tasks.

Inspired by the tube MPC paradigm, this work aims to endow diffusion-based imitation learning policies with similar robustness and reactivity by learning a local feedback flow around nominal trajectories. Instead of explicitly modeling system dynamics or deriving analytical feedback gains, we leverage the expressive capacity of generative models to directly learn an observation-conditioned feedback policy from demonstrations. To this end, we propose *Tube Diffusion Policy* (TDP), a reactive visual–tactile policy learning framework that augments diffusion-based imitation learning with learned feedback control during action execution. The central idea is that, rather than committing to a fixed action sequence as *action chunk*, we construct an *action tube* that implicitly constrains execution within a bounded neighborhood around a nominal trajectory while allowing actions to be locally adjusted through a learned feedback flow. In particular, at the beginning of each action chunk horizon, a diffusion model generates an initial action that accounts for the full system dynamics and serves as a nominal initialization. The learned feedback flow then streams subsequent actions conditioned on fresh observations, enabling local regulation within the action tube during execution. The entire structure is learned directly from demonstrations without requiring explicit system models, making it well suited for nonlinear and contact-rich manipulation.

To reconcile expressive generative modeling with efficient feedback control within a unified policy architecture, we introduce a dual-time formulation that explicitly separates diffusion and streaming flow in policy learning. A diffusion timescale governed by diffusion dynamics captures the multimodal action distribution while accounting for full contact dynamics, whereas a trajectory timescale governs action evolution by streaming a learned flow defined by a velocity field. This dual-time architecture enables accurate action generation through diffusion while supporting fast observation-conditioned corrections during execution. As a result, TDP mitigates the open-loop execution inherent in action chunking approaches, while retaining the expressiveness of diffusion-based policy learning.

Overall, this framework can be viewed as a learning-based realization of tube MPC under unknown dynamics. Rather than solving a model-based optimization problem, TDP leverages a learned action velocity field to induce locally stabilizing behavior.

Extensive simulation and real-world experiments on challenging visual–tactile manipulation tasks demonstrate substantial improvements over state-of-the-art imitation learning baselines in both task success and reaction under uncertainty.

The main contributions of this work are summarized as follows:

- We introduce *action tube*, which extends the widely used *action chunking* with a learned local feedback flow, enabling reactive control during execution. This formulation also relaxes the need for highly accurate action generation through denoising and thereby significantly reduces control latency.

- We propose *Tube Diffusion Policy* (TDP), a novel architecture that integrates diffusion-based action generation with learned feedback flow, enabling reactive visual–tactile policy learning for contact-rich manipulation.

- We provide a theoretical analysis showing that reactive control can be formulated as learning an observation-conditioned action velocity field under locally linearized dynamics, while diffusion performs chunk-level corrections that account for full system dynamics.

- We validate the proposed approach through extensive simulation and real-world experiments on challenging visual–tactile manipulation tasks, demonstrating consistent improvements over state-of-the-art imitation learning baselines in task success and reactivity under perturbations.

7.2 Related Work

The proposed Tube Diffusion Policy sits at the intersection of three research threads: (1) *chunk-based policy learning*, which provides the temporal abstraction and generative modeling that TDP uses for action generation; (2) *visual–tactile manipulation*, which motivates the need for high-frequency reactive control under contact uncertainty; and (3) *learning dynamical systems*, which supplies the velocity-field formulation that TDP adopts for streaming execution. As summarized in Table 7.1, each thread individually addresses only a subset of the requirements for reactive contact-rich manipulation. We review each thread below, focusing on the specific limitation that TDP is designed to overcome.

Table 7.1: Comparison of policy learning paradigms for reactive visual-tactile manipulation.

Method	Visual-Tactile Observation	Closed-loop Execution	Contact-rich Dynamics	Fast Inference
Action Chunking with Transformers (ACT) (Zhao et al., 2023)	✗	✗	✓	✓
Diffusion Policy (DP) (Chi et al., 2024)	✗	✗	✓	✗
Dynamical Systems (Ijspeert et al., 2013)	✗	✓	✗	✓
Streaming Flow Policy (SFP) (Jiang et al., 2025)	✗	✗	✗	✓
Reactive Diffusion Policy (RDP) (Xue et al., 2025a)	✓	✓	✓	✗
Tube Diffusion Policy (TDP)	✓	✓	✓	✓

7.2.1 Chunk-based Policy Learning

Action chunking is a concept originating in psychology that groups a sequence of actions into a single chunk and executes it as a unit (Lai et al., 2022). It has since been widely adopted to address long-horizon decision making in continuous control (Li et al., 2025; Zhao et al., 2023). In imitation learning, chunk-based formulations enable policies to model expert behavior at the level of short trajectory segments rather than individual actions. Early works focus on learning reusable latent skills or motion primitives (Hausman et al., 2018; Lynch et al., 2019; Xue et al., 2024a). More recent approaches directly predict action chunks using sequence models. Notably, Action Chunking Transformer (ACT) (Zhao et al., 2023) demonstrates that predicting fixed-length action sequences can significantly reduce error accumulation and enable long-horizon bimanual manipulation, even on low-cost hardware platforms. However, ACT executes the predicted chunks largely in an open-loop manner, and its performance can degrade under contact uncertainty or external disturbances within each chunk.

Generative models further enhance chunk-based policies by enabling expressive multi-step action generation. They generate entire action chunks conditioned on current observations, allowing the policy to represent multimodal behaviors and capture long-horizon dependencies (Chi et al., 2024; Zhang and Gienger, 2024; Jiang et al., 2025). While effective in visuomotor manipulation, these methods typically replan only at the chunk boundary and provide limited mechanisms for correcting high-frequency deviations caused by modeling errors or contact variations during execution.

As a result, existing chunk-based and generative policy learning methods provide powerful abstractions for long-horizon planning but remain fundamentally limited by their weak intra-chunk feedback mechanisms, particularly in contact-rich settings with tactile feedback (Liu et al., 2025; Xue et al., 2025a). Our work aims to address this issue by integrating chunk-based generative policies with structured local feedback flows, enabling systematic disturbance rejection while preserving the benefits of temporal abstraction.

7.2.2 Visual-tactile Manipulation

Visual–tactile manipulation has attracted increasing attention, particularly in contact-rich tasks that require force regulation and fine-grained interaction. Early work leveraged tactile signals for contact detection, slip estimation, and state inference, demonstrating that tactile feedback is an essential component in tasks such as grasp stabilization (Calandra et al., 2018; Bauza et al., 2024), in-hand manipulation (Suresh et al., 2024), and deformable object manipulation (Yuan et al., 2017).

Recent advances in visuomotor policy learning (Chi et al., 2024; Zhao et al., 2023; Ze et al., 2024; Zhang and Gienger, 2024) have further enabled the learning of complex manipulation behaviors, motivating the integration of tactile sensing into policy learning frameworks. Most existing approaches incorporate tactile signals as additional sensory inputs by concatenating them with visual observations into a unified observation space, and train policies end-to-end using standard visuomotor learning pipelines (Lee et al., 2020b; Hansen et al., 2022; Zhang and Demiris, 2023). While effective in many settings, such formulations often rely on computationally intensive generative models with relatively slow inference, and therefore overlook the fact that tactile sensing is inherently high-frequency and contact-driven, requiring rapid reactive responses during execution.

To better exploit tactile feedback, visual–tactile policy learning should adopt a reactive formulation. One representative approach is Reactive Diffusion Policy (RDP) (Xue et al., 2025a), which employs a slow–fast architecture that combines diffusion-based planning with fast tactile-driven refinement, enabling improved reactivity to external disturbances in contact-rich manipulation tasks. Nevertheless, it has been primarily validated on parallel grippers rather than dexterous hands, leaving its effectiveness in high-dimensional, multi-contact manipulation settings unclear. Moreover, it still relies on slow denoising at the beginning of each action chunk, introducing latency and limiting responsiveness, which can induce periodic jerky motions similar to those observed in standard diffusion policies.

Overall, most existing visual–tactile policies either execute planned action chunks in an open-loop manner or rely on auxiliary controllers that operate in latent space, limiting their ability to support fully reactive visual–tactile control for high-DoF dexterous manipulation. In contrast, TDP embeds reactivity directly into the diffusion policy through a streaming integration phase, enabling online action updates under an observation-conditioned vector field.

7.2.3 Learning Dynamical Systems

Another line of work approaches policy learning from the perspective of dynamical systems, representing robot behaviors as continuous-time vector fields or attractor dynamics (Khansari-Zadeh and Billard, 2010, 2011; Ijspeert et al., 2013; Schaal et al., 2003). These approaches impose strong structural constraints to guarantee stability, which simplifies the underlying dynamics and facilitates analysis, but limits expressiveness when dealing with highly nonlinear or contact-rich interactions.

Recent learning-based methods extend this perspective by parameterizing continuous-time dynamics with neural networks. Neural ODEs and their variants provide a flexible framework for learning nonlinear dynamical systems from data (Chen et al., 2018; Rubanova et al., 2019; Kidger et al., 2020), while recent work explicitly enforces stability properties in deep dynamical systems (Sochopoulos et al., 2024). Although more expressive, these methods are typically used as motion generators or nominal dynamics models, rather than as fully reactive controllers under complex disturbances.

In parallel, generative modeling techniques such as diffusion models, continuous normalizing flows, and flow matching reinterpret trajectory or action generation as the integration of learned vector fields (Song et al., 2021b; Ho et al., 2020; Lipman et al., 2023a; Liu et al., 2023). Building on this perspective, Streaming Flow Policy (SFP) (Jiang et al., 2025) formulates policy generation as a continuous-time dynamical system evolving in action space, enabling smooth action evolution and improved coverage of multimodal behaviors. However, due to the computational cost of generative inference, these methods primarily produce open-loop or chunk-based action trajectories, with limited ability to react to execution-time disturbances.

Overall, most prior learning-based dynamical system approaches either rely on simplified or abstracted dynamics, or structure execution around chunk-based or open-loop evolution, limiting their ability to represent complex behaviors or regulate deviations arising from disturbances or contact interactions. Our approach combines the strengths of diffusion-based policies and streaming flows, where diffusion is used to capture complex nonlinear behaviors, while observation-conditioned streaming flow enables real-time, reactive control during execution.

7.3 Problem Formulation

We consider contact-rich manipulation systems with observation $\boldsymbol{o}_t = (\boldsymbol{o}_t^o, \boldsymbol{o}_t^r) \in \mathbb{O} \subseteq \mathbb{R}^{n_o}$, where $\boldsymbol{o}_t^r$ denotes the robot's proprioceptive state and $\boldsymbol{o}_t^o$ represents the object state, which may consist of either low-dimensional object configurations or high-

dimensional, multimodal sensory inputs such as visual and tactile observations. The control input $\boldsymbol{u}_t \in \mathbb{U} \subseteq \mathbb{R}^{n_u}$ specifies the commanded robot action. The system evolves according to the continuous-time dynamics

$$\dot{\boldsymbol{o}}(t) = f\Big(\boldsymbol{o}(t), \boldsymbol{u}(t)\Big), \tag{7.1}$$

where $f(\cdot)$ captures the underlying contact-rich interaction dynamics between the robot and its environment.

To enable reactive feedback control, we adopt a tube-based control formulation. Specifically, by locally linearizing the system over a short time horizon, we can obtain the following control law with a proportional feedback correction:

$$\boldsymbol{u}(t) = \bar{\boldsymbol{u}}(t) + K\Big(\bar{\boldsymbol{o}}(t) - \boldsymbol{o}(t)\Big), \tag{7.2}$$

where $\bar{\boldsymbol{u}}(t)$ and $\bar{\boldsymbol{o}}(t)$ denote the nominal action and observation trajectories, respectively, and the feedback gain $K \succ 0$ is held constant within the horizon. Differentiating (7.2) with respect to time yields

$$\dot{\boldsymbol{u}}(t) = \dot{\bar{\boldsymbol{u}}}(t) + K\Big(\dot{\bar{\boldsymbol{o}}}(t) - \dot{\boldsymbol{o}}(t)\Big). \tag{7.3}$$

We consider a neighborhood of the nominal trajectory in which the observation dynamics are differentiable and admit a first-order approximation. In particular, we apply a Taylor expansion around $(\bar{\boldsymbol{o}}(t), \bar{\boldsymbol{u}}(t))$ of system dynamics (7.1), leading to

$$\dot{\boldsymbol{o}}(t) \approx \dot{\bar{\boldsymbol{o}}}(t) + A(t)\Big(\boldsymbol{o}(t) - \bar{\boldsymbol{o}}(t)\Big) + B(t)\Big(\boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t)\Big), \tag{7.4}$$

where $A(t) \triangleq \frac{\partial f}{\partial \boldsymbol{o}}\Big|_{(\bar{\boldsymbol{o}}(t), \bar{\boldsymbol{u}}(t))}$ and $B(t) \triangleq \frac{\partial f}{\partial \boldsymbol{u}}\Big|_{(\bar{\boldsymbol{o}}(t), \bar{\boldsymbol{u}}(t))}$.

Within each time interval, we adopt the widely used quasi-dynamic assumption for contact-rich systems: over the short duration of each interval, the observation error $\boldsymbol{e}(t) = \bar{\boldsymbol{o}}(t) - \boldsymbol{o}(t)$ remains bounded and varies slowly, such that its contribution can be neglected in the local approximation. This yields

$$\dot{\bar{\boldsymbol{o}}}(t) - \dot{\boldsymbol{o}}(t) \approx -\Gamma(t)\Big(\boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t)\Big), \qquad \Gamma(t) \triangleq B(t), \tag{7.5}$$

where $\Gamma(t)$ summarizes the local input-to-observation sensitivity induced by contact interactions and the environment, and is understood to be valid locally.

Substituting (7.5) into (7.3) yields the action-space velocity field

$$\dot{\boldsymbol{u}}(t) \approx \dot{\bar{\boldsymbol{u}}}(t) - K\Gamma(t)\Big(\boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t)\Big). \tag{7.6}$$

Equation (7.6) decomposes the controller into a trajectory-following term $\dot{\bar{\boldsymbol{u}}}(t)$ and a stabilizing contraction term that drives system deviations toward the nominal trajectory, thereby locally maintaining the execution error within a bounded tube.

For simplicity, we approximate the product $K\Gamma(t)$ by an isotropic contraction term λI. This approximation is motivated by the observation that action–observation coupling in contact-rich manipulation is often dominated by a few low-dimensional interaction modes within a local contact regime. For instance, in planar pushing with persistent contact, the interaction is typically governed by contact modes such as sticking or sliding at the contact interface (Xue et al., 2023a, 2025b). Within each mode, the resulting action–observation coupling is effectively low-dimensional, making an isotropic contraction a reasonable local approximation. Here, $\lambda > 0$ denotes the effective local contraction rate in action space. Substituting this approximation into (7.3) yields the simplified action-space dynamics

$$\dot{\boldsymbol{u}}(t) \approx \dot{\bar{\boldsymbol{u}}}(t) - \lambda\Big(\boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t)\Big), \qquad \lambda \triangleq \gamma k > 0, \tag{7.7}$$

which describes a contracting flow around the nominal action trajectory.

This formulation is structurally consistent with (Jiang et al., 2025), which frames policy learning as the imitation of action trajectories. While effective in some settings, (Jiang et al., 2025) abstracts away robot–environment coupling and is generally insufficient, as control policies are fundamentally designed to regulate system states or observations rather than actions alone. We instead adopt a full-system perspective and identify physically grounded assumptions under which action-trajectory imitation provides a valid local approximation of closed-loop control. We show that this approximation holds only in a neighborhood of the nominal trajectory under a quasi-dynamic execution regime with bounded tracking errors. This analysis provides the theoretical motivation for the proposed approach, in which diffusion provides chunk-level corrections that account for full system dynamics, while the streaming flow enables local feedback control during execution.

7.4 Methodology

From (7.7), we observe that after local linearization, the resulting feedback flow depends only on the nominal action trajectory. This insight suggests that the conventional

decision-making process can be conceptually decomposed into two stages: (i) periodically generating actions that account for the full nonlinear dynamics to correct accumulated drift, and (ii) applying a local feedback controller based on linearized dynamics.

Motivated by this decomposition, we propose an approach based on *Tube Diffusion Policy* (TDP) as reactive policy learning under real-time observations. At the beginning of each action chunk, a diffusion model performs multi-step denoising to generate an initial action that accounts for the full nonlinear contact dynamics. Subsequent actions are then generated through a streaming feedback flow learned from demonstrations, which locally regulates deviations based on fresh observations. Although the diffusion stage does not explicitly generate a full nominal action trajectory, it implicitly defines one by providing the initial condition for the subsequent action evolution. The resulting action sequence can therefore be interpreted as a nominal trajectory around which closed-loop regulation is performed. This structure aligns with the action chunk paradigm in chunk-based control, but crucially augments it with step-wise closed-loop feedback during execution. We refer to this formulation as an *action tube*, where diffusion provides chunk-level action initialization and the learned feedback flow enforces real-time corrections within the tube. This section presents the overall architecture for learning such reactive control policies from demonstrations.

Given a dataset of demonstration episodes, similar to chunk-based approaches (Chi et al., 2024; Zhao et al., 2023), we decompose each episode into multiple observation and action sequences of length H_p, yielding a dataset $\mathcal{D} = \{(\zeta_i^o, \zeta_i^u)\}_{i=1}^N$. Our goal is to learn the conditional velocity field $v_\theta(\boldsymbol{u}, t_1, t_2 \mid h_\tau)$ by minimizing (7.8), enabling both diffusion-based action generation and flow-based reactive control. Here, $h_\tau = \{\boldsymbol{o}_t\}_{t=\tau}^{\tau+H_o}$ denotes a finite observation history of length H_o. At execution time, the learned policies are deployed in a receding-horizon fashion, with an action horizon H_a that may differ from the prediction horizon H_p used during training. The complete pseudocode for training and inference is provided in Alg. 5 and Alg. 6.

7.4.1 Overall Architecture

Dual-Time Formulation: TDP introduces two time variables with distinct semantics:

- $t_1 \in [0, T_{\text{diff}}]$: **diffusion time**, governing the diffusion denoising process while the physical system remains stationary;
- $t_2 \in [0, 1]$: **trajectory time**, governing the streaming control process that evolves in real time with the physical system.

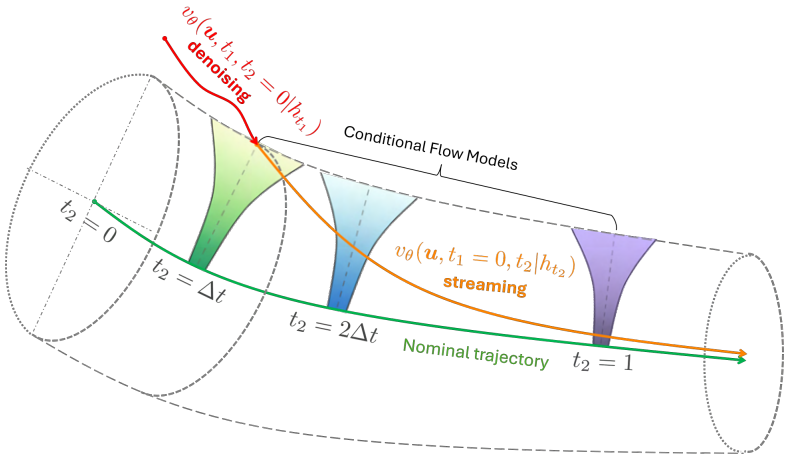


Figure 7.2: Overview of Tube Diffusion Policy (TDP). TDP adopts a dual-time formulation with a multi-step denoising stage along t_1 and an observation-conditioned streaming stage along t_2. The method operates in two phases: a *denoising phase* and a *streaming phase*. At the beginning of each action chunk horizon, an initial action is generated through multi-step denoising, accounting for the full nonlinear system dynamics; the resulting trajectory is shown in red. Subsequent actions are produced by streaming action flow conditioned on the observation at each timestep, yielding the orange trajectory. The green curve denotes the nominal trajectory. This process can be interpreted as executing actions within an *action tube* (dashed boundary), where local deviations are continuously regulated by switching among a sequence of observation-conditioned flow models (illustrated as funnels). Here, h_t denotes the observation history, and Δt controls the streaming temporal granularity. By combining diffusion-based chunk-level initialization with tube-based reactive feedback, TDP enables robust, high-frequency closed-loop control for contact-rich manipulation.

The policy is conditioned on both variables, allowing a single network to operate in two modes: diffusion-based action generation and flow-based streaming.

Training Objective: The overall training objective combines losses from both modes:

$$\mathcal{L} = \mathcal{L}_{\text{diff}} + \mathcal{L}_{\text{stream}}, \tag{7.8}$$

where $\mathcal{L}_{\text{diff}}$ is the diffusion noise-prediction loss and $\mathcal{L}_{\text{stream}}$ supervises the streaming velocity field. We use equal weighting for the two objectives, as they correspond to complementary roles—global action generation and local feedback correction—and operate on comparable scales in practice.

7.4.2 Diffusion Phase: Diffusion-based Action Generation

The diffusion phase follows the standard Denoising Diffusion Probabilistic Models (DDPM) formulation with ϵ-prediction (Ho et al., 2020). Given a clean action $\boldsymbol{u}_0 = \zeta^u[H_o - 1 : H_o]$ that is aligned with the observation history $h_{t_1} = \zeta^o[: H_o]$, the forward noising process is defined as

$$\boldsymbol{u}_{t_1} = \sqrt{\bar{\alpha}_{t_1}}\boldsymbol{u}_0 + \sqrt{1-\bar{\alpha}_{t_1}}\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{7.9}$$

where $\bar{\alpha}_{t_1}$ is specified by the noise schedule in (Ho et al., 2020). The network $v_\theta(\boldsymbol{u}, t_1, t_2 \mid h_{t_1})$ is trained to predict the injected noise conditioned on the aligned observation history h_{t_1} by minimizing

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\boldsymbol{u}_0,\, h_{t_1},\, \boldsymbol{\epsilon},\, t_1}\left[\left\|\boldsymbol{\epsilon} - v_\theta(\boldsymbol{u}_{t_1}, t_1, t_2 = 0 \mid h_{t_1})\right\|^2\right], \tag{7.10}$$

where the expectation is taken over aligned observation–action pairs $(h_{t_1}, \boldsymbol{u}_0)$ sampled from the demonstration dataset $\mathcal{D}$, and t_1 is sampled uniformly from $[0, T_{\text{diff}}]$, with $[0, T_{\text{diff}}]$ specifying the length of the diffusion process.

7.4.3 Streaming Phase: Flow-based Action Streaming

By locally linearizing the system dynamics, the tube-based feedback controller can be projected from the observation–action space into the action space. Under this formulation, the streaming phase learns a reactive controller by directly modeling action evolution as a velocity field over the trajectory time, enabling high-frequency closed-loop execution. This design is inspired by (Jiang et al., 2025).

Following (7.7), we define the target velocity field along the action trajectory as

$$v_\zeta(\boldsymbol{u}, t_2) = \frac{d\zeta^u}{dt_2} - \lambda\Big(\boldsymbol{u} - \zeta^u(t_2)\Big), \tag{7.11}$$

where $\zeta^u(t_2)$ denotes the continuous nominal action trajectory parameterized by the trajectory time $t_2 \in [0, 1]$. The first term provides feedforward progression along it, while the second induces feedback toward the nominal trajectory.

To enable reactive control within the action tube, we extract step-wise observation history aligned with the trajectory time via

$$h_{t_2} = \zeta^o[l : l + H_o], \tag{7.12}$$

Algorithm 5 Training

Require: Training set $\mathcal{D} = \{(\zeta_i^o, \zeta_i^u)\}_{i=1}^N$, H_o, H_p, T_{diff}

ζ^o and ζ^u have time horizon T_{pred} rescaled to $[0,1]$
while not converged **do**
  $(\zeta^o, \zeta^u) \sim \mathcal{D}$
  // ===== Diffusion phase (t_1) =====
  $t_1 \sim \text{Uniform}\{0, \dots, T_{\text{diff}}\}$
  $h_{t_1} \leftarrow \zeta^o[: H_o]$
  $\boldsymbol{u}_0 \leftarrow \zeta^u[H_o - 1 : H_o]$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\boldsymbol{u}_{t_1} = \sqrt{\bar{\alpha}_{t_1}}\boldsymbol{u}_0 + \sqrt{1-\bar{\alpha}_{t_1}}\boldsymbol{\epsilon}$
  $\mathcal{L}_{\text{diff}} \leftarrow \|\boldsymbol{\epsilon} - v_\theta(\boldsymbol{u}_{t_1}, t_1, t_2 = 0 \mid h_{t_1})\|^2$
  // ===== Streaming phase (t_2) =====
  $t_2 \sim \text{Uniform}(0,1)$
  $\boldsymbol{u} \sim p_{\zeta^u}(\boldsymbol{u} \mid t_2)$
  $l \leftarrow \lfloor t_2 H_p \rfloor$
  $h_{t_2} \leftarrow \zeta^o[l : l + H_o]$
  $v_\zeta(\boldsymbol{u}, t_2) \leftarrow$ *Eq.* (7.11)
  $\mathcal{L}_{\text{stream}} \leftarrow \|v_\zeta(\boldsymbol{u}, t_2) - v_\theta(\boldsymbol{u}, t_1 = 0, t_2 | h_{t_2})\|^2$
  $\theta \leftarrow \theta - \gamma\nabla_\theta(\mathcal{L}_{\text{diff}} + \mathcal{L}_{\text{stream}})$
end while
return v_θ

Algorithm 6 Inference

Require: $v_\theta(\boldsymbol{u}, t_1, t_2 \,|\, h)$, H_o, H_p, H_a, T_{ddim}, Δt

$h \leftarrow \{\boldsymbol{o}_{\text{curr}}\}_{i=1}^{H_o}$
while True **do**
  // === Phase 1: Denoising (t_1) ===
  $\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $t_1 = T_{\text{ddim}}, \dots, 1$ **do**
    $\hat{\boldsymbol{\epsilon}} \leftarrow v_\theta(\boldsymbol{u}, t_1, t_2 = 0 \,|\, h)$
    $\boldsymbol{u} \leftarrow \text{DDIMStep}(\boldsymbol{u}, \hat{\boldsymbol{\epsilon}}, t_1)$
  end for
  // === Phase 2: Streaming (t_2) ===
  $t_2 \leftarrow \Delta t$
  while $t_2 \le H_a / H_p$ **do**
    $o \leftarrow \text{Execute}(\boldsymbol{u})$
    $h \leftarrow h \cup \{o\}$
    $\boldsymbol{u} \leftarrow \boldsymbol{u} + v_\theta(\boldsymbol{u}, t_1 = 0, t_2 \,|\, h) \cdot \Delta t$
    $t_2 \leftarrow t_2 + \Delta t$
  end while
end while

where $l = \lfloor t_2 H_p \rfloor$ is the index corresponding to t_2 on the discretized observation trajectory, with $\lfloor \cdot \rfloor$ denoting the floor (round-down) operation. H_p denotes the prediction horizon, and H_o is the length of the observation history window.

The network $v_\theta(\boldsymbol{u}, t_1, t_2 \mid h_{t_2})$ is trained to predict the streaming velocity field conditioned on the aligned observation history h_{t_2} by minimizing

$$\mathcal{L}_{\text{stream}} = \mathbb{E}_{\boldsymbol{u}, h, t_2}\left[\left\| v_\theta(\boldsymbol{u}, t_1 = 0, t_2 \mid h_{t_2}) - v_\zeta(\boldsymbol{u}, t_2) \right\|^2\right], \tag{7.13}$$

where $t_2 \sim \mathcal{U}([0,1])$, and $\boldsymbol{u}$ is sampled from the stabilizing conditional distribution

$$\boldsymbol{u} \sim p_\zeta(\boldsymbol{u} \mid t_2) = \mathcal{N}\Big(\zeta^u(t_2),\ \sigma_0^2 e^{-2kt_2}\mathbf{I}\Big), \tag{7.14}$$

which is a Gaussian centered at the action trajectory $\zeta^u(t_2)$ with a small variance σ_0^2, as introduced in (Jiang et al., 2025).

By extracting step-wise observation histories, we learn a family of observation-conditioned flow models of the form $v_\theta(\boldsymbol{u}, t_1 = 0, t_2 \mid h_{t_2})$, each conditioned on a distinct observation history h_{t_2}. During execution, the action sequence within the chunk horizon is not governed by a single conditional flow model; instead, actions are generated by sequentially switching among multiple observation-conditioned flows as new observation becomes available and provide updated conditioning.

At each step, the action is updated by integrating the currently active streaming velocity field,

$$\boldsymbol{u} \leftarrow \boldsymbol{u} + v_\theta(\boldsymbol{u}, t_1 = 0, t_2 \mid h_{t_2})\, \Delta t_2, \tag{7.15}$$

where Δt_2 denotes the discretization step size of the trajectory time t_2. This step-wise integration incorporates fresh observations at every step, enabling closed-loop and reactive control.

7.4.4 Network Architecture and Encoding

We employ a conditional 1D U-Net as the shared backbone for both diffusion-based action generation and streaming-based reactive flow. The network predicts action evolution conditioned on observations, outputting either diffusion noise or action velocity depending on the time conditioning, which enables parameter sharing across the diffusion and streaming phases.

To support the dual-time formulation, the network uses separate sinusoidal embeddings for the diffusion time t_1 and the streaming time t_2. The two embeddings are scaled differently to reflect their distinct semantics: diffusion time follows the standard DDPM time scale, while trajectory time is amplified to provide finer resolution over the normalized execution horizon $[0, 1]$. This separation prevents interference between diffusion and streaming behaviors while allowing a unified architecture.

Observation features and time embeddings are injected into the network via Feature-wise Linear Modulation (FiLM) (Perez et al., 2018). This conditioning mechanism enables the policy to adapt its action predictions to both the current observation context and the operating mode (diffusion or streaming) in a smooth and modular manner.

Visual observations are encoded using ResNet-18 backbones (He et al., 2016). In simulation, tactile inputs are processed using a lightweight multilayer perceptron, as

the tactile sensor provides array-based measurements. In real-world experiments, tactile observations from Digit360 sensors are image-based and are therefore encoded using multiple ResNet-18 backbones. The resulting features are concatenated with proprioceptive signals to form a global observation representation, which conditions the action policy through FiLM modulation.

Actions consist of end-effector position, orientation, and hand joint commands. For orientation, we adopt the continuous 6D rotation representation (Zhou et al., 2019), which avoids discontinuities associated with quaternion parameterizations and is better suited for regression-based learning.

7.4.5 Stability Analysis

TDP decomposes control into two complementary mechanisms: diffusion correction, which periodically contracts the accumulated drift, and streaming imitation, which provides high-frequency reactive control but may introduce bounded tracking error. Together they form a hybrid control system that admits a practical stability guarantee.

Consider the discrete-time system

$$\boldsymbol{o}_{t+1} = g(\boldsymbol{o}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t, \tag{7.16}$$

where $g(\cdot)$ denotes the discrete-time dynamics corresponding to (7.1), and $\boldsymbol{w}_t$ is a bounded disturbance.

Let $(\boldsymbol{o}_t^*, \boldsymbol{u}_t^*)$ be a demonstrated reference trajectory satisfying

$$\boldsymbol{o}_{t+1}^* = g(\boldsymbol{o}_t^*, \boldsymbol{u}_t^*), \tag{7.17}$$

and define the tracking error $\boldsymbol{e}_t = \boldsymbol{o}_t - \boldsymbol{o}_t^*$ and the value function $V(\boldsymbol{e}_t) = \|\boldsymbol{e}_t\|$.

To facilitate the stability analysis, we introduce the following conditions.

Assumption 1 (Lipschitz dynamics). The discrete-time dynamics satisfy

$$\|g(\boldsymbol{o}_1, \boldsymbol{u}_1) - g(\boldsymbol{o}_2, \boldsymbol{u}_2)\| \leq L_x\|\boldsymbol{o}_1 - \boldsymbol{o}_2\| + L_u\|\boldsymbol{u}_1 - \boldsymbol{u}_2\|. \tag{7.18}$$

The Lipschitz condition bounds the sensitivity of the system dynamics to state and action deviations, enabling tractable propagation of tracking errors.

Assumption 2 (Streaming imitation accuracy). The streaming flow policy is trained to

imitate the reference action trajectory. As a result, the action error remains bounded:

$$\|\boldsymbol{u}_t - \boldsymbol{u}_t^*\| \leq \varepsilon_a. \tag{7.19}$$

This condition reflects the training objective of the streaming phase in TDP and is expected to hold when the policy accurately reproduces the demonstrated actions.

Assumption 3 (Bounded disturbance). The external disturbance is bounded:

$$\|\boldsymbol{w}_t\| \leq \bar{w}. \tag{7.20}$$

Assumption 4 (Diffusion correction property). At correction steps t_k, the diffusion module produces a corrective action that reduces the accumulated tracking error:

$$V(\boldsymbol{e}_{t_k+1}) \leq \lambda V(\boldsymbol{e}_{t_k}) + \varepsilon_d, \qquad 0 < \lambda < 1. \tag{7.21}$$

This condition corresponds to the training objective of the diffusion phase in TDP, which aims to contract the tracking error at correction steps by accounting for the full system dynamics.

Lemma 1 (Streaming-phase bound). During streaming execution, the value function satisfies

$$V(\boldsymbol{e}_{t+1}) \leq L_x V(\boldsymbol{e}_t) + L_u \varepsilon_a + \bar{w}. \tag{7.22}$$

Proof. From the system dynamics,

$$\boldsymbol{e}_{t+1} = g(\boldsymbol{o}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t - g(\boldsymbol{o}_t^*, \boldsymbol{u}_t^*).$$

Applying Lipschitz continuity and Assumption 2 yields

$$\|\boldsymbol{e}_{t+1}\| \leq L_x \|\boldsymbol{e}_t\| + L_u \varepsilon_a + \bar{w}.$$

Substituting the definition of V proves the result. □

The lemma shows that the streaming policy may introduce bounded tracking drift due to imitation error and disturbances, but the error growth remains bounded at each control step.

Theorem 1 (Practical stability of TDP). Suppose diffusion correction occurs every H_a steps and $L_x < 1$. Then the closed-loop system under TDP satisfies the following

properties:

(1) Uniform ultimate boundedness. The tracking error sequence is uniformly ultimately bounded.

(2) Bounded error within each diffusion cycle. During each correction cycle $t \in \{t_k, \ldots, t_k + H_a - 1\}$, the tracking error remains bounded as a function of the post-correction error.

Proof sketch. Lemma 1 shows that during streaming execution the value function grows at most linearly due to imitation error and disturbances. At correction instants, diffusion reduces the accumulated tracking error according to Assumption 4. Combining bounded growth with periodic contraction yields a stable affine recursion on the post-correction error, which implies uniform ultimate boundedness of the tracking error. Moreover, the streaming bound ensures that the error remains bounded throughout each diffusion cycle. □

In the ideal case where the imitation error, disturbance, and diffusion residual vanish ($\varepsilon_a = 0$, $\bar{w} = 0$, and $\varepsilon_d = 0$), the equilibrium $\boldsymbol{e} = 0$ is asymptotically stable and the tracking error satisfies $\lim_{t\to\infty} \boldsymbol{e}_t = 0$. The full derivation of the stability bound is provided in Appendix.

7.5 Experiments

We first use a toy 1-D point-mass system to illustrate the effectiveness of *action tube* under external disturbances. We then evaluate TDP on the widely used Push-T task and three additional visual–tactile dexterous manipulation tasks in simulation, followed by two challenging real-world experiments that require reactive control under contact uncertainty and external disturbances.

7.5.1 1-D Point-mass System

We consider a 1-D point-mass system with single-integrator dynamics $x_{t+1} = x_t + u_t \Delta t$. The goal is to use imitation learning to reproduce a demonstrated trajectory, as illustrated in Fig. 7.3. The horizontal axis denotes time, while the vertical axis shows the evolution of the state x_t. To highlight the difference between *action chunk* and *action tube*, we assume that the entire trajectory is executed within a single action-chunk horizon. The white curves depict the movement flow of the point mass under different initializations in both cases. Without loss of generality, we consider the example in

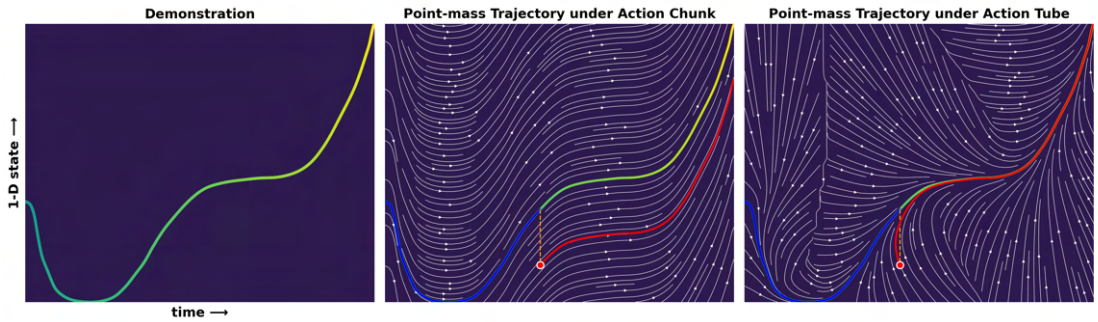


Figure 7.3: 1-D Point-Mass Example: Action-Tube Robustness to External Disturbances. We consider a 1-D point-mass system with a single demonstration to contrast action chunk and action tube. The x-axis denotes execution time and the y-axis denotes system state. **Left:** Nominal trajectory without disturbance, with color indicating progression along the trajectory from dark green (start) to light green (end). **Middle:** Action chunk execution. A disturbance is applied at the midpoint (orange dashed line), after which the system continues executing the precomputed action sequence (i.e., the 1-D velocity) without feedback, resulting in deviation from the target. **Right:** Action tube execution under the same disturbance. A local feedback controller drives the system back toward the nominal trajectory and ultimately reaches the target. Blue curves denote trajectories before disturbance, and red curves denote trajectories after disturbance.

which the system is initialized at the same state as the demonstration, and an external disturbance is injected at the midpoint of the trajectory, as indicated by the orange dashed line. This disturbance can be interpreted as either human perturbation or physical uncertainty.

Chunk-based approaches generate a fixed sequence of actions at the beginning of execution. As a result, after the disturbance occurs, the controller continues to apply the precomputed action sequence, ultimately driving the system to an incorrect final state. In contrast, the action-tube formulation employs a local feedback controller around the nominal trajectory, which actively steers the system back toward the desired path after the disturbance.

This toy example illustrates the importance of *action tube* for achieving reactive and robust control in the presence of external disturbances or contact uncertainty.

7.5.2 Push-T Task

We further benchmark TDP on Push-T task against state-of-the-art imitation learning methods, including Diffusion Policy (DP) (Chi et al., 2024) with 100 DDPM steps, DP with 10 Denoising Diffusion Implicit Models (DDIM) (Song et al., 2021a) steps, Flow Matching Policy (Zhang and Gienger, 2024), and Streaming Flow Policy (SFP) (Jiang et al., 2025). We also have one ablation study that removes the diffusion phase of TDP, namely TDP (streaming only). Following (Jiang et al., 2025), we use the current robot configuration as a warm-start initialization for SFP, and also apply the same strategy to the TDP (streaming only) ablation, as it does not include an initial denoising stage. Push-T is a widely-used task to benchmark imitation learning approaches, and it is particularly suitable to be used in this work since it has nonlinear dynamics and physical uncertainty. For example, the friction coefficients are hard to be identified and varied with different objects. We report the average score of the 5 best checkpoints and also the maximum score across all checkpoints. Moreover, there is another metric called *steps to success*, which describes the task completion time for the successful trials. This further evaluates the effectiveness of each action across all cases that ultimately lead to success. We report the average steps of the 5 best checkpoints and the minimum steps of the best checkpoint. Table 7.2 demonstrates the results for both state-based and image-based Push-T tasks, where state-based Push-T task uses low-dim state as the input, including 2-D position of the cylindrical pusher and 3-D pose of the T-shape block on planar space. Image-based push-T task use RGB image as the input. The actions are 2-D setpoints tracked by a PD controller. This task contains 200 demonstrations and 50 evaluation episodes.

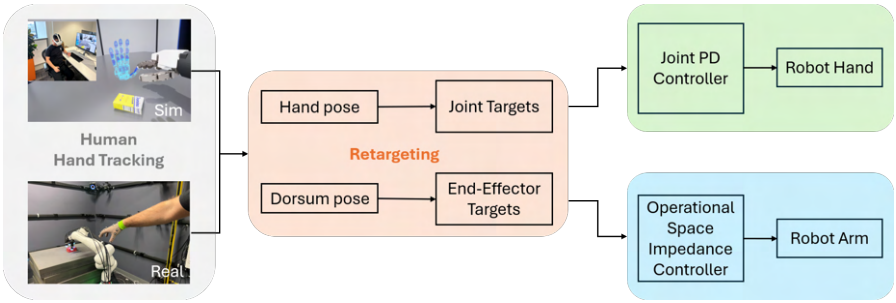


Figure 7.4: Teleoperation system. Human demonstrations are collected through teleoperation in both simulation and the real world. In simulation, a Meta Quest 3 headset is used for VR-based hand tracking, while in the physical setup an OptiTrack motion-capture system provides mocap-based hand tracking. The captured hand pose and dorsum pose are then processed by the same retargeting module and controllers to actuate the dexterous hand and robot arm.

As reported in Table 7.2, TDP achieves the highest performance on both tasks while requiring the fewest steps, demonstrating the effectiveness of real-time reactions at each step given the current observation for task accomplishment. Moreover, the ablation results show that removing the diffusion phase degrades performance, highlighting the limitations of relying solely on local linearization without a diffusion-based correction that accounts for the full system dynamics.

We further conduct an ablation study on the number of DDIM inference steps used in the denoising phase for state-based PushT task, as shown in Fig. 7.5. We conduct five trials, each consisting of 20 episodes, and report both the average and maximum scores. The results indicate that competitive performance can already be achieved with as few as two DDIM steps. This observation suggests that our method does not require highly accurate action generation during the denoising stage. Instead, the diffusion model only needs to produce an initial action that initializes execution while accounting for the full system dynamics, after which the streaming policy regulates deviations through closed-loop feedback. This property highlights a key advantage of the proposed tube-based formulation over conventional chunk-based methods that rely on precise action chunks. Moreover, reducing the number of DDIM steps substantially lowers inference latency, demonstrating the potential of TDP for real-time, high-frequency control.

7.5.3 Visual–Tactile Manipulation in a Virtual Environment

We evaluate our method on three additional visual–tactile dexterous manipulation tasks in a simulation environment built with Unreal Engine. The simulator integrates a

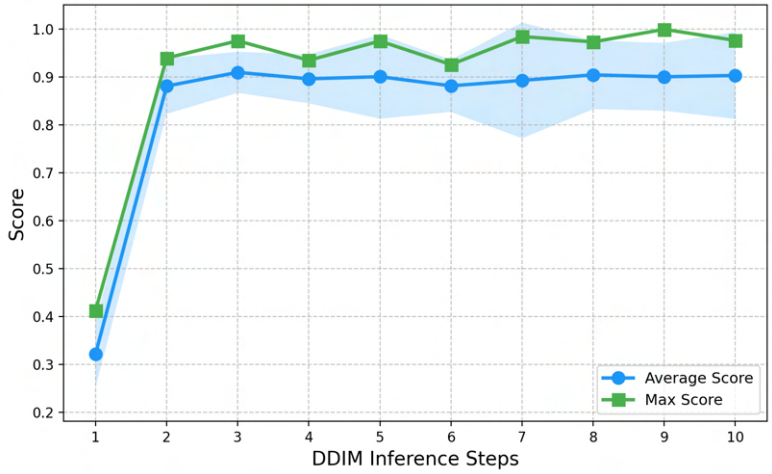
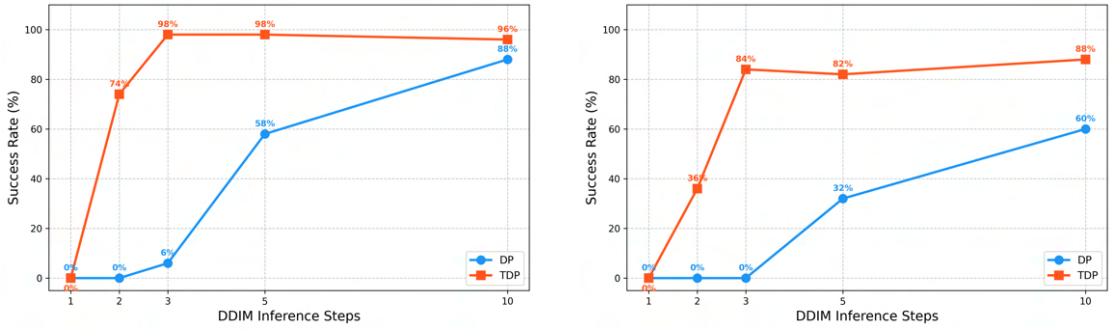


Figure 7.5: Ablation study on DDIM inference steps for the virtual PushT task. We vary the number of DDIM inference steps used in the denoising phase of TDP. The results show that TDP maintains competitive performance with as few as two denoising steps, highlighting its suitability for high-frequency control.



(a) Grasping (b) Stable grasping

Figure 7.6: Ablation study on DDIM inference steps. We compare TDP and DP under varying numbers of DDIM inference steps on both grasping and stable grasping tasks. TDP consistently performs well with fewer inference steps than DP, demonstrating its ability to reduce control latency in practice. Furthermore, with the same number of inference steps, TDP achieves higher success rates owing to its step-wise reactive control mechanism.

	Push-T with state input		Push-T with image input	
	Avg/Max scores ↑	Avg/Min steps to success ↓	Avg/Max scores ↑	Avg/Min steps to success ↓
DP (DDPM)	93.2%/93.8%	125.9/128.5	76.0%/77.5%	114.7/107.9
DP (DDIM)	91.8%/93.9%	124.6/117.1	68.5%/69.0%	107.3/104.8
Flow Matching	91.8%/93.3%	122.8/115.9	65.3%/68.3%	108.9/102.5
SFP	88.6%/89.7%	124.4/116.1	73.3%/74.8%	112/108.6
TDP (streaming only)	84.3%/85.0%	109.6/108.1	77.1%/77.7%	117.5/96.3
TDP	**96.1%/96.9%**	**106.2/105.1**	**82.0%/85.6%**	**103.5/91.3**

Table 7.2: Push-T results with state and image inputs. Our method (green) is compared against baselines (orange) and ablation variants (blue).

	Stable Grasping			On-table Reorientation	
	Success rate (Grasp) ↑	Success rate (Stable grasp) ↑	Steps to success ↓	Success rate ↑	Steps to success ↓
DP (DDIM)	88%	60%	28.0 ± 3.70	82%	41.7 ± 8.51
SFP	78%	72%	29.6 ± 4.75	80%	43.1 ± 12.9
TDP (streaming only)	72%	44%	$\mathbf{25.5 \pm 5.1}$	76%	60.6 ± 25.3
TDP	**96%**	**88%**	27.8 ± 3.06	**90%**	$\mathbf{40.0 \pm 15.3}$

Table 7.3: Comparison on stable grasping and on-table reorientation. Our method (green) is compared against baselines (orange) and ablation variants (blue).

real-time finite element solver through a native plugin, enabling physically realistic interactions between rigid and compliant objects. By leveraging reduced-order models, the solver efficiently simulates contact dynamics at interactive rates while providing detailed per-contact state information for dexterous manipulation (Tao et al., 2025; Zhu et al., 2022; Chang et al., 2023; Zong et al., 2023). The simulated system consists of a UR5 robotic arm and a 20-DoF Tesollo DG5F dexterous hand equipped with a dense tactile array containing 768 sensing elements. Each element measures three-axis forces, resulting in a tactile observation of 2304 dimensions. Visual observations are obtained from two RGB cameras: one mounted above the workspace and another attached to the robot wrist. To collect demonstrations, we use a VR-based teleoperation interface with a Meta Quest 3 headset, as illustrated in Fig. 7.4. The tracked human hand pose determines the pose of the robot end-effector, while the dexterous hand is controlled through fingertip retargeting. Specifically, the human fingertip positions are scaled to match the kinematic workspace of the DG5F hand. The corresponding finger joint targets are then computed using RelaxedIK (Rakita et al., 2018; Wang et al., 2023b). All simulation experiments are performed on a Lenovo P8 desktop equipped with an NVIDIA RTX 4080 GPU, responsible for robot control, visual–tactile sensing, and policy

	Score ↑	Latency	
		Denoising (↓)	Streaming (↓)
DP (DDIM-10)	**98.4%**	72.9 ms	
DP (DDIM-2)	5.3%	17.1 ms	
TDP	98.0%	**13.3 ms**	**6.50 ms**

Table 7.4: Comparison of DP and TDP on the dish-cleaning task. TDP achieves comparable success rate while significantly reducing inference latency.

inference.

The first task, *stable grasping*, requires the robot to grasp a mustard bottle and maintain a secure hold. Performance is evaluated using grasp success rate, stable grasp success rate, and steps to success, where a stable grasp requires holding the object for at least three seconds. The second task, *on-table reorientation*, involves rotating a sugar box by 90 degrees via in-hand manipulation. The third task, *dish cleaning*, requires cleaning a dirty dish using a sponge. Task performance is measured by the percentage of the dirty region cleaned. We collect 150 demonstrations for each task and evaluate each method over 50 episodes.

We first compare our approach against state-of-the-art baselines and conduct ablation studies on the *stable grasping* and *on-table reorientation* tasks, with results summarized in Table 7.3. Across both tasks, our method consistently outperforms all baselines in terms of success rate while requiring comparable or fewer steps to reach successful outcomes. In the *stable grasping* task, our approach achieves a grasp success rate of 96% and a stable grasp success rate of 88%, substantially exceeding both DP and SFP, while maintaining efficient execution. Similar improvements are observed in the *on-table reorientation* task, where our method attains the highest success rate (90%) and reduces the number of steps compared to prior approaches. These results demonstrate the effectiveness of the proposed tube-based formulation in leveraging real-time visual–tactile feedback for robust and efficient manipulation. We further conduct an ablation study by removing the diffusion phase and relying solely on the streaming controller. In this setting, success rates drop significantly for both tasks. For *on-table reorientation*, small errors tend to accumulate and often require multiple retries, leading to a higher number of steps to success despite eventual task completion. Interestingly, the streaming-only variant achieves the lowest number of steps in the *stable grasping* task, which makes sense since this metric only accounts for completed episodes. In contrast to reorientation, failed grasp attempts typically cause the object to fall, making retries infeasible within the same episode. As a result, the

	**On-table Reorientation**	**Jar Opening**	**Latency**	
	Success rate (↑)	Success rate (↑)	Denoising (↓)	Streaming (↓)
DP	60%	84%	0.037s	
TDP	**96%**	**96%**	**0.008s**	**0.003s**

Table 7.5: Comparison of DP and TDP on two physical experiments. TDP achieves higher success rate with significantly lower inference latency.

steps-to-success values for grasping remain on a similar scale (and are even slightly better) than those of the full TDP method, as this metric mainly reflects the simplest successful scenarios. This further highlights the efficiency of step-wise reactive control for contact-rich manipulation, as this variant preserves a fully reactive formulation with fresh observations at every time step.

Furthermore, we perform an ablation study on the number of DDIM inference steps to evaluate execution efficiency on the *stable grasping* task (Fig. 7.6). With a single DDIM step, both TDP and DP fail to complete the task. As the number of inference steps increases, TDP exhibits a rapid performance gain with only two to three steps, whereas DP remains at a 0% success rate and requires substantially more denoising steps (e.g., 5–10) to achieve similar performance. This indicates that the step-wise feedback regulation of the action tube can effectively compensate for coarse diffusion outputs, allowing TDP to attain high success rates with significantly fewer denoising steps and, consequently, reduced diffusion-stage inference latency for high-frequency, real-time control.

Based on this observation, we further evaluate both methods on the *dish cleaning* task, with results shown in Table 7.4. While DP with 10 DDIM steps achieves the highest cleaning score, it leads to significant control latency due to the long inference time. In contrast, TDP requires only 2 DDIM steps, making it possible to run at approximately 75 Hz during the denoising phase and up to 150 Hz during the streaming phase. When DP is constrained to a comparable inference budget (i.e., 2 DDIM steps), its performance drops significantly. This result highlights the capability of TDP for real-time, high-frequency control.

7.5.4 Real-world Experiments

We further validate TDP on two real-world visual–tactile dexterous manipulation tasks: jar opening and on-table reorientation, as illustrated in Fig. 7.1. These tasks involve

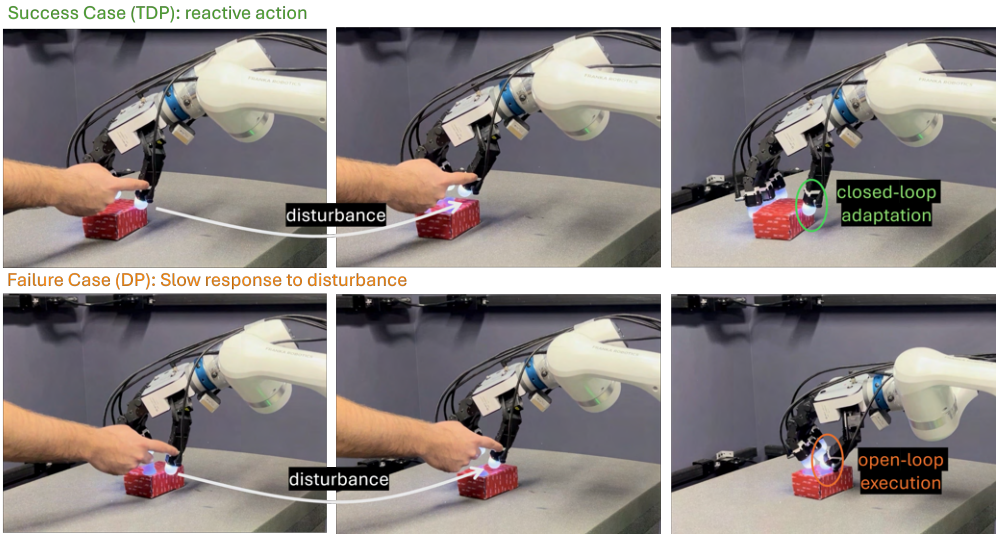


Figure 7.7: On-table manipulation task under external disturbance. The disturbance is introduced by bending the thumb fingers of the Allegro hand. TDP reacts quickly to the perturbation, while DP exhibits a delayed response and the fingers become stuck, resulting in task failure.

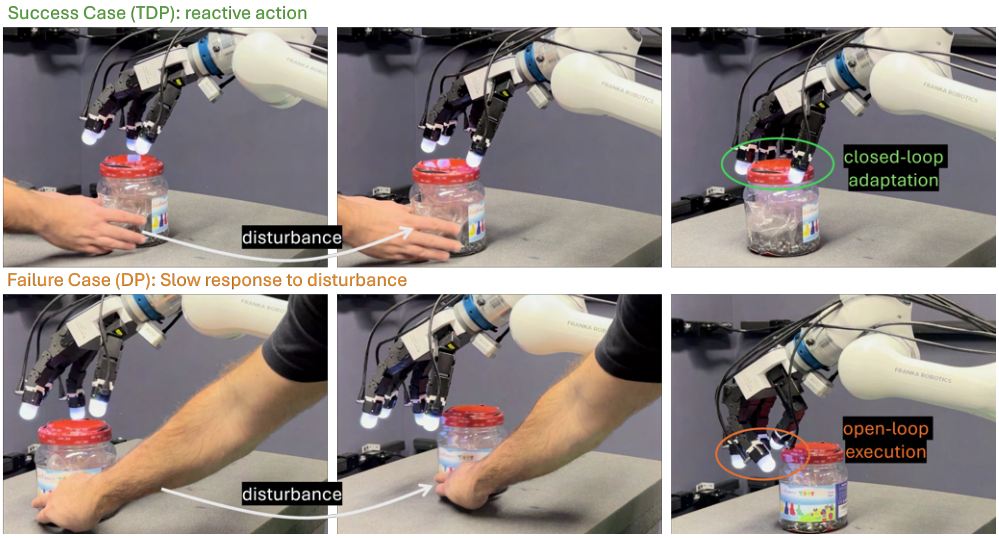
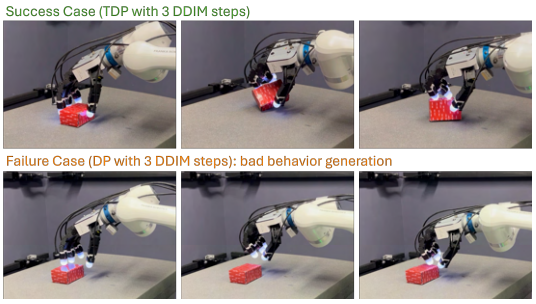
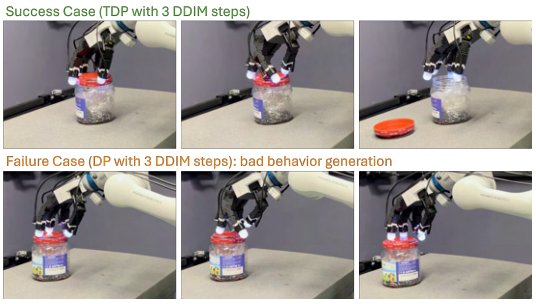


Figure 7.8: Jar-opening task under external disturbance. We introduce disturbance by shifting the jar's position. TDP quickly adapts to the new position and continues opening the jar, whereas DP blindly executes the pre-generated action sequence, resulting in slow response to the disturbance.



(a) On-table reorientation with 3 DDIM steps.



(b) Jar-opening with 3 DDIM steps.

Figure 7.9: Real-world manipulation with few DDIM inference steps. TDP maintains strong performance with only three DDIM inference steps, enabling high-frequency control in the real world. In contrast, DP produces inaccurate behaviors under the same inference budget, indicating its reliance on more DDIM steps for stable execution and consequently higher control latency.

contact-rich interactions, pose uncertainty, and frequent variations in hand–object contact, posing significant challenges for policies relying on open-loop execution.

All physical experiments are conducted using a Franka Research 3 robotic arm equipped with an Allegro V5 dexterous hand. Visual observations are provided by an Intel RealSense D405 camera for the wrist-mounted view and a D435F camera for the global view. In addition, Digit360 tactile sensors are attached to the fingertips to capture tactile feedback during manipulation. The system runs on two workstations. A Lenovo P620 desktop (NVIDIA RTX 3080 GPU) is used for low-level robot control and real-time visual–tactile sensing processing, while a Lenovo P8 desktop (NVIDIA RTX 4080 GPU) is dedicated to policy inference. To collect demonstrations, we employ a teleoperation system based on an OptiTrack motion-capture setup together with an instrumented glove for hand tracking, as illustrated in Fig. 7.4. The pose of the robot end-effector is determined by tracking the pose of the human hand. For hand teleoperation, we adopt a fingertip retargeting strategy: fingertip positions estimated from the glove are mapped to the workspace of the Allegro hand while ignoring the human pinky finger. We then solve the inverse kinematics using RelaxedIK (Rakita et al., 2018; Wang et al., 2023b) to obtain the desired finger joint targets.

We first conduct 25 evaluation episodes for each task using both TDP and DP. The quantitative results are summarized in Table 7.5. Benefiting from its closed-loop reactive control, TDP achieves significantly higher success rates on both tasks. Moreover, TDP allows the use of fewer DDIM inference steps thanks to the tube-based correction mechanism applied at each timestep. In our experiments, we use only 3 DDIM steps during denoising, which results in substantially lower inference latency and enables

control frequencies above 100 Hz, approximately four times faster than DP ($\sim$ 25 Hz).

To further examine this property, we also evaluate DP using the same number of DDIM inference steps for high-frequency control. As shown in Fig. 7.9, DP with only 3 DDIM steps produces inaccurate behaviors and fails to complete the tasks, indicating that DP requires more inference steps for stable action generation.

We additionally evaluate the step-wise reactive capability of TDP by introducing external disturbances during task execution. Fig. 7.7 and Fig. 7.8 show representative keyframes before and after disturbances. In the on-table reorientation task, we introduce perturbations by bending the thumb finger of the Allegro hand. As shown in Fig. 7.7, TDP quickly reacts to the perturbation and recovers to a stable configuration. In contrast, DP continues executing the pre-generated action sequence, causing the fingers to become stuck on the object and resulting in task failure. In the jar-opening task, we introduce disturbance by shifting the position of the jar during execution. As shown in Fig. 7.8, TDP rapidly tracks the updated jar pose and continues the opening motion. In contrast, DP blindly executes the pre-generated action chunk, failing to adapt to the changed object position.

Overall, the physical experiments demonstrate that TDP consistently achieves higher success rates and more stable manipulation behaviors than DP. When unexpected contact changes or disturbances occur, TDP can react immediately and adjust the hand configuration online. In contrast, DP tends to continue executing pre-generated action sequences in an open-loop manner, which often leads to failure.

Importantly, TDP supports inference with very few DDIM steps, significantly reducing latency and enabling fast, high-frequency control. Combined with its per-step reactive control within the learned action tube, this makes TDP particularly well suited for real-world contact-rich dexterous manipulation.

7.6 Discussion

In this chapter, we proposed a reactive visual–tactile policy learning approach, termed *Tube Diffusion Policy* (TDP), which augments diffusion-based imitation learning with feedback control during action execution. In contrast to conventional action chunking approaches that execute pre-generated action sequences in an open-loop manner, TDP constructs an *action tube* that enables step-wise feedback control conditioned on continuously updated observations.

A key insight of this work is that, by locally linearizing contact dynamics, the con-

trol problem can be decomposed into two complementary components. First, a diffusion-based generative policy periodically produces corrective actions to account for nonlinear dynamics and compensate for accumulated drift. Second, between these updates, a tube-bounded velocity field provides a geometric approximation of action trajectories, enabling real-time inference and fast reactive control.

We evaluate TDP on three visual–tactile manipulation tasks in simulation and two tasks in the real world. The experimental results demonstrate that the proposed approach consistently improves task success rates and robustness compared with state-of-the-art imitation learning methods. In real-world experiments, TDP further exhibits strong resilience to external disturbances and unexpected contact variations during execution. Notably, the method does not require highly accurate generation of full action sequences. Instead, the diffusion phase only needs to produce coarse corrective actions that account for global system dynamics, after which actions are generated through streaming conditional feedback flows. This design significantly reduces inference latency and enables high-frequency control in practice.

Despite these promising results, several limitations remain and point to directions for future work. First, the current approach employs simple concatenation to fuse visual, tactile, and proprioceptive observations, which neglects the inherent sparsity and asynchronous nature of tactile sensing. Future work could explore more structured or learned tactile representations to better exploit contact information. Second, although the tube-based formulation allows the use of a small number of DDIM inference steps to reduce latency, further acceleration could be achieved by leveraging recent advances in diffusion model acceleration techniques, such as one-step or distillation-based denoising methods (Song et al., 2023a; Prasad et al., 2024; Frans et al., 2025). Third, while this work focuses on improving diffusion policies, the proposed tube-based formulation is broadly applicable to other chunk-based or open-loop policy learning methods. Extending this framework to alternative generative or sequence-based control approaches remains an important direction for future research. For instance, many Vision–Language–Action (VLA) models (Black et al., 2024; Intelligence et al., 2025; Black et al., 2025) rely on flow-based action chunking and could benefit from incorporating the *action tube* formulation to enable fast and reactive execution.

Overall, by reformulating time-dependent action chunking as time-independent velocity field, this approach provides a general pathway for mitigating the limitations of open-loop execution and enabling high-frequency closed-loop control in contact-rich manipulation.

8 Conclusion and Future Work

8.1 Conclusion

Contact-rich manipulation tasks can be naturally formulated as decision-making problems comprising two tightly coupled stages: objective approximation and objective optimization. While these two stages are conceptually linked, many existing approaches do not consider them jointly. As a result, optimization is often performed over approximations that are not well suited for this purpose. Consequently, action generation relies on black-box solvers or computationally intensive sampling methods that are prone to poor local optima and high control latency. This dissertation argues that approximation and optimization should instead be treated in a coupled manner, a paradigm referred to as *structured approximation*, to enable efficient planning, control, and learning for complex physical interaction.

The central premise of this dissertation is that, although contact-rich manipulation is inherently non-smooth and combinatorial, it also contains rich latent structure that can be systematically exploited. By embedding appropriate structure into the approximation itself, otherwise intractable problems can be transformed into computationally manageable ones. In particular, the factorized nature of localized contact interactions enables objective functions to be represented through tensor networks, supporting efficient global optimization over large combinatorial spaces. At the same time, the piecewise-smooth nature of contact dynamics enables feedback control to be formulated as a geometric action flow, facilitating high-frequency reactive behavior.

Building on this perspective, the dissertation makes three main contributions.

First, we introduce Tensor Train (TT), a specific class of tensor networks, as a structured function approximator for addressing the large-scale non-convex optimization prob-

lems inherent in contact-rich manipulation. In this work, we formulate contact-rich manipulation as a decision-making process over a multi-layer decision tree, where each layer represents selections over continuous contact configurations or discrete modes. To mitigate the resulting combinatorial explosion, TT is employed to approximate this decision tree in a structured manner, enabling efficient storage and search over the combinatorial solution space. This provides a principled framework for tackling the highly non-convex optimization landscapes encountered in contact-rich manipulation tasks.

Second, given the efficient optimization capability of TT approximation, this dissertation integrates TT with symbolic reasoning and optimal control to address the combinatorial complexity induced by hybrid contact dynamics. Within this framework, TT is used to approximate value functions in approximate dynamic programming, enabling efficient skill policy learning and the construction of a task-agnostic skill library. *Logic-Skill Programming* is further proposed to sequence these pretrained skills for multi-stage contact-rich manipulation tasks. The resulting approach reformulates traditional task and motion planning as sequential value function optimization, enabling practical real-world execution of long-horizon logic-geometric plans.

Third, the dissertation addresses contact uncertainty and the challenge of sim-to-real transfer, as well as reactive control in visual–tactile, contact-rich settings. To enable robust policy learning, TT provides a structured approximation of parameter-conditioned value functions, facilitating rapid motor adaptation given probabilistic estimates of the physical parameters across varying contact conditions. In parallel, this dissertation introduces tube-bounded velocity fields as a geometric approximation of action trajectories for reactive policy learning, enabling fast and reactive closed-loop control under high-frequency tactile feedback. Together, these approaches provide both robustness to physical uncertainty and real-time visual–tactile responsiveness in contact-rich manipulation.

Overall, this dissertation demonstrates that physical intelligence emerges from the structural coupling of approximation and optimization. By moving beyond unstructured models toward structure-aware formulations, we show that tensor networks provide a principled way to capture the separable structure of task objectives, enabling efficient optimization over large combinatorial decision spaces, while the action velocity field supports high-speed inference for reactive visual–tactile manipulation. Together, these components establish a unified perspective for advancing contact-rich manipulation, bridging the gap between high-level task approximation and the low-level motion dexterity required for real-world physical interaction.

8.2 Limitations

This dissertation demonstrates that structured approximations, in particular tensor train (TT) representations and velocity field formulations, provide a powerful framework for enabling efficient decision making in contact-rich manipulation. By exploiting the underlying structure of objective functions and action representations, the proposed approaches achieve significant improvements in computational efficiency across planning, control, and learning. Despite these advantages, several important limitations remain. While each chapter has discussed the limitations of the proposed methods in their respective contexts, we here highlight several overarching challenges that persist across the entire thesis.

8.2.1 Low-Rank Assumption in Tensor Approximation

The effectiveness of tensor train (TT) approximation fundamentally relies on an implicit low-rank assumption. Throughout this work, we have empirically demonstrated that objective functions (e.g., value functions) in many contact-rich manipulation problems, such as pushing, pivoting, and related interaction tasks, admit compact low-rank representations. However, general theoretical guarantees for the emergence of such structure remain an open question. We hypothesize that objective functions are inherently smoother than the underlying dynamics, making them more amenable to low-rank approximation. While these empirical findings align with the observed regularity in contact-rich systems, a more rigorous theoretical analysis of the relationship between contact manifolds and tensor ranks is needed to further strengthen the generality of this framework.

8.2.2 Limited Scalability to High-Dimensional Observations

Despite the improved efficiency of TT-Cross compared to classical decomposition methods such as TT-SVD, as it avoids the explicit construction or storage of full tensors, the overall scalability of the approach remains limited. In its current form, the method does not readily extend to very high-dimensional inputs such as visual observations. As a result, the proposed framework is primarily restricted to state or configuration spaces with moderate dimensionality, limiting its applicability in visuomotor control settings.

8.2.3 Dependence on Models or Simulators

TT-Cross requires access to known or analytically derived functions, such as forward dynamics, cost functions, or value functions. In practice, this results in a reliance on either explicit models or black-box simulators, which limits the applicability of the approach in fully model-free settings. For example, extending tensor networks to support model-free imitation learning would require the development of alternative techniques that can learn TT representations directly from data, rather than relying on TT-Cross.

8.2.4 Limited Exploitation of Visual–Tactile Structure

In the visual–tactile manipulation setting, the current approach adopts a relatively simple feature concatenation strategy to fuse inputs. This design does not fully exploit the intrinsic properties of tactile sensing, such as sparsity and spatial structure. By flattening inputs into vectors, the model discards spatial inductive biases and treats independent sensing channels from different fingers as uncorrelated dimensions. This lack of structural awareness obscures the relative spatial configuration between contact points, preventing the agent from perceiving coordinated patterns such as object symmetry, grasp geometry, and tactile pressure distribution.

8.3 Future Work

The limitations discussed above point to several promising directions for extending the proposed framework. In particular, addressing these challenges requires advancing the integration of tensor networks with data-driven learning, dynamics modeling, and multimodal perception. The following directions outline potential avenues for future research.

8.3.1 Data-driven Neural Tensor Networks

As discussed in the limitations, the current framework relies on TT-Cross, which assumes access to known or analytically derived functions and does not scale well to high-dimensional observations. A promising direction is therefore the development of data-driven neural tensor networks that integrate the expressive capacity of neural networks (NNs) with the structured representations of tensor networks (TNs). Rather than explicitly constructing TNs from discretized analytical functions, such approaches would learn factorized representations directly from data, including sampled trajec-

tories, human demonstrations, or interaction experience. In doing so, higher-order geometric and physical relationships could be preserved in their structured form, avoiding conventional flattening while benefiting from reduced parameter counts and improved optimization efficiency (Wang et al., 2023a).

One potential avenue is to draw inspiration from latent generative models such as Stable Diffusion (Rombach et al., 2022). Rather than operating directly in the high-dimensional space, tensor approximation could be performed in a learned latent space that captures the essential structure of the underlying function. This approach may significantly reduce the dimensionality of the representation while preserving key structural relationships among variables.

8.3.2 Tensor Networks for Contact Dynamics Learning

As discussed in the limitations, the current framework relies on an implicit low-rank assumption, but this assumption has not been explicitly investigated in the thesis. This is because the present work primarily studies the tensor approximation of objective functions, such as value functions, cost functions, or policy-related quantities, through which the low-rank structure of contact-rich manipulation is only examined indirectly.

A promising direction for future work is therefore to study contact dynamics more directly through tensor-based dynamics model learning. Instead of approximating downstream objective functions, we could represent the underlying contact dynamics itself in TT format and fit models with different rank budgets. By controlling the TT rank and evaluating the resulting approximation accuracy, it becomes possible to quantitatively investigate whether contact dynamics admit low-rank structure, and under what conditions such structure emerges.

If contact dynamics are indeed well approximated by low-rank tensor representations, this would make it possible to learn structured dynamics models that support efficient prediction, planning, and control in contact-rich manipulation. In this context, tensor networks would offer a complementary perspective to existing world model approaches (Ha and Schmidhuber, 2018; Hafner et al., 2023), which aim to learn predictive models of the environment directly from high-dimensional observations such as images, often without explicitly exploiting the underlying physical structure.

8.3.3 Tensor Networks for Belief-space Planning

While the aforementioned dynamics learning typically assumes full state observability, practical robot manipulation tasks often operate under partial observability due to occlusions, sensor noise, or inherent state ambiguities. In such scenarios, the robot must maintain and plan over a probability distribution of the environment state, known as the belief space. However, belief-space planning suffers acutely from the curse of dimensionality. The representation, prediction, and Bayesian updating of high-dimensional joint probability distributions are computationally prohibitive, as a dense grid representation scales exponentially with the state dimension. Traditional methods often resort to particle filters or Gaussian approximations, which either suffer from particle deprivation in high-dimensional spaces or struggle to capture multimodal, non-Gaussian uncertainties common in contact-rich interactions.

To address these challenges, a promising future direction is to leverage tensor networks, particularly the Tensor Train (TT) format, for compact belief representation and efficient belief-space planning. A multidimensional joint probability density function (PDF) can be naturally conceptualized as a high-order tensor. By factorizing this tensor into a low-rank TT representation, the storage and computational complexities can be reduced from exponential to linear with respect to the state dimension.

More importantly, fundamental probabilistic operations required for belief evolution, such as marginalization, conditioning, and prediction, can be performed analytically and directly within the TT compressed format without full tensor reconstruction. This algebraic advantage bypasses the need for sample-based approximations and opens up the possibility of integrating TT-based belief dynamics with sample-efficient tree search or trajectory optimization algorithms. Investigating this direction would provide a scalable and mathematically rigorous framework for robust planning under uncertainty, bridging the gap between structural tensor approximations and stochastic optimal control.

8.3.4 Tensor Networks for Energy-based Imitation Learning

Recent advances in imitation learning increasingly leverage generative models, including energy-based models, diffusion models, and flow matching approaches, to represent policies as distributions over actions conditioned on the current observation. Despite their strong expressive power, these methods typically rely on iterative inference procedures, which introduce substantial computational overhead. As a result, policy execution can be slow, making these approaches difficult to deploy in high-frequency control settings required for contact-rich manipulation.

To address this limitation, a promising direction is to approximate such energy functions using tensor networks. In particular, tensor train (TT) representations provide a compact and structured parametrization that captures interactions between states and actions while enabling fast evaluation and efficient optimization. This formulation enables policies that retain the expressiveness of generative models while achieving substantially faster inference.

8.3.5 Structured Tactile Representation

As discussed in the limitations section, flattening or concatenating multi sensor signals discards the inherent spatial structure and sparsity of tactile data. This structural neglect obscures the relative spatial configuration across contact points, forcing policies to overfit to instance specific signatures rather than learning generalizable contact geometry.

To address this, future research could focus on structured representations that preserve tactile spatial topology. Tensor networks provide a principled framework to exploit this structure by modeling tactile observations as multi-dimensional arrays with explicit factorization of cross sensor dependencies. Furthermore, cross attention (Vaswani et al., 2017; Dosovitskiy et al., 2021) can dynamically align these localized features with visual observations. By replacing simple concatenation with spatial alignment, the agent can better resolve coordinated patterns such as grasp geometry and pressure distribution, facilitating robust generalization across diverse objects and embodiments.

A Appendix

A.1 Appendix to Chapter 4

A.1.1 Experimental Hyperparameter

In our experiments, we utilized an NVIDIA GeForce RTX 3090 GPU with 24 GB of memory. The tolerance for the TT-Cross approximation was set to $\epsilon = 10^{-3}$. Table A.1 summarizes the hyperparameters applied across different tasks. *Batch size* indicates the maximum number of parallel environments; *Num. MCTS* denotes the number of independent MCTS instances executed concurrently; and *Num. Sim.* specifies the number of parallel environments used during the simulation phase. *Num. Discret.* refers to the discretization granularity of the state and action spaces. The parameter $r_{\max}$ defines the maximum TT rank employed during TT-Tree initialization via TT-Cross. *Pop. Size* and *CMA-ES Iter.* are the population size and iteration count used in the CMA-ES refinement step, respectively.

Table A.1: Hyperparameters employed for different tasks.

Task	Batch Size	Num. MCTS	Num. Discret.	Num. Sim.	r_{max}	Pop. Size	CMA-ES Iter.
3-joint IK	1000	5	20	1000	21	25	20
7-joint IK	500	5	20	1000	21	25	20
3-joint MP	1000	5	20	1000	41	25	20
7-joint MP	500	5	20	1000	21	25	20
LegMani	500	5	20	1000	21	25	20
Multi-stage MP	1000	5	50	1000	41	25	20
Whole-body Manipulation	500	1	20	500	21	25	20

A.1.2 Cost Function Used in Experiments

Continuous non-convex function. To illustrate the capability of TTTS in handling non-convex optimization problems and its advantage over TTGO, we construct the following full-rank two-dimensional function as a toy example. The performance of TTTS on this function is reported in Section 4.4.2.

$$\begin{aligned} f_1(\boldsymbol{x}) &= -\tfrac{1}{2}\left(0.8z - 2\ \mathrm{sign}(z)\right)^2 + 0.1\|\boldsymbol{x}\|_2^2 + 2, \\ \boldsymbol{x} &= (x_1, x_2)^\top, \quad z = \tfrac{x_1+x_2}{\sqrt{2}} \end{aligned} \tag{A.1}$$

Mixed-integer non-convex function. MsMP problems can be generally formulated as mixed-integer non-convex programs. To demonstrate the effectiveness of TTTS in addressing such problems, we consider the following piecewise-defined mixed-integer non-convex function:

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \boldsymbol{x},$$

$$f_2(\boldsymbol{x}) = \tfrac{1}{2}\, g(\tilde{x}_1) + 0.1\, h(\tilde{x}_1, \tilde{x}_2), \tag{A.2}$$

where the piecewise components are defined as

$$g(\tilde{x}_1) = \begin{cases} |-(\tilde{x}_1+5)^2 + 8|, & -7 \le \tilde{x}_1 < -3, \\ |-(\tilde{x}_1+1)^2 + 3|, & -3 \le \tilde{x}_1 < 1, \\ |-(\tilde{x}_1-3)^2 + 4|, & 1 \le \tilde{x}_1 \le 5, \\ |-(\tilde{x}_1-7)^2 + 5|, & 5 < \tilde{x}_1 \le 9, \\ |-(\tilde{x}_1-11)^2 + 10|, & 9 < \tilde{x}_1 \le 13, \\ 0, & \text{otherwise}, \end{cases}$$

$$h(\tilde{x}_1, \tilde{x}_2) = \begin{cases} (\tilde{x}_1+5)^2 + (\tilde{x}_2+5)^2, & -7 \le \tilde{x}_1 < -3, \\ (\tilde{x}_1+1)^2 + (\tilde{x}_2+1)^2, & -3 \le \tilde{x}_1 < 1, \\ (\tilde{x}_1-3)^2 + (\tilde{x}_2-3)^2, & 1 \le \tilde{x}_1 \le 5, \\ (\tilde{x}_1-7)^2 + (\tilde{x}_2-7)^2, & 5 < \tilde{x}_1 \le 9, \\ (\tilde{x}_1-11)^2 + (\tilde{x}_2-11)^2, & 9 < \tilde{x}_1 \le 13, \\ 0, & \text{otherwise}. \end{cases}$$

subject to the constraints

$$\boldsymbol{x} = (x_1, x_2)^\top, \quad x_1 \in \{0, 1, \ldots, 10\}, \quad x_2 \in [-5, 5].$$

The performance of TTTS on this function is reported in Section 4.4.2.

Cost function for inverse kinematics. In the inverse kinematics (IK) setting, the goal is to find a joint configuration $\boldsymbol{q} \in \mathbb{R}^n$ such that the robot's end-effector reaches a desired target position while avoiding collisions and maintaining reasonable deviation from the current posture. The total cost function used is defined as:

$$c_{\text{total}} = 50\, c_{\text{goal}} + c_{\text{obst}}.$$

The term c_{goal} penalizes the distance between the forward kinematics output of the proposed joint configuration and the desired end-effector position:

$$c_{\text{goal}} = \left\| \text{eef}_{\text{pos}}(\boldsymbol{q}) - \boldsymbol{x}_{\text{goal}} \right\|_2.$$

Here, $\boldsymbol{x}_{\text{goal}}$ is the target pose of the end-effector, and $\text{eef}_{\text{pos}}(\boldsymbol{q})$ denotes the pose of the end-effector computed using forward kinematics at joint configuration $\boldsymbol{q}$.

The term c_{obst} penalizes collisions by summing binary collision indicators for the final robot configuration:

$$c_{\text{obst}} = \sum_{i=1}^{N_{\text{links}}} \text{collides}_i(\boldsymbol{q}),$$

where $\text{collides}_i(\boldsymbol{q}) \in \{0, 1\}$ indicates whether the i-th link is in collision.

Cost function for motion planning. The total cost function used for trajectory optimization is composed of three terms: a goal-reaching cost, a collision avoidance cost, and a control smoothness cost. The overall cost is defined as:

$$c_{\text{total}} = 50\, c_{\text{goal}} + c_{\text{obst}} + 0.1\, c_{\text{control}}.$$

The first term, c_{goal}, encourages the robot to reach the desired target position. It is computed as the Euclidean distance between the current end-effector position and

the goal:

$$c_{\text{goal}} = \left\| \text{eef}_{\text{pos}} - \boldsymbol{x}_{\text{goal}} \right\|_2,$$

where eef_{pos} denotes the Cartesian position of the robot's end-effector at the final time step, and $\boldsymbol{x}_{\text{goal}}$ is the desired target position.

The second term, c_{obst}, penalizes trajectories that result in collisions. It is computed by summing binary collision indicators over the trajectory:

$$c_{\text{obst}} = \sum_{t=1}^{T} \text{collisions}_t,$$

where $\text{collisions}_t \in \{0, 1\}$ is a binary variable indicating whether a collision occurs at time step t.

The third term, c_{control}, measures the smoothness of the joint trajectory. It compares the total length of the path in joint space with the direct distance between the start and end configurations:

$$c_{\text{control}} = \frac{\sum_{t=1}^{T} \|\boldsymbol{q}_t - \boldsymbol{q}_{t-1}\|_2}{\|\boldsymbol{q}_T - \boldsymbol{q}_0\|_2 + \varepsilon}.$$

Here, $\boldsymbol{q}_t$ represents the robot's joint configuration at time step t, and ε is a small positive constant added for numerical stability. We set $\varepsilon = 10^{-6}$ in our experiments.

Cost function for legged robot manipulation. In the legged robot manipulation setting, the objective is to find optimal robot trajectories which move the manipulated box toward the target pose. The total cost is defined as

$$c_{\text{total}} \;=\; c_{\text{orn}} + c_{\text{vel}} + 5\, c_{\text{pos}},$$

where the three components are given below.

The first term penalizes deviation between the final box orientation and the desired orientation:

$$c_{\text{orn}} \;=\; 0.1 \left\| \mathbf{o}_T^{\text{box}} - \mathbf{o}_{\text{target}} \right\|_2,$$

where o_T^{box} is the yaw angle of the final box orientation.

The second term encourages stable manipulation by penalizing abrupt orientation

changes of the box during the trajectory:

$$c_{\text{vel}} \;=\; \frac{0.1}{T-1}\sum_{t=0}^{T-2}\left|\, o_{t+1}^{\text{box}} - o_{t}^{\text{box}} \,\right|,$$

where o_t^{box} is the yaw angle of the box at time t.

The third term penalizes final position error of the box in the horizontal plane:

$$c_{\text{pos}} \;=\; 0.1\left\|\mathbf{p}_T^{\text{box}} - \mathbf{p}_{\text{target}}\right\|_2,$$

where $\mathbf{p}_T^{\text{box}}$ denotes the final box position projected onto the xy-plane.

The cost function therefore balances final pose accuracy (position and orientation) with the smoothness of the manipulated box trajectory, promoting stable and efficient robot–box interaction.

Cost function for multi-stage motion planning. In the planar pushing setting, the objective is to compute a sequence of discrete contact modes and continuous robot velocities that move an object from its initial state to a target pose. The total cost combines two components: a state reaching cost and a control effort cost. It is defined as:

$$c_{\text{total}} = c_{\text{state}} + 0.01\, c_{\text{action}}.$$

The state cost c_{state} evaluates how far the final state $\boldsymbol{x}_T \in \mathbb{R}^3$ (position and orientation of the object) is from the target pose $\boldsymbol{x}_{\text{target}} \in \mathbb{R}^3$. It is defined as:

$$c_{\text{state}} = \left\|\boldsymbol{x}_T^{\text{pos}} - \boldsymbol{x}_{\text{target}}^{\text{pos}}\right\|_2 + 0.1\;\left|\theta_T - \theta_{\text{target}}\right|,$$

where:

- $\boldsymbol{x}_T^{\text{pos}} = [x_{1_T}, x_{2_T}]^\top$ is the object's final position
- θ_T is the final orientation
- $\boldsymbol{x}_{\text{target}}^{\text{pos}}$ and θ_{target} are the desired position and orientation

The action cost c_{action} penalizes the total effort of the pushing trajectory, where each control input $\mathbf{u}_t \in \mathbb{R}^2$ represents a planar velocity vector. The total control cost is the

sum of the norms of all pushing actions:

$$c_{\text{action}} = \sum_{k=1}^{K} \sum_{t=1}^{T} \|\mathbf{u}_t\|_2 \,.$$

These actions are generated based on a discrete contact face selection and continuous parameters defining a trajectory. The discrete mode $i_t \in \{1, 2, 3, 4\}$ specifies which face of the object is being pushed at each time.

The final cost balances reaching the target pose accurately with minimizing the pushing effort, with a small weight on the latter to avoid excessive motion without overconstraining the optimization.

Cost function for whole-body manipulation. In the bimanual whole-body manipulation setting, the objective is to control two arms collaboratively to manipulate an object (e.g., a box) toward a target pose while maintaining effective contact and avoiding awkward configurations. The total cost function is composed of five terms:

$$c_{\text{total}} = c_{\text{pos}} + 50\, c_{\text{orn}} + 0.1\, c_{\text{control}} - c_{\text{contact}} + 5\, \delta_{\text{eef}},$$

where c_{pos} penalizes deviation of the box's final position from the target:

$$c_{\text{pos}} = \left\| \mathbf{p}_T^{\text{box}} - \mathbf{p}_{\text{target}} \right\|_2 .$$

$\mathbf{p}_T^{\text{box}} \in \mathbb{R}^3$ is the final box position and $\mathbf{p}_{\text{target}}$ is the desired position. c_{orn} measures orientation alignment between the final box orientation and the target orientation, namely

$$c_{\text{orn}} = \left| \theta_T^{\text{box}} - \theta_{\text{target}} \right| .$$

c_{control} is a regularization term that penalizes excessive joint movement across the trajectory:

$$c_{\text{control}} = \sum_{t=0}^{T} \|\boldsymbol{q}_{t+1} - \boldsymbol{q}_t\|_2 \,,$$

where $\boldsymbol{q}_t \in \mathbb{R}^n$ is the full-body joint configuration at time step t.

c_{contact} rewards the maintenance of valid bimanual contact. Let $c_t^L, c_t^R \in \{0, 1\}$ be

binary contact flags for the left and right hands. Then:

$$c_{\text{contact}} = \sum_{t=1}^{T} \mathbb{I}\left[c_t^L = 1 \wedge c_t^R = 1\right],$$

where $\mathbb{I}[\cdot]$ is the indicator function. This term is subtracted from the total cost to encourage simultaneous dual-arm contact.

δ_{eef} is a binary term that penalizes configurations where the two arms are too close to each other, in order to avoid self-collision:

$$\delta_{\text{eef}} = \mathbb{I}\left[\left\|\mathbf{x}^{\text{eef}_0} - \mathbf{x}^{\text{eef}_1}\right\|_2 < 0.3\right],$$

where $\mathbf{x}^{\text{eef}_i}$ is the Cartesian position of the i-th end-effector.

The cost function encourages accurate placement and orientation of the manipulated object, smooth and efficient joint trajectories, persistent bimanual contact, and physically feasible arm configurations.

A.2 Appendix to Chapter 7

A.2.1 Stability Analysis of Tube Diffusion Policy

This section provides a more detailed analysis and formal proof of the stability properties of Tube Diffusion Policy (TDP). TDP decomposes control into two complementary mechanisms: diffusion correction, which periodically contracts the accumulated drift, and streaming imitation, which provides high-frequency reactive control but introduces bounded drift. Together, these mechanisms form a hybrid control system that admits a practical stability guarantee.

Problem Setup. Consider the discrete-time nonlinear system

$$\boldsymbol{o}_{t+1} = g(\boldsymbol{o}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t, \tag{A.3}$$

where $\boldsymbol{o}_t \in \mathbb{R}^{n_o}$ is the system state, $\boldsymbol{u}_t \in \mathbb{R}^{n_u}$ is the control input, and $\boldsymbol{w}_t$ is a bounded disturbance.

We assume access to a reference demonstration trajectory

$$(\boldsymbol{o}_t^*, \boldsymbol{u}_t^*) \tag{A.4}$$

that satisfies the nominal dynamics

$$\boldsymbol{o}_{t+1}^{*} = g(\boldsymbol{o}_t^{*}, \boldsymbol{u}_t^{*}). \tag{A.5}$$

Tube Diffusion Policy consists of two components:

Diffusion correction. At correction instants t_k, a diffusion model produces a corrective action that compensates for the drift accumulated during streaming execution.

Streaming flow policy. A learned policy generates actions sequentially,

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + v_\theta(\boldsymbol{u}, t_1 = 0, t_2 \mid h) \cdot \Delta t, \tag{A.6}$$

and is trained to imitate the demonstration action trajectory.

We define the tracking error

$$\boldsymbol{e}_t := \boldsymbol{o}_t - \boldsymbol{o}_t^{*}. \tag{A.7}$$

Assumptions. To analyze the stability of Tube Diffusion Policy, we introduce the following assumptions. Assumptions 1 and 3 characterize properties of the system dynamics and disturbances, while Assumptions 2 and 4 describe the desired behavior of the learned streaming and diffusion policies induced by training.

Assumption 1 (Lipschitz dynamics). There exist constants $L_x > 0$ and $L_u > 0$ such that for all admissible states and actions,

$$\|g(\boldsymbol{o}_1, \boldsymbol{u}_1) - g(\boldsymbol{o}_2, \boldsymbol{u}_2)\| \le L_x \|\boldsymbol{o}_1 - \boldsymbol{o}_2\| + L_u \|\boldsymbol{u}_1 - \boldsymbol{u}_2\|. \tag{A.8}$$

The Lipschitz condition bounds the sensitivity of the system dynamics to state and action deviations, enabling tractable propagation of tracking errors.

Assumption 2 (Streaming imitation accuracy). During streaming execution, the learned policy approximately imitates the reference actions, resulting in a bounded action error:

$$\|\boldsymbol{u}_t - \boldsymbol{u}_t^{*}\| \le \varepsilon_a. \tag{A.9}$$

This condition reflects the training objective of the streaming phase and is expected to hold when the policy accurately reproduces the demonstrated actions.

Assumption 3 (Bounded disturbance). The external disturbance is bounded:

$$\|\boldsymbol{w}_t\| \leq \bar{w}. \tag{A.10}$$

Assumption 4 (Diffusion correction property). At each correction instant t_k, the diffusion module produces a corrective action that contracts the tracking error:

$$\|\boldsymbol{e}_{t_k+1}\| \leq \lambda \|\boldsymbol{e}_{t_k}\| + \varepsilon_d, \qquad 0 \leq \lambda < 1. \tag{A.11}$$

This condition corresponds to the training objective of the diffusion phase, which seeks to reduce the accumulated tracking error at correction steps by accounting for the full system dynamics.

Streaming Phase Error Dynamics. From (A.3), (A.5), and (A.7), the error dynamics during the streaming phase satisfy

$$\begin{aligned} \boldsymbol{e}_{t+1} &= g(\boldsymbol{o}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t - g(\boldsymbol{o}_t^*, \boldsymbol{u}_t^*) && \text{(A.12)} \\ &= g(\boldsymbol{o}_t, \boldsymbol{u}_t) - g(\boldsymbol{o}_t^*, \boldsymbol{u}_t^*) + \boldsymbol{w}_t. && \text{(A.13)} \end{aligned}$$

Applying Assumption 1 gives

$$\|\boldsymbol{e}_{t+1}\| \leq L_x \|\boldsymbol{e}_t\| + L_u \|\boldsymbol{u}_t - \boldsymbol{u}_t^*\| + \|\boldsymbol{w}_t\|. \tag{A.14}$$

Using Assumptions 2 and 3,

$$\|\boldsymbol{e}_{t+1}\| \leq L_x \|\boldsymbol{e}_t\| + c, \qquad c := L_u \varepsilon_a + \bar{w}. \tag{A.15}$$

Lyapunov Function for the Error Dynamics. Consider the Lyapunov candidate

$$V(\boldsymbol{e}_t) := \|\boldsymbol{e}_t\|. \tag{A.16}$$

This function is positive definite and radially unbounded with respect to the tracking error $\boldsymbol{e}_t$. Then (A.15) implies

$$V(\boldsymbol{e}_{t+1}) \leq L_x V(\boldsymbol{e}_t) + c. \tag{A.17}$$

Thus, the streaming phase is input-to-state stable with respect to the combined input c, although it does not necessarily contract to zero in the absence of diffusion correction.

Lemma 1 (Streaming-phase ISS bound)**.** *For any integer $j \geq 0$, the tracking error during the streaming phase satisfies*

$$V(\boldsymbol{e}_{t+j}) \leq L_x^j V(\boldsymbol{e}_t) + \sum_{i=0}^{j-1} L_x^i c. \tag{A.18}$$

If $L_x < 1$, then

$$V(\boldsymbol{e}_{t+j}) \leq L_x^j V(\boldsymbol{e}_t) + \frac{1 - L_x^j}{1 - L_x} c. \tag{A.19}$$

Proof. The result follows by recursively unrolling (A.17). When $L_x < 1$, the geometric series identity

$$\sum_{i=0}^{j-1} L_x^i = \frac{1 - L_x^j}{1 - L_x}$$

yields (A.19). □

Error Evolution Across Correction Cycles. The diffusion correction is applied every H_a steps. Let $z_k := V(\boldsymbol{e}_{t_k})$ denote the tracking error immediately after the diffusion correction at time t_k.

After the correction at t_k, the system evolves under the streaming controller for the next $H_a - 1$ steps, from $t_k + 1$ to $t_k + H_a - 1$. By Lemma 1, the error at time $t_k + H_a - 1$ satisfies

$$V(\boldsymbol{e}_{t_k + H_a - 1}) \leq L_x^{H_a - 1} z_k + \frac{1 - L_x^{H_a - 1}}{1 - L_x} c. \tag{A.20}$$

At time $t_{k+1} = t_k + H_a$, a new diffusion correction is applied. Using Assumption 4 gives

$$z_{k+1} \leq \lambda \left(L_x^{H_a - 1} z_k + \frac{1 - L_x^{H_a - 1}}{1 - L_x} c \right) + \varepsilon_d. \tag{A.21}$$

Equivalently,

$$z_{k+1} \leq \alpha z_k + \beta, \tag{A.22}$$

where

$$\alpha := \lambda L_x^{H_a - 1}, \qquad \beta := \lambda \frac{1 - L_x^{H_a - 1}}{1 - L_x} c + \varepsilon_d. \tag{A.23}$$

Lemma 2 (Contraction across correction cycles)**.** *Suppose* $L_x < 1$*. Then* $0 \leq \alpha < 1$*, and the post-correction error sequence satisfies*

$$z_k \leq \alpha^k z_0 + \frac{1-\alpha^k}{1-\alpha}\beta. \tag{A.24}$$

Hence,

$$\limsup_{k\to\infty} z_k \leq \frac{\beta}{1-\alpha}. \tag{A.25}$$

Proof. Since $0 \leq \lambda < 1$ and $L_x < 1$, we have $0 \leq \alpha = \lambda L_x^{H_a-1} < 1$. The result follows by unrolling the affine recursion (A.22). □

Proposition 3 (Practical stability of Tube Diffusion Policy)**.** *Suppose Assumptions 1–4 hold and* $L_x < 1$*. Then the closed-loop system under TDP satisfies the following properties:*

(1) Uniform ultimate boundedness. The tracking error sequence is uniformly ultimately bounded, i.e.,

$$\limsup_{t\to\infty} \|\boldsymbol{e}_t\| < \infty.$$

(2) Bounded error within each correction cycle. For any correction cycle $t \in \{t_k, \ldots, t_k + H_a - 1\}$*, the tracking error remains bounded as a function of the post-correction error* z_k*.*

Proof. First, by Lemma 2, the post-correction error satisfies the affine recursion

$$z_{k+1} \leq \alpha z_k + \beta,$$

with $0 \leq \alpha < 1$. Standard results for linear difference inequalities imply

$$\limsup_{k\to\infty} z_k \leq \frac{\beta}{1-\alpha}.$$

Hence the sequence of post-correction errors is uniformly ultimately bounded.

Next, consider any time $t \in \{t_k, \ldots, t_k + H_a - 1\}$. By Lemma 1,

$$\|\boldsymbol{e}_t\| \leq L_x^{H_a-1} z_k + \frac{1-L_x^{H_a-1}}{1-L_x} c.$$

Thus the tracking error remains bounded during each streaming phase as a function of z_k. Combining this result with the boundedness of z_k establishes uniform ultimate

boundedness of the full error trajectory.

In the ideal case where $\varepsilon_a = 0$, $\bar{w} = 0$, and $\varepsilon_d = 0$, we have $c = 0$ and $\beta = 0$, so the recursion reduces to

$$z_{k+1} \leq \alpha z_k, \qquad 0 \leq \alpha < 1.$$

Hence $z_k \to 0$. Using Lemma 1 with $c = 0$ further implies $\boldsymbol{e}_t \to 0$, and the closed-loop system is asymptotically stable.

□

In summary, Theorem 3 formalizes the role of the two components in TDP. At each correction cycle, the diffusion module first performs a correction step, followed by $H_a - 1$ streaming control steps. The streaming flow provides high-frequency reactive control. However, due to bounded imitation error and external disturbances, it introduces a bounded tracking drift during the streaming phase. The diffusion module periodically contracts this accumulated drift at the beginning of each correction cycle. Their combination yields a hybrid closed-loop system in which the tracking error remains within a bounded tube around the demonstrated reference trajectory. This matches the intuition behind TDP: streaming flow ensures responsive step-wise control, while diffusion correction prevents long-term drift accumulation.

A.2.2 Network Architecture and Hyperparameters

We provide additional details of the policy network architecture. Our method employs a conditional 1D U-Net backbone for diffusion-based action generation and streaming flow, augmented with visual and tactile encoders.

U-Net backbone. The policy network is a 1D U-Net with three resolution levels (channel dimensions 256, 512, and 1024), consisting of symmetric downsampling and upsampling stages with 12 FiLM-conditioned residual blocks.

Conditioning. Observation conditioning is incorporated via feature-wise linear modulation (FiLM). The conditioning dimension is 3118, and FiLM layers produce per-channel scale and shift parameters.

Observation encoders. Visual and tactile inputs are encoded using six ResNet-18 backbones (two for vision and four for tactile inputs, without parameter sharing).

Model size. The policy U-Net contains 44.7M parameters. The full policy network, including the U-Net backbone and action prediction heads but excluding observation encoders, contains 105.2M parameters. The complete model, including all visual and tactile encoders, contains 172.8M parameters.

Table A.2: Network architecture and hyperparameters

Component	Specification
U-Net backbone	1D U-Net with 3 resolution levels (channels 256, 512, 1024)
Residual blocks	12 FiLM-conditioned residual blocks
Backbone encoders	6 ResNet-18 encoders (2 vision, 4 tactile; no weight sharing)
FiLM conditioning	Linear layer with conditioning dimension 3118 generating scale and shift
U-Net backbone	44.7M
Policy network	105.2M
Full model	172.8M

Table A.2 summarizes the key components and hyperparameters.

Bibliography

Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., Weld, D., SRI, D. W., Barrett, A., Christianson, D., et al. (1998). Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*

Agia, C., Migimatsu, T., Wu, J., and Bohg, J. (2023). Stap: Sequencing task-agnostic policies. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7951–7958. IEEE.

Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36.

Arndt, K., Hazara, M., Ghadirzadeh, A., and Kyrki, V. (2020). Meta reinforcement learning for sim-to-real domain adaptation. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 2725–2731.

Bauza, M., Bronars, A., Hou, Y., Taylor, I., Chavan-Dafle, N., and Rodriguez, A. (2024). Simple, a visuotactile method learned in simulation to precisely pick, localize, regrasp, and place objects. *Science Robotics*, 9(91):eadi8808.

Biamonte, J. and Bergholm, V. (2017). Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006*.

Bianchini, B., Halm, M., and Posa, M. (2023). Simultaneous learning of contact and continuous dynamics. In *Conference on Robot Learning*, pages 3966–3978. PMLR.

Billard, A. G., Calinon, S., and Dillmann, R. (2016). Learning from humans. In Siciliano, B. and Khatib, O., editors, *Handbook of Robotics*, chapter 74, pages 1995–2014. Springer, Secaucus, NJ, USA. 2nd Edition.

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., et al. (2024). π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*.

Black, K., Galliker, M. Y., and Levine, S. (2025). Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*.

Bonami, P., Kilinç, M., and Linderoth, J. (2012). Algorithms and software for convex mixed integer nonlinear programs. In *Mixed integer nonlinear programming*, pages 1–39. Springer.

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 4243–4250.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121(1-2):49–107.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Burer, S. and Letchford, A. N. (2012). Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106.

Calandra, R., Owens, A., Jayaraman, D., Lin, J., Yuan, W., Malik, J., Adelson, E. H., and Levine, S. (2018). More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3300–3307.

Calinon, S. (2018). Robot learning with task-parameterized generative models. In *Robotics Research*, pages 111–126. Springer.

Calinon, S. and Lee, D. (2019). Learning control. In Vadakkepat, P. and Goswami, A., editors, *Humanoid Robotics: a Reference*, pages 1261–1312. Springer.

Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). The YCB object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE.

Chang, Y., Chen, P. Y., Wang, Z., Chiaramonte, M. M., Carlberg, K., and Grinspun, E. (2023). Licrom: Linear-subspace continuous reduced order modeling with neural fields. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–12.

Chaslot, G. M. B., Winands, M. H., and van Den Herik, H. J. (2008). Parallel monte-carlo tree search. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 60–71. Springer.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., and Fox, D. (2019). Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 8973–8979.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems (NIPS)*, 31.

Chen, T., Simeonov, A., and Agrawal, P. (2019). AIRobot. https://github.com/Improbable-AI/airobot.

Cheng, S. and Xu, D. (2023). League: Guided skill learning and abstraction for long-horizon manipulation. *IEEE Robotics and Automation Letters (RA-L)*, 8(10):6451–6458.

Cheng, X., Patil, S., Temel, Z., Kroemer, O., and Mason, M. T. (2023). Enhancing dexterity in robotic manipulation via hierarchical contact exploration. *IEEE Robotics and Automation Letters (RA-L)*, 9(1):390–397.

Chi, C., Xu, Z., Feng, S., Cousineau, E., Du, Y., Burchfiel, B., Tedrake, R., and Song, S. (2024). Diffusion policy: Visuomotor policy learning via action diffusion. *International Journal of Robotics Research (IJRR)*, 0(0).

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. (2016). Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9(4-5):249–429.

Cong, L., Liang, H., Ruppel, P., Shi, Y., Görner, M., Hendrich, N., and Zhang, J. (2022). Reinforcement learning with vision-proprioception model for robot planar pushing. *Frontiers in Neurorobotics*, 16:829437.

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning.(2016). *URL http://pybullet. org*.

Coumans, E. and Bai, Y. (2016–2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. https://pybullet.org.

Cunningham, P. and Delany, S. J. (2021). K-nearest neighbour classifiers-a tutorial. *ACM Computing Surveys (CSUR)*, 54(6):1–25.

De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67.

De Moura, J. P., Stouraitis, T., and Vijayakumar, S. (2022). Non-prehensile planar manipulation via trajectory optimization with complementarity constraints. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*.

Dolgov, S., Anaya-Izquierdo, K., Fox, C., and Scheichl, R. (2020). Approximation and sampling of multivariate probability distributions in the tensor train decomposition. *Statistics and Computing*, 30(3):603–625.

Dolgov, S., Kalise, D., and Saluzzi, L. (2023). Data-driven tensor train gradient cross approximation for hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 45(5):A2153–A2184.

Doshi, N., Hogan, F. R., and Rodriguez, A. (2020). Hybrid differential dynamic programming for planar manipulation primitives. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 6759–6765.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Yevgen, C., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., Zeng, A., Mordatch, I., and Florence, P. (2023). Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning*, pages 8469–8488. PMLR.

Du, Y. and Mordatch, I. (2019). Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems (NIPS)*, volume 32.

Envall, J., Poranne, R., and Coros, S. (2023). Differentiable task assignment and motion planning. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 2049–2056.

Florence, P., Lynch, C., Zeng, A., Ramirez, O. A., Wahid, A., Downs, L., Wong, A., Lee, J., Mordatch, I., and Tompson, J. (2022). Implicit behavioral cloning. pages 158–168. PMLR.

Franco, A. L. D., Bourles, H., De Pieri, E. R., and Guillard, H. (2006). Robust nonlinear control associating robust feedback linearization and H_∞ control. *IEEE Transactions on Automatic Control*, 51(7):1200–1207.

Frans, K., Hafner, D., Levine, S., and Abbeel, P. (2025). One step diffusion via shortcut models. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Gao, J., Sarkar, B., Xia, F., Xiao, T., Wu, J., Ichter, B., Majumdar, A., and Sadigh, D. (2024). Physically grounded vision-language models for robotic manipulation. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 12462–12469. IEEE.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., and Lozano-Pérez, T. (2021). Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293.

Garrett, C. R., Lozano-Perez, T., and Kaelbling, L. P. (2018). Ffrob: Leveraging symbolic planning for efficient task and motion planning. *International Journal of Robotics Research (IJRR)*, 37(1):104–136.

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2020). Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448.

Gasparetto, A., Boscariol, P., Lanzutti, A., and Vidoni, R. (2015). Path planning and trajectory planning algorithms: A general overview. *Motion and operation planning of robotic systems: Background and practical approaches*, pages 3–27.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D. E., Sun, Y., and Weld, D. (1998). PDDL|The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control.

Girgin, H., Jankowski, J., and Calinon, S. (2022). Reactive anticipatory robot skills with memory. In *International Symposium on Robotics Research (ISRR)*.

Goldenberg, A., Benhabib, B., and Fenton, R. (2003). A complete generalized solution to the inverse kinematics of robots. *IEEE Journal on Robotics and Automation*, 1(1):14–20.

Gorodetsky, A., Karaman, S., and Marzouk, Y. (2015). Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. In *Proc. Robotics: Science and Systems (R:SS)*, pages 1–8.

Goyal, S., Ruina, A., and Papadopoulos, J. (1989). Limit surface and moment function descriptions of planar sliding. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 794–795.

Grasedyck, L. (2010). Hierarchical singular value decomposition of tensors. *SIAM journal on matrix analysis and applications*, 31(4):2029–2054.

Green, M. and Limebeer, D. J. (2012). *Linear robust control*. Courier Corporation.

Guo, Z., Deshpande, A., Wang, X., Kiedrowski, B. C., and Gorodetsky, A. A. (2026). Hierarchical tensor network structure search for high-dimensional data. *arXiv preprint arXiv:2603.27856*.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. *Advances in Neural Information Processing Systems (NIPS)*, 31.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Hansen, J., Hogan, F., Rivkin, D., Meger, D., Jenkin, M., and Dudek, G. (2022). Visuotactile-rl: Learning multimodal manipulation policies with deep reinforcement learning. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 8298–8304.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

Hargraves, C. R. and Paris, S. W. (1987). Direct trajectory optimization using nonlinear programming and collocation. *Journal of guidance, control, and dynamics*, 10(4):338–342.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hartmann, V. N., Orthey, A., Driess, D., Oguz, O. S., and Toussaint, M. (2022). Long-horizon multi-robot rearrangement planning for construction assembly. *IEEE Transactions on Robotics*, 39(1):239–252.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 770–778.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems (NIPS)*, 33:6840–6851.

Hogan, F. R., Grau, E. R., and Rodriguez, A. (2018). Reactive planar manipulation with convex hybrid mpc. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 247–253.

Hogan, F. R. and Rodriguez, A. (2020a). Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. In *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pages 800–815. Springer.

Hogan, F. R. and Rodriguez, A. (2020b). Reactive planar non-prehensile manipulation with hybrid model predictive control. *International Journal of Robotics Research (IJRR)*, 39(7):755–773.

Holladay, R., Lozano-Pérez, T., and Rodriguez, A. (2024). Robust planning for multi-stage forceful manipulation. *International Journal of Robotics Research (IJRR)*, 43(3):330–353.

Horowitz, M. B., Damle, A., and Burdick, J. W. (2014). Linear Hamilton Jacobi Bellman equations in high dimensions. *IEEE Conference on Decision and Control (CDC)*, pages 5880–5887.

Huang, D.-A., Xu, D., Zhu, Y., Garg, A., Savarese, S., Fei-Fei, L., and Niebles, J. C. (2019). Continuous relaxation of symbolic planner for one-shot imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2635–2642. IEEE.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373.

Intelligence, P., Black, K., Brown, N., Darpinian, J., Dhabalia, K., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., et al. (2025). $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*.

Jacobson, D. H. and Mayne, D. Q. (1970). *Differential Dynamic Programming*. American Elsevier Publishing Company, New York.

Jankowski, J., Brudermüller, L., Hawes, N., and Calinon, S. (2023). Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 10125–10131.

Jiang, S., Fang, X., Roy, N., Lozano-Pérez, T., Kaelbling, L. P., and Ancha, S. (2025). Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories. In *Conference on Robot Learning (CoRL)*.

Jones, C. N. and Morari, M. (2010). Polytopic approximation of explicit model predictive controllers. *IEEE Transactions on Automatic Control*, 55(11):2542–2553.

Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580.

Khansari-Zadeh, S. M. and Billard, A. (2010). Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 2676–2683.

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957.

Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems (NIPS)*, 33:6696–6707.

Kim, B., Lee, K., Lim, S., Kaelbling, L., and Lozano-Pérez, T. (2020). Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9916–9924.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Kolda, T. and Bader, B. (2009). Tensor decompositions and applications. *SIAM Review*.

Kouvaritakis, B. and Cannon, M. (2016). Model predictive control. *Switzerland: Springer International Publishing*, 38(13-56):7.

Kumar, A., Fu, Z., Pathak, D., and Malik, J. (2021). RMA: Rapid motor adaptation for legged robots. In *Proc. Robotics: Science and Systems (RSS)*.

Lai, L., Huang, A. Z., and Gershman, S. J. (2022). Action chunking as policy compression. *PsyArXiv*.

Lancaster, P. (1995). *Algebraic Riccati Equations*. Oxford Science Publications/The Clarendon Press, Oxford University Press.

Langson, W., Chryssochoos, I., Raković, S., and Mayne, D. Q. (2004). Robust model predictive control using tubes. *Automatica*, 40(1):125–133.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2020a). Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986.

Lee, M. A., Zhu, Y., Zachares, P., Tan, M., Srinivasan, K., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2020b). Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Transactions on Robotics*, 36(3):582–596.

Lee, S. H. and Cutkosky, M. (1991). Fixture planning with friction. *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, 113(3):320–327.

Lembono, T. S., Paolillo, A., Pignat, E., and Calinon, S. (2020). Memory of motion for warm-starting trajectory optimization. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):2594–2601.

Li, Q., Zhou, Z., and Levine, S. (2025). Reinforcement learning with action chunking. In *Advances in Neural Information Processing Systems (NIPS)*.

Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 222–229. SciTePress.

Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. (2018). Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. (2023a). Flow matching for generative modeling. *Proc. Intl Conf. on Learning Representations (ICLR)*.

Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., and Le, M. (2023b). Flow matching for generative modeling. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Liu, M., Zhu, M., and Zhang, W. (2022). Goal-conditioned reinforcement learning: Problems and solutions. In *Proc. Intl Joint Conf. on Artificial Intelligence (IJCAI)*, pages 5502–5511.

Liu, X., Gong, C., and Liu, Q. (2023). Flow straight and fast: Learning to generate and transfer data with rectified flow. *Proc. Intl Conf. on Learning Representations (ICLR)*.

Liu, Y., Hamid, J. I., Xie, A., Lee, Y., Du, M., and Finn, C. (2025). Bidirectional decoding: Improving action chunking via guided test-time sampling. *Proc. Intl Conf. on Learning Representations (ICLR)*.

Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. (2019). Learning latent plans from play. In *Conference on Robot Learning (CoRL)*.

Lynch, K. M., Maekawa, H., and Tanie, K. (1992). Manipulation and active sensing by pushing using tactile feedback. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, volume 1, pages 416–421.

Lynch, K. M. and Mason, M. T. (1996). Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research (IJRR)*, 15(6):533–556.

Malyuta, D., Reynolds, T. P., Szmuk, M., Lew, T., Bonalli, R., Pavone, M., and Açıkmeşe, B. (2022). Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently. *IEEE Control Systems Magazine*, 42(5):40–113.

Marcucci, T., Petersen, M., von Wrangel, D., and Tedrake, R. (2023). Motion planning around obstacles with convex optimization. *Science robotics*, 8(84):eadf7843.

Marcucci, T. and Tedrake, R. (2019). Mixed-integer formulations for optimal control of piecewise-affine systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 230–239.

Mason, M. (2001a). *Mechanics of Robotic Manipulation*.

Mason, M. T. (1986). Mechanics and planning of manipulator pushing operations. *International Journal of Robotics Research (IJRR)*, 5(3):53–71.

Mason, M. T. (1999). Progress in nonprehensile manipulation. *International Journal of Robotics Research (IJRR)*, 18(11):1129–1141.

Mason, M. T. (2001b). *Mechanics of robotic manipulation*. MIT press.

Mastalli, C., Budhiraja, R., Merkt, W., Saurel, G., Hammoud, B., Naveau, M., Carpentier, J., Righetti, L., Vijayakumar, S., and Mansard, N. (2020). Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 2536–2542.

Mayne, D. (1966). A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95.

Mayne, D. Q., Kerrigan, E. C., Van Wyk, E., and Falugi, P. (2011). Tube-based robust nonlinear model predictive control. *International journal of robust and nonlinear control*, 21(11):1341–1353.

Mehta, B., Diaz, M., Golemo, F., Pal, C. J., and Paull, L. (2020). Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR.

Migimatsu, T. and Bohg, J. (2020). Object-centric task and motion planning in dynamic environments. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):844–851.

Mishra, U. A., Xue, S., Chen, Y., and Xu, D. (2023). Generative skill chaining: Long-horizon skill planning with diffusion models. In *Conference on Robot Learning*, pages 2905–2925. PMLR.

Morari, M. and Lee, J. H. (1999). Model predictive control: past, present and future. *Computers & chemical engineering*, 23(4-5):667–682.

Moura, J., Stouraitis, T., and Vijayakumar, S. (2022). Non-prehensile planar manipulation via trajectory optimization with complementarity constraints. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 970–976.

Muratore, F., Gruner, T., Wiese, F., Belousov, B., Gienger, M., and Peters, J. (2022). Neural posterior domain randomization. In *Conference on Robot Learning*, pages 1532–1542. PMLR.

Muratore, F., Treede, F., Gienger, M., and Peters, J. (2018). Domain randomization for simulation-based policy optimization with transferability assessment. In *Conference on Robot Learning*, pages 700–713. PMLR.

Orús, R. (2019). Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1(9):538–550.

Oseledets, I. (2011a). Tensor-train decomposition. *SIAM Journal on Scientific Computing*.

Oseledets, I. and Tyrtyshnikov, E. (2010). TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88.

Oseledets, I. V. (2011b). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

Pang, T., Suh, H., Yang, L., and Tedrake, R. (2022). Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *arXiv preprint arXiv:2206.10787*.

Pashenkova, E., Rish, I., and Dechter, R. (1996). Value iteration and policy iteration algorithms for Markov decision problem. In *AAAI'96: Workshop on Structural Issues in Planning and Temporal Reasoning*, volume 39. Citeseer.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 3803–3810.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Proc. AAAI Conference on Artificial Intelligence*, volume 32.

Pertsch, K., Lee, Y., and Lim, J. (2021). Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR.

Peterson, L. E. (2009). K-nearest neighbor. *Scholarpedia*, 4(2):1883.

Posa, M., Cantu, C., and Tedrake, R. (2014). A direct method for trajectory optimization of rigid bodies through contact. *International Journal of Robotics Research (IJRR)*, 33(1):69–81.

Potters, S. (2022). Learning-based control for pushing with a non-holonomic mobile robot. Master's thesis, Delft University of Technology, Delft.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons.

Prasad, A., Lin, K., Wu, J., Zhou, L., and Bohg, J. (2024). Consistency policy: Accelerated visuomotor policies via consistency distillation. In *Proc. Robotics: Science and Systems (R:SS)*.

Qi, H., Kumar, A., Calandra, R., Ma, Y., and Malik, J. (2023). In-hand object rotation via rapid motor adaptation. In *Conference on Robot Learning*, pages 1722–1732. PMLR.

Quintero-Pena, C., Kingston, Z., Pan, T., Shome, R., Kyrillidis, A., and Kavraki, L. E. (2023). Optimal Grasps and Placements for Task and Motion Planning in Clutter. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 3707–3713.

Rabanser, S., Shchur, O., and Günnemann, S. (2017). Introduction to tensor decompositions and their applications in machine learning. *arXiv:1711.10781*, pages 1–13.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Rakita, D., Mutlu, B., and Gleicher, M. (2018). Relaxedik: Real-time synthesis of accurate and feasible robot arm motion. In *Proc. Robotics: Science and Systems (R:SS)*, volume 14, pages 26–30.

Ramos, F., Possas, R. C., and Fox, D. (2019). Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. In *Proc. Robotics: Science and Systems (RSS)*.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

Rubanova, Y., Chen, R. T., and Duvenaud, D. K. (2019). Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems (NIPS)*, 32.

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190.

Savostyanov, D. V. and Oseledets, I. V. (2011). Fast adaptive interpolation of multi-dimensional arrays in tensor train format. *The 2011 International Workshop on Multidimensional (nD) Systems*, pages 1–8.

Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shah, D., Xu, P., Lu, Y., Xiao, T., Toshev, A. T., Levine, S., et al. (2021). Value Function Spaces: Skill-Centric State Abstractions for Long-Horizon Reasoning. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Shetty, S., Lembono, T., Löw, T., and Calinon, S. (2024a). Tensor Train for Global Optimization Problems in Robotics. *International Journal of Robotics Research (IJRR)*, 43(6):811–839.

Shetty, S., Lembono, T., Löw, T., and Calinon, S. (2024b). Tensor train for global optimization problems in robotics. *International Journal of Robotics Research (IJRR)*, 43(6):811–839.

Shetty, S., Xue, T., and Calinon, S. (2024c). Generalized policy iteration using tensor approximation for hybrid control. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Shetty, S. N. (2024). *Robot Learning using Tensor Networks*. PhD thesis, EPFL, Lausanne.

Shtessel, Y., Edwards, C., Fridman, L., and Levant, A. (2014). *Sliding mode control and observation*, volume 10. Springer.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Simeonov, A., Du, Y., Kim, B., Hogan, F., Tenenbaum, J., Agrawal, P., and Rodriguez, A. (2021). A long horizon planning framework for manipulating rigid pointcloud objects. In *Conference on Robot Learning*, pages 1582–1601. PMLR.

Sochopoulos, A., Gienger, M., and Vijayakumar, S. (2024). Learning deep dynamical systems using stable neural odes. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 11163–11170.

Song, J., Meng, C., and Ermon, S. (2021a). Denoising diffusion implicit models. In *Proc. Intl Conf. on Learning Representations (ICLR)*.

Song, Y., Dhariwal, P., Chen, M., and Sutskever, I. (2023a). Consistency models. In *Proc. Intl Conf. on Machine Learning (ICML)*.

Song, Y., Romero, A., Müller, M., Koltun, V., and Scaramuzza, D. (2023b). Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2021b). Score-based generative modeling through stochastic differential equations. *Proc. Intl Conf. on Learning Representations (ICLR)*.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

Stoudenmire, E. and Schwab, D. J. (2016). Supervised learning with tensor networks. *Advances in neural information processing systems*, 29.

Suh, H. J. T. (2025). *Leveraging Structure for Efficient and Dexterous Contact-Rich Manipulation*. PhD thesis, Massachusetts Institute of Technology.

Suh, H. T., Pang, T., Zhao, T., and Tedrake, R. (2025). Dexterous contact-rich manipulation via the contact trust region. *International Journal of Robotics Research (IJRR)*, page 02783649251398875.

Suresh, S., Qi, H., Wu, T., Fan, T., Pineda, L., Lambeta, M., Malik, J., Kalakrishnan, M., Calandra, R., Kaess, M., et al. (2024). Neuralfeels with neural fields: Visuotactile perception for in-hand manipulation. *Science Robotics*, 9(96):eadl0628.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Tal, E., Gorodetsky, A., and Karaman, S. (2018). Continuous tensor train-based dynamic programming for high-dimensional zero-sum differential games. In *2018 Annual American Control Conference (ACC)*, pages 6086–6093. IEEE.

Tao, Y., Chiaramonte, M., and Fernandez, P. (2025). Interpolated adaptive linear reduced order modeling for deformation dynamics. *arXiv preprint arXiv:2509.25392*.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 23–30.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033.

Torabi, F., Warnell, G., and Stone, P. (2018). Behavioral cloning from observation. In *Proc. Intl Joint Conf. on Artificial Intelligence (IJCAI)*, pages 4950–4957.

Toussaint, M. (2015). Logic-geometric programming: an optimization-based approach to combined task and motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1930–1936.

Toussaint, M., Allen, K. R., Smith, K. A., and Tenenbaum, J. B. (2018). Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. In *Proc. of Robotics: Science and Systems (R:SS). Best Paper Award.*

Toussaint, M., Ha, J.-S., and Driess, D. (2020). Describing physics for physical reasoning: Force-based sequential manipulation planning. *IEEE Robotics and Automation Letters (RA-L)*, 5(4):6209–6216.

Toussaint, M., Harris, J., Ha, J.-S., Driess, D., and Hönig, W. (2022). Sequence-of-constraints mpc: Reactive timing-optimal control of sequential manipulation. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 13753–13760.

Tucker, L. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems (NIPS)*, 30.

Viña Barrientos, F., Karayiannidis, Y., Smith, C., and Kragic, D. (2016). Adaptive control for pivoting with visual and tactile feedback. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA).*

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272.

Wang, M., Pan, Y., Xu, Z., Li, G., Yang, X., Mandic, D., and Cichocki, A. (2023a). Tensor networks meet neural networks: A survey and future perspectives. *arXiv preprint arXiv:2302.09019.*

Wang, Y., Praveena, P., Rakita, D., and Gleicher, M. (2023b). Rangedik: An optimization-based robot motion generation method for ranged-goal tasks. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 9700–9706.

Wang, Z. and Grant, M. J. (2017). Constrained trajectory optimization for planetary entry via sequential convex programming. *Journal of Guidance, Control, and Dynamics*, 40(10):2603–2615.

Wang, Z., Hunt, J. J., and Zhou, M. (2022). Diffusion policies as an expressive policy class for offline reinforcement learning. In *Proc. Intl Conf. on Learning Representations (ICLR).*

Werbos, P. (1992). Approximate dynamic programming for real-time control and neural modeling. *Handbook of intelligent control.*

Williams, B. C. (2016). Cognitive Robotics: Monte Carlo Tree Search. Cambridge MA. MIT OpenCourseWare.

Wu, B., Nair, S., Fei-Fei, L., and Finn, C. (2021). Example-Driven Model-Based Reinforcement Learning for Solving Long-Horizon Visuomotor Tasks. In *5th Annual Conference on Robot Learning*.

Xu, D., Mandlekar, A., Martín-Martín, R., Zhu, Y., Savarese, S., and Fei-Fei, L. (2021). Deep affordance foresight: Planning through what can be done in the future. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6206–6213. IEEE.

Xue, H., Ren, J., Chen, W., Zhang, G., Fang, Y., Gu, G., Xu, H., and Lu, C. (2025a). Reactive diffusion policy: Slow-fast visual-tactile policy learning for contact-rich manipulation. In *Proc. Robotics: Science and Systems (R:SS)*.

Xue, T., Girgin, H., Lembono, T., and Calinon, S. (2023a). Demonstration-guided optimal control for long-term non-prehensile planar manipulation. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 4999–5005.

Xue, T., Girgin, H., Lembono, T. S., and Calinon, S. (2023b). Demonstration-guided optimal control for long-term non-prehensile planar manipulation. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 4999–5005.

Xue, T., Razmjoo, A., Shetty, S., and Calinon, S. (2024a). Logic-skill programming: An optimization-based approach to sequential skill planning.

Xue, T., Razmjoo, A., Shetty, S., and Calinon, S. (2024b). Robust manipulation primitive learning via domain contraction. In *Proc. Conference on Robot Learning (CoRL)*.

Xue, T., Razmjoo, A., Shetty, S., and Calinon, S. (2025b). Robust contact-rich manipulation through implicit motor adaptation. *International Journal of Robotics Research (IJRR)*.

Xue, T., Rigo, A., Huang, B., Shen, J., Xu, Z., Colonnese, N., and Memar, A. (2026). Tube diffusion policy: reactive visual-tactile policy learning for contact-rich manipulation. *under review*.

Xue, T., Zhang, Y., Razmjoo, A., and Calinon, S. (2025c). Monte carlo tree search with tensor factorization for robot optimization. *under review, available as arXiv:2507.04949*.

Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., and Yang, M.-H. (2023a). Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4):1–39.

Yang, Y., Krompass, D., and Tresp, V. (2017). Tensor-train recurrent neural networks for video classification. In *International Conference on Machine Learning*, pages 3891–3900. PMLR.

Yang, Z., Garrett, C. R., Lozano-Pérez, T., Kaelbling, L., and Fox, D. (2023b). Sequence-based plan feasibility prediction for efficient task and motion planning. In *Proc. Robotics: Science and Systems (R:SS)*, Daegu, Republic of Korea.

Yu, W., Tan, J., Liu, C. K., and Turk, G. (2017). Preparing for the unknown: Learning a universal policy with online system identification. In *Proc. Robotics: Science and Systems (RSS)*.

Yuan, W., Zhu, C., Owens, A., Srinivasan, M. A., and Adelson, E. H. (2017). Shape-independent hardness estimation using deep learning and a gelsight tactile sensor. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pages 951–958.

Ze, Y., Zhang, G., Zhang, K., Hu, C., Wang, M., and Xu, H. (2024). 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Proc. Robotics: Science and Systems (R:SS)*.

Zeilinger, M. N., Morari, M., and Jones, C. N. (2014). Soft constrained model predictive control with robust stability guarantees. *IEEE Transactions on Automatic Control*, 59(5):1190–1202.

Zhang, F. and Demiris, Y. (2023). Visual-tactile learning of garment unfolding for robot-assisted dressing. *IEEE Robotics and Automation Letters (RA-L)*, 8(9):5512–5519.

Zhang, F. and Gienger, M. (2024). Affordance-based robot manipulation with flow matching. *arXiv preprint arXiv:2409.01083*.

Zhao, T., Kumar, V., Levine, S., and Finn, C. (2023). Learning fine-grained bimanual manipulation with low-cost hardware. In *Proc. Robotics: Science and Systems (R:SS)*.

Zhou, K. and Doyle, J. C. (1998). *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ.

Zhou, X., Qiao, Y., Xu, Z., Wang, T.-H., Chen, Z., Zheng, J., Xiong, Z., Wang, Y., Zhang, M., Ma, P., Wang, Y., Dou, Z., Kim, B., Tian, Y., Chen, Y., Qiu, X., Lin, C., He, T., Si, Z., Zhang, Y., Yang, Z., Liu, T., Li, T., Yamazaki, K., Zhang, H., Ha, H., Zhang, Y., Liu, M., Zheng, S., Fu, Z., Wu, Q., Geng, Y., Chen, F., Hu, Y., Shi, G., Liu, L., Komura, T., Erickson, Z., Held, D., Li, M., Fan (Jim), L., Zhu, Y., Matusik, W., Gutfreund, D., Song, S., Rus, D., Lin, M., Zhu, B., Fragkiadaki, K., and Gan, C. (2024). Genesis: A generative and universal physics engine for robotics and beyond.

Zhou, Y., Barnes, C., Lu, J., Yang, J., and Li, H. (2019). On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5745–5753.

Zhu, H., Meduri, A., and Righetti, L. (2023). Efficient object manipulation planning with monte carlo tree search. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 10628–10635.

Zhu, Y., Joshi, A., Stone, P., and Zhu, Y. (2022). Viola: Imitation learning for vision-based manipulation with object proposal priors. *arXiv preprint arXiv:2210.11339*.

Zimmermann, S., Hakimifard, G., Zamora, M., Poranne, R., and Coros, S. (2020). A multi-level optimization framework for simultaneous grasping and motion planning. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):2966–2972.

Zong, Z., Li, X., Li, M., Chiaramonte, M. M., Matusik, W., Grinspun, E., Carlberg, K., Jiang, C., and Chen, P. Y. (2023). Neural stress fields for reduced-order elastoplasticity and fracture. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11.

Teng Xue

Research Assistant, École Polytechnique Fédérale de Lausanne (EPFL)/Idiap Research Institute
teng.xue@epfl.ch — +41 76 576-2763 — Linkedin — Personal Webpage: https://schortenger.github.io/

RESEARCH INTERESTS

Dexterous Manipulation, Visual-Tactile Imitation Learning, Task and Motion Planning, Optimal Control

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL) Nov. 2021 — May 2026 (Expected)
Ph.D. in Electrical Engineering
Thesis Topic: Contact-rich Manipulation with Tensor Networks
Thesis Director: Dr. Sylvain Calinon, Prof. Amir Zamir

ETH Zurich Oct. 2019 — Mar. 2020
Visiting Student, Robotic Systems Lab (RSL)
Semester Project: Learning-based Pose Estimation and Control of Festo BionicSofthand
Supervisors: Prof. Dr. Marco Hutter, Dr. David Hoeller, Dr. Martin Wermelinger

Shanghai Jiao Tong University Sep. 2017 — Dec. 2020
M.S. in Mechanical Engineering
GPA: 3.73/4.0 (90/100)
Thesis Title: Stable Robot Grasping with Vision and Tactile Priors
Supervisor: Prof. Weiming Wang

Nanjing University of Aeronautics and Astronautics Sep. 2013 — Jul. 2017
B.S. in Mechanical Engineering (Changkong Honors College)
GPA: 4.2/5.0 (92/100)
Thesis Title: Development of a Recirculating Friction-Driven Skateboard System for Car Assembly
Supervisor: Prof. Peihuang Lou

EXPERIENCE

Meta Reality Labs Research **London, UK**
Research Scientist Intern Sep. 2025 — Feb. 2026

- Designed and developed robotic data collection pipelines for dexterous manipulation tasks.
- Implemented algorithms to train multimodal (vision and tactile) control policies using generative models, including diffusion and flow matching.
- Developed a novel reactive visual–tactile imitation learning method that improves diffusion policy, achieving higher success rates and enabling high-frequency feedback control.

Idiap Research Institute **Martigny, Switzerland**
Research Assistant, Robot Learning and Interaction Group Nov. 2021 — Present

- Developed algorithms to combine symbolic AI and geometric motion planning for long-horizon dexterous manipulation.
- Investigated fast and memory-efficient algorithm for contact-rich policy learning based on tensor factorization.

Flexiv Robotics Inc. **Shanghai, China**
Research Intern Mar. 2021 — Aug. 2021

- Applied deep reinforcement learning for peg-in-hole task.

Stanford Artificial Intelligence laboratory (SAIL), Stanford University **Stanford, CA**
Research Intern May. 2020 — Oct. 2020

- Developed in-hand manipulation simulator for Roller Grasper and applied model-free reinforcement learning for control policy learning.
- Developed universal policy learning through behavior cloning.

Shenzhen DJI Innovation and Technology Co., Ltd **Shenzhen, China**
Mechanical Engineer Intern Jul. 2016 — Aug. 2016

- Designed and fabricated a lightweight gripper using carbon fiber for UAV grasping.

PUBLICATIONS

UNDER REVIEW

- **T. Xue**, Y. Zhang, A. Razmjoo, and S. Calinon. **Monte Carlo Tree Search with Tensor Factorization for Robot Optimization**, submitted to *International Journal of Robotics Research (IJRR)*.

- **T. Xue**, A. Rigo, B. Huang, J. Shen, Z. Xu, N. Colonnese and A. Memar. **Tube Diffusion Policy: reactive Visual-Tactile Policy Learning for Contact-Rich Manipulation**, submitted to *IEEE Transactions on Robotics (T-RO)*.

JOURNALS

- **T. Xue**, A. Razmjoo, S. Shetty, and S. Calinon. (2025). **Robust Contact-rich Manipulation through Implicit Motor Adaptation**. International Journal of Robotics Research (IJRR).

- A. Razmjoo, **T. Xue**, S. Shetty, and S. Calinon. **Sampling-Based Constrained Motion Planning with Products of Experts**. International Journal of Robotics Research (IJRR).

- Y. Zhang, **T. Xue**, A. Razmjoo, and S. Calinon. (2025). **Learning Problem Decomposition for Efficient Sequential Multi-object Manipulation Planning**. IEEE Robotics and Automation Letters (RA-L)

- S. Yuan, L. Shao, Y. Feng, J. Sun, **T. Xue**, C. Yako, J. Bohg, K. Salisbury. (2024). **Design and Control of Roller Grasper V3 for In-Hand Manipulation**. IEEE Transactions on Robotics (T-RO), 40, 4222-4234.

- Y. Zhang, **T. Xue***, A. Razmjoo*, and S. Calinon. (2024). **Logic Dynamic Movement Primitives for Long-horizon Manipulation Tasks in Dynamic Environments**. IEEE Robotics and Automation Letters (RA-L), 9:8, 7214-7221.

- W. Liu, W. Wang, Y. You, **T. Xue**, Z. Pan, J. Qi, J. Hu. (2022). **Robotic Picking in Dense Clutter via Domain Invariant Learning from Synthetic Dense Cluttered Rendering**. Robotics and Autonomous Systems, 147, 103901.

- **T. Xue**, W. Wang, J. Ma, W. Liu, Z. Pan, M. Han. (2020). **Progress and Prospects of Multimodal Fusion Methods in Physical Human–Robot Interaction: A Review**. IEEE Sensors Journal, 20:18, 10355-10370.

CONFERENCES

- X. Chi, H. Girgin, T. Löw, Y. Xie, **T. Xue**, J. Huang, C. Hu, Z. Liu and S. Calinon. (2025) **Efficient and Real-Time Motion Planning for Robotics Using Projection-Based Optimization**. In Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS).

- **T. Xue**, A. Razmjoo, S. Shetty, and S. Calinon. (2024). **Robust Manipulation Primitive Learning via Domain Contraction**. In Proc. of Conference on Robot Learning (CoRL).

- **T. Xue**, A. Razmjoo, S. Shetty, and S. Calinon. (2024). **Logic-Skill Programming: An Optimization-based Approach to Sequential Skill Planning**. In Proc. of Robotics: Science and Systems (RSS).

- **T. Xue**, A. Razmjoo, and S. Calinon. (2024). **D-LGP: Dynamic Logic-Geometric Program for Combined Task and Motion Planning**. In Proc. IEEE Intl Conf. on Robotics and Automation (ICRA), pp. 14888-14894.

- S. Shetty, **T. Xue**, and S. Calinon. (2024). **Generalized Policy Iteration using Tensor Approximation for Hybrid Control**. In Proc. Intl Conf. on Learning Representations (ICLR). (Spotlight)

- **T. Xue**, H. Girgin, T. Lembono, and S. Calinon. (2023). **Demonstration-guided Optimal Control for Long-term Non-prehensile Planar Manipulation**. In Proc. IEEE Intl Conf. on Robotics and Automation (ICRA), pp. 4999–5005.

WORKSHOPS

- **T. Xue**, A. Razmjoo, S. Shetty, and S. Calinon. (2024). **Learning Contact-rich Abstractions using Tensor Factorization**. Learning Effective Abstractions for Planning. Workshop at Conference on Robot Learning (CoRL).

- **T. Xue**, A. Razmjoo, S. Shetty, and S. Calinon. (2024). **Logic-Geometric Planning and Control Using Graph of Tensor Networks**. Frontiers of optimization for robotics. Workshop at Robotics: Science and Systems (RSS).

- **T. Xue***, S. Shetty*, and S. Calinon. (2023). **Dynamic Programming using Tensor Approximation for Contact-rich Manipulation**. Embracing Contacts. Workshop at IEEE Intl Conf. on Robotics and Automation (ICRA).

ACADEMIC SERVICE

Workshop Organizer

- Workshop on Manipulation Robustness: Towards Human-Level Robustness under Real-World Challenges, ICRA 2026

Reviewer

- **Journal**: IEEE Transactions on Robotics (T-RO), IEEE Robotics and Automation Letters (RA-L)
- **Conference**: IEEE International Conference on Robotics and Automation (ICRA), International Conference on Automated Planning and Scheduling (ICAPS)

AWARDS

- **Fist Place**, ICRA - Tidy Up My Room Challenge, 2018
- **Outstanding Winner (1/8085)**, The 2017 Mathematics Contest in Modeling held by American Consortium for Mathematics and Its Application (COMAP), 2017
- **Runner-up**, Robomaster-DJI Summer Camp Robot Challenge, 2016;
- **First Prize**, The 6th National Mathematics Contest, 2014
- **National Scholarship (Top 1%)**, 2014 & 2018
- **Outstanding Volunteer**, Youth Olympic Games (International Olympic Committee), 2014

Extracurricular and Social Activities

Vice President, Graduate Student Union in School of Mechanical Engineering Jun. 2018 — Jun. 2019

- Organizing educational and social events catering to 2500 students enrolled in the School of Mechanical Engineering.
- Communicating and collaborating with other student associates.

SKILLS

- **Programming:** Python, MATLAB, ROS, LaTeX, Linux, PDDL
- **Softwares:** Pybullet, Mujoco, IsaacGym, Crocoddyl, Pytorch, OpenCV, CasADi, CATIA, Solidworks, AutoCAD
- **Languages:** Chinese (Native), English (Fluent), French (Beginner)