EPFL

Thèse n° 10 273

Robot Learning using Tensor Networks

Présentée le 14 juin 2024

Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Suhan Narayana SHETTY

Acceptée sur proposition du jury

Prof. A. Ijspeert, président du jury
Prof. D. Gatica-Perez, Dr S. Calinon, directeurs de thèse
Dr P.-B. Wieber, rapporteur
Prof. R. Pajarola, rapporteur
Prof. D. Kressner, rapporteur

École
polytechnique
fédérale
de Lausanne

2024

To my mother and sister,
for their boundless love and the sacrifices they have made.

Abstract

In various robotics applications, the selection of function approximation methods greatly influences the feasibility and computational efficiency of algorithms. Tensor Networks (TNs), also referred to as tensor decomposition techniques, present a versatile approach for approximating functions involving continuous variables, discrete variables, or combinations of these variable types. Apart from their approximation capabilities, TNs offer efficient methods for conducting algebraic operations, calculus, probability modeling, and optimization, which are particularly essential in robotics applications. This thesis highlights the importance of a specific TN known as Tensor Train (TT) for function approximation in robotics by addressing a diverse range of previously challenging problems. Initially, utilizing TT, the thesis enhances the scalability and deployability of an ergodic exploration algorithm commonly employed in robotic exploration. Subsequently, the thesis introduces a novel numerical optimization algorithm named Tensor Train for Global Optimization (TTGO) to determine the optima of functions represented in TT format. Given that numerous robotics tasks are framed as numerical optimization problems, TTGO provides efficient solutions to several optimization-based problems in robotics, including inverse kinematics with obstacles, motion planning, and policy learning, as demonstrated in the thesis. In summary, this thesis underscores the promising potential of TNs as valuable tools in the field of robotics.

Keywords: Tensor Networks, Tensor Train, Matrix Product States, Tensor Decomposition, Ergodic Exploration, Global Optimization, Density Modeling, Inverse Kinematics, Motion Planning, Optimal Control, Dynamic programming, Imitation Learning, Learning from Demonstration, Reinforcement Learning, Peg-in-hole Insertion, Hybrid Control.

Résumé

Dans diverses applications robotiques, la sélection des méthodes d'approximation des fonctions influence grandement la faisabilité et l'efficacité informatique des algorithmes. Les Tensor Networks (TN), également appelés techniques de décomposition tensorielle, présentent une approche polyvalente pour approximer des fonctions impliquant des variables continues, des variables discrètes ou des combinaisons de ces types de variables. Outre leurs capacités d'approximation, les TN offrent des méthodes efficaces pour effectuer des opérations algébriques, des calculs, des modélisations probabilistes et des optimisations, particulièrement essentielles dans les applications robotiques. Cette thèse met en évidence l'importance d'un TN spécifique connu sous le nom de Tensor Train (TT) pour l'approximation de fonctions en robotique, en abordant un large éventail de problèmes auparavant difficiles. Initialement, en utilisant TT, la thèse améliore l'évolutivité et la déployabilité d'un algorithme d'exploration ergodique couramment utilisé en robotique. Par la suite, la thèse introduit un nouvel algorithme d'optimisation numérique nommé Tensor Train for Global Optimization (TTGO) pour déterminer les optima des fonctions représentées dans le format TT. Étant donné que de nombreux problèmes de robotique sont présentés comme des défis d'optimisation numérique, TTGO fournit des solutions efficaces à plusieurs problèmes basés sur l'optimisation, notamment la cinématique inverse avec obstacles, la planification de mouvement et l'apprentissage de politiques, comme le démontre cette thèse. En résumé, cette thèse souligne le potentiel prometteur des TN en tant qu'outils précieux dans le domaine de la robotique.

Mots-clés : Tensor Network, Tensor Train, Matrix Product State, Décomposition tensorielle, Exploration ergodique, Optimisation globale, Modélisation de distributions, Cinématique inverse, Planification de mouvement, Contrôle optimal, Programmation dynamique, Apprentissage par imitation, Apprentissage par démonstration, Apprentissage par renforcement, Tâches d'insertion, Contrôle hybride.

Acknowledgements

The journey toward the PhD thesis has spanned a considerable portion of my life thus far, influencing me both professionally and personally. I express gratitude to my colleagues whose influence directly shaped this thesis. Additionally, I am thankful for the support of friends and family, as well as the beautiful landscapes of Switzerland, which had a role in shaping my character and thus indirectly aided me in joyfully completing this thesis.

First and foremost, I extend my gratitude to my thesis advisors, Dr Sylvain Calinon and Prof. Daniel Gatica-Perez, for granting me the opportunity to pursue a PhD jointly with the Idiap Research Institute and EPFL. I appreciate Sylvain's unwavering support, the flexibility he allowed in exploring various research directions, and his invaluable guidance. His enthusiasm for my research served as a constant source of motivation for me to strive for excellence in my work.

I wish to express my thanks to the jury members of my thesis: Prof. Auke Ijspeert, Prof. Daniel Kressner, Prof. Renato Pajarola, and Dr Pierre-Brice Wieber, for dedicating their time to review my thesis and for their constructive feedback. I am also grateful to the Idiap secretariats and administrative staff for facilitating my stay in Switzerland and for fostering a conducive research environment.

My gratitude also goes to Disney Research, Zurich, particularly my mentors Dr Moritz Bächer and Dr Ruben Grandia, for giving me an internship opportunity to work on an exciting project on legged robots.

This thesis has been shaped by numerous enjoyable discussions with my colleagues at the RLI group held over tea, during lunch breaks, or in group meetings. I am particularly grateful to Joao, Teguh, Teng, and Tobias for their invaluable contributions and memorable collaborations, without which this thesis would not have been possible. I also want to express my appreciation to other members of the RLI group, including Amir, Emmanuel, Boyang, Julius, and Martin, for their presence and support. In particular, I owe a special thank you to Martin for his friendship and support during the initial year of my PhD journey.

Acknowledgements

Reflecting on my PhD journey, I am filled with a warm nostalgia for the cherished moments shared with my friends at Idiap. Whether it was hiking adventures, board game evenings, cooking sessions, or engaging conversations over a drink, each memory brings a smile to my face. The presence of Sargam, Amir, Zohreh, Suraj, Rudy, Apoorv, Anshul, Enno, Skanda, Dhananjay, Teja, Eklavya, Bogdan, Laurent, Fabio, Anshul, and all the others, whose names I may have unintentionally omitted, filled my days at Idiap with excitement and anticipation. Special mention goes to my dear friends Parvaneh and Tilak, my trusted confidants. I am grateful for Parvaneh's steadfast friendship, which has supported me through the highs and lows of my PhD journey.

I extend my gratitude to my friends in Lausanne: Julian, Pablo, and Neha, for the wonderful times we shared. Special thanks to Pablo and Neha for their care and the delicious meals they prepared in our shared flat; you truly made me feel at home. I feel lucky to have shared both the office and living space with my dear friend Pablo and see his smile every day.

I feel incredibly fortunate to have met Daniel and Monique right at the beginning of my PhD journey; they have since become like family to me. I am deeply grateful to my Sahajmarg community for their Satsangh and for introducing me to such remarkable individuals. I extend my heartfelt appreciation for the picnics, soulful hikes, and meditation sessions we have experienced together. Additionally, I would like to express my gratitude to Delara and my Ashtanga Yoga Montreux community. When I reflect on the middle phase of my PhD journey, it is your warm faces that immediately come to mind. Thank you for your love and support.

Just as the strength of a tree resides in its roots, my mother, Sulochana, and my sister, Sajna, have served as the steadfast roots of my life's journey. Their boundless love and selfless sacrifices have enabled me to ascend the academic ladder and be more humane. I am also indebted to my brother-in-law, Pradeep, my uncle, Karunakar Shetty, and the rest of my family for their unwavering support. The wisdom and values they all have instilled in me empower me to stand firm and grow through the trials of life, much like a tree grows through the changing seasons.

Lausanne, February 23, 2024 Suhan Shetty

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Function approximation is a pivotal element in numerous robotics algorithms, playing a crucial role in algorithms for control, exploration, motion planning, imitation learning, and reinforcement learning. The choice of function approximation significantly influences algorithm performance. These algorithms often involve functions with continuous variables, discrete variables, or a combination of both. When dealing with discrete variables over a rectangular domain, the function is referred to as a tensor or multidimensional array.

Beyond storage efficiency and expressibility, computational efficiency is an important consideration for function approximation techniques in robotics as we often need to perform various algebraic operations and calculus including multivariate integration, differentiation, computing the mean, volume, etc. As robotics tasks often involve decision-making or policy learning, they are typically formulated as optimization problems. For example, inverse kinematics (IK) and motion planning tasks are framed as the minimization of a cost function, and optimal control is defined as the minimization of a cost-to-go function. Therefore, in addition to the computational efficiency, it is beneficial if the function approximation technique facilitates optimization. i.e., we want finding the optima would be easier with the approximated function. If we are dealing with probability models, it would be desirable if the approximation model allows fast ways to sample from the distribution.

Popular function approximation techniques in robotics include Gaussian Mixture Models (GMMs) and Neural Networks (NNs). GMMs facilitate algebraic operations and optimization but lack scalability. In contrast, NNs offer scalability but are less adept at multivariate calculus operations. They excel in data-driven settings but are inefficient in exploiting the prior knowledge of the functions being modeled and do not facilitate optimization. For example, in solving IK problems and motion planning problems involving the minimization of cost function and optimal control involving minimization of a cost-to-go function, we know a priori the procedure that returns the values of these functions (i.e., full knowledge of the target function).

In this thesis, we demonstrate that Tensor Networks (TNs), particularly a specific type known as Tensor Train (TT), emerge as a promising solution, addressing these challenges in many applications in robotics. They can handle mixtures of continuous and discrete variables, and allow efficient algebraic operations, calculus, and optimization. If we are dealing with density modeling, they allow efficient ways to sample from the approximate model. If we have the full knowledge of the function being approximated, in addition to data-driven approaches, there exist efficient approaches like cross-approximation to

exploit this information to model the target functions in TT format.

The research done in this thesis builds upon the extensive knowledge within the physics community where it was originally used, particularly for solving partial differential equations and in quantum computing [1] [2]. With a growing presence in signal processing [3] and recent applications in machine learning [4], computer graphics [5] and control theory [6][7], TNs showcase their adaptability and effectiveness. Our emphasis is on demonstrating the practical relevance of tensor networks in a diverse set of applications, spanning exploration, inverse kinematics, motion planning, and policy learning. By showcasing their versatility in these contexts, we aim to contribute valuable insights and methodologies, solidifying the position of TNs as powerful tools in advancing robotics.

1.2 Thesis Outline

In Chapter 2, we furnish the essential groundwork on TNs, with a specific focus on the TT. We illustrate the applicability of TT in modeling functions and delve into the details of how it enables efficient algebraic and calculus operations.

In Chapter 3, we introduce the ergodic exploration algorithm, addressing the historical challenge of the curse of dimensionality in high-dimensional exploration tasks. Our contribution lies in demonstrating the effectiveness of our solution, enabling the ergodic exploration algorithm to operate in a closed-loop manner. This paves the way for practical applications, exemplified by the successful application of the algorithm to a complex task—specifically, the 6D exploration required for peg-in-hole insertion.

In Chapter 4, we introduce Tensor Train for Global Optimization (TTGO), a novel technique enabling the identification of optima of functions in TT format. This significantly enhances the relevance of TT in robotics, particularly in optimization-based problems such as inverse kinematics and motion planning. Our approach demonstrates the capability to obtain both global and multiple optima.

Chapter 5 leverages the TTGO framework to propose an efficient Approximate Dynamic Programming (ADP) algorithm for optimal control synthesis in nonlinear control systems. The policy learning framework accommodates hybrid state and action spaces without specific assumptions on the dynamic model, distinguishing it from existing ADP algorithms. We also showcase its significance for policy learning paradigms like imitation learning and Reinforcement Learning (RL).

The thesis concludes in Chapter 6 where we summarize our findings. The possible extensions provided in each of the chapters in this thesis in addition to the future

directions outlined in Chapter 6, offer a comprehensive perspective on the significance of tensor networks in robotics.

1.3 Publications

List of publications by the author during the doctorate study:

- S. Shetty, J. Silvério, and S. Calinon, "Ergodic exploration using tensor train: Applications in insertion tasks," *IEEE Trans. on Robotics*, vol. 38, no. 2, pp. 906–921, 2022
- S. Shetty, T. Lembono, T. Löw, and S. Calinon, "Tensor trains for global optimization problems in robotics," *International Journal of Robotics Research (IJRR)*, 2023
- S. Shetty, T. Xue, and S. Calinon, "Generalized policy iteration using tensor approximation for hybrid control," in *International Conference on Learning Representations (ICLR)*, 2024, (spotlight paper, 5% acceptance rate)
- L. Brudermüller, T. Lembono, S. Shetty, and S. Calinon, "Trajectory prediction with compressed 3d environment representation using tensor train decomposition," in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, 2021, pp. 633–639
- B. Nemec, M. M. Hrovat, M. Simonič, S. Shetty, S. Calinon, and A. Ude, "Robust execution of assembly policies using a pose invariant task representation," in *2023 20th International Conference on Ubiquitous Robots (UR)*, 2023, pp. 779–786

2 Background

In this chapter, we begin by offering intuitive insights into function approximation using the Tensor Networks (TNs). We draw parallels with the familiar concept of 2D function approximation through matrix factorization, as elucidated in Section 2.1.1 and 2.1.2. To lay the groundwork, we elucidate the concept of tensors in Section 2.2 and delve into the application of tensors as a discrete analogue for approximating multivariate functions in Section 2.3.

Moving forward, in Section 2.4 we briefly introduce TNs and describe how it effectively addresses the curse of dimensionality associated with tensors. In Section 2.5, we describe a particular TN called Tensor Train (TT) in more detail as it is the TN considered in this thesis. Section 2.6 outlines various algebraic operations applicable to TT models, commonly employed in practical applications. We explore the TT cross approximation in Section 2.8, a pivotal algorithm employed extensively in this thesis to represent a given function in TT format.

In Section 2.9.1, we extend our discussion to conceptualize any TT model as a probability distribution. Finally, in Section 2.10, we elaborate on the application of TT models in data-driven supervised and unsupervised learning, including density estimation.

2.1 Function Approximation using Matrix Factorization

We begin by introducing the familiar concept of matrix factorization, demonstrating its application in representing 2D functions in a separable manner, i.e., as a sum-of-product of univariate functions. This understanding will serve as a precursor to the subsequent generalization of such principles for high-dimensional functions using the TNs.

2.1.1 Separation of Variables using Matrix Factorization

Consider a continuous 2D function:

$$P(x_1,\ x_2)\colon \Omega_{\boldsymbol{x}} \subset \mathbb{R}^2 \to \mathbb{R}. \tag{2.1}$$

Let $\Omega_{\boldsymbol{x}} = \Omega_{x_1} \times \Omega_{x_2}$ be the rectangular domain formed by the Cartesian product of intervals, such that $x_1 \in \Omega_{x_1}$ and $x_2 \in \Omega_{x_2}$, and $\boldsymbol{x} = (x_1,\ x_2)$. We derive a discrete analogue $\boldsymbol{P}$ of this function, representing it as a matrix in the 2D case, by evaluating the function on a grid-like discretization of the domain $\Omega_{\boldsymbol{x}}$. Discretizing the intervals Ω_{x_1} and Ω_{x_2} with n_1 and n_2 points, respectively, and denoting the corresponding discretization points as $(x_1^1, \ldots, x_1^{n_1})$ and $(x_2^1, \ldots, x_2^{n_2})$, we define the discretization set $\mathcal{X}$ containing $\{(x_1^{i_1},\ x_2^{i_2})\colon\ i_k \in \{1, \ldots, n_k\},\ k \in \{1, 2\}\}$ and the corresponding index set $\mathcal{I}_{\mathcal{X}}$ containing $\{(i_1,\ i_2)\colon\ i_k \in \{1, \ldots, n_k\},\ k \in \{1, 2\}\}$. The discrete analogue of the function is then given by the matrix:

$$\boldsymbol{P}_{i_1,\,i_2} = P(x_1^{i_1},\ x_2^{i_2}), \quad \forall (i_1,\ i_2) \in \mathcal{I}_{\mathcal{X}}. \tag{2.2}$$

We proceed to find a factorization of the matrix $\boldsymbol{P}$ into two factors $(\boldsymbol{P}^1,\ \boldsymbol{P}^2)$, with $\boldsymbol{P}^1 \in \mathbb{R}^{n_1 \times r}$ and $\boldsymbol{P}^2 \in \mathbb{R}^{r \times n_2}$, such that the elements of $\boldsymbol{P}$ can be approximated as:

$$\boldsymbol{P}_{i_1,\,i_2} \approx \boldsymbol{P}^1_{i_1,\,:}\, \boldsymbol{P}^2_{:,\,i_2}. \tag{2.3}$$

This matrix factorization, achievable through techniques like QR, SVD, or LU decompositions, offers compact representation advantages, especially when the rank r is low. Furthermore, it allows us to represent the function P in a separable form. While (2.3) is limited to evaluating the function P at discretized points in $\mathcal{X}$, we extend its application to general points $(x_1,\ x_2) \in \Omega_{\boldsymbol{x}}$ using linear interpolation between the rows (or columns), defining vector-valued functions $\boldsymbol{p}^1(x_1)$ and $\boldsymbol{p}^2(x_2)$ as outlined in (2.4) below:

$$\begin{aligned}
\boldsymbol{p}^1(x_1) &= \frac{x_1 - x_1^{i_1}}{x_1^{i_1+1} - x_1^{i_1}}\, \boldsymbol{P}^1_{i_1+1,\,:} + \frac{x_1^{i_1+1} - x_1}{x_1^{i_1+1} - x_1^{i_1}}\, \boldsymbol{P}^1_{i_1,\,:}, \\
\boldsymbol{p}^2(x_2) &= \frac{x_2 - x_2^{i_2}}{x_2^{i_2+1} - x_2^{i_2}}\, \boldsymbol{P}^2_{:,\,i_2+1} + \frac{x_2^{i_2+1} - x_2}{x_2^{i_2+1} - x_2^{i_2}}\, \boldsymbol{P}^2_{:,\,i_2},
\end{aligned} \tag{2.4}$$

where $x_k^{i_k} \leq x_k \leq x_k^{i_k+1}$, $\boldsymbol{p}^1(x_1)\colon \Omega_{\boldsymbol{x}_1} \subset \mathbb{R} \to \mathbb{R}^{1\times r}$ and $\boldsymbol{p}^2(x_2)\colon \Omega_{\boldsymbol{x}_2} \subset \mathbb{R} \to \mathbb{R}^{r\times 1}$. Note that we could also use a spline-based interpolation here.

The approximation for the function P in a separable form is then given by:

$$\begin{aligned} P(x_1,\ x_2) &\approx \boldsymbol{p}^1(x_1)\ \boldsymbol{p}^2(x_2) \\ &= \sum_{j=1}^{r} \boldsymbol{p}_j^1(x_1)\ \boldsymbol{p}_j^2(x_2), \quad \forall (x_1,\ x_2) \in \Omega_{\boldsymbol{x}}. \end{aligned} \tag{2.5}$$

This factorization of multivariate functions as a sum of products of univariate functions proves to be a powerful representation. For instance, integrating the multivariate function becomes computationally efficient through integration of the univariate functions (factors). Moreover, when dealing with probability density functions, such separable representations facilitate efficient sampling procedures, including conditional distribution sampling, as discussed in Section 2.9.2.

In numerous engineering applications, functions often exhibit such separable forms. Moreover, we often have functions characterized by some smoothness improving separability [13]. The degree of separability in the function P correlates with a low-rank structure in the discrete analogue $\boldsymbol{P}$ (indicated by the number of sums in the sum-of-products-of-univariate-functions representation). This suggests a low rank r for the factors, resulting in a reduced number of parameters for representation.

The accuracy of the approximation in (2.5) depends on the number of discretization points and the decomposition technique used to find the factors. For 2D functions, common approaches involve matrix decomposition techniques such as SVD, QR or LU decomposition. However, a standard implementation of these algorithms necessitates computing and storing the entire matrix $\boldsymbol{P}$ in memory, incurring a computational cost of $\mathcal{O}(n_1 n_2)$. If the discretization is fine (i.e., large n_1 and n_2), this can become inefficient. A technique called *cross approximation* (or skeleton decomposition) [14] mitigates this issue, directly finding separable factors without the need to compute and store the entire tensor in memory. In the next section, we briefly delve into the matrix cross approximation technique and its relevant features utilized in this thesis.

2.1.2 Matrix Cross Approximation

Suppose we have a rank-r matrix $\boldsymbol{P} \in \mathbb{R}^{n_1\times n_2}$. Using cross-approximation (a.k.a. CUR decomposition or skeleton decomposition), this matrix can be exactly recovered using r

independent rows (given by the index vector $\boldsymbol{i}_1 \subset \{1, \ldots, n_1\}$) and r independent columns (given by the index vector $\boldsymbol{i}_2 \subset \{1, \ldots, n_2\}$) of the matrix $\boldsymbol{P}$ as

$$\hat{\boldsymbol{P}} = \boldsymbol{P}_{:,\, \boldsymbol{i}_2}\, \boldsymbol{P}^{-1}_{\boldsymbol{i}_1,\, \boldsymbol{i}_2}\, \boldsymbol{P}_{\boldsymbol{i}_1,\, :},$$

provided the intersection matrix $\boldsymbol{P}_{\boldsymbol{i}_1,\, \boldsymbol{i}_2}$ (called submatrix) is non-singular. Thus, the matrix $\boldsymbol{P}$, which has $n_1 n_2$ elements, can be reconstructed using only $(n_1 + n_2 - r)r$ of its elements (see Figure 2.1).

Now suppose we have a noisy version of the matrix $\boldsymbol{P} = \tilde{\boldsymbol{P}} + \boldsymbol{E}$ with $\|\boldsymbol{E}\| < \epsilon$ and $\tilde{\boldsymbol{P}}$ is of low rank. For a sufficiently small ϵ, $\operatorname{rank}(\tilde{\boldsymbol{P}}) = r$ so that the matrix $\boldsymbol{P}$ can be approximated with a lower rank r (i.e., $\operatorname{rank}(\boldsymbol{P}) \approx r$). Then, the choice of the submatrix $\boldsymbol{P}_{\boldsymbol{i}_1,\, \boldsymbol{i}_2}$ (or index vectors $\boldsymbol{i}_1,\ \boldsymbol{i}_2$) for the cross approximation requires several considerations. The maximum volume principle can be used in choosing the submatrix which states that the submatrix with maximum absolute value of the determinant is the optimal choice. If $\boldsymbol{P}_{\boldsymbol{i}_1^*,\, \boldsymbol{i}_2^*}$ is chosen to have the maximum volume, then by skeleton decomposition we have an approximation of the matrix $\boldsymbol{P}$ given by $\hat{\boldsymbol{P}} = \boldsymbol{P}_{:,\, \boldsymbol{i}_2^*}\, \boldsymbol{P}^{-1}_{\boldsymbol{i}_1^*,\, \boldsymbol{i}_2^*}\, \boldsymbol{P}_{\boldsymbol{i}_1^*,\, :}$. This results in a quasi-optimal approximation:

$$\|\boldsymbol{P} - \hat{\boldsymbol{P}}\|_2 < (r+1)^2\, \sigma_{r+1}(\boldsymbol{P}),$$

where $\sigma_{r+1}(\boldsymbol{P})$ is the $(r+1)$-th singular value of $\boldsymbol{P}$ (i.e., the approximation error in the best rank r approximation in the spectral norm) [14]. Thus, we have an upper bound on the error incurred in the approximation which is slightly higher than the best rank r approximation (Eckart–Young–Mirsky theorem).

Finding the maximum volume submatrix is, however, an NP-hard problem. However, many heuristic algorithms that work well exist in practice by using a submatrix with a sufficiently large volume, trading off the approximation accuracy for the computation speed. One of the widely used methods is the MAXVOL algorithm [15] which can provide, given a tall matrix $\boldsymbol{P} \in \mathbb{R}^{r \times n_2}$ (or $\mathbb{R}^{n_1 \times r}$), the maximum volume submatrix $\boldsymbol{P}_{\boldsymbol{i}_1^*,\, \boldsymbol{i}_2^*} \in \mathbb{R}^{r \times r}$. The cross approximation algorithm uses the MAXVOL algorithm in an iterative fashion to find the skeleton decomposition. We describe it below for intuition and refer to [14] for more details:

- *Input*: $\boldsymbol{P} \in \mathbb{R}^{n_1 \times n_2}$, the approximation rank r for the skeleton decomposition.
- Find the columns index set $\boldsymbol{i}_2^*$ and the row index set $\boldsymbol{i}_1^*$ corresponding to the maximum volume submatrix:
  - Randomly choose r columns $\boldsymbol{i}_2$ of the matrix $\boldsymbol{P}$ and repeat the following until convergence:

1. Use MAXVOL to find r row indices $\boldsymbol{i}_1$ so that $\boldsymbol{P}_{\boldsymbol{i}_1,\,\boldsymbol{i}_2}$ is the submatrix with maximum volume in $\boldsymbol{P}_{:,\,\boldsymbol{i}_2}$.
2. Use MAXVOL to find r column indices $\boldsymbol{i}_2$ so that $\boldsymbol{P}_{\boldsymbol{i}_1,\,\boldsymbol{i}_2}$ is the submatrix with maximum volume in $\boldsymbol{P}_{\boldsymbol{i}_1,\,:}$.

- *Output*: Using the column index set $\boldsymbol{i}_2^*$ and the row-index set $\boldsymbol{i}_1^*$ corresponding to the maximum volume submatrix, we have the skeleton decomposition $\hat{\boldsymbol{P}} \approx \boldsymbol{P}_{:,\,\boldsymbol{i}_2^*}\,\boldsymbol{P}^{-1}_{\boldsymbol{i}_1^*,\,\boldsymbol{i}_2^*}\,\boldsymbol{P}_{\boldsymbol{i}_1^*,\,:}$.

In the above algorithm, during the iterations the matrices $\boldsymbol{P}_{:,\,\boldsymbol{i}_2}$ (or $\boldsymbol{P}_{\boldsymbol{i}_1,\,:}$) might be singular. Thus, a more practical implementation uses the pseudo-inverses [14].

Note that, in the above algorithm, the input is only a function to evaluate the elements of the matrix $\boldsymbol{P}$ (i.e., we do not need the whole matrix $\boldsymbol{P}$ in computer memory). Some features of cross approximation algorithms are highlighted below:

- The factors in a cross approximation method consist of elements of the actual data (rows and columns) of the original matrix and hence it improves interpretability. For example, SVD does projection onto the eigenvectors which could be abstract, whereas cross approximation does projection onto the vectors formed by rows and columns of the actual data of the matrix which are more meaningful.
- Cross approximation algorithms directly find the factors without computing and storing the whole matrix.

2.2 Tensors

A tensor is a multidimensional array and as such, it is a higher-dimensional generalization of vectors and matrices. It can be treated as a function of discrete variables. In this context, a vector can be viewed as a first-order tensor, and a matrix as a second-order tensor, with the order of a tensor denoting the number of dimensions (or modes) in the multidimensional array.

The shape of a d-th order tensor $\boldsymbol{\mathcal{P}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is defined by a tuple of integers $\boldsymbol{n} = (n_1, \ldots, n_d)$. We designate n as $\max(n_1, \ldots, n_d)$ and introduce the index set $\mathcal{I}$ for the tensor $\boldsymbol{\mathcal{P}}$ as $\mathcal{I} = \{\boldsymbol{i} = (i_1, \ldots, i_d),\ i_k \in \{1, \ldots, n_k\},\ k \in \{1, \ldots, d\}\}$, which uniquely identifies its elements. The $\boldsymbol{i}$-th element of the tensor $\boldsymbol{\mathcal{P}}$ is denoted as $\boldsymbol{\mathcal{P}}_{\boldsymbol{i}}$.

In tensor terminology, a *fiber* represents the higher-order equivalent of a matrix row or

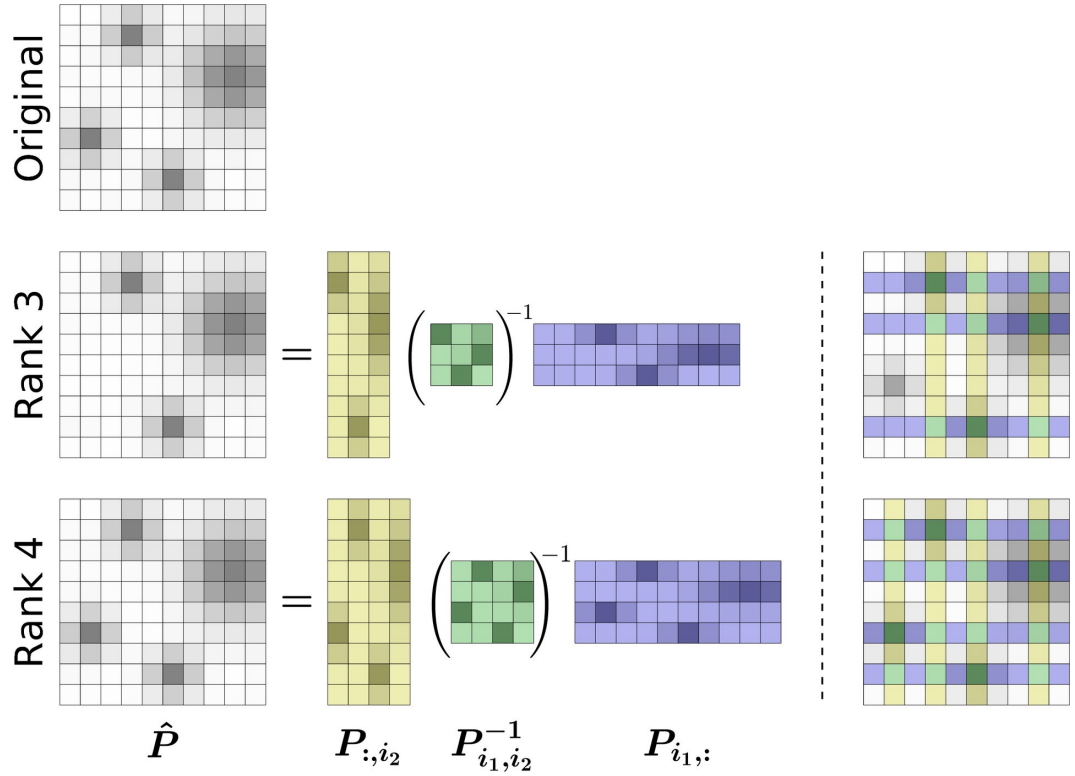


Figure 2.1: For a given matrix $\boldsymbol{P}$ (top-left), suppose we know r independent columns indexed by $\boldsymbol{i}_2 = (i_2^1, \ldots, i_2^r)$, i.e., $\boldsymbol{P}_{:,\,\boldsymbol{i}_2} \in \mathbb{R}^{n_1 \times r}$ and r independent rows indexed by $\boldsymbol{i}_1 = (i_1^1, \ldots, i_1^r)$, i.e., $\boldsymbol{P}_{\boldsymbol{i}_1,\,:} \in \mathbb{R}^{r \times n_2}$, with their intersection $\boldsymbol{P}_{\boldsymbol{i}_1,\,\boldsymbol{i}_2} \in \mathbb{R}^{r \times r}$ being nonsingular. Then, by skeleton decomposition we have $\hat{\boldsymbol{P}} = \boldsymbol{P}_{:,\,\boldsymbol{i}_2}\, \boldsymbol{P}^{-1}_{\boldsymbol{i}_1,\,\boldsymbol{i}_2}\, \boldsymbol{P}_{\boldsymbol{i}_1,\,:}$. If $\operatorname{rank}(\boldsymbol{P}) = r$, then $\hat{\boldsymbol{P}} = \boldsymbol{P}$ (bottom row). For $r < \operatorname{rank}(\boldsymbol{P})$ we obtain a quasi-optimal approximation, $\hat{\boldsymbol{P}} \approx \boldsymbol{P}$ (middle row). The right figures show the rows and columns selected from the original matrix by the cross approximation algorithm to find the skeleton decomposition.

column; it is a vector obtained by fixing every index but one. Likewise, a *slice* of a tensor is a matrix acquired by fixing every index but two.

2.3 Tensors as Discrete Analogue of a Function

In numerous applications, tensors naturally emerge from discretizing multivariate functions defined on a rectangular domain. Let $P : \Omega_{\boldsymbol{x}} \subset \mathbb{R}^d \to \mathbb{R}$ be a function with a rectangular domain $\Omega_{\boldsymbol{x}} = \times_{k=1}^{d} \Omega_{x_k}$, a Cartesian product of intervals for each dimension. Various discretization approaches for specifying these intervals can be considered, with a default assumption of uniform discretization unless otherwise stated. Bounded intervals $\Omega_{x_k} \subset \mathbb{R}$ are discretized into n_k elements. The discretization set is denoted

as $\mathcal{X} = \{\boldsymbol{x} = (x_1^{i_1}, \ldots, x_d^{i_d}) : x_k^{i_k} \in \Omega_{x_k},\ i_k \in \{1, \ldots, n_k\}\}$, and the corresponding index set is defined as $\mathcal{I}_\mathcal{X} = \{\boldsymbol{i} = (i_1, \ldots, i_d) :\ i_k \in \{1, \ldots, n_k\},\ k \in \{1, \ldots, d\}\}$. A canonical bijective discretization map $X : \mathcal{I}_\mathcal{X} \to \mathcal{X}$ defined as $X(\boldsymbol{i}) = (x_1^{i_1}, \ldots, x_d^{i_d}), \forall \boldsymbol{i} = (i_1, \ldots, i_d) \in \mathcal{I}_\mathcal{X}$. With this discretization, a tensor $\boldsymbol{\mathcal{P}}$, a discrete analogue of the function P, is obtained by evaluating the function at the discretization points given by $\mathcal{X}$, i.e., $\boldsymbol{\mathcal{P}}_{\boldsymbol{i}} = P(X(\boldsymbol{i})),\ \boldsymbol{i} \in \mathcal{I}_\mathcal{X}$. To simplify notation, we overload the terminology and define $\boldsymbol{\mathcal{P}}_{\boldsymbol{x}} = \boldsymbol{\mathcal{P}}_{X^{-1}(\boldsymbol{x})},\ \forall \boldsymbol{x} \in \mathcal{X}$. Notably, given a discrete analogue $\boldsymbol{\mathcal{P}}$ of a function P, we can approximate the value $P(\boldsymbol{x})$ for any $\boldsymbol{x} \in \Omega_{\boldsymbol{x}}$ by interpolating between specific nodes of the tensor $\boldsymbol{\mathcal{P}}$.

However, naively approximating a high-dimensional function using a tensor becomes intractable due to the computational and storage complexities of the tensor ($\mathcal{O}(n^d)$). TNs address the storage issue by representing a tensor with factors that have a smaller number of elements by exploiting the separability (or low-rank structure).

2.4 Tensor Networks

Similar to matrix factorization described in Section 2.1.1 Tensor Networks (TNs) represent a high-dimensional tensor or multidimensional array (a function of only discrete variables) using several low-order tensors. They allow powerful interpolation schemes for handling continuous variables. Alternatively, it can be viewed as a function approximation tool that uses the sum-of-product of univariate functions (separable form) to represent functions. This feature allows us to perform various algebraic operations and calculus efficiently.

There are various types of TNs such as Tucker TNs, Tensor Train (TT) TNs, Tree TNs, MERA, PEPS. Each of them is parameterized to represent functions in separable form. However, they differ in their parameter efficiency, scalability, and expressivity. Given the data of the target function (i.e., data for supervised or unsupervised learning) or the target function itself (i.e., a procedure to evaluate its elements), there exist various techniques to represent the target function in these formats.

Tensor diagrams provide a elegant tool to describe tensors and TNs pictorially. We describe the tensors, tensor contraction operations, and TNs in Figure 2.2, 2.3 and 2.4. For more details, we refer the readers to [16] and [1].

Tensor Train provides a very good balance in terms of expressivity, scalability, and parameter efficiency among these TNs. Thus, in the remainder of this thesis, we will use TT for all the applications considered, and it is described in more detail in the subsequent sections.

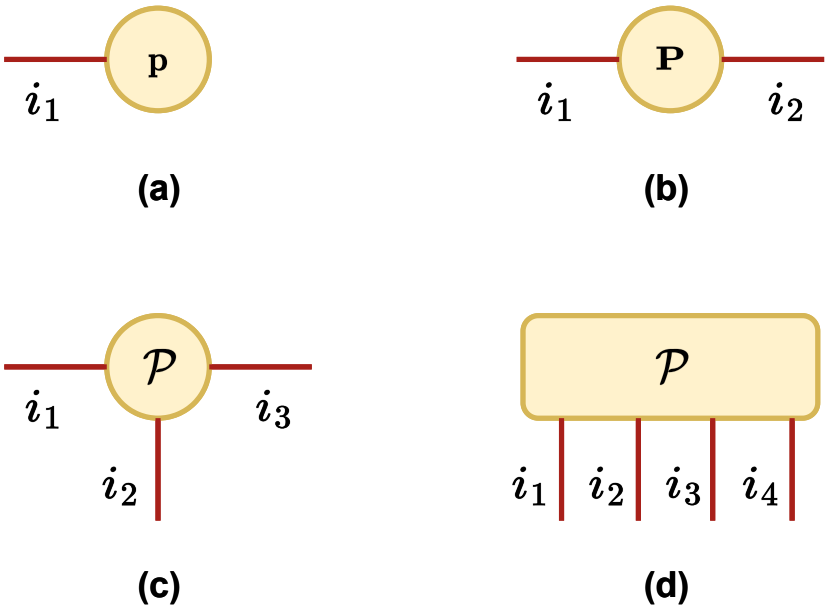
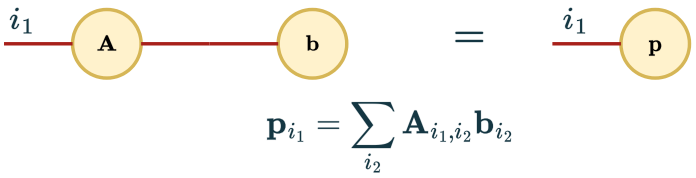
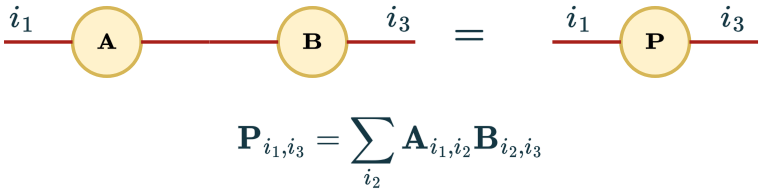


Figure 2.2: Tensor Diagram: In the diagrammatic notation for tensors, each tensor (a multidimensional array) is represented as a solid shape. The number of lines branching out of the solid shape corresponds to its order. In the figure, (a), (b), (c), and (d) represent a first-order tensor (a vector), a second-order tensor (a matrix), a third and a fourth-order tensor respectively.



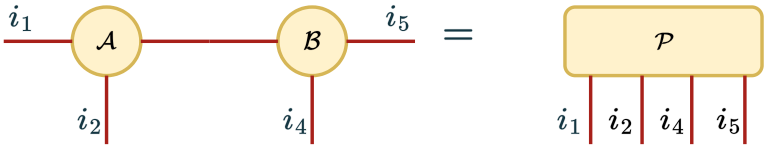
$$\mathbf{p}_{i_1} = \sum_{i_2} \mathbf{A}_{i_1,i_2} \mathbf{b}_{i_2}$$

(a)



$$\mathbf{P}_{i_1,i_3} = \sum_{i_2} \mathbf{A}_{i_1,i_2} \mathbf{B}_{i_2,i_3}$$

(b)



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

(c)

Figure 2.3: Tensor contraction: We can use tensor contraction (networking) to form a new tensor using multiple tensors. The shared line (closed line) between two tensors represents summation over that index. In the figure, (a) represents a matrix-vector product that results in another vector (first-order tensor), (b) represents the familiar matrix factorization which is a matrix-matrix product resulting in a new matrix (second-order tensor), (c) represents a fourth-order tensor formed by a contraction over two third order tensors. Note that the number of open-ended lines in the diagram corresponds to the order of the resultant tensor after tensor contractions.

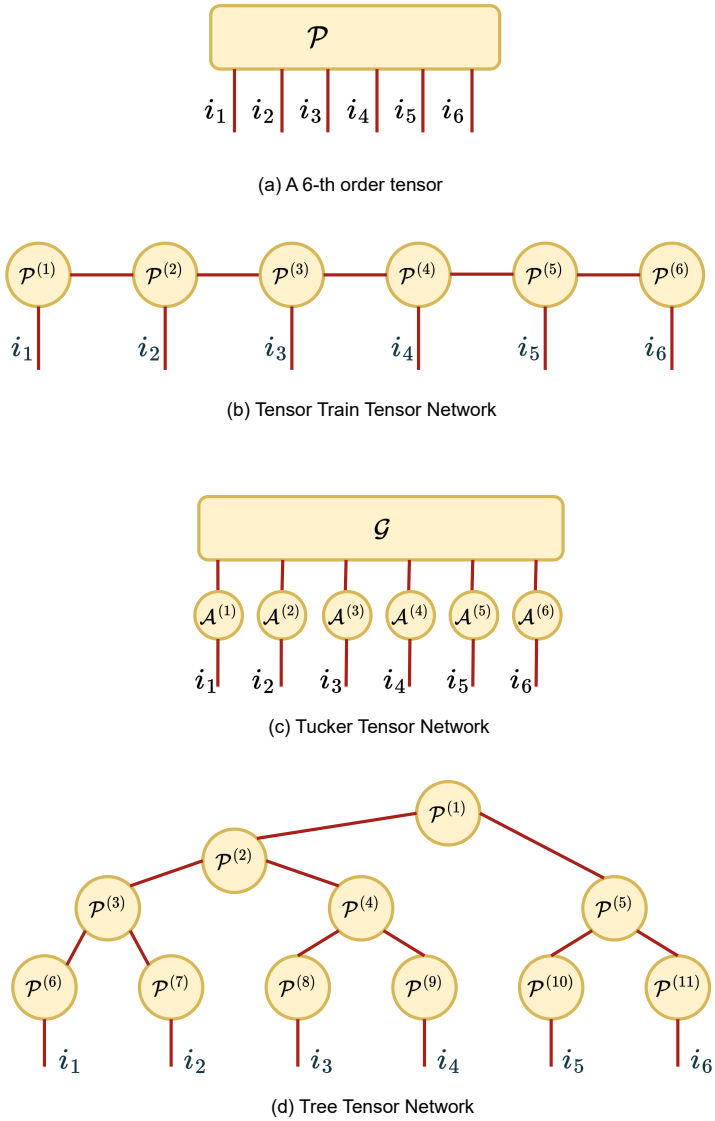


Figure 2.4: Tensor Networks: We can use the tensor diagram (Fig.2.2) and tensor contraction (Fig. 2.3) to pictorially describe various tensor networks (TNs). The figure describes how a sixth-order tensor in (a) can be represented using Tensor Train TN in (b), Tucker TN in (c), and Tree TN in (d).

2.5 Tensor Train

Among the popular tensor networks, we concentrate in this work on the Tensor Train (TT). They are a special type of Tree Tensor Networks and also called by Matrix Product States (MPS) in the physics community [16]. TT decomposition encodes a given tensor compactly using a set of third-order tensors called *cores*. A d-th order tensor $\boldsymbol{\mathcal{P}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ in TT format is represented using a tuple of d third-order tensors $(\boldsymbol{\mathcal{P}}^1, \ldots, \boldsymbol{\mathcal{P}}^d)$. The dimension of the cores are given as $\boldsymbol{\mathcal{P}}^1 \in \mathbb{R}^{1 \times n_1 \times r_1}, \boldsymbol{\mathcal{P}}^k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$, $k \in \{2, \ldots, d-1\}$, and $\boldsymbol{\mathcal{P}}^d \in \mathbb{R}^{r_{d-1} \times n_d \times 1}$ with $r_0 = r_d = 1$. As shown in Figure 2.5, the $\boldsymbol{i}$-th element of the tensor in this format, with $\boldsymbol{i} \in \mathcal{I} = \{(i_1, \ldots, i_d) \colon i_k \in \{1, \ldots, n_k\},\ k \in \{1, \ldots, d\}\}$, is simply given by multiplying matrix slices from the cores:

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{i}} = \boldsymbol{\mathcal{P}}^1_{:,\, i_1,\, :}\, \boldsymbol{\mathcal{P}}^2_{:,\, i_2,\, :} \cdots \boldsymbol{\mathcal{P}}^d_{:,\, i_d,\, :}, \tag{2.6}$$

where $\boldsymbol{\mathcal{P}}^k_{:,\, i_k,\, :} \in \mathbb{R}^{r_{k-1} \times r_k}$ represents the i_k-th frontal slice (a matrix) of the third-order tensor $\boldsymbol{\mathcal{P}}^k$. The dimensions of the cores are such that the above matrix multiplication yields a scalar. Note that, for $d = 2$, it reduces into the familar matrix factorization as described in Section 2.1.1. The *TT-rank* of the tensor in TT representation is then defined as the tuple $\boldsymbol{r} = (r_1,\ r_2, \ldots, r_{d-1})$. We call $r = \max(r_1, \ldots, r_{d-1})$ as the *maximal rank*. For any given tensor, there always exists a TT representation (2.6) [17].

Due to its structure, the TT representation offers several advantages for storage and computation. Let $n = \max(n_1, \ldots, n_d)$. Then, the number of elements in the TT representation is $\mathcal{O}(ndr^2)$ as compared to $\mathcal{O}(n^d)$ elements in the original tensor. For a small r and a large d, the representation is thus very efficient. As explained in Section 2.1.1, the existence of a low-rank structure (i.e., a small r) of a given tensor is closely related to the separability of the underlying multivariate function. Although separability of functions is not a very well understood concept, it is known that smoothness and symmetry of functions often induces better separability of the functions [13]. By *better*, we mean fewer low-dimensional functions in the sum of products representation. The degree of *smoothness* can be formally defined using the properties of higher-order derivatives, however, roughly speaking, it implies the degree of variation of the function across its domain. For example, a probability density function in the form of a Gaussian Mixture Model (GMM) is considered to become less smooth as the number of mixture components (i.e., multi-modality) increases or the variance of the component Gaussians decreases (i.e., sharper peaks).

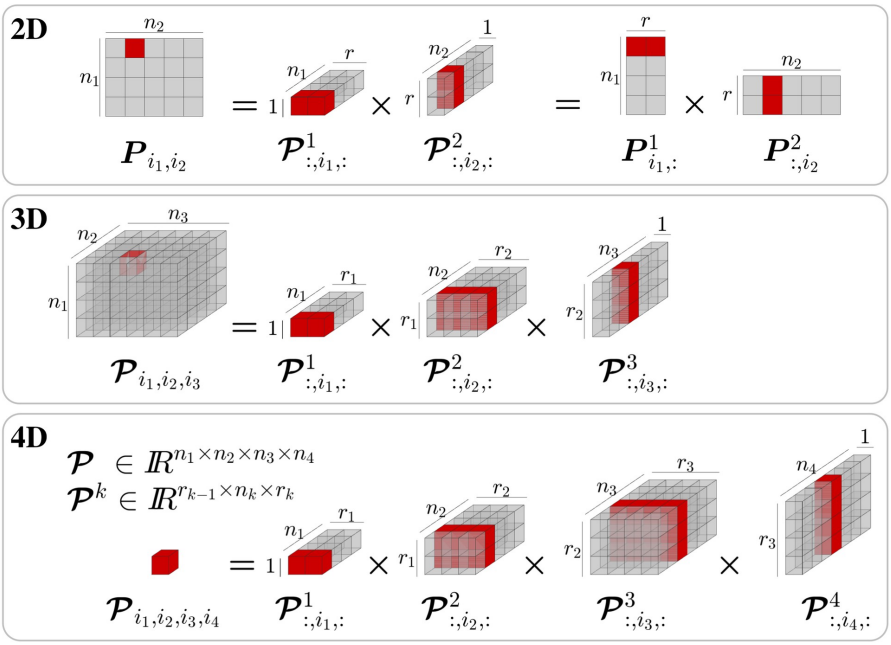


Figure 2.5: TT is an extension of matrix decomposition techniques to higher dimensional arrays. With a matrix decomposition, we can access an element of the original matrix by multiplying appropriate rows or columns of the factors. Similarly, an element of a tensor in TT format can be accessed by multiplying the selected slices (matrices represented in red color) of the core tensors (factors). The figure depicts examples for a 2nd order, 3rd order, and a 4th order tensor.

2.5.1 Continous Function Approximation using Tensor Train

Given the discrete analogue tensor $\boldsymbol{\mathcal{P}}$ of a function P in TT format, we can obtain the continuous approximation by interpolating the TT cores, in a similar way as in the matrix case in Section 2.1.1. For example, we can use a linear interpolation for each core (i.e., between the matrix slices of the core) and define a matrix-valued function corresponding to each core $k \in \{1, \ldots, d\}$,

$$\boldsymbol{P}^k(x_k) = \frac{x_k - x_k^{i_k}}{x_k^{i_k+1} - x_k^{i_k}} \, \boldsymbol{\mathcal{P}}^k_{:,\, i_k+1,\, :} + \frac{x_k^{i_k+1} - x_k}{x_k^{i_k+1} - x_k^{i_k}} \, \boldsymbol{\mathcal{P}}^k_{:,\, i_k,\, :}, \tag{2.7}$$

where $x_k^{i_k} \leq x_k \leq x_k^{i_k+1}$ and $\boldsymbol{P}^k : \Omega_{x_k} \subset \mathbb{R} \to \mathbb{R}^{r_{k-1} \times r_k}$ with $r_0 = r_d = 1$. This induces a continuous approximation of P given by

$$P(x_1, \ldots, x_d) \approx \boldsymbol{P}^1(x_1) \cdots \boldsymbol{P}^d(x_d). \tag{2.8}$$

Note that a higher-order spline-based interpolation can also be used if needed. In this thesis, we used Catmull-Rom splines for the interpolation.

We overload the terminology again to define the continuous TT representation as:

$$\begin{aligned} \boldsymbol{\mathcal{P}}^{k}_{:,\, x_k,\, :} &= \boldsymbol{P}^{k}(x_k), \\ \boldsymbol{\mathcal{P}}_{\boldsymbol{x}} &= P(x_1, \ldots, x_d), \\ &= \boldsymbol{\mathcal{P}}^{1}_{:,\, x_1,\, :} \cdots \boldsymbol{\mathcal{P}}^{d}_{:,\, x_d,\, :}, \quad \forall x_k \in \Omega_{x_k},\ \forall k \in \{1, \ldots, d\}, . \end{aligned} \tag{2.9}$$

2.5.2 Refining Tensor Train Model

Suppose we have a TT model $\boldsymbol{\mathcal{P}}$ defined on a domain $\Omega_{\boldsymbol{x}} = \Omega_{x_1} \times \cdots \times \Omega_{x_d}$ with discretization set $\mathcal{X} = \{\boldsymbol{x} = (x_1^{i_1}, \ldots, x_d^{i_d}) : x_k^{i_k} \in \Omega_{x_k},\ i_k \in \{1, \ldots, n_k\}\}$. We can obtain a refined TT model $\hat{\boldsymbol{\mathcal{P}}}$ defined on a finer discretization $\hat{\mathcal{X}} = \{\boldsymbol{x} = (x_1^{i_1}, \ldots, x_d^{i_d}) : x_k^{i_k} \in \Omega_{x_k}, i_k \in \{1, \ldots, \hat{n}_k\}\}$ of the domain with $\hat{n}_k > n_k$ using interpolation of the TT cores. The cores of the corresponding TT model $\hat{\boldsymbol{\mathcal{P}}}$ defined over the refined discretization can be determined using $\hat{\boldsymbol{\mathcal{P}}}^{k}_{:,\, x_k,\, :} = \boldsymbol{P}^{k}(x_k),\ \forall k \in \{1, \ldots, d\},\ (x_1, \ldots, x_d) \in \hat{\mathcal{X}}$ using interpolation scheme as described in Section 2.5.1.

This proves beneficial in specific applications where a coarse discretization is employed during the learning phase to acquire the TT representation of a target function, which can be computationally intensive. During the inference phase, however, in some applications, the TT model might be required to be defined over a finer discretization for more accurate results. For instance, in Chapter 4, we introduce a method for determining optima of a tensor in TT format, with these optima representing an element of the underlying discretization set. To apply this technique to find optima of an arbitrary function over continuous variables, we initially use a coarse discretization of the domain during the learning phase to find a TT representation over this discretization set (i.e., modeling the function in TT format). Subsequently, before employing the optimization technique, we refine the TT model over a finer discretization set, ensuring the attainment of a more accurate optimum.

2.6 Algebraic Operations over Tensor Train

Working with TT models offers an advantage as it enables the efficient execution of various commonly used algebraic operations directly in the TT format. In this thesis, we briefly describe the operations employed, and for a more in-depth understanding, we refer readers to [18].

Suppose we have two TT models $\boldsymbol{\mathcal{P}}$ with TT cores $(\boldsymbol{\mathcal{P}}^1, \ldots, \boldsymbol{\mathcal{P}}^d)$ and $\boldsymbol{\mathcal{Q}}$ with cores $(\boldsymbol{\mathcal{Q}}^1, \ldots, \boldsymbol{\mathcal{Q}}^d)$ defined over the same discretization set $\mathcal{X} = \{\boldsymbol{x} = (x_1^{i_1}, \ldots, x_d^{i_d}) : \ x_k^{i_k} \in \Omega_{x_k}, \ i_k \in \{1, \ldots, n_k\}\}$.

Let the dimension of the cores of $\boldsymbol{\mathcal{P}}$ be given as $\boldsymbol{\mathcal{P}}^k \in \mathbb{R}^{r_{k-1}^P \times n_k \times r_k^P}$ for $k \in \{1, \ldots, d\}$, with $r_0^P = r_d^P = 1$. Similarly, $\boldsymbol{\mathcal{Q}}^k \in \mathbb{R}^{r_{k-1}^Q \times n_k \times r_k^Q}$ for $k \in \{1, \ldots, d\}$, with $r_0^Q = r_d^Q = 1$.

Scalar Multiplication

Mulitplying a TT $\boldsymbol{\mathcal{P}}$ with a scalar $c \in \mathbb{R}$ results in a new TT model $\boldsymbol{\mathcal{S}} = c\boldsymbol{\mathcal{P}}$ whose cores are given by $\boldsymbol{\mathcal{S}}^1 = c\boldsymbol{\mathcal{P}}^1$ and $\boldsymbol{\mathcal{S}}^k = \boldsymbol{\mathcal{P}}^k, \quad k \in \{2, \ldots, d\}$. The rank of $\boldsymbol{\mathcal{S}}$ is same as that of $\boldsymbol{\mathcal{P}}$.

Addition

Adding two TT models $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$ results in another TT model $\boldsymbol{\mathcal{S}}$ defined as $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{P}} + \boldsymbol{\mathcal{Q}}$ whose cores are determined as follows:

$$
\begin{aligned}
\boldsymbol{\mathcal{S}}^1_{:,\, i_1,\, :} &= [\boldsymbol{\mathcal{P}}^1_{:,\, i_1,\, :} \quad \boldsymbol{\mathcal{Q}}^1_{:,\, i_1,\, :}], \\
\boldsymbol{\mathcal{S}}^d_{:,\, i_d,\, :} &= [\boldsymbol{\mathcal{P}}^d_{:,\, i_d,\, :} \quad \boldsymbol{\mathcal{Q}}^d_{:,\, i_d,\, :}]^\top, \\
\boldsymbol{\mathcal{S}}^k_{:,\, i_k,\, :} &= \begin{bmatrix} \boldsymbol{\mathcal{P}}^k_{:,\, i_k,\, :} & 0_{r_{k-1}^P \times r_k^Q} \\ 0_{r_{k-1}^Q \times r_k^P} & \boldsymbol{\mathcal{Q}}^k_{:,\, i_k,\, :} \end{bmatrix}, \quad k \in \{2, \ldots, d-1\}, \quad i_k \in \{1, \ldots, n_k\}.
\end{aligned}
\tag{2.10}
$$

The rank of the TT model $\boldsymbol{\mathcal{S}}$ is given by $(r_1^S, \ldots, r_{d-1}^S)$ where $r_k^S = r_k^P + r_k^Q, \ k \in \{1, \ldots, d-1\}$.

Hadamard Product

Hadamard product (elementwise multiplication) of two TT models $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$ results in another TT model $\boldsymbol{\mathcal{S}}$ defined as $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{P}} * \boldsymbol{\mathcal{Q}}$ whose cores are determined as follows using Kronecker product: $\boldsymbol{\mathcal{S}}^k_{:,\, i_k,\, :} = \boldsymbol{\mathcal{P}}^k_{:,\, i_k,\, :} \otimes \boldsymbol{\mathcal{Q}}^k_{:,\, i_k,\, :}, \quad k \in \{1, \ldots, d\}$. The rank of the TT model $\boldsymbol{\mathcal{S}}$ is given by $(r_1^S, \ldots, r_{d-1}^S)$ where $r_k^S = r_k^P \ r_k^Q, \ k \in \{1, \ldots, d-1\}$.

Computing the Mean

The mean of a TT model $\boldsymbol{\mathcal{P}}$ can be obtained as:

$$\mu = \frac{1}{\prod_{i=1}^{d} n_i} \Big(\sum_{i_1} \boldsymbol{\mathcal{P}}^{1}_{:,\, i_1,\, :} \Big) \times \cdots \times \Big(\sum_{i_d} \boldsymbol{\mathcal{P}}^{d}_{:,\, i_d,\, :} \Big) \tag{2.11}$$

Computing the Frobenius Norm

The Frobenius norm of a TT model $\boldsymbol{\mathcal{P}}$ is defined as:

$$\begin{aligned} \|\boldsymbol{\mathcal{P}}\| &= \Big(\sum_{i_1} \cdots \sum_{i_d} \boldsymbol{\mathcal{P}}(i_1, \ldots, i_d)^2 \Big)^{\frac{1}{2}}, \\ &= \Big(\sum_{i_1} \cdots \sum_{i_d} \Big(\boldsymbol{\mathcal{P}}^{k}_{:,\, i_1,\, :} \cdots \boldsymbol{\mathcal{P}}^{d}_{:,\, i_d,\, :} \Big) \Big(\boldsymbol{\mathcal{P}}^{1}_{:,\, i_1,\, :} \cdots \boldsymbol{\mathcal{P}}^{d}_{:,\, i_d,\, :} \Big)^{\top} \Big)^{\frac{1}{2}}. \end{aligned} \tag{2.12}$$

This can be computed efficiently in a recursive manner:

$$A_k = \sum_{i_k} \boldsymbol{\mathcal{P}}^{k}_{:,\, i_k,\, :}\, A_{k+1}\, (\boldsymbol{\mathcal{P}}^{k}_{:,\, i_k,\, :})^{\top}, \quad k \in \{d, \ldots, 2, 1\}, \tag{2.13}$$

where $A_{d+1} = 1$. Then $\|\boldsymbol{\mathcal{P}}\| = \sqrt{A_1}$.

2.7 Compression of Tensor Train with Rounding Operation

TT-rounding [17] is an important operation on a tensor in TT format. Most binary operations on tensors in TT format, although efficient, increase the rank of the resultant tensor in TT format, where the resultant tensor is often not in its optimal TT representation in terms of the number of parameters involved. A repeated application of binary operations to a given tensor may result in an explosion of its TT-rank, which would affect the efficiency of subsequent operations on the tensor. For example, the addition of two tensors, both with TT-rank $\boldsymbol{r} = (r_1, \ldots, r_d)$, results in a tensor in TT format with rank $\boldsymbol{r} = (2r_1, \ldots, 2r_d)$. TT-rounding is an operation applied to tensors already in TT format to compress it to optimal TT representation and hence reduce its TT-rank. For a d-th order tensor $\boldsymbol{\mathcal{P}}$ in TT-format with maximal TT-rank r, TT-rounding has computational complexity $\mathcal{O}(ndr^3)$. The TT-rounding procedure returns a tensor $\hat{\boldsymbol{\mathcal{P}}} = \text{TT-round}(\boldsymbol{\mathcal{P}}, \hat{r})$, for a given $\hat{r} < r$, such that its maximal TT-rank is less than $\hat{r}$ and

the Frobenius norm of the residual $\|\hat{\boldsymbol{\mathcal{P}}} - \boldsymbol{\mathcal{P}}\|$ is as small as possible. Alternatively, we can specify an approximation accuracy ϵ to a tensor $\boldsymbol{\mathcal{P}}$ in TT format and the TT-rounding returns a tensor $\hat{\boldsymbol{\mathcal{P}}} = \text{TT-round}(\boldsymbol{\mathcal{P}}, \epsilon)$ with optimal TT-rank and $\|\hat{\boldsymbol{\mathcal{P}}} - \boldsymbol{\mathcal{P}}\| \leq \epsilon$.

2.8 Approximating Functions in Tensor Train using Cross Approximation

When the function to be approximated is known (i.e., there exists a procedure that can return the values of the function given input from its domain), the popular methods to find the TT representation are TT-SVD [17], TT-DMRG [19], and TT-cross [20]. TT-SVD and TT-DMRG, like matrix SVD, require the full tensor (discrete analog of the function) in memory to find the decomposition, and hence they are infeasible for higher-order tensors. TT Cross approximation (TT-cross) overcomes this issue by using cross-approximation techniques to find the decomposition.

TT-cross approximation (TT-cross) is an extension of the cross approximation technique for matrix decomposition explained in Section 2.1.2 for obtaining the TT representation of a tensor. It is appealing for many practical problems as it approximates the given tensor (e.g., the discrete analogue of a function) with a controlled accuracy, by evaluating only a small number of its elements and without having to compute and store the entire tensor in the memory. The method needs to compute only certain fibers of the original tensor at a time and hence works in a black-box fashion. We refer the readers to [21, 20] for more details.

Suppose we have a function P and its discrete analogue $\boldsymbol{\mathcal{P}}$ (a tensor). Given the desired accuracy for the approximation ϵ, TT-cross returns an approximate tensor in TT format $\hat{\boldsymbol{\mathcal{P}}} = \text{TT-cross}(P, \epsilon)$ to the tensor $\boldsymbol{\mathcal{P}}$ by querying only a portion of its elements ($\mathcal{O}(ndr^2)$ evaluations instead of $\mathcal{O}(n^d)$). The maximal TT-rank r is determined by the algorithm depending on the ϵ specified. The model is very efficient if the rank r of the tensor is low, which is typically the case in many engineering applications, including robotics. Thus, TT-cross avoids the need to compute and store explicitly the original tensor, which may not be possible for higher-order tensors. It only requires computing the function P that can return the elements of the tensor $\boldsymbol{\mathcal{P}}$ at various query points, i.e., the fibers of the tensor $\boldsymbol{\mathcal{P}}$.

In practical applications, it operates in an unsupervised manner, requiring only the target function (a procedure that provides the function values at specified query points) to be approximated in TT format. Additionally, the desired accuracy for representation and the maximum allowed rank of the TT representation are input parameters. The TT-cross

algorithm then determines the final rank of the resulting TT representation.

2.9 Probability Modeling using Tensor Train

In this section, we describe how TT representation can be leveraged to model probability functions. In Chapter 4, this interpretation serves as a foundational concept for developing an algorithm crucial to the core content of this thesis: the global optimization of functions. The underlying approach involves representing the objective functions for optimization in TT format and interpreting it as a distribution in TT format as described below. The probabilistic interpretation is harnessed to derive an algorithm for locating the maxima of the distribution, effectively identifying the optima of the objective function.

2.9.1 Tensor Train Distribution

Suppose we have a tensor $\boldsymbol{\mathcal{P}}$ in TT format corresponding to a function P within the discretization set $\mathcal{X}$ of the domain $\Omega_{\boldsymbol{x}}$. We can then construct the corresponding probability distribution that we call TT distribution,

$$\Pr(\boldsymbol{x}) = \frac{\boldsymbol{\mathcal{P}}_{\boldsymbol{x}}^2}{Z}, \quad \boldsymbol{x} \in \mathcal{X}, \tag{2.14}$$

where Z is the corresponding normalization constant.

Due to the separable structure of the TT model, as described in Section 2.9.2, we can get the exact samples from the TT distribution in an efficient manner without requiring to compute the normalization factor Z as described in the next section. However,if needed, the normalization constant Z for (2.14) can be computed efficiently as described in (2.12).

2.9.2 Sampling from Tensor Train distribution

Consider the discrete probability distribution given by (2.14). For the simplicity of the presentation, we assume $Z = 1$ as we will not require the normalization constant to be known for sampling from the above distribution. Any probability distribution can be expressed as a product of conditional distributions

$$\Pr(x_1, \ldots, x_d) = \Pr_1(x_1)\, \Pr_2(x_2|x_1) \cdots \Pr_d(x_d|x_1, \ldots, x_{d-1}),$$

where

$$\Pr_k(x_k|x_1,\ldots,x_{k-1}) = \frac{\sigma_k(x_1,\ldots,x_k)}{\sigma_{k-1}(x_1,\ldots,x_{k-1})},$$

is the conditional distribution defined using the marginals

$$\sigma_k(x_1,\ldots,x_k) = \sum_{x_{k+1}} \cdots \sum_{x_d} \Pr(x_1,\ldots,x_d).$$

Let $\sigma_0 = 1$. Now, using the above definitions, we can generate samples $\boldsymbol{x} \sim \Pr$ by sampling from each of the conditional distributions in turn. Each conditional distribution is a function of only one variable, and in the discrete case it is a multinomial distribution, with

$$x_k \sim \Pr_k(x_k|x_1,\ldots,x_{k-1}),\ \forall k \in \{1,\ldots,d\}.$$

However, this process becomes computationally intensive during sampling x_k, as it necessitates the conditional distribution $\Pr_k$, which, in turn, requires the evaluation of the summation over several variables to find the marginal σ_k. Consequently, this approach incurs a computational cost that grows exponentially with the number of dimensions. Here, the TT format offers an elegant solution by capitalizing on the separability of the function.

If Pr is a TT distribution (see (2.14)) corresponding to a TT model $\boldsymbol{\mathcal{P}}$ with the discretization set $\mathcal{X}$ and the cores $(\boldsymbol{\mathcal{P}}^1,\ldots,\boldsymbol{\mathcal{P}}^d)$, we have:

$$\begin{aligned}\sigma_k(x_1,\ldots,x_k) &= \sum_{x_{k+1}} \cdots \sum_{x_d} \boldsymbol{\mathcal{P}}^2_{\boldsymbol{x}},\ k \in \{1,\ldots,d\} \\ &= \left(\boldsymbol{\mathcal{P}}^1_{:,\,x_1,\,:} \cdots \boldsymbol{\mathcal{P}}^k_{:,\,x_k,\,:}\right) \beta^{k+1} \left(\boldsymbol{\mathcal{P}}^1_{:,\,x_1,\,:} \cdots \boldsymbol{\mathcal{P}}^k_{:,\,x_k,\,:}\right)^\top,\end{aligned} \tag{2.15}$$

where we can compute β^k efficiently in a recursive manner as follows:

$$\begin{aligned}\beta^k &= \sum_{x_k} \cdots \sum_{x_d} \left(\boldsymbol{\mathcal{P}}^k_{:,\,x_k,\,:} \cdots \boldsymbol{\mathcal{P}}^d_{:,\,x_d,\,:}\right)\left(\boldsymbol{\mathcal{P}}^k_{:,\,x_k,\,:} \cdots \boldsymbol{\mathcal{P}}^d_{:,\,x_d,\,:}\right)^\top, \\ &= \sum_{x_k} \boldsymbol{\mathcal{P}}^k_{:,\,x_k,\,:}\ \beta^{k+1}\ (\boldsymbol{\mathcal{P}}^k_{:,\,x_k,\,:})^\top, \quad k \in \{d,\ldots,2,1\},\end{aligned} \tag{2.16}$$

where $\beta^{d+1} = 1$. Alternatively, there exists a procedure called Tensor Train orthogonalization [18] which re-parameterizes the core tensors of $\boldsymbol{\mathcal{P}}$ in TT format so that β^k is an identity matrix. This will eliminate the need for the computational of (2.15). Thus, the

Algorithm 1 TT-CD: Sampling from TT distribution given by (2.14).

Input: TT Blocks $\boldsymbol{\mathcal{P}} = (\boldsymbol{\mathcal{P}}^1, \ldots, \boldsymbol{\mathcal{P}}^d)$ corresponding to the distribution Pr, sample priority $\alpha \in (0, 1)$
Output: N samples $\{(x_1^l, \ldots, x_d^l)\}_{l=1}^N$ from the distribution Pr (see (2.14))
$\boldsymbol{\beta}_{d+1} \leftarrow 1$
for $k \leftarrow d$ to 2 **do**
  $\boldsymbol{\beta}^k = \sum_{x_k} \boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :}\, \boldsymbol{\beta}_{k+1}\, (\boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :})^\top$
end for
$\boldsymbol{\Phi}_1 \leftarrow \mathbf{1} \in \mathbb{R}^{N \times 1}$
for $k \leftarrow 1$ to d **do**
  $\pi^k(x_k) = \boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :}\, \boldsymbol{\beta}^{k+1}\, (\boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :})^\top, \quad \forall x_k$
  for $l = 1, \ldots, N$ **do**
    $\boldsymbol{p}_k(x_k) = |\Phi_k(l,\, :)\, \pi^k(x_k)\, \Phi_k(l,\, :)^\top|, \quad \forall x_k$
    Sample x_k^l from the multinomial distribution $\boldsymbol{p}_k$
    $\boldsymbol{\Phi}_{k+1}(l,\, :) = \boldsymbol{\Phi}_k(l,\, :)\, \boldsymbol{\mathcal{P}}^k_{:,\, x_k^l,\, :}$
  end for
end for

TT format reduces the complicated multidimensional summation to evaluate σ_k into several one-dimensional summations. As the same summation terms appear over several conditionals $\Pr_k$, we can use an algorithm called Tensor Train Conditional Distribution (TT-CD) sampling [22], to efficiently get the samples from Pr. This is described in Algorithm 1.

2.9.3 Conditioning Tensor Train Distribution

Suppose we want to fix a subset of variables in $\boldsymbol{x}$ and find the corresponding conditional distribution of the remaining variables. Without loss of generality, let $\boldsymbol{x}$ be segmented as $\boldsymbol{x} = (\boldsymbol{x}_1,\ \boldsymbol{x}_2) \in \Omega_{\boldsymbol{x}} = \Omega_{\boldsymbol{x}_1} \times \Omega_{\boldsymbol{x}_2}$ with $\boldsymbol{x}_1 \in \Omega_{\boldsymbol{x}_1} \subset \mathbb{R}^{d_1}$, $\boldsymbol{x_2} \in \Omega_{\boldsymbol{x}_2} \subset \mathbb{R}^{d_2}$. i.e., $\boldsymbol{x}_1$ corresponds to the first d_1 variables in $\boldsymbol{x}$. We are interested in finding the conditional distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1)$ of the TT distribution given in (2.14).

Suppose $\boldsymbol{x}_1$ takes a particular value $\boldsymbol{x}_t = (x_1, \ldots, x_{d_1})$. We can obtain $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$ by defining a conditional TT model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_1 = \boldsymbol{x}_t}$ using TT model $\boldsymbol{\mathcal{P}}$ as

$$\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}_{\boldsymbol{x_2}} = \boldsymbol{\mathcal{P}}_{(\boldsymbol{x}_t,\, \boldsymbol{x}_2)}, \quad \forall \boldsymbol{x_2} \in \Omega_{\boldsymbol{x}_2}.$$

In other words, the TT cores of $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_1 = \boldsymbol{x}_t}$ are then given by

$$
\begin{aligned}
(\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t})^1_{:,\,x,\,:} &= \Big(\prod_{i=1}^{d_1} \boldsymbol{\mathcal{P}}^i_{:,\,x_i,\,:} \Big) \boldsymbol{\mathcal{P}}^{d_1+1}_{:,\,x,\,:}, \quad \forall x \in \Omega_{x_{d+1}} \\
(\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t})^k &= \boldsymbol{\mathcal{P}}^{k+d_1},\ k \in \{2, \dots, d_2\},
\end{aligned}
\tag{2.17}
$$

Given the above-defined conditional TT model, we can obtain the conditional distribution as

$$
\Pr(\boldsymbol{x}_2 | \boldsymbol{x}_1 = \boldsymbol{x}_t) = \frac{(\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}_{\boldsymbol{x}_2})^2}{Z_2}, \quad \forall \boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \tag{2.18}
$$

Given $\boldsymbol{x}_1 = \boldsymbol{x}_t$, we can sample $\boldsymbol{x}_2$ from this distribution using Algorithm 1 with the conditional TT model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_1 = \boldsymbol{x}_t}$.

2.10 Tensor Train for Data-Driven Learning

In this section, we describe a popular way the TT model is employed for data-driven function approximation. It is used to represent the weights for the basis function representation (linear combination of basis functions) of a high-dimensional function to overcome the curse of dimensionality. It can be used for regression problems or density modeling as explained below.

Suppose we are interested in a parameterized function approximation over a domain $\Omega_{\boldsymbol{x}} = \Omega_{x_1} \times \cdots \times \Omega_{x_d} \subset \mathbb{R}^d$ in the form of linear combinations of pre-defined basis functions. Consider a collection of basis functions $\boldsymbol{\phi}^i(x_i) = (\phi^i_1(x_i), \dots, \phi^i_n(x_i)) \in \mathbb{R}^{n_i}$, for each dimension $i \in \{1, \dots, d\}$. Then, denoting $\boldsymbol{x} = (x_1, \dots, x_d)$, the basis functions for the domain can be obtained by the elements of the d-th order tensor formed by the outer product of these vectors: $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{\phi}^1(x_1) \circ \cdots \circ \boldsymbol{\phi}^d(x_d) \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. So elements of $\boldsymbol{\Phi}(\boldsymbol{x})$ forms a basis function for $\mathbb{R}$.

Let $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, a d-th order tensor, represent the coefficients for this basis. i.e., for a given index $\boldsymbol{i} = (i_1, \dots, i_d),\ i_k \in \{1, \dots, n_k\}$, we have $\boldsymbol{\Phi}_{\boldsymbol{i}}(\boldsymbol{x}) = \phi^1_{i_1}(x_1) \times \cdots \times \phi^d_{i_d}(x_d)$, and $\boldsymbol{\mathcal{W}}_{\boldsymbol{i}}$ represents the coefficient corresponding to the basis $\boldsymbol{\Phi}_{\boldsymbol{i}}$. A typical choice for $\boldsymbol{\phi}^i_j(x_i) \in \mathbb{R}$ includes radial basis functions, splines, and Fourier basis if x_i is a continuous variable. Alternatively, it could be a delta function if the variable is discrete.

The parametric function approximation model can then be defined as:

$$\begin{aligned} F(\boldsymbol{x};\ \boldsymbol{\mathcal{W}}) &= \Big\langle \boldsymbol{\mathcal{W}},\ \boldsymbol{\Phi}(\boldsymbol{x}) \Big\rangle, \\ &= \sum_{\boldsymbol{i} \in \mathcal{I}} \boldsymbol{\mathcal{W}}_{\boldsymbol{i}}\ \boldsymbol{\Phi}_{\boldsymbol{i}}(\boldsymbol{x}). \end{aligned} \tag{2.19}$$

However, (2.19) suffers from the curse of dimensionality as the number of parameters in $\boldsymbol{\mathcal{W}}$ scales exponentially with the number of dimensions d ($\mathcal{O}(n^d)$ with $n = \max(n_1, \ldots, n_d)$). We can overcome this by assuming TT representation for $\boldsymbol{\mathcal{W}}$ with TT cores $(\boldsymbol{\mathcal{W}}^1, \ldots, \boldsymbol{\mathcal{W}}^d)$. Thus, the number of parameters will be $\mathcal{O}(ndr^2)$ in TT format where r is the maximal rank.

Thus, if $\boldsymbol{\mathcal{W}}$ is in TT format, then the above equation (2.19) can be written as:

$$\begin{aligned} F(\boldsymbol{x};\ \boldsymbol{\mathcal{W}}) &= \Big\langle \boldsymbol{\mathcal{W}},\ \boldsymbol{\Phi}(\boldsymbol{x}) \Big\rangle, \\ &= \sum_{\boldsymbol{i} \in \mathcal{I}} \boldsymbol{\mathcal{W}}_{\boldsymbol{i}}\ \boldsymbol{\Phi}_{\boldsymbol{i}}(\boldsymbol{x}), \\ &= \sum_{i_1} \cdots \sum_{i_d} \boldsymbol{\mathcal{W}}^1_{:,\, i_1,\, :} \cdots \boldsymbol{\mathcal{W}}^d_{:,\, i_d,\, :}\ \phi^1_{i_1}(x_1) \cdots \phi^d_{i_d}(x_d), \\ &= \Big(\sum_{i_1} \boldsymbol{\mathcal{W}}^1_{:,\, i_1,\, :}\ \phi^1_{i_1}(x_1) \Big) \cdots \Big(\sum_{i_d} \boldsymbol{\mathcal{W}}^d_{:,\, i_d,\, :}\ \phi^d_{i_d}(x_d) \Big), \end{aligned} \tag{2.20}$$

which is computationally efficient.

This parametric model given in (2.20) has been used for supervised learning for regression and classification tasks in [23, 24]. Such representation can approximate arbitrary functions and the weight parameters (the TT model) can be found using gradient descent procedures given a dataset of input-output pairs. The properties of the TT model also allow more advanced methodologies (e.g., Riemannian optimization, DMRG scheme) for parameter training as described in [23, 24] with mean squared error as the loss function.

The parametric model given in (2.20) can be adapted for density estimation with a minor modification:

$$P(\boldsymbol{x};\ \boldsymbol{\mathcal{W}}) = \frac{\langle \boldsymbol{\mathcal{W}},\ \boldsymbol{\Phi}(\boldsymbol{x}) \rangle^2}{Z}, \tag{2.21}$$

where Z is the normalization constant. Unlike many other function approximation techniques such as Neural Networks, if $\boldsymbol{\mathcal{W}}$ is in TT format, the normalization constant can be computed efficiently. This has been used in [25, 26, 27] for density estimation with negative likelihood (or negative log-likelihood) as the loss function. The main advantage is that the above TT-based density model allows exact sampling in a fast manner as described in Section 2.9.2. Moreover, it would also allow finding optima of the distribution through techniques introduced in this thesis in Chapter 4. This allows interesting frameworks for imitation learning and offline reinforcement learning as described in Section 5.6.1.

3 Ergodic Exploration using Tensor Train

In robotics, ergodic control extends the tracking principle by specifying a probability distribution over an area to cover instead of a trajectory to track. This provides a systematic approach for designing control policies for exploration tasks with robotic systems across applications like manipulation, surveillance, and human-robot collaboration. The original problem, formulated as a spectral multiscale coverage problem, typically requires the spatial distribution to be decomposed as a Fourier series. This approach does not scale well to control problems requiring exploration in search space of more than 2 dimensions. In this chapter, we show how we can tackle the scalability challenge by using the tensor train (TT) representation. By mitigating the curse of dimensionality, TT facilitates real-time implementation and enables ergodic control applicable to complex robotic systems operating in high-dimensional state spaces. The approach is applied to a peg-in-hole insertion task requiring full 6D end-effector poses, implemented with a 7-axis Franka Emika Panda robot without the use of force/torque sensors.

Publication Note

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

- S. Shetty, J. Silvério, and S. Calinon, "Ergodic exploration using tensor train: Applications in insertion tasks," *IEEE Trans. on Robotics*, vol. 38, no. 2, pp. 906–921, 2022. The work was also invited to present at IROS 2022.

Supplementary materials including videos and source codes related to this chapter are available at: https://sites.google.com/view/ergodic-exploration/

3.1 Introduction

Autonomous systems are often encountered with coverage tasks in applications such as localization, tracking, and active learning. In such tasks, the agent might be required to explore a region of its state space, either due to the nature of the task at hand (e.g. surveillance) or due to uncertainties induced by sensory inaccuracies (e.g. peg-in-hole insertion). In such problems, the coverage task can be specified by a reference probability density function, which encodes the importance of exploration at any point of the state space. For such problems, a pattern-based coverage approach (e.g., a "lawnmower-type" strategy), as commonly used in low-dimensional state space, is not scalable, and hence not applicable to most of the applications encountered in practice [28]. Maximizing information gain, another popular approach to circumvent uncertainty, is not suitable for exploration since the coverage is likely to be concentrated in regions around information maxima disproportionately over the period of exploration [29].

Ergodic control provides an elegant solution to design control policies for such autonomous systems, in order to equip them with natural search behaviors (see Fig. 3.1). For a given reference probability density function over a domain of interest in the state space of the robot, a dynamical system is said to be ergodic if the fraction of time spent in a given region is proportional to the probability mass of that region [30]. This is formalized in ergodic theory, where the goal is to characterize how ergodic a given dynamical system is. Ergodic control, on the other hand, aims to design a control policy for a given autonomous system so that the trajectory evolution of the resulting dynamical system is ergodic for the reference probability distribution. Systems engineered in such a way have already found applications in robotics [29, 31].

A popular approach to ergodic control is called Spectral Multiscale Coverage (SMC) and involves spectral analysis of the dynamical system evolution [30]. This elegant method was proposed for point-mass systems having receding horizon control with infinitesimal control horizon. This original work paved the way for various extensions, with other types of dynamical systems and finite-horizon controllers [31, 29, 32]. The idea behind SMC [30] is to minimize a metric, called the ergodic metric, that quantifies the match between the Fourier coefficients of a reference distribution and those of the time-averaged statistics of the system trajectory. As we will see in Section 3.3, this method suffers from the curse of dimensionality, prohibiting its applications to search spaces with more than 2 or 3 dimensions, which are often encountered in robot manipulation problems (e.g. exploration in the task space of an end-effector is often a 6D problem).

We propose a solution to overcome the challenges in SMC by using tensor train (TT) representation and hence expanding the domain of ergodic control to robot manipulation.

Figure 3.2 shows an ergodic exploration behavior generated by the proposed method for the peg-in-hole task considered in this chapter.

We showcase our approach in a 6D peg-in-hole insertion task using a robot manipulator, by considering the position and orientation of the end-effector. In peg-in-hole scenarios, perception and modeling inaccuracies often compromise success, requiring the robot to leverage smart control strategies. Here, we propose to apply ergodic control to facilitate the insertion by letting the robot explore around the hole location in the 6D state space of the end-effector. In this application, we rely on human demonstrations to specify the distribution that the robot should use for an ergodic exploration.

The main contribution of this work is an algorithm for SMC to generate ergodic exploration in multidimensional spaces, which was previously considered to be an intractable problem. In particular, we improve the state-of-the-art by proposing:

- Fast ways to compute the Fourier coefficients of multivariate functions, a well-known bottleneck in the ergodic control literature;
- The use of tensor train to exploit the inherent low-rank structure in the problem, which is used to overcome the curse of dimensionality in both real-time computation and storage requirements and facilitate implementation of ergodic control online on robotic systems.

The proposed ergodic control algorithm paves the way for two additional contributions in robotics. Particularly, we:

- Extend ergodic control to peg-in-hole tasks solved with an online policy, with a novel, principled and theoretically-grounded exploratory strategy for insertion tasks that does not rely on specialized sensors but only on human demonstrations;
- Provide a formal way to perform ergodic exploration in orientation by relying on the $\mathcal{S}^3$ Riemannian manifold.

To the best of our knowledge, this is the first time ergodic control is implemented online on a physical robot for exploration in dimension greater than 2. Note that the strategies to mitigate the curse of dimensionality introduced in this chapter have the potential to be applied to many other applications in robotics to address real-time computation and limited storage requirements. Similarly, the proposed control strategy is not limited to manipulation applications, and can be extended to other robotics scenarios requiring high-dimensional coverage (e.g. 3D-object modeling, 6D surveillance).[1] Finally, even though

[1]The proposed algorithm has been numerically evaluated for state space of up to 15 dimensions.

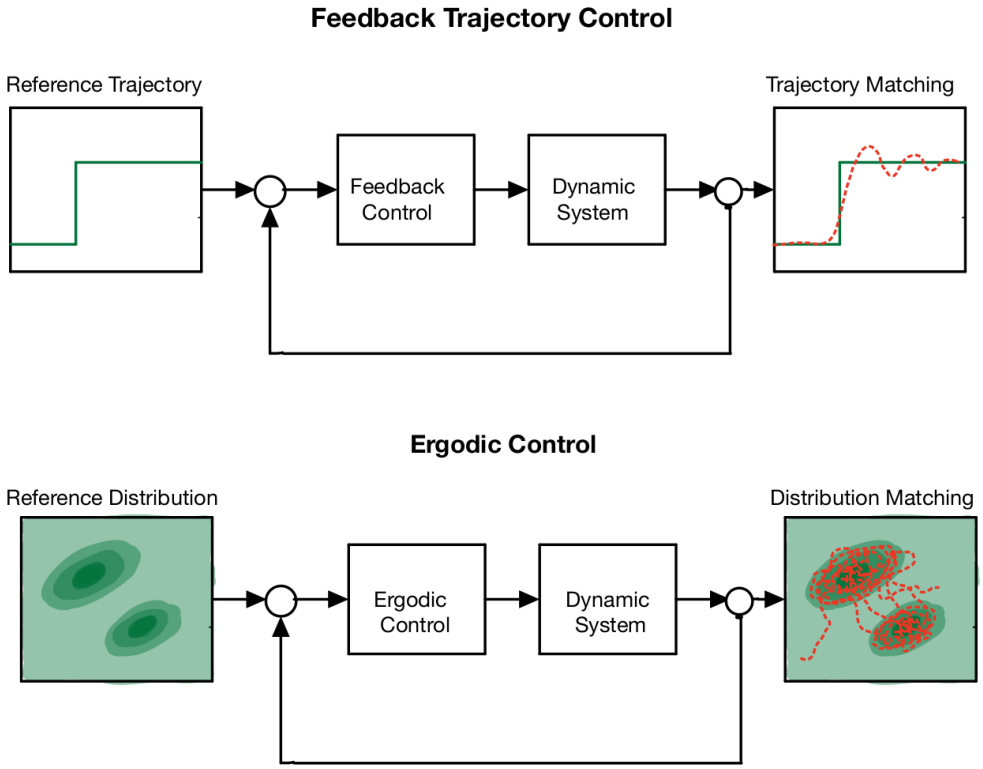


Figure 3.1: In contrast to feedback trajectory control the goal of the ergodic control is to cover a reference distribution.

we rely on human demonstrations to obtain the reference distribution, this reference can in practice be specified/learned in different ways (e.g. from sensor uncertainty models).

The chapter is organized as follows: Section 3.2 gives a literature survey on ergodic control and control strategies for insertion tasks. In Section 3.3, the mathematical formulation of ergodic control is described, where the underlying challenges of the algorithm are outlined. In Section 3.4, we propose low-rank approximation using the tensor train as a solution for multidimensional ergodic exploration. In Section 3.5, we evaluate the proposed algorithm in simulation. Lastly, in Section 3.6 we showcase the results of our approach in a peg-in-hole insertion task using a torque-controlled 7-axis Franka Emika robot.

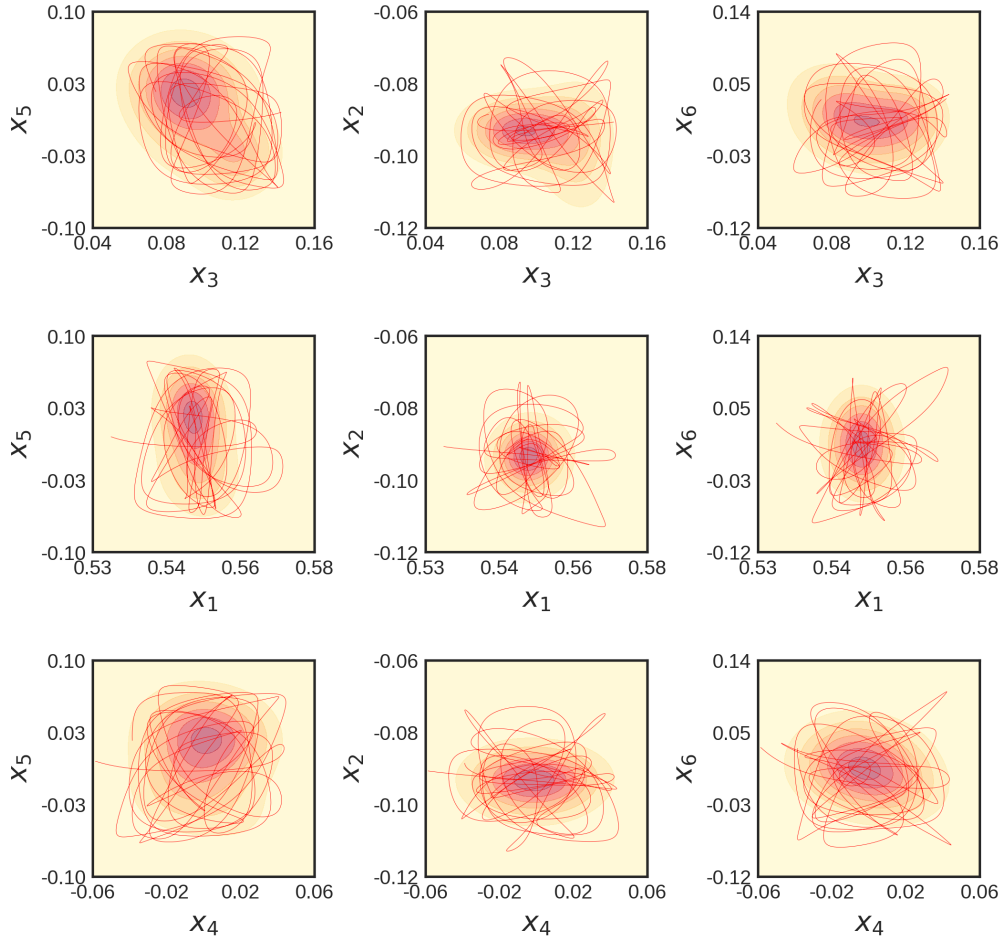


Figure 3.2: The 6D robot trajectory generated by ergodic controller offline for the insertion task.

3.2 Motivation and Related Work

3.2.1 Challenges in Ergodic Control

A solution to ergodic control was originally proposed by [30] using Spectral Multiscale Coverage (SMC) in the form of a feedback control law designed for multi-agent systems, with an objective defined so that the agents trajectories cover a reference probability distribution. Here, the system considered is a point-mass system. The control policy is obtained by solving an optimization problem with an ergodic metric as the cost function (see Section 3.3). The ergodic metric compares the Fourier series coefficients associated to the spatial reference distribution and the trajectory evolution of the system.

Although other possible choices of basis functions would be interesting to investigate (e.g. wavelets), the Fourier transform holds essential properties that are relevant to the considered problem. The use of Fourier series within ergodic control, including their links to cosine basis functions, and their properties for reference distributions in the form of mixtures of Gaussians is discussed in [33].

The ergodic metric can be used as a starting point to design other forms of controllers. For example, ergodic controllers have been proposed using nonlinear dynamical systems and finite control horizons [29], using projection-based trajectory optimization [34], or using hybrid systems theory [32]. An overview of these methods with finite control horizon can be found in [35].

The main drawback of these methods is that they suffer from the curse of dimensionality (see Section 3.3) when the dimension of the state space for ergodic exploration increases. This is due to the computational complexity and storage demanded in working with the ergodic metric and the control policy derived from it. For low dimensional exploration tasks (2D), Dressel and Kochenderfer used supervised learning to reduce the computational burden [36]. However, it does not scale to higher dimensional problems. Several authors deviated from the approach used in SMC to tackle this limitation. In [37] and [38], the authors relied on a different ergodic metric based on a Kullback-Leibler (KL) divergence measure for finite sensor footprint, where the control policy is obtained using sampling-based techniques. Here, the ergodic metric (KL-divergence) is approximated using the samples from the reference distribution. Sampling-based methods avoid the curse of dimensionality but they can still be computationally expensive to address the real-time computational requirements of robotics systems. Moreover, the performance of the method is heavily dependent on the quality of the samples obtained, which is hard to assess. While most sampling-based methods generate the ergodic trajectory offline, its online implementation on real robots, which is the focus of the current chapter, is still a challenging problem. Based on [37], Abraham *et al.* provide an online version of ergodic control with KL-divergence as ergodic metric [38], at the expense of potentially losing ergodicity in the exploration (e.g., by limiting the search to high density regions).

This chapter keeps the original methodology (SMC) proposed by [30], which is the foundation of most literature on ergodic control, and which has the advantage of providing closed form solutions for many of the commonly used models of dynamical systems (kinematics-based) [31], while providing multi-scale coverage behaviour. To do so, we propose a solution based on tensor train (TT) representation to overcome the curse of dimensionality. The proposed algorithm has intuitive hyperparameters that can be adjusted to address the storage and computational constraints of the application.

3.2.2 Challenges in Peg-in-hole Insertion Task

Peg-in-hole insertion is a typical and important problem in robotics. Many strategies to solve this problem depend on expensive force and torque sensors [39, 40]. Sensorless strategies [41, 42, 43], on the other hand, rely only on the state of the end-effector and provide a low cost solution. However, most of the sensorless strategies depend either on a predetermined trajectory that the robot end-effector needs to follow [43], or a full modeling of the insertion behavior [41, 42]. In [43], insertion is treated as a 3D trajectory tracking problem (2D position and 1D orientation of the end-effector), where the reference trajectory for the robot end-effector is generated offline by using a coverage strategy inspired by ergodic control. As we will show, this strategy fails for the peg-in-hole insertion task considered in this chapter, as a trajectory generated offline is often not possible to track due to obstructions from the surface of the hole and the peg. To address this challenge, our approach instead formulates the coverage problem in an online manner, with exploration in the full 6D state space of the robot end-effector (3D position and 3D orientation).

In order to handle the exploration in orientation jointly with the position, we extend the control strategy to Riemannian manifolds by modeling the probability distribution of orientations, see [44, 45] for details. Subsequently, we use an online implementation of ergodic control as the solution for coverage. The algorithm proposed in this chapter allows us to run the ergodic controller online on the robot for 6D insertion tasks.

3.3 Problem Definition and Background

In this section we lay out the mathematical background of our contribution.

Ergodic control considers a point-mass dynamical system whose trajectory evolves such that its time-averaged statistics matches a desired reference probability distribution. In the method proposed originally in [30], the problem reduces to minimizing a cost function called ergodic metric, evaluating the distance between the Fourier coefficients of the reference distribution and that of the time-averaged statistics of the trajectory evolution of the dynamical system.

We assume a bounded d-dimensional rectangular domain: $\Omega_{\boldsymbol{x}} = [0, L_1] \times \cdots \times [0, L_d]$ with $L_i > 0, \forall i \in \{1, \ldots, d\}$. Without loss of generality, we will consider $L_i = L, \forall i \in \{1, ..., d\}$. $\boldsymbol{x}(t) \in \mathbb{R}^d$ represents the trajectory of the dynamical system in the domain. The spatial statistics of the trajectory $\boldsymbol{x}(t)$ is defined as the fraction of time spent by the dynamical

system at each point of the domain:

$$C_t(\boldsymbol{x}) = \frac{1}{t}\int_{\tau=0}^{t} \delta\Big(\boldsymbol{x}(\tau) - \boldsymbol{x}\Big)d\tau,$$

where δ is the Dirac delta function, and $\boldsymbol{x} = (x_1, \ldots, x_d) \in \Omega_{\boldsymbol{x}}$ is a point in the domain.

Let $P(\boldsymbol{x})$ be the reference probability distribution for the exploration defined on $\Omega_{\boldsymbol{x}}$. The goal of ergodic control is to match the time-averaged spatial statistic $C_t(\boldsymbol{x})$ with the spatial distribution $P(\boldsymbol{x})$. The idea is to choose $K \in \mathbb{Z}^+$ orthonormal Fourier basis functions[2] satisfying the Neumann boundary conditions on the boundary of $\Omega_{\boldsymbol{x}}$: ϕ_k, $\forall k \in \{1, \ldots, K\}$, for each variable x, which is then organized as $\boldsymbol{\phi}(x) = (\phi_1(x), \ldots, \phi_K(x)) \in \mathbb{R}^K$. Although the results that follow apply to any such choice of basis function, we will use $\phi_k(x) = \cos\Big(\frac{2\pi(k-1)x}{L}\Big)$ for numerical evaluation, see [33] for details. Then, orthonormal Fourier basis functions for $\Omega_{\boldsymbol{x}}$ can be obtained by the elements of the d-th order tensor formed by the outer product of these vectors: $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{\phi}(x_1) \circ \cdots \circ \boldsymbol{\phi}(x_d) \in \mathbb{R}^{K \times \cdots \times K}$. With respect to this basis, the Fourier coefficients (cosine transforms) of $P(\boldsymbol{x})$ can be represented by a d-th order tensor $\hat{\boldsymbol{\mathcal{W}}}$. For a given index $\boldsymbol{k} = (k_1, \ldots, k_d) \in \mathcal{K}$, we have $\boldsymbol{\Phi}_{\boldsymbol{k}}(\boldsymbol{x}) = \phi_{k_1}(x_1) \times \cdots \times \phi_{k_d}(x_d)$, and $\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}}$ represents the Fourier coefficient w.r.t. the basis $\boldsymbol{\Phi}_{\boldsymbol{k}}$, namely

$$\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}} = \int_{x_1=0}^{L} \cdots \int_{x_d=0}^{L} P(\boldsymbol{x})\ \boldsymbol{\Phi}_{\boldsymbol{k}}(\boldsymbol{x})\ dx_1 \ldots dx_d. \tag{3.1}$$

The ergodic metric is then defined as

$$\xi(t) = \sum_{\boldsymbol{k} \in \mathcal{K}} \boldsymbol{\Lambda}_{\boldsymbol{k}} \Big(\boldsymbol{\mathcal{W}}_{\boldsymbol{k}}(t) - \hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}}\Big)^2, \tag{3.2}$$

where $\boldsymbol{\Lambda}_{\boldsymbol{k}} = (1 + \|\boldsymbol{k}\|^2)^{-\frac{d+1}{2}}$ are the weights for different frequencies, $\mathcal{K} = \{\boldsymbol{k} = (k_1, \ldots, k_d) : k_i \in \{1, \ldots, K\}\}$ and K is a sufficiently large positive integer. This way, higher priority is given to lower frequency contents of the reference distribution (i.e., exploration of large scale features), hence resulting in a multi-scale exploration behavior. $\boldsymbol{\mathcal{W}}(t)$ is the Fourier coefficients for the spatial statistics of the trajectory evolution $\boldsymbol{x}(t)$ at time t (i.e., of $C_t(\boldsymbol{x})$), which is given by

$$\boldsymbol{\mathcal{W}}(t) = \frac{1}{t}\int_{\tau=0}^{t} \boldsymbol{\Phi}(\boldsymbol{x}(\tau))d\tau. \tag{3.3}$$

[2] In general, we can choose a different number of basis functions K_i for each dimension $i \in \{1, \ldots, d\}$. Without loss of generality, we will assume $K_i = K, \forall i$.

The ergodic control objective is $\lim_{t\to\infty} \xi(t) = 0$. For a dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ where $\boldsymbol{x} \in \mathbb{R}^d$, we want the ergodic dynamics w.r.t. the evolution of the states $\boldsymbol{x}(t) \in \Omega_{\boldsymbol{x}} \subset R^d$. For infinitesimal control horizon, the solution for first-order systems is given as (see [30, 33] for details)

$$\begin{aligned} \dot{x}_i(t) &= \alpha \frac{b_i(t)}{\|\boldsymbol{b}(t)\|}, i \in \{1, \dots, d\}, \\ \text{with} \quad b_i(t) &= \sum_{\boldsymbol{k}\in\mathcal{K}} \boldsymbol{\Lambda}_{\boldsymbol{k}} \Big(\boldsymbol{\mathcal{W}}_{\boldsymbol{k}}(t) - \hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}} \Big) \nabla_i \boldsymbol{\Phi}_{\boldsymbol{k}} \Big(\boldsymbol{x}(t) \Big), \\ \boldsymbol{b} &= (b_1, \dots, b_d), \end{aligned} \tag{3.4}$$

$$\nabla_i \boldsymbol{\Phi}\Big(\boldsymbol{x}(t)\Big) = \boldsymbol{\phi}(x_1) \circ \cdots \circ \frac{\partial \boldsymbol{\phi}(x_i)}{\partial x} \circ \cdots \circ \boldsymbol{\phi}(x_d), \tag{3.5}$$

where $\alpha > 0$ is a small real number.

For a fully actuated system with point-mass dynamics $\dot{\boldsymbol{x}} = \boldsymbol{u}$, with $\boldsymbol{u} = (u_1, \dots, u_d)$ and maximum velocity $u_{\max}$, the control commands $u_i, i \in \{1, \dots, d\}$ that minimize the ergodic metric are given by

$$u_i(t) = u_{\max} \frac{b_i(t)}{\|\boldsymbol{b}(t)\|}.$$

Similar expressions for control policies are available for other types of dynamical systems, such as second-order point-mass systems (acceleration command), or first order and second order Dubin's car models, see e.g. [30, 31]. The controller can also be extended to other nonlinear dynamic systems, see e.g. [35, 34, 32]. We demonstrate our approach using the simple point-mass system, as it captures the key challenges in scaling the ergodic control algorithm to higher-dimensional exploration, and because it remains a classical choice for ergodic exploration [31, 46].

The ergodic control algorithm (see Algorithm 1) is simple but it suffers from the curse of dimensionality when the dimension d of the exploration domain $\Omega_{\boldsymbol{x}}$ increases, which typically limits the use of the algorithm to problems of 2 or 3 dimensions. This is a drawback since many applications, in practice, are of dimensions $d > 3$. For example, the end-effector of a robot manipulator has 6 DOF (position and orientation). A naive implementation of the above algorithm for an exploration in the task space of a manipulator is then not feasible in practice.

Algorithm 2 Ergodic control algorithm

Input: d, L, K, $u_{\max}$, T, and $P(\boldsymbol{x})$
Preprocessing:
Compute $\hat{\boldsymbol{\mathcal{W}}}$ (evaluate K^d multivariate integrals with (3.1))
Compute $\boldsymbol{\Lambda}$ (K^d function evaluations)
Initialise: $t = 0$, dt (time step), $\boldsymbol{x}(0)$, $\boldsymbol{\mathcal{W}}(0)$, $\boldsymbol{u}(0)$
 # Control Loop
while $t < T$ **do**
  $t \leftarrow t + dt$
  Update $\boldsymbol{x}(t)$
  Update $\boldsymbol{\mathcal{W}}(t)$ (use numerical integration)
  Compute $\nabla_i \boldsymbol{\Phi}\Big(\boldsymbol{x}(t)\Big), \forall i \in \{1, \ldots, d\}$
  for i=1,. . . ,d **do**
    $b_i(t) = \sum_{\boldsymbol{k} \in \mathcal{K}} \boldsymbol{\Lambda}_{\boldsymbol{k}} \Big(\boldsymbol{\mathcal{W}}_{\boldsymbol{k}}(t) - \hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}}\Big) \nabla_i \boldsymbol{\Phi}_{\boldsymbol{k}} \Big(\boldsymbol{x}(t)\Big)$
  end for
  Update $\boldsymbol{u}(t) = u_{\max} \frac{\boldsymbol{b}(t)}{\|\boldsymbol{b}(t)\|}$
end while
Note: In the control loop, each of the variables $\hat{\boldsymbol{\mathcal{W}}}, \boldsymbol{\mathcal{W}}(t), \boldsymbol{\Lambda}, \nabla_i \boldsymbol{\Phi}\Big(\boldsymbol{x}(t)\Big)$ need $\mathcal{O}(K^d)$ floating-point elements and each binary operation involving them has computational complexity $\mathcal{O}(K^d)$.

The main challenges in the algorithm are:

1. *Computation and storage of the Fourier series coefficients of the reference distribution $\hat{\boldsymbol{\mathcal{W}}}$:*

   This requires evaluation of the multidimensional integral in (3.1) K^d times to completely determine $\hat{\boldsymbol{\mathcal{W}}}$. Although this is a preprocessing step, the computational complexity involved and the storage requirement make it infeasible for engineering applications.

2. *Computation and storage of $\boldsymbol{\Lambda}$:*

   This is a preprocessing step. The computation of each element $\boldsymbol{\Lambda}_{\boldsymbol{k}}$ is straightforward, but the number of function evaluations to find the complete tensor $\boldsymbol{\Lambda}$ and the required storage grow exponentially (i.e., K^d).

3. *Real-time implementation of the control loop:*

   The control loop will be very slow as the algebraic operations (such as addition, element-wise product, summation, Frobenius norm of tensors) are more time-consuming as the order of the tensors involved in computing the control policy increases. So, an online implementation of the control loop may not be possible.

To give an example of the computation time involved in ergodic control, we used the Python software implementation of ergodic control described in [30][3]. By using $K = 10$ for the Fourier series coefficients and a spherical Gaussian of variance 0.01 at the center of $\Omega_{\boldsymbol{x}}$ with $L = 1$ as the reference probability distribution, the preprocessing time to compute the coefficients. Here, each coefficient is computed independently. $\hat{\boldsymbol{\mathcal{W}}}$ is approximately 16s in 2D, 5400s in 3D, and 2300000s in 4D ($d = 4$) with the multivariate integration (3.1) evaluated using the Python package *scipy.integrate.nquad*.[4] Note that the number of elements to be stored in these cases is 10^d, and for larger d (such as $d > 6$) it is highly likely that memory/storage requirement for each of the multidimensional arrays involved exceeds the limit. Moreover, the average time taken per control loop in the above setting for $d = 3, 4, 5, 6, 7$ are 7×10^{-4}s, 1×10^{-3}s, 4×10^{-3}s, 6×10^{-2}s, and 8×10^{-1}s correspondingly. For $d = 2$, in [36], Dressel et al use a supervised learning approach based on convolutional neural networks for the fast computation of the Fourier coefficients. For higher dimensional problems, as we will see next, the above-mentioned challenges can be solved using TT. Using the proposed solution, for the above reference distribution with $d = 7$, $\hat{\boldsymbol{\mathcal{W}}}$ can be represented using approximately Kd parameters and it can be computed in less than 2×10^{-3}s and the average control loop takes less than 1×10^{-3}s. Also, for other reference distributions and larger d, the pre-processing step and the control loop can be processed very fast.

3.4 Ergodic Control using Tensor Train

In this section, we give details of the solution proposed in this chapter to overcome the challenges mentioned in Section 3.3. Additionally, as part of our proposed strategy for 6D exploration in manipulation tasks, we propose a Riemannian manifold extension to allow ergodic exploration for orientation data represented as unit quaternions.

We use the TT representation for the variables involved in the algorithm, namely $\hat{\boldsymbol{\mathcal{W}}}, \boldsymbol{\mathcal{W}}(t), \boldsymbol{\Lambda}, \nabla_i \boldsymbol{\Phi}\Big(\boldsymbol{x}(t)\Big)$. By its definition in (3.5), $\nabla_i \boldsymbol{\Phi}\Big(\boldsymbol{x}(t)\Big), \forall i \in \{1, \ldots, d\}$ can be represented as a rank-1 TT. Its TT cores are $\Big(\boldsymbol{\phi}(x_1), \ldots, \boldsymbol{\phi}(x_{i-1}), \frac{\partial \boldsymbol{\phi}(x_i)}{\partial x}, \boldsymbol{\phi}(x_{i+1}), \ldots, \boldsymbol{\phi}(x_d)\Big)$. Similarly, $\boldsymbol{\Phi}(\boldsymbol{x})$ by its definition can be represented as a rank-1 TT, with TT cores $\Big(\boldsymbol{\phi}(x_1), \ldots, \boldsymbol{\phi}(x_d)\Big)$. These variables, as described below, can be compactly represented in TT format, within a desired accuracy. Since all the operations to find the control policy, at each step of the control loop, are done on variables in TT format, the computational complexity is significantly reduced.

$\boldsymbol{\Lambda}$ can be found using the TT-cross approximation (see Section 2.8). Because of the

[3]The specifications of the used computing system are given in Section 3.5.

[4]https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

involved symmetry (see the definition of $\mathbf{\Lambda}$), the resultant tensor in TT format is of very low rank. Maximal TT rank of 2 accurately captures the tensor with error less than 10^{-2} in Frobenius norm and it can be computed in a fraction of a second (for d<15). We also represent $\boldsymbol{\mathcal{W}}(t)$ in TT format and use time integration in TT format [47] to compute it efficiently at each iteration of the control loop. Due to integration, as the TT-rank of this tensor may increase in an unbounded manner over time, we specify an upper bound (a hyperparameter) on its TT-rank. If other integration schemes are used, one can periodically use TT-rounding to cut off the TT-rank.

Finding $\hat{\boldsymbol{\mathcal{W}}}$ is a preprocessing step for the algorithm, and it is the most challenging part as it requires to evaluate K^d times a d-dimensional integral given by (3.1), if implemented naively. By using the properties of the TT format, the Fourier coefficients can be computed without having to perform any multidimensional integration directly. By exploiting the properties of Gaussian quadrature rule for the integration of multivariate functions, we derive an analytical expression for the Fourier coefficients $\hat{\boldsymbol{\mathcal{W}}}$ in the TT representation for arbitrary functions. In this method, we exploit the smoothness of the reference probability distribution $P(\boldsymbol{x})$ to find $\hat{\boldsymbol{\mathcal{W}}}$ indirectly by evaluating $P(\boldsymbol{x})$ at a few points in its domain.

3.4.1 Finding the Fourier Series Coefficients

In this section, we provide an analytical expression for the Fourier coefficients of a smooth but arbitrary distribution. The proof is inspired by [48] where they have used separability structure in TT representation to find polynomial approximations of multivariate functions. A similar strategy has been used in [19] to evaluate high-dimensional integration of smooth functions. We use this strategy to find an analytical expression for the Fourier coefficients of functions directly in TT representation. The proof relies on quadrature rules (see Appendix A.1 for the details) for numerical integration of multivariate functions. There are many possible options to choose the quadrature rule depending on the type of function $P(\boldsymbol{x})$ to be integrated. The following result applies to any choice of quadrature rule, however, for simplicity, we use Gaussian quadrature rule (G-Q) in this chapter.

The idea is to find the TT representation $\boldsymbol{\mathcal{P}}$ of the multivariate function $P(\boldsymbol{x})$ evaluated at the discretization induced from the quadrature rule. Then, as we will see below, the TT representation of $\hat{\boldsymbol{\mathcal{W}}}$ can be obtained directly using $\boldsymbol{\mathcal{P}}$.

Let $x_j \in \mathbb{R}$ be the discretization points of the interval $[0, L]$ and α_j be the weights obtained from the quadrature rule, where $j \in \{1, \ldots, N\}$, with N representing the specified degree of approximation of the function. Then, we can discretize the domain $\Omega_{\boldsymbol{x}}$ at $\boldsymbol{x_j} = (x_{j_1}, \ldots, x_{j_d})$, with $\boldsymbol{j} \in \mathcal{J}$, where $\mathcal{J} = \{\boldsymbol{j} = (j_1, \ldots, j_d) : j_i \in \{1, \ldots, N\}\}$ is

the index set. Let $\boldsymbol{\mathcal{P}}$ be the tensor formed by evaluating the reference distribution $P(\boldsymbol{x})$ at the discretization points, so that $\boldsymbol{\mathcal{P}}_{\boldsymbol{j}} = P(\boldsymbol{x}_{\boldsymbol{j}}), \forall \boldsymbol{j} \in \mathcal{J}$.

Let $(\boldsymbol{\mathcal{P}}^1, \boldsymbol{\mathcal{P}}^2, \ldots, \boldsymbol{\mathcal{P}}^d)$ be the TT cores of $\boldsymbol{\mathcal{P}}$ in its TT representation[5], so that for $\boldsymbol{j} = (j_1, \ldots, j_d) \in \mathcal{J}$ we have,

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{j}} = \boldsymbol{\mathcal{P}}^1_{:,\, j_1,\, :}\, \boldsymbol{\mathcal{P}}^2_{:,\, j_2,\, :} \cdots \boldsymbol{\mathcal{P}}^d_{:,\, j_d,\, :},$$

then the TT cores of $\hat{\boldsymbol{\mathcal{W}}}$ are (see Appendix A.1 for the proof)

$$\hat{\boldsymbol{\mathcal{W}}}^i_{:,\, k,\, :} = \sum_{j=1}^{N} \alpha_j\, \boldsymbol{\mathcal{P}}^i_{:,\, j,\, :}\, \phi_k(x_j), \qquad \begin{matrix} \forall k \in \{1, \ldots, K\}, \\ \forall i \in \{1, \ldots, d\}, \end{matrix} \tag{3.6}$$

so that

$$\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}} = \hat{\boldsymbol{\mathcal{W}}}^1_{:,\, k_1,\, :} \cdots \hat{\boldsymbol{\mathcal{W}}}^d_{:,\, k_d,\, :}, \qquad \forall \boldsymbol{k} \in \mathcal{K}.$$

Thus, we can compute the Fourier coefficients $\hat{\boldsymbol{\mathcal{W}}}$ by only investing in computing the TT decomposition $\boldsymbol{\mathcal{P}}$ of the discretized reference distribution. This can be done in $\mathcal{O}(Ndr^2)$ function evaluations of $P(\boldsymbol{x})$ using the TT-cross algorithm (see Section 2.8). The TT-rank of the tensor $\hat{\boldsymbol{\mathcal{W}}}$ will be same as that of the TT-rank of $\boldsymbol{\mathcal{P}}$. For a smooth reference distribution, $\boldsymbol{\mathcal{P}}$ will have low TT-rank. This is a tremendous saving in computing the Fourier coefficients $\hat{\boldsymbol{\mathcal{W}}}$, and thus it overcomes the curse of dimensionality. The TT based algorithm for ergodic control is outlined in Algorithm 2.

3.4.2 Ergodic Control on Riemannian Manifolds

Most manipulation tasks concern the full robot end-effector pose, which includes both its position and orientation. Hence, when designing exploration strategies for manipulation it is desirable to consider both. In the case of position, the ergodic control formulation in Section 3.4.1 can be directly applied. However, since orientation data do not lie on a Euclidean space, exploration in orientation requires a special mathematical treatment. In this section, we extend ergodic control to handle data on a Riemannian manifold [44, 45], particularly the orientation manifold $\mathcal{S}^3$.

The orientation of the robot end-effector can be represented by a unit quaternion $\boldsymbol{q} \in \mathcal{S}^3$, comprised of a scalar part $q_s \in \mathbb{R}$ and a vector part $\boldsymbol{q}_v \in \mathbb{R}^3$ such that $\boldsymbol{q} = [q_s \;\; \boldsymbol{q}_v^\top]^\top$. For any point on the manifold $\boldsymbol{g} \in \mathcal{S}^3$ there exists a tangent space $\mathcal{T}_{\boldsymbol{g}}\mathcal{S}^3$ in which standard Euclidean methods can be applied to orientation. The function that maps a quaternion

[5] This can be obtained using TT-cross: $\boldsymbol{\mathcal{P}} = \text{TT-Cross}(P(\boldsymbol{x}), \epsilon)$.

Algorithm 3 Ergodic Control using TT

Input: d, L, K, $u_{\max}$, T, and $P(\boldsymbol{x})$
Pre-Processing:
Compute $\boldsymbol{\mathcal{P}}$ using TT-cross
Find $\hat{\boldsymbol{\mathcal{W}}}$ using (3.6)
Compute $\boldsymbol{\Lambda}$ using TT-cross
TT-rounding of $\hat{\boldsymbol{\mathcal{W}}}$ (remove low-energy contents)
Initialise: dt (time step), $\boldsymbol{x}(0)$, $\boldsymbol{\mathcal{W}}(0)$ in TT format, $\boldsymbol{u}(0)$, and $t = 0$
 # Control Loop
while $t < T$ **do**
  $t \leftarrow t + dt$
  Update $\boldsymbol{x}(t)$ (time integration)
  Update $\boldsymbol{\mathcal{W}}(t)$ (Use TT time integration [47] with a fixed maximal TT-rank. Alternatively, use numerical integration such as Euler integration followed by TT-rounding)
  Compute $\nabla_i \boldsymbol{\Phi}\Big(\boldsymbol{x}(t)\Big), i \in \{1, \ldots, d\}$ (rank-1 TT)
  # Using algebraic operations in TT
  for i=1,. . . ,d **do**
    $b_i(t) = \sum_{\boldsymbol{k} \in \mathcal{K}} \boldsymbol{\Lambda}_{\boldsymbol{k}} \Big(\boldsymbol{\mathcal{W}}_{\boldsymbol{k}}(t) - \hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}}\Big) \nabla_i \boldsymbol{\Phi}_{\boldsymbol{k}} \Big(\boldsymbol{x}(t)\Big)$
  end for
  Update $\boldsymbol{u}(t) = u_{\max} \frac{\boldsymbol{b}(t)}{\|\boldsymbol{b}(t)\|}$
end while
Note: In the control loop, the memory of each variable and the computational complexity of each algebraic operation has complexity grow linearly with d. Thus it avoids the curse of dimensionality. In particular, the computation of $b_i(t)$ requires a subtraction, a Hadamard product and an inner product involving tensors in TT format, hence it can be computed efficiently.

$\boldsymbol{q}$ from the manifold to a tangent space is called the *logarithmic map* and is given by

$$\mathrm{Log}(\boldsymbol{q}) = \begin{cases} \mathrm{acos}_*(q_s)\frac{\boldsymbol{q}_v}{\|\boldsymbol{q}_v\|}, & q_s \neq 1 \\ [0,0,0]^\top, & q_s = 1 \end{cases}, \tag{3.7}$$

where $\mathrm{acos}_*(\cdot)$ is a modified arc-cosine function [44]. Equation (3.7) maps $\boldsymbol{q}$ to the tangent space of the manifold origin. The mapping of $\boldsymbol{q}$ to the tangent space of an arbitrary point $\boldsymbol{g}$ is given by [44]

$$\mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{q}) = \mathrm{Log}(\bar{\boldsymbol{g}} * \boldsymbol{q}), \tag{3.8}$$

where $\bar{(\cdot)}$ and $*$ denote the quaternion conjugate and product, respectively. The logarithmic map represents a unit quaternion $\boldsymbol{q}$ as a 3-dimensional Euclidean vector $\boldsymbol{v} \in \mathbb{R}^3$. Quaternions can be retrieved from the tangent space through the *exponential map*

$$\mathrm{Exp}(\boldsymbol{v}) = \begin{cases} \left[\cos(\|\boldsymbol{v}\|),\ \sin(\|\boldsymbol{v}\|)\frac{\boldsymbol{v}^\top}{\|\boldsymbol{v}\|}\right]^\top, & \|\boldsymbol{v}\| \neq 0 \\ [1,0,0,0]^\top, & \|\boldsymbol{v}\| = 0 \end{cases}, \tag{3.9}$$

which, analogously to (3.8), maps from an arbitrary tangent space $\mathcal{T}_{\boldsymbol{g}}\mathcal{S}^3$ back to the manifold through

$$\mathrm{Exp}_{\boldsymbol{g}}(\boldsymbol{v}) = \boldsymbol{g} * \mathrm{Exp}(\boldsymbol{v}). \tag{3.10}$$

Given a set of unit quaternions (e.g. obtained from demonstrations), we formulate orientation-ergodic control by modeling their distribution in the tangent space of their mean. For M end-effector orientations $\{\boldsymbol{q}_i\}_{i=1}^M$, the mean on the manifold $\boldsymbol{\mu} \in \mathcal{S}^3$ is computed iteratively with (see [44, 45] for details)

$$\boldsymbol{v} = \frac{1}{M}\sum_{i=1}^{M} \mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{q}_i), \qquad \boldsymbol{\mu} \leftarrow \mathrm{Exp}_{\boldsymbol{\mu}}(\boldsymbol{v}). \tag{3.11}$$

All quaternions in the dataset can thus be mapped to the tangent space of the mean through $\{\boldsymbol{v}_i\}_{i=1}^M = \{\mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{q}_i)\}_{i=1}^M$, allowing the proposed ergodic exploration (Algorithm 2) to be performed in $\mathbb{R}^3$, even for orientation. As desired orientations are computed, in the tangent space, at each time step by $\hat{\boldsymbol{v}}(t) = \boldsymbol{v}(t) + \boldsymbol{u}(t)dt$, the exponential map generates a desired unit quaternion for the robot to track, using

$$\hat{\boldsymbol{q}}(t) = \mathrm{Exp}_{\boldsymbol{\mu}}(\hat{\boldsymbol{v}}(t)). \tag{3.12}$$

In this way, ergodic control for end-effector poses is formulated as a 6D problem, where the first 3 dimensions correspond to position and the last 3 to orientation.

			Gaussian mixture model						Uniform distribution	
			2 components		4 components		6 components			
			With TT	Without TT	With TT	Without TT	With TT	Without TT	With TT	Without TT
5D	# parameters	$\nabla_i \mathbf{\Phi}$	50	10^5	50	10^5	50	10^5	50	10^5
		$\mathbf{\Lambda}$	160	10^5	160	10^5	160	10^5	160	10^5
		$\hat{\mathcal{W}}$	160	10^5	548	10^5	1032	10^5	50	10^5
	Average time per loop		2×10^{-3}	4×10^{-3}	3×10^{-3}	4×10^{-3}	3.6×10^{-3}	4×10^{-3}	2×10^{-3}	4×10^{-3}
	Pre-processing time		0.2	–	0.87	–	2.4	–	33×10^{-3}	–
6D	# parameters	$\nabla_i \mathbf{\Phi}$	60	10^6	60	10^6	60	10^6	60	10^6
		$\mathbf{\Lambda}$	200	10^6	200	10^6	200	10^6	200	10^6
		$\hat{\mathcal{W}}$	200	10^6	695	10^6	1431	10^6	60	10^6
	Average time per loop		3.2×10^{-3}	63×10^{-3}	4.6×10^{-3}	63×10^{-3}	5.4×10^{-3}	63×10^{-3}	3×10^{-3}	63×10^{-3}
	Pre-processing time		30×10^{-3}	–	1.26	–	3.6	–	40×10^{-3}	–
7D	# parameters	$\nabla_i \mathbf{\Phi}$	70	10^7	70	10^7	70	10^7	70	10^7
		$\mathbf{\Lambda}$	240	10^7	240	10^7	240	10^7	240	10^7
		$\hat{\mathcal{W}}$	233	10^7	860	10^7	1801	10^7	70	10^7
	Average time per loop		4.8×10^{-3}	0.8	6.8×10^{-3}	0.8	7.5×10^{-3}	0.8	3.6×10^{-3}	0.8
	Pre-processing time		35×10^{-3}	–	1.53	–	4.9	–	43×10^{-3}	–

Table 3.1: Computational speed and storage requirements in ergodic control for different reference probability distributions with $K = 10$ and $L = 1$m. The components of GMM are spherical Gaussians with 0.005 variance. All the tensors in TT format are approximated with an accuracy of 10^{-2} in the Frobenius norm. The preprocessing time for the naive approach (without using TT) is not given in the table as it is computationally infeasible in the computing device used for the experiment.

3.5 Numerical Evaluation

In this section, we demonstrate the computational efficiency of the TT-based algorithm for ergodic control through simulations. We use Method 2 described in Section 3.4 to compute the Fourier series coefficients. The simulations are performed on a Lenovo Thinkpad personal computer with Intel(R) Core(TM) i7-8565U CPU at 1.80GHz with 24GB RAM. We use ttpy, a Python-based toolbox for working with TT.[6]

A naive implementation without using tensor decomposition techniques would require K^d elements to store each of the tensors: $\hat{\boldsymbol{\mathcal{W}}}$, $\mathbf{\Lambda}$, and $\nabla_i \mathbf{\Phi}\Big(\boldsymbol{x}\Big)$ $(i \in \{1, \ldots, d\})$. However, a TT representation requires less than $4Kd$ elements for $\mathbf{\Lambda}$ (with approximation error 10^{-2} in the Frobenius norm) and only Kd elements to exactly represent $\nabla_i \mathbf{\Phi}\Big(\boldsymbol{x}\Big)$. As $\nabla_i \mathbf{\Phi}\Big(\boldsymbol{x}\Big)$ is a rank-1 tensor with explicit analytical expressions for its TT cores, it can be computed very fast. Computing the weights $\mathbf{\Lambda}$ can be done in a fraction of a second for $d \leq 15$.

The computation and storage of $\hat{\boldsymbol{\mathcal{W}}}$ depends on the smoothness of the reference probability distribution. For the evaluation, we define our reference distribution as an isotropic Gaussian distribution at the centre of the domain with variance 0.015, where we used

[6] https://github.com/oseledets/ttpy

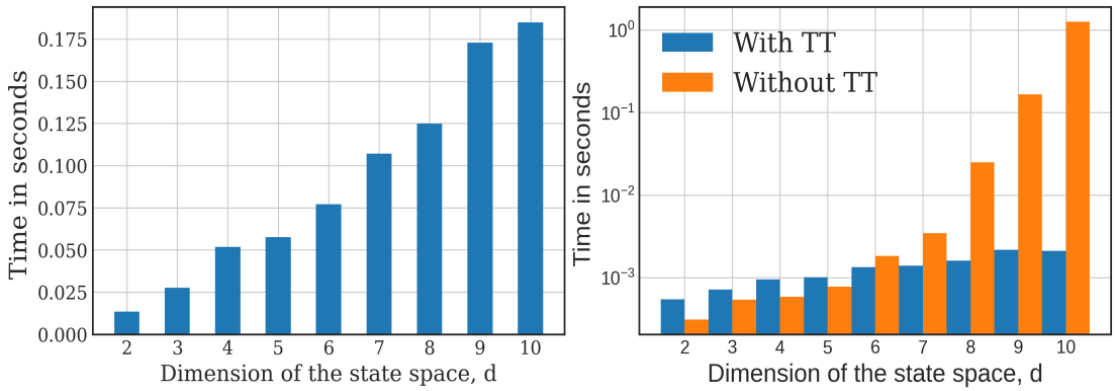


Figure 3.3: Time taken (linear scale) to compute the Fourier coefficients $\hat{\boldsymbol{\mathcal{W}}}$ (**left**) and the average time taken (in log-10 scale) per control loop (**right**) of the ergodic control algorithm using the proposed technique for $K = 5$, $l = 1$, and a reference distribution in the form of an isotropic Gaussian with variance 0.015. In the right figure, the exponential growth in the computational complexity in the control loop can be observed for the standard approach (without using tensor decomposition), whereas the proposed approach avoids the curse of dimensionality. In the left figure, it can be observed that the computational complexity in computing the Fourier series coefficients using the proposed approach tends to grow linearly with d.

$L = 1$, $K = 5$, $N = 10$ and an approximation accuracy of 10^{-2} in the TT representation of $\hat{\boldsymbol{\mathcal{W}}}$ and $\boldsymbol{\Lambda}$. As can be seen in Fig. 3.3, the time taken to compute $\hat{\boldsymbol{\mathcal{W}}}$ grows approximately linearly with d, and it is less than a second for the chosen reference distribution with $d \leq 10$ and the average time taken per control loop increases almost linearly with the number of dimensions d, whereas with a naive implementation (without using TT) the time taken in the control loop grows exponentially with d. The trend remains the same for other reference distributions, see Table 3.1.

The computation of $\boldsymbol{\mathcal{W}}(t)$ in the control loop requires some attention. At each iteration of the control loop, the rank of the tensor $\boldsymbol{\mathcal{W}}(t)$ may keep increasing due to the integration, see (3.3). This could be a problem if the time period of ergodic exploration is very high. So, it is necessary to upper bound the TT-rank of this tensor using TT-rounding with a specified maximal TT-rank. Setting an excessively low value for the maximal rank may lead to convergence issues and specifying a large value for maximal rank slows down the speed of computation of each control loop. Thus, this hyperparameter must be chosen carefully. In the numerical evaluation, our experiments revealed that a maximal rank of $d \cdot r$ for $\boldsymbol{\mathcal{W}}(t)$, where r is maximal TT-rank of $\hat{\boldsymbol{\mathcal{W}}}$, worked well for $d < 15$.

The TT representation allows compact representation of the tensors involved and the storage complexity also grows linearly with d. These properties allow our algorithm to

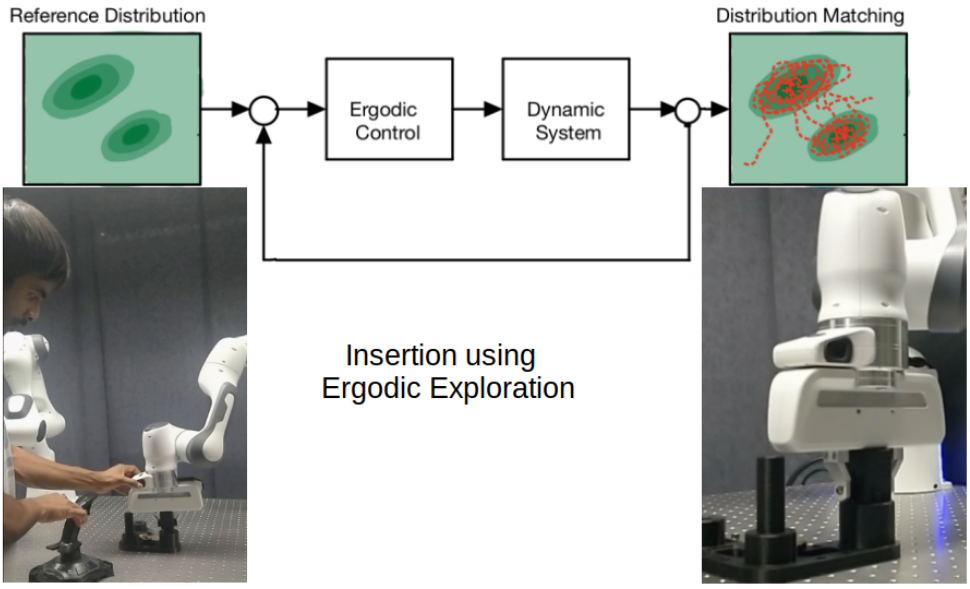


Figure 3.4: We propose Peg-in-hole insertion as an ergodic exploration problem. The reference distribution for the exploration is given by human demonstrations of successful insertions.

be implemented for real-time applications on devices with small memory and limited computational power, which are often the case in robotic systems. Another important property of our algorithm is that the approximation of the tensors involved such as $\hat{\boldsymbol{\mathcal{W}}}$, $\boldsymbol{\mathcal{W}}(t)$ and $\boldsymbol{\Lambda}$ can be controlled precisely using TT-rounding (see Section 2.7). In practice, TT-rounding with accuracy 10^{-2} is observed to be sufficient for all practical purposes considered here. Furthermore, doing these approximations in the spectral domain results in negligible impact on the time domain behaviour of the system, thanks to Parseval's theorem as the approximation can be considered as a noise filter in the ergodic motion. This also provides a convenient trade-off between accuracy of approximation in ergodic exploration and the speed of computation in the control loop.

3.6 Sensorless Peg-in-Hole Insertion using Ergodic Exploration

We evaluate the proposed approach in an insertion task (see Fig. 3.4). We formulate the insertion task as a 6D exploration problem where we simultaneously address the uncertainty about the insertion pose in position (3D) and orientation (3D) in the robot task space. Our method is well suited for peg-in-hole insertion tasks where uncertainties may

arise from several sources, including variable grasps of the peg, unprecise locations of the hole, and unmodeled manufacturing defects of the involved components (gripper fingers, pegs and holes). In the considered experiment, the reference probability distribution for exploration is found using information from human demonstrations. The human demonstrates the key regions for exploration in the state space of the end-effector and we use a Gaussian mixture model (GMM) to model the reference probability distribution based on the datapoints collected during the demonstrations. As a means to intuitively show the effectiveness of our approach, we begin by comparing it to three baselines of exploratory behaviors commonly used in insertion tasks. For this we use a toy example in 2D and 3D.

3.6.1 Simulation experiments

In this section we provide the motivation for using ergodic control for exploration, and its significance for insertion tasks, using simulation of exploration behavior in 2D and 3D.

We use a GMM as the reference distribution in the space $\Omega_{\boldsymbol{x}}$ with $L = 1$m. The GMM is chosen such that it has 6 equally weighted mixture components with its centers placed randomly in the exploration space $\Omega_{\boldsymbol{x}}$ and each component is a spherical Gaussian with variance 0.01. As a first metric, we compute the average time taken to reach a spherical region with volume 0.5% of the volume of $\Omega_{\boldsymbol{x}}$. The spherical region is representative of the target detection region during exploration. For the insertion task, this corresponds to the set of end-effector states at which the peg is inside the hole. For all the trials, we fix a maximum duration of 1000s for detection (i.e., reaching the target region) and the magnitude of the point-mass-system velocity is constant and fixed to $u_{\max} = 0.1$ m/s. For the analysis we choose 10 different GMMs as described above and for each choice of GMM, we conduct 10 trials. For each trial, the center of the target region is chosen by sampling in $\Omega_{\boldsymbol{x}}$ from the reference GMM and the point-mass system starts with the same initial state: $(0.5, 0.5)$ for 2D and $(0.5, 0.5, 0)$ for 3D.

We compare four different exploration strategies:

1. **Strategy 1:** *Ergodic exploration* (proposed approach),
2. **Strategy 2:** *Sampling-based exploration*
3. **Strategy 3:** *Cylindrical spiral*, as a representative of sweeping patterns,
4. **Strategy 4:** *Mixture of ellipsoidal spirals*, as a sweeping pattern customized for GMM.

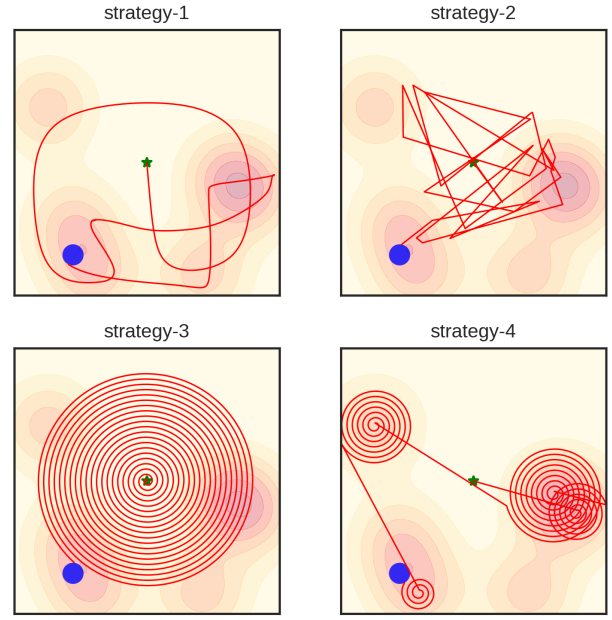
Fig. 3.5 shows an example of exploration behaviors for these different strategies.

In strategy 2, we explore by tracking samples from the GMM sequentially. In this approach, the simulated system tracks GMM samples using a constant speed, with a new sample being computed every time the previous one is reached. Unlike ergodic exploration, such approaches based on sampling from the reference distribution are typically inefficient at handling distributed information [29].

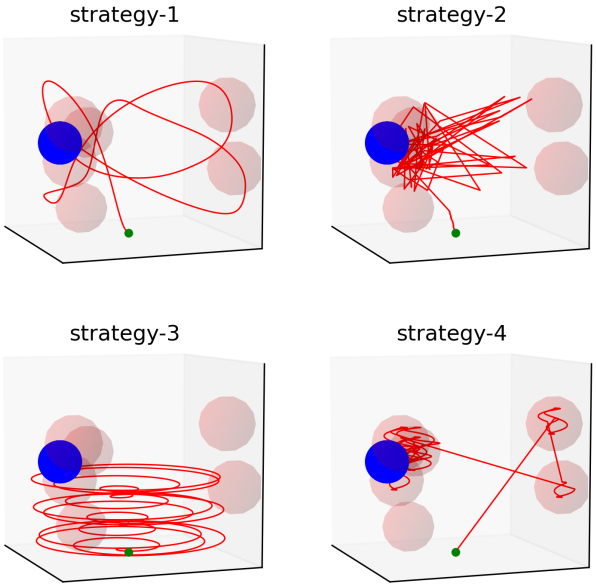
In strategy 3, we use an Archimedean spiral for 2D and its cylindrical extension for 3D (see Fig. 3.5). These are conventionally used in robotics as heuristic search strategies for uniform coverage in 2D and 3D search spaces (i.e. for uniform reference probability distributions) [49] [50]. Strategy 4 is similar to strategy 3 but uses spherical/ellipsoidal spiral trajectories that are customized to sweep the GMM search space. In this approach, the Gaussians are swept in sequence with spherical/ellipsoidal spiral paths starting from the centers of the Gaussian and sweeping the area up to a given number of standard deviations before moving to the next component. These approaches suffer from the curse of dimensionality and perform poorly for $d > 3$. Moreover, they require careful tuning of hyperparameters to generate efficient spiral paths. Furthermore, the resulting trajectories are deterministic and do not consider the stochasticity of target detection. Namely, the trajectory generated by such sweeping pattern passes through a given point in the search space only once. If the measurement system fails to detect the target during its first pass, the strategy has no future possibilities for detection.

The first metric we use for comparison is the average time taken to reach the target region for the first time. Table 3.2 shows the obtained results for 2D and 3D. We see that, on average, ergodic exploration reaches the target region faster due to its multiscale search behavior. Additionally, ergodic control has a 100% success rate. Despite being slower, spiral search is equally successful, but this success comes at the cost of optimally choosing the spiral parameters. This is often cumbersome in practice, especially in higher dimensions and considering that the tolerance of the detection region is often not known with high certainty.

A desirable property for exploration strategies is that the system takes into account the already visited regions to cover the unvisited regions more often. In order to evaluate this property, we consider a second metric: the cumulative average time to reach the target region over several successful attempts. We define this as $\frac{T_c}{c}$, where c is the number of successful attempts and T_c is the cumulative time until successful attempt c. In this evaluation, as soon as the system reaches the boundary of the target region, we re-initialize it to the initial state and repeat this process for a fixed number of times. The results in Fig. 3.6 show that, for ergodic control, the cumulative average decreases



(a)



(b)

Figure 3.5: Example trajectories (red) of the four different exploration strategies to reach an target region (blue sphere) within a reference probability distribution (in this case a GMM) for $d = 2$ (top) and $d = 3$ (bottom). The green point indicates the initial state of the point-mass system. The goal is to reach this target region the information about which is known to the search strategies only through the reference probability distribution.

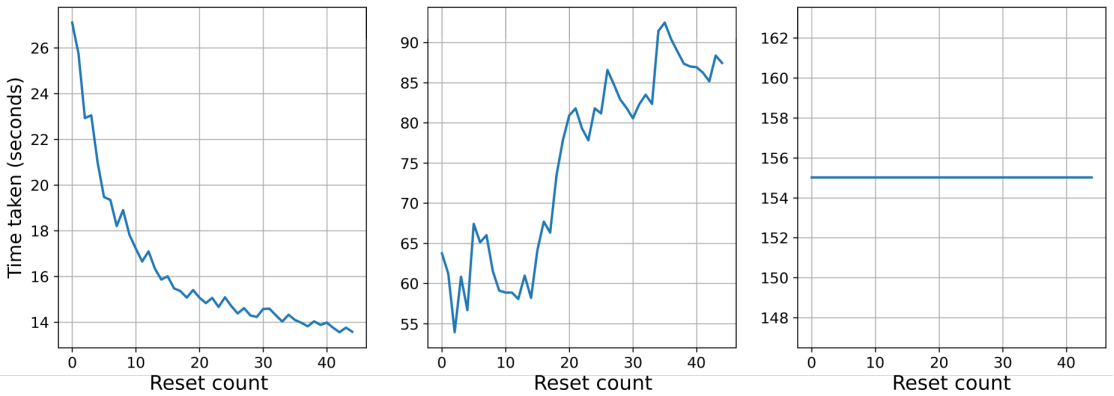


Figure 3.6: Cumulative average time to reach a specified target region (spherical region with 1% of the volume of $\Omega_{\boldsymbol{x}}$ with $d = 3$) with ergodic exploration (**left**), sampling-based strategy (**center**) and spiral movement (**right**). The x-axis represents the number of attempts to reach the target (reset count) and the y-axis represents the cumulative average of the time taken to reach the target at each number of attempts. Every time the target is reached, the system is re-initialized to the same initial state (starting a new attempt). The ergodic controller is aware of the lack of exploration inside the target region and tries to visit it more frequently. The cumulative average therefore converges to the time it takes to go from the initial state directly to the target at every re-initialization.

with the number of successful attempts and converges to a fixed value. This suggests that the ergodic exploration tries to visit the unexplored regions more frequently as the number of attempts increases. In this case, the value that the cumulative average converges to is the time it takes to go from the initial state directly to the target at every re-initialization. This is an essential feature for insertion tasks as the exploration inside the hole (representing successful insertion of the peg) is not easy due to obstacles (e.g., uncertainties and collision of the surface of the hole against the peg) and we need the exploration strategy to drive exploration towards the unexplored region as the time evolves. This is an inherent property of ergodic exploration that the other approaches lack. This property plays a crucial role to cope with the stochasticity involved either in the measurement systems for detection, or the dynamics of the process (e.g., insertion task). To exploit this feature in real robot experiments, it is necessary that the ergodic controller is implemented *online* on the robot manipulator, i.e., that the controller knows the actual observed end-effector states. Our proposed algorithm for ergodic controller allows this online implementation on robot manipulators for $d = 6$, which is demonstrated in the next section.

In the insertion task, the target region for detection corresponds to a set of states of the peg that are necessary to be passed through for a successful insertion of the peg. This information is obtained from the reference probability distribution for exploration

Table 3.2: Average time taken to reach a target region for different exploration strategies.

	Strategy	Success rate		Time taken (seconds)
		# Trials	# Success	
2D	Strategy 1	100	100	**66.9**
	Strategy 2	100	96	106.8
	Strategy 3	100	100	122.9
	Strategy 4	100	98	155.4
3D	Strategy 1	100	100	**84.7**
	Strategy 2	100	92	141.4
	Strategy 3	100	98	292.3
	Strategy 4	100	95	247.5

in the search space. Considering stochasticity is important for a search strategy to be useful in practice. In general, stochasticity may arise either from the measurement system (e.g., uncertainties in the location of the hole, grasp of the peg or manufacturing defects) and/or the dynamics of the system (e.g., stochasticity in the contact dynamics involved in the insertion). For example, during the insertion, the peg might be at the correct relative location to the hole according to the sensor system, but the insertion may still not be successful every time in that configuration due to the stochasticity of the process. We need the search strategy to explore more often in these target regions (correct configurations for insertion) to increase the likelihood of insertion. Ergodic control considers this stochasticity by driving the system to regions in the state space such that the amount of time it spends there is in proportion to the probability mass of that region. For more details on search strategies and their desired characteristics, see the seminal work of Koopman on the theory of search [51, 52, 53]. Ergodic exploration satisfies the standards for optimal search behavior set by Koopman. Although strategies 3 and 4 do not satisfy these properties, we included them in our evaluation for completeness.

3.6.2 Experimental Setup for Peg-in-hole Task

We use a torque-controlled 7-axis Franka Emika robot, with an insertion task based on the Siemens gear set benchmark (see Fig. 3.7)[7], by using the 25.4mm-diameter peg and the 26.29mm-diameter receptacle, with the length of insertion of 47mm. We employed a Cartesian impedance controller to control the robot end-effector by computing a desired

[7]https://new.siemens.com/us/en/company/fairs-events/robot-learning.html

Cartesian wrench according to

$$\hat{\boldsymbol{f}} = \boldsymbol{K}_p \begin{bmatrix} \hat{\boldsymbol{p}} - \boldsymbol{p} \\ \mathrm{Log}\left(\hat{\boldsymbol{q}} * \bar{\boldsymbol{q}}\right) \end{bmatrix} - \boldsymbol{K}_d \begin{bmatrix} \dot{\boldsymbol{p}} \\ \boldsymbol{\Omega}_x \end{bmatrix},$$

where $\hat{\boldsymbol{p}} \in \mathbb{R}^3$, $\hat{\boldsymbol{q}} \in \mathcal{S}^3$ are, respectively, the desired position and orientation of the end-effector (with $\hat{\boldsymbol{q}}$ obtained from (3.12)), $\boldsymbol{p}, \boldsymbol{q}, \dot{\boldsymbol{p}}, \boldsymbol{w}$ are the end-effector position, orientation, linear and angular velocity and $\boldsymbol{K}_p, \boldsymbol{K}_d$ are $6{\times}6$ diagonal stiffness and damping gains, experimentally set to $\boldsymbol{K}_p = \mathrm{diag}\left(500, 500, 500, 160, 160, 160\right)$ and $\boldsymbol{K}_v = \mathrm{diag}\left(40, 40, 40, 10, 10, 10\right)$. The symbol $*$ denotes the unit quaternion product and $\bar{\boldsymbol{q}}$ the conjugate of quaternion $\boldsymbol{q}$. $\mathrm{Log}(\cdot)$ is the logarithmic map defined in (3.7).

We obtain the desired robot joint torques with $\hat{\boldsymbol{\tau}} = \boldsymbol{J}(\boldsymbol{\theta})^\top \hat{\boldsymbol{f}} + \boldsymbol{g}\left(\boldsymbol{\theta}\right) + \boldsymbol{h}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$, where $\boldsymbol{\theta}, \dot{\boldsymbol{\theta}} \in \mathbb{R}^7$ are the robot joint positions and velocities and $\boldsymbol{J} \in \boldsymbol{R}^{6\times7}$, $\boldsymbol{g} \in \mathbb{R}^7$, $\boldsymbol{h} \in \mathbb{R}^7$ are the Jacobian matrix of the end-effector, gravity and Coriolis terms. Note that the impedance gains were selected such that the robot remained compliant enough to safely interact with the environment during exploration, while still being able to track the ergodic trajectory.

We compare three different implementations of our approach. First, a *closed loop* version where the ergodic controller runs in real-time on the robot as an online coverage problem (Fig. 3.9). In that case, at every time step, we read the end-effector pose and feed it back to the ergodic controller, which computes the next pose to track based on the current state. Second, an *open loop* version where the controller tracks a reference ergodic trajectory computed offline (as used in [43] for a lower dimensional state space). Lastly, GMM-sampling-based exploration as presented in Section 3.6.1.

3.6.3 Initialization and Preprocessing for Ergodic Control

In our setup, the location of the hole is fixed (but unknown to the robot) and the main source of uncertainty comes from the different possible grasps of the peg by the end-effector, see Fig. 3.8. We model these uncertainties using a probability density function that indicates the importance of spending time in each region of the state space of the robot end-effector. We use ergodic control to insert the peg under such uncertainties.

To model the reference probability distribution for ergodic exploration of insertion, we collected $M = 204$ data points using kinesthetic teaching, which corresponds to less than 2 minutes of recording, see Fig. 3.10. Each datapoint corresponds to the position and orientation of the end-effector holding the peg. The datapoints in the vicinity of the location of the hole were recorded for successful insertions with different orientation and

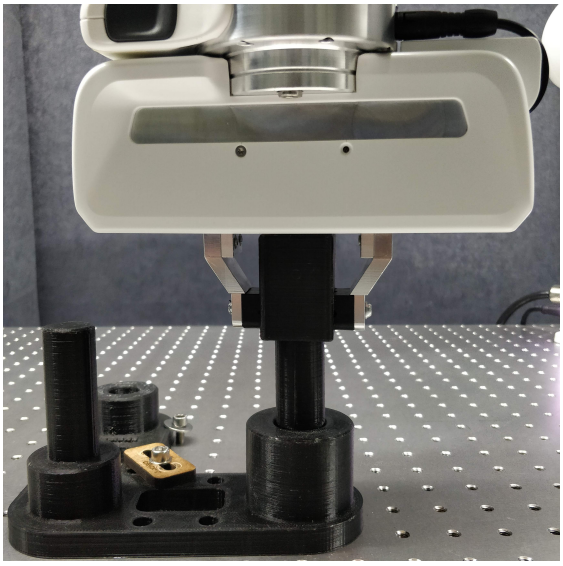
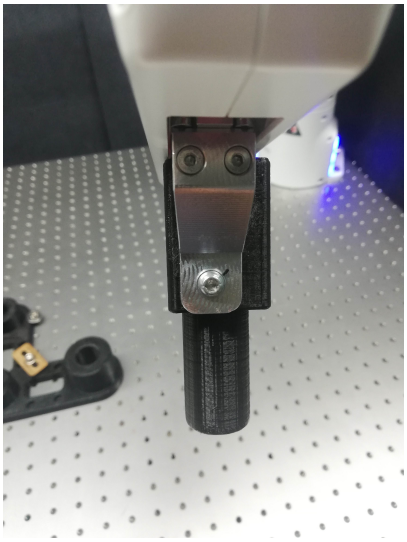


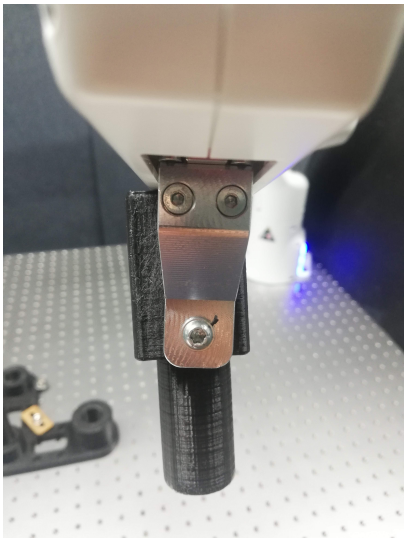
Figure 3.7: The Siemens gear benchmark used for evaluation of the peg-in-hole insertion task using ergodic control.

position offsets inherent to the grasps of the peg. To give higher importance to insertion, almost half the data points were taken from the states corresponding to the peg within the hole. A variation of about 15mm in position of the grasp for each axis and about $\sim 10^{\circ}$ variation in the orientation of the grasp were demonstrated (see accompanying video of the experiment).

Once data are collected, we preprocess the end-effector orientations as described in Section 3.4.2. Subsequently, we concatenate the position data with the obtained 3D orientation representation into a 6D vector. We then transform the data into the domain of ergodic control $\Omega_{\boldsymbol{x}}$ with $L = 1$ (ergodic space) using a bijective linear transformation. We model the data points in this transformed space using a Gaussian mixture model (GMM) with full covariances. We empirically selected 8 mixture components with a minimal isotropic covariance prior of 5×10^{-3}. Figure 3.11 shows the obtained GMM for position and orientation (marginal distributions). The GMM is used as reference probability distribution $P(\boldsymbol{x})$ for ergodic control in $\Omega_{\boldsymbol{x}}$. With $K = 10$ and $N = 10$, the computation of Fourier coefficients $\hat{\boldsymbol{\mathcal{W}}}$ for the reference probability distribution took less than 2 minutes. The closed-loop ergodic controller could be run at 100Hz ($dt = 0.01$ in Algorithm 3) on the robot. The same settings are applied to the open-loop version of the insertion task. In this case, the trajectory is generated offline using ergodic control. It is then tracked using the above-mentioned settings of the impedance control. Figure 3.11 shows examples of generated ergodic trajectories. We allow a maximum of 40 seconds for insertion. Figure 3.2 shows a trajectory generated from ergodic control in one of the



(a) A grasp with an offset in position



(b) A grasp with an offset in orientation

Figure 3.8: Two instances of grasps typically appearing when testing the ergodic control for insertion. The demonstrations included different types of grasps to let the robot cope with this uncertainty during ergodic exploration.

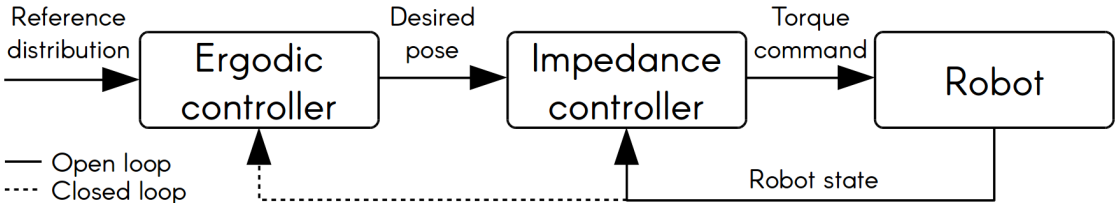


Figure 3.9: Control diagram of *open* and *closed* loop implementations. In the latter, the ergodic controller takes the robot state (end-effector pose) into account while computing the desired pose. In this way, the history of observed states is kept and new desired poses are computed accordingly.

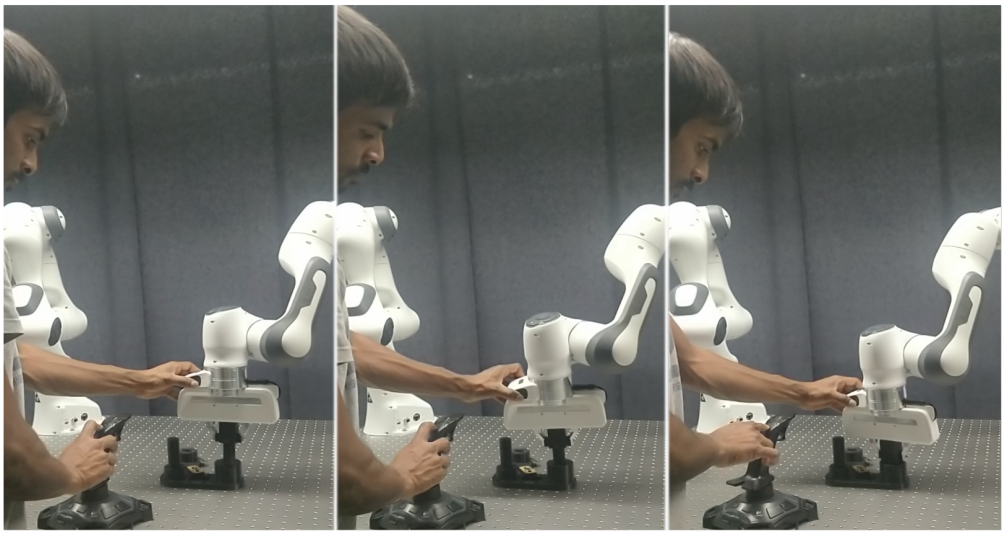


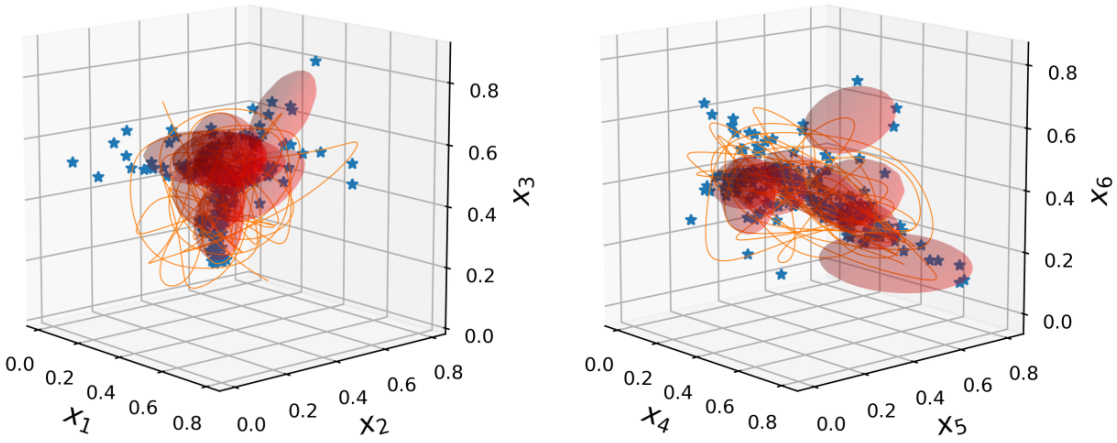
Figure 3.10: Human demonstration of peg-in-hole insertion task. Datapoints are collected for different grasps of the peg through kinesthetic teaching to show the regions in the vicinity of the receptacle to be used by the ergodic controller.

trials of the experiment.

3.6.4 Experimental Results

The obtained results, summarized in Table 3.3, show that the *closed loop* approach clearly outperforms the other approaches that do not consider the history of observed states during the exploration. Indeed, while using the former, the robot was able to successfully insert the peg in the hole in 20 out of the 20 trials performed using the grasps in Fig. 3.8. On average, the insertion using the *closed loop* approach succeeded in 9.9s with a standard deviation of 8.5s. Figure 3.12 shows snapshots of the insertion using ergodic exploration in the *closed loop* setting. Note that the only required input was a set of demonstrations to show the robot the regions it should explore. This has proven important to deal with the uncertainty in the peg grasping pose. Note that interaction forces during exploration can also cause the grasp pose to change due to the limited gripping force of the robot (either from hardware and software limitations, or set on purpose to handle fragile objects). Our experimental results show that the closed loop ergodic strategy is also robust in these situations.

The open loop version succeeded in only 2 out of 10 trials and the naive approach of GMM-sampling-based exploration succeeded in none of the 5 trials. This shows the importance of exploiting the history of the real observed end-effector poses to compute



(a) Position distribution

(b) Orientation distribution

Figure 3.11: Position and orientation marginals from the distribution used for full pose exploration (the figure does not show correlations between position and orientation). The 6D pose distribution is encoded as a GMM with 8 components (red ellipsoids) and full covariances, trained on a dataset of $M = 204$ datapoints (blue points). The trajectory for the exploration (orange lines) is generated by ergodic control for the insertion task.

Table 3.3: Performance of the peg-in-hole task for different variations of the grasps.

Strategy	**Success rate**		**Time taken (seconds)**
	# Trials	Successful Trials	
Closed loop	20	20	9.9 ± 8.5
Open loop	10	2	-
GMM-sampling	5	0	-

control commands. This is particularly important for tasks requiring contacts with the environment, where the commands need to be re-evaluated according to the history of poses retrieved by the impedance controller. Notably, this allows the use of low gains to remain compliant and enable safe contacts. In the closed loop approach, the algorithm is aware of the locations that were previously effectively visited, which is exploited to fulfill the insertion task in an online and robust fashion.

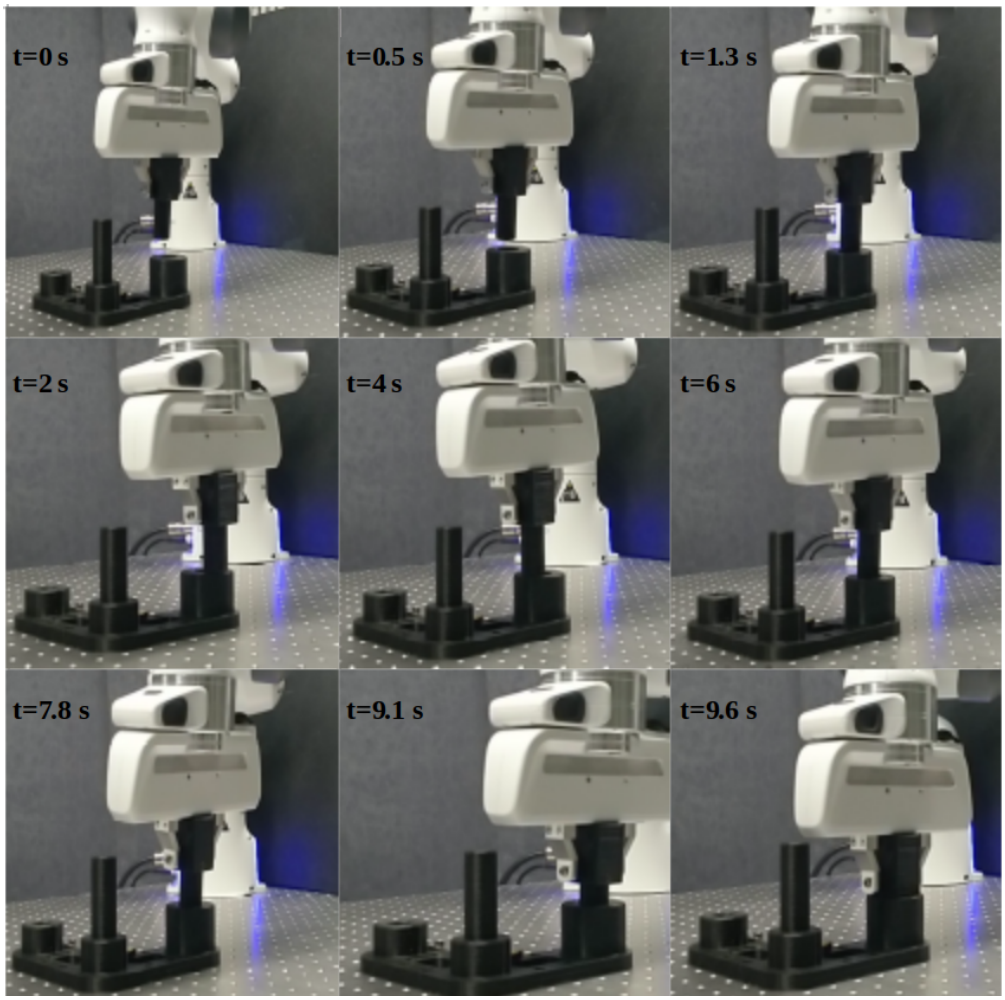


Figure 3.12: Snapshots of an insertion using ergodic control in closed loop.

3.7 Future Work

There is still room for improvement in the proposed approach. First, designing demonstrations for insertion tasks can be further optimized. Furthermore, alternative demonstration strategies (including different distributions), could speed up the insertion time using ergodic control. We will also apply the insertion tasks to different physical setups by using different sensory modalities [12]. The approach could be applied to other applications in robotics requiring exploration and active sensing. We demonstrated how TT can overcome the scalability issues in a particular ergodic exploration algorithm, namely SMC. Another popular ergodic exploration algorithm called HEDAC [54] also suffers from similar scalability issues due to convolution operations involved in computing the control commands. We could potentially use the fast ways to perform convolution in TT

format as proposed in [55] to overcome this issue in HEDAC.

3.8 Conclusion

We proposed a solution to the multidimensional ergodic exploration problem with $d > 3$, which was previously considered to be intractable for applications in robotic manipulation. The proposed approach relies on tensor train decompositions and is evaluated in simulated examples of target detection and a real robot experiment using a peg-in-hole insertion task. The obtained results show that ergodic control has a 100% success rate in all tasks, succeeding faster than the competing approaches. In addition to the novelty of exploiting ergodic control in an online fashion for insertion tasks, the computational techniques we used demonstrate how algorithms in robotics, in general, can benefit from tensor decomposition techniques (tensor networks) to overcome the limitations in computational speed and storage requirements. To achieve this, tensor networks exploit the smoothness and symmetry of the functions underlying the problem. If the underlying problem does not have low rank structure, there may not be significant saving in the storage and computational cost using tensor methods. However, ergodic control problem as formulated in SMC has such low rank structure by design.

The main challenges of extending ergodic control to state space of more than 3 dimensions concern the computation of Fourier coefficients (preprocessing step) and the speed of the control loop (for online implementation). This chapter addressed both of these challenges using the properties of tensor train decomposition. We leveraged this improved ergodic control formulation to propose a sensorless strategy for peg-in-hole insertion tasks. We validated our approach with a compliant robot manipulator, where the 6D regions to explore are obtained from kinesthetic human demonstrations. Our experimental results show that by using our approach, the robot is capable of achieving challenging insertion tasks without force/torque sensors.

By using the reproducible Siemens gears benchmark, insertion tasks with unknown gripping variations could be achieved in less than 10 seconds on average. This is, in part, due to the multi-scale exploration behavior that is inherent to the ergodic metric we employed. With such metric, the resulting controller first crudely explores the region of interest, and then progressively refines the search until insertion, efficiently exploiting information about the already covered regions. This is also due to the fast processing that we propose (through TT), which allows ergodic control to be run in an online manner. Indeed, re-estimating the control commands on-the-fly allows the proposed ergodic control strategy to be used together with an impedance controller with low gains, which is important for tasks requiring contacts with the environment.

4 Optimization using Tensor Train

In this chapter, we present a new approach for optimizing functions in Tensor Train (TT) format called Tensor Train for Global Optimization (TTGO). We show that TTGO often yields globally optimal solutions and can provide multiple solutions. In particular, it can handle the challenging task of numerical optimization involving both continuous and discrete variables—an issue existing solvers struggle with. The integration of powerful techniques, like cross-approximation, enables the representation of complex robotics objective functions in the TT format, facilitating TTGO's application to solve challenging optimization problems. Tests on benchmark functions for numerical optimization demonstrate the effectiveness of our method in generating solutions close to global optima and capturing multiple solutions. We further showcase the versatility of our framework in robotics by applying it to inverse kinematics with obstacles and motion planning for a 7-DoF manipulator. In a subsequent chapter, this will be extended to control synthesis using dynamic programming for hybrid control tasks.

Publication Note

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

- S. Shetty, T. Lembono, T. Löw, and S. Calinon, "Tensor trains for global optimization problems in robotics," *International Journal of Robotics Research (IJRR)*, 2023

Supplementary material including videos and source codes related to this chapter are available at: https://sites.google.com/view/ttgo/home

4.1 Introduction

Numerical optimization has been one of the major tools for solving a variety of robotics problems including inverse kinematics, motion planning, state estimation, and control. In this framework, the robotics task to be accomplished is formulated as the minimization of a cost function. Although we ideally seek a solution that incurs the least cost (i.e., global optima), any solution that has a cost comparable to the global optima is usually sufficient. It is essential in robotics applications that a feasible solution is found fast. In practice, the optimization problems in robotics involve non-convex cost functions and the existing optimization techniques often can not quickly find a feasible solution.

On the one hand, there are stochastic procedures, often called evolutionary strategies (e.g., CMA-ES [56], Genetic Algorithm [57], Simulated Annealing [58]), that can find the global optima of non-convex functions. However, such techniques are too slow for most robotics applications. On the other hand, Newton-type optimization techniques are fast in general—a desirable feature for robotics applications. Hence, most of the existing numerical optimization techniques used in robotics are variants of Newton-type optimization techniques. However, such techniques are iterative and require a good initial guess that determines the solution quality and the time required to find a solution.

In this chapter, we show that we can find globally optimal solutions when the objective function is in TT format. We propose a novel approach that we call Tensor Train for Global Optimization (TTGO) to produce optima of a function represented in TT format. We leverage TTGO along with the cross approximation (see Section 2.8) technique [59] for applications in robotics including inverse kinematics and motion planning. For such problems in robotics, we propose to approximate the objective function in TT format using cross approximation and then use TTGO to obtain a solution close to the optima. This solution is further refined using a local solver (e.g., Newton-type optimization) to compensate for modelling error. This approach allows us to obtain a richer set of solutions, especially for multimodal problems (namely, problems with multiple solutions). As TTGO does not use any gradient information, it is also less susceptible to getting stuck at local optima.

For applications considered in this chapter, the objective is to minimize a cost function. We view the cost function as a function of both the *task parameters* (e.g., desired end-effector pose to reach an object) and the *optimization variables* (e.g., valid robot configurations for inverse kinematics or joint angle trajectories for motion planning). Minimizing the cost function can be reformulated as maximizing a probability density function. Existing methods, such as [60, 61, 62], use Variational Inference to approximate the probability density function by treating the task parameters as constant, making the

cost function a function of only the optimization variables. In contrast, we can handle diverse task parameters by approximating the joint probability distribution of both the task parameters and optimization variables. By exploiting the structure between the two, which often exhibit a *low-rank structure*, the TT model can compactly approximate the density function. After training the model, during the online execution, we can condition the TT Model on specific task parameters, obtain a new TT model corresponding to the given task, and use TTGO to obtain an approximate solution in a fast manner.

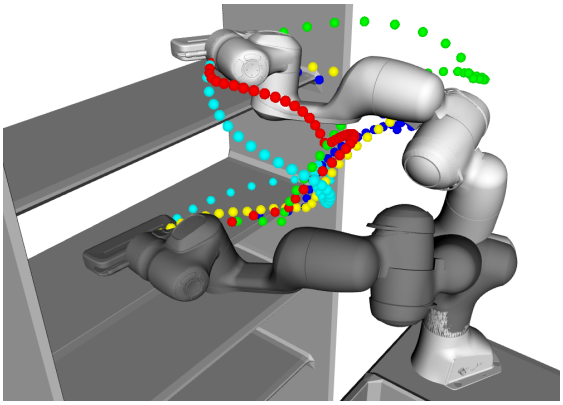
From the viewpoint of warm-starting optimization solvers, our approach builds an implicit database in Tensor Train (TT) format for diverse tasks by only using the definition of the cost function in an unsupervised manner, i.e., without requiring any gradient information or another solver.

In summary, our contributions are as follows:

- We introduce a principled approach called TTGO (Tensor Train for Global Optimization) to obtain optima of functions in TT format
- When the underlying function is multimodal, our approach can find multiple solutions that correspond to a given task.
- The approach is first demonstrated on some benchmark optimization functions to show that it can find global optima and multiple solutions robustly. We show the relevance of the approach to robotics problems by applying it to inverse kinematics with obstacles and motion planning problems with a 7-DoF manipulator where the solution was obtained in a few milliseconds.

Additionally, even though we have outlined the procedure for functions of continuous variables in this chapter, TTGO is capable of managing both continuous and discrete variables effortlessly. As a result, it can be utilized as a supplement or substitute for mixed integer programming. This feature is exploited in the subsequent chapter for control synthesis for robotic systems involving hybrid state and action space.

The chapter is organized as follows. In Section 4.2, we provide a literature survey on initializing numerical optimization solvers and multimodal optimization. We refer to Section 2 for the necessary background on Tensor Train modeling that is used in this chapter. Then, in Section 4.3, we describe the proposed TTGO algorithm and Section 4.4 describes how it is adapted to problems in robotics. Section 4.5 presents the evaluation of our algorithm. We first test it on benchmark functions for numerical optimization and then apply it to inverse kinematics with obstacles and motion planning problems with



(a)

Figure 4.1: Solutions from TTGO for motion planning of a manipulator from a given initial configuration (white) to a final configuration (dark). The obtained joint angle trajectories result in different paths for the end effector which are highlighted by dotted curves in different colors. The multimodality is clearly visible from these solutions.

manipulators. In Sections 4.6 and 4.9, we conclude the chapter by discussing how our approach could lead to new ways of solving a variety of problems in robotics.

4.2 Related work

This work intersects with several research directions. Firstly, we target robotics applications that are formulated as optimization problems. Our framework provides a way to predict a good initialization for the optimization solver. At the same time, it also provides a principled way to obtain multiple solutions of a given optimization problem. Finally, the proposed framework relies on tensor train decomposition (see Chapter 2). We discuss each topic briefly in this section.

4.2.1 Optimization in Robotics

Many problems in robotics are formulated as optimization problems. For example, recent work in motion planning relies on trajectory optimization to plan the robot motion (e.g., CHOMP [63], STOMP [64], TrajOpt [65], GPMP [66]). Inverse kinematics for high dimensional robots is usually formulated as nonlinear least squares optimization [67] or Quadratic Programming (QP) [68]. The optimization framework offers a convenient way to transfer the high-level requirement (e.g., energy efficiency, maintaining orientation) to cost functions or constraints. Furthermore, the availability of off-the-shelf optimization solvers and tools for automatic gradient computations allow researchers to focus more on

the problem formulation.

However, most of the solvers used in robotics are local optimizers whose performance depend highly on the initialization, especially since most robotics problems are highly non-convex. Even state-of-the-art solvers such as TrajOpt can fail on a simple problem with poor initialization [69]. The initialization determines both the convergence speed, the solution quality, and the success rate of the solver. This motivates further research on how to predict good initialization for a given optimization problem.

4.2.2 Predicting Good Initialization

A majority of works that attempt to predict good initialization rely on a database approach, often called trajectory library [70] [71] or memory of motion [72] for motion planning. In [73], they provided a data driven framework for solving globally optimal collision-free inverse kinematics (IK) problems. The idea is to first build a database of precomputed solutions offline. This database can be constructed from expert demonstrations [74], using the optimization solver itself [75], or using the combination of a global planner and the optimization solver [76]. Once the database is constructed, we can predict a good initial guess (i.e., a *warm start*) for a given task by formulating it as a regression problem that maps the task to the initial guess. While the formulation is easy to implement, especially since there are many function approximators easily available, the database approach suffers from two main issues: non-convexity and multimodality.

Firstly, the database approach requires computing good solutions to be stored in the database. With the complexity of general robotics problems, computing the solutions is often not trivial. Assuming we have access to a suitable database and are prepared to train function approximators, we must take into account the fact that many robotics problems have multiple solutions, making it challenging for most function approximators to accurately approximate this one-to-many mapping. Often, such approximators tend to average the different solutions, resulting in poor predictions.

4.2.3 Multimodal Optimization

In problems requiring multiple solutions, common heuristics involve initializing from a uniform distribution. SMTO [60] was developed for multimodal trajectory optimization, transforming the task of minimizing a cost function into identifying the modes of a corresponding probability density function (PDF). It approximates the PDF using a Gaussian mixture model (GMM) through Variational Inference. However, SMTO faces practical challenges as it relies heavily on GMM for function approximation.

To address some of the limitations in approximating a PDF with GMM including scalability and the need for samples from the PDF, SMTO employs variational inference and importance sampling from a proposal distribution. The approach is primarily suitable for trajectory optimization problems with finite homotopic solutions. In contrast, LSMO [61] explores infinite homotopic solutions for trajectory optimization using a neural network, albeit with high computational demands for online operations. Our method offers a more versatile solution by distributing computation offline and online, efficiently handling both finite and uncountably many solutions. We have described the challenges in approximating a given PDF using GMM and the advantages of TT in more detail in Appendix B.5.

4.2.4 Optimization using Tensor Train

The studies conducted in [77, 59] have shown that the TT decomposition can serve as an effective means for gradient-free optimization and can compete with top-performing global optimization algorithms, such as CMA-ES and GA. However, the use of TT decomposition for global optimization in these approaches resembles evolutionary strategies in that they tackle one optimization problem at a time and can only yield one solution, which is too slow for robotics applications.

In contrast, our work takes a distinct approach by first approximating the objective function using the TT format and then performing the optimization over the TT representation. The computationally intensive component is distributed to an offline phase for modeling the objective function over the task parameters and decision varibles, allowing us to quickly address numerous optimization problems in an online phase using TTGO.

4.3 Tensor Train for Global Optimization

In this section, we propose efficient methodologies to find the optima of a function represented in TT format. We first consider the function to be a TT-distribution given by (2.14) and aim to find the maxima. In the subsequent sections, we show how this can be easily extended to find the optima of an arbitrary function represented in TT format. In Section 4.4, we demonstrate how TTGO can be used to tackle optimization problems in robotics.

The methodology was proposed originally in our work [9] as a stochastic procedure. It was originally introduced to obtain multiple solutions and global optimality as described in Section 4.3.1. Following this work, a deterministic version of this approach was proposed in [78]. We further improved this deterministic version for applicability in robotics in our

Algorithm 4 Stochastic TTGO.
This algorithm has a parallel implementation so that the K samples can be obtained in parallel

Input: TT Blocks $\boldsymbol{\mathcal{P}} = (\boldsymbol{\mathcal{P}}^1, \ldots, \boldsymbol{\mathcal{P}}^d)$ corresponding to the distribution Pr (see (2.14)), sample priority $\alpha \in (0, 1)$
Output: N $\alpha-$prioritized samples $\{(x_1^l, \ldots, x_d^l)\}_{l=1}^N$ from the distribution Pr in (2.14)
$\boldsymbol{\beta}_{d+1} \leftarrow 1$
for $k \leftarrow d$ to 2 **do**
  $\boldsymbol{\beta}^k = \sum_{x_k} \boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :}\, \boldsymbol{\beta}_{k+1}\, (\boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :})^\top$
end for
$\boldsymbol{\Phi}_1 \leftarrow \mathbf{1} \in \mathbb{R}^{N \times 1}$
for $k \leftarrow 1$ to d **do**
  $\pi^k(x_k) = \boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :}\, \boldsymbol{\beta}^{k+1}\, (\boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :})^\top, \quad \forall x_k$
  for $l = 1, \ldots, N$ **do**
    $\boldsymbol{p}_k(x_k) = |\Phi_k(l,\, :)\, \pi^k(x_k)\, \Phi_k(l,\, :)^\top|, \quad \forall x_k$
    $\boldsymbol{p}_k \leftarrow \frac{\boldsymbol{p}_k}{\max(\boldsymbol{p}_k)}$
    $\boldsymbol{p}_k \leftarrow \boldsymbol{p}_k^{\frac{1}{1-\alpha+\epsilon}}$, where ϵ is positive and $\epsilon \approx 0$
    $\boldsymbol{p}_k(x_k) \leftarrow \frac{\boldsymbol{p}_k(x_k)}{\sum_{x_k} \boldsymbol{p}_k(x_k)}, \quad \forall x_k$
    Sample x_k^l from the multinomial distribution $\boldsymbol{p}_k$
    $\boldsymbol{\Phi}_{k+1}(l, :) = \boldsymbol{\Phi}_k(l, :)\, \boldsymbol{\mathcal{P}}^k_{:, x_k^l, :}$
  end for
end for
Maxima: $\boldsymbol{x^*} = \underset{(x_1^l, \ldots, x_d^l)}{\arg\max} |\boldsymbol{\mathcal{P}}(x_1^l, \ldots, x_d^l)|, \quad \forall l \in \{1, \ldots, N\}$

subsequent work [10] and it is described in this chapter in Section 4.3.2.

4.3.1 Stochastic Approaches

In Section 2.9.2, we described an efficient way to sample from a TT distribution. Note that the samples from the high-density region provide an approximation to the maxima. So, we do not necessarily want to sample from the whole distribution, but instead focus on obtaining samples from the high-density regions (e.g., when we only want to find the modes of the distribution). We achieve this using *prioritized sampling* by modifying the Algorithm 1 and it is described in the Algorithm 4.

Algorithm 5 Deterministic TTGO

Input: TT Cores $\boldsymbol{\mathcal{P}} = (\boldsymbol{\mathcal{P}}^1, \ldots, \boldsymbol{\mathcal{P}}^d)$, Domain $\Omega_x = \{(x_1^{i_1}, \ldots, x_d^{i_d}) : i_k \in \{1, \ldots, n_k\}\}$
Hyperparameters: N, $n_{\textbf{sweeps}} \in \{1, \ldots, d\}$, ϵ # default: $K = n_1$, $n_{\text{sweeps}} = 1$, $\epsilon = 0.001$
Output: Maxima $\boldsymbol{x^*} = (x_1^*, \ldots, x_d^*)$ of the TT distribution given by $\boldsymbol{\mathcal{P}}$ (see (2.14))
$\boldsymbol{\beta}^{d+1} \leftarrow 1$
for $k \leftarrow d$ to 2 **do**
  $\boldsymbol{\beta}^k = \sum_{x_k} \boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :}\, \boldsymbol{\beta}^{k+1}\, \boldsymbol{\mathcal{P}}^k_{:,\, x_k,\, :}{}^\top$
end for
Definition:

$$\pi^m(x_1^{i_1}, \ldots, x_m^{i_m}) = (\boldsymbol{\mathcal{P}}^1_{:,\, x_1^{i_1},\, :} \cdots \boldsymbol{\mathcal{P}}^m_{:,\, x_m^{i_m},\, :}) \boldsymbol{\beta}^{m+1} (\boldsymbol{\mathcal{P}}^1_{:,\, x_1^{i_1},\, :} \cdots \boldsymbol{\mathcal{P}}^m_{:,\, x_m^{i_m},\, :})^\top$$

$$q^m(x_1^{i_1}, \ldots, x_m^{i_m}) = \begin{cases} 1 & \text{if} \quad (\pi^m(x_1^{i_1}, \ldots, x_{m-1}^{i_{m-1}},\ x_m^{i_m}) > \pi^m(x_1^{i_1}, \ldots, x_{m-1}^{i_{m-1}},\ x_m^{i_m+a}), \\ & \quad \forall a \in \{1, -1\}) \text{ OR } (x_m \text{ is discrete}) \\ \epsilon & \text{else } \#\text{ i.e., lower weight if } \pi^m \text{ is not a concave peak w.r.t. } x_m^{i_m} \end{cases}$$

$$\hat{\pi}^m(x_1^{i_1}, \ldots, x_m^{i_m}) = q^m(x_1^{i_1}, \ldots, x_m^{i_m})\ \pi^m(x_1^{i_1}, \ldots, x_m^{i_m})$$

Initialize: $\mathcal{D}_1^1 = \{(x_1^{j_1^k}) : k \in \{1, \ldots, \min(N, n_1)\},\ j_i^k \in \{1, \ldots, n_1\},\ \pi^1(x_1^{j_1^k}) \geq \pi^1(x_1^{j_1^{k-1}})\}$
Set $p_{max} = 0$
for $s \leftarrow 1$ to n_{sweeps} **do**
  for $m \leftarrow \max(2, s)$ to d **do**
    $\mathcal{D}_m^s = \{(x_1^{j_1^k}, \ldots, x_m^{j_m^k}) :$

$$\begin{aligned} & k \in \{1, \ldots, \min(N,\ \text{size}(\mathcal{D}_{m-1}^s)\ n_m)\}, \\ & j_i^k \in \{1, \ldots, n_1\}, \\ & \hat{\pi}^m(x_1^{j_1^k}, \ldots, x_m^{j_m^k}) \geq \hat{\pi}^m(x_1^{j_1^{k-1}}, \ldots, x_m^{j_m^{k-1}}), \\ & (x_1^{j_1^k}, \ldots, x_{m-1}^{j_{m-1}^k}) \in \mathcal{D}_{m-1}^s)\} \end{aligned}$$

  end for
  $\boldsymbol{x} = (x_1, \ldots, x_d) \leftarrow (x_1^{j_1^1}, \ldots, x_d^{j_d^1}) \in \mathcal{D}_d^s$
  $p = |\pi^d(\boldsymbol{x})|$
  if $p \geq p_{max}$ **then**
    $p_{max} \leftarrow p$
    $\boldsymbol{x^*} \leftarrow \boldsymbol{x}$
  end if
  $\mathcal{D}_s^s = \{(x_1^*, \ldots, x_s^*)\}$
end for
Note: The associated software provides a highly parallel implementation of the above algorithm in PyTorch where $\mathcal{D}$ are tensors. If we are interested in multiple solutions (say $K < N$), the sweeping iteration over s needs to be done separately for the top K candidate solution in $\mathcal{D}_d^1$ obtained after the first inner *for* loop over m.

Recall that the sampling procedure in Algorithm 1 consists of repeated sampling of each dimension separately from a multinomial distribution. We modify this by providing a sampling parameter $\alpha \in (0, 1)$ that can be chosen to adjust the sampling priority. When $\alpha = 0$, the samples will be generated from the whole distribution (i.e., exact sampling), including from the low-density region (albeit with a lower probability). Higher α will focus the sampling around the area with higher density. This is ideal for robotics applications, as some applications require a very good initial solution for fast optimization (in that case, α is set near to one to obtain the best possible solution) while some others prefer the diversity of the solutions (by setting a small α). As the sampling procedures can be done in parallel, we can generate a large number of samples quickly and select the best few samples according to their cost function values as the solution candidates.

4.3.2 Deterministic Approaches

Alternatively, instead of using prioritized samples with a specified α value, we can choose to use a deterministic version of the algorithm. This involves selecting the top K elements from the multinomial distribution $\boldsymbol{p}_k$ at iteration k instead of sampling N elements independently and we keep track of the history of the selected indices from the previous modes $(k-1)$. This deterministic version of TTGO was proposed in [78]. In [10], we further improve this methodology to include smarter choices for top-N which improves the quality and diversity of the solution obtained. The idea is to give higher priority to the local maxima (peaks) of the multinomials involved in each iteration (i.e., $\Pr_k(x_k|x_1, \ldots, x_{k-1}), \forall k \in \{1, \ldots, d\}$ as described in Section 2.9.2). In addition, we introduce an iterative procedure to improve the solution and scalability of the approach. This is sketched in Algorithm 5.

In practice, the stochastic version is more suitable when the approximation error in TT modeling is large and the application needs diversity in the solutions. However, we observed that the deterministic version provides better-quality solutions. Both algorithms are available in the software accompanying the articles [9, 10] with fully parallel implementation.

In the remainder of this chapter, the results are demonstrated using the stochastic version as we are interested in multiple solutions for a given optimization problem and as we expect large approximation error in TT modeling of the objective functions involved. In the next chapter, we will use the deterministic version of TTGO for control synthesis using dynamic programming.

4.3.3 Finding Optima of Arbitrary Tensor Train Model

Given a TT model $\boldsymbol{\mathcal{P}}$, Algorithms 4 and 5 provide maxima $\arg\max_{\boldsymbol{x}} |\boldsymbol{\mathcal{P}}(\boldsymbol{x})|$ (i.e. maximum of the corresponding TT distribution given by (2.14)). To find the $\arg\max_{\boldsymbol{x}} \boldsymbol{\mathcal{P}}(\boldsymbol{x})$ using TTGO, we first need to pre-process the TT model. We first find the maxima w.r.t the absolute value $\boldsymbol{x}_a = \arg\max_{\boldsymbol{x}} |\boldsymbol{\mathcal{P}}(\boldsymbol{x})|$ which can be done using TTGO with $\boldsymbol{\mathcal{P}}$. Next, we find a shifted TT model $\hat{\boldsymbol{\mathcal{P}}} = \boldsymbol{\mathcal{P}} - \boldsymbol{\mathcal{P}}(\boldsymbol{x}_a)$ (using algebraic operations over TT model, see Section 2.6). Now we again use TTGO to find $\boldsymbol{x}_b = \arg\max_{\boldsymbol{x}} |\hat{\boldsymbol{\mathcal{P}}}(\boldsymbol{x})|$. So $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ are the two extrema (a maxima and a minima) of $\boldsymbol{\mathcal{P}}$. So, $\boldsymbol{x}_{\min} = \underset{\boldsymbol{x}\in\{\boldsymbol{x}_a,\, \boldsymbol{x}_b\}}{\arg\min}\ \boldsymbol{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{x}_{\max} = \underset{\boldsymbol{x}\in\{\boldsymbol{x}_a,\, \boldsymbol{x}_b\}}{\arg\max}\ \boldsymbol{\mathcal{P}}(\boldsymbol{x})$.

4.3.4 Normalizing Tensor Train Model

Let $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ are the minima and maxima of an arbitrary TT model $\boldsymbol{\mathcal{P}}$ found using the methodology described in Section 4.3.3. Let $p_{\min} = \boldsymbol{\mathcal{P}}(\boldsymbol{x}_{\min})$ and $p_{\max} = \boldsymbol{\mathcal{P}}(\boldsymbol{x}_{\max})$. Then, we can shift and scale the TT model $\boldsymbol{\mathcal{P}}$ to obtain a new TT model: $\hat{\boldsymbol{\mathcal{P}}} = \hat{p}_{\min} + (\boldsymbol{\mathcal{P}} - p_{\min})\left(\frac{\hat{p}_{\max} - \hat{p}_{\min}}{p_{\max} - p_{\min}}\right)$. Then, $\hat{\boldsymbol{\mathcal{P}}}(\boldsymbol{x}) \in (\hat{p}_{\min},\ \hat{p}_{\max}),\ \forall \boldsymbol{x} \in \Omega_{\boldsymbol{x}}$. Note that $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ are also the extrema of $\hat{\boldsymbol{\mathcal{P}}}$. By specifying $\hat{p}_{\min} > 0$ and $\hat{p}_{\max} > \hat{p}_{\min}$, we can ensure that $\hat{\boldsymbol{\mathcal{P}}}$ is non-negative.

This is a useful pre-processing step in practice to apply TTGO. To find the maxima of an arbitrary tensor $\boldsymbol{\mathcal{P}}$ we can work with the corresponding non-negative TT model $\hat{\boldsymbol{\mathcal{P}}}$. For instance, as we describe later in this chapter, a typical use case of TTGO in robotics is to find $\boldsymbol{x}_d^* = \arg\max_{\boldsymbol{x}_d} \boldsymbol{\mathcal{P}}(\boldsymbol{x}_t,\ \boldsymbol{x}_d)$ where $\boldsymbol{\mathcal{P}}(\boldsymbol{x}_t,\ \boldsymbol{x}_d)$ could be negative. So in all our applications, we first find the normalized TT model $\hat{\boldsymbol{\mathcal{P}}}$ and then $\boldsymbol{x}_d^* = \arg\max_{\boldsymbol{x}_d} \hat{\boldsymbol{\mathcal{P}}}(\boldsymbol{x}_t,\ \boldsymbol{x}_d)$ can be found using TTGO for various $\boldsymbol{x}_t$ on the conditioned TT model $\hat{\boldsymbol{\mathcal{P}}}_{\boldsymbol{x}_d}^{\boldsymbol{x}_t}$ (as defined in Section 2.9.3). So, unless otherwise specified in this chapter, we assume the TT model $\boldsymbol{\mathcal{P}}$ is normalized to be non-negative while using TTGO.

4.4 Applications to Function Optimization in Robotics

In this section, we outline how we can leverage TTGO to solve some challenging tasks in robotics formulated as optimization problems.

Cost functions in robotics applications typically rely on two types of variables: *task*

parameters and *decision variables*. Task parameters are constant for a given optimization problem and describe the range of tasks that may arise in a particular robotic application. For example, in an inverse kinematics (IK) problem with obstacles, the task parameters could be the desired end-effector pose to reach an object, while the decision variables are the variables being optimized (e.g., the robot configuration or joint angles). In most cases, we can anticipate the possible range of task parameters, such as the robot workspace for IK. Ideally, we can solve the optimization problem offline numerous times for the complete range of task parameters and leverage this knowledge to expedite online optimization for new tasks.

It is important to note that in robotics, the cost function is frequently a piecewise smooth function that incorporates a specific structure (i.e., low-rank structure explained in Chapter 2) among the cost function variables. For instance, similarities among task parameters correspond to similarities among solutions to the optimization problem. By capturing this structure, we can compactly model the relationships among variables rather than relying on database approaches that store each data point. Although such a structure is prevalent in many robotics applications, it has not been extensively utilized.

We regard the cost function as a function of both the optimization variables and the task parameters that define the optimization problem. Our approach involves first transforming the cost function into an unnormalized Probability Density Function (PDF) and then approximating it with TT representation using TT-cross (Section 2.8). So the surrogate probability model is a TT distribution (see Section 2.9). The TT model defining the TT distribution corresponds to the discrete analogue of the given unnormalized PDF. During online execution, when a task parameter is specified, we condition the TT distribution on the corresponding parameter to obtain another TT model corresponding to the conditional distribution (see Section 2.9.3). Subsequently, we use TTGO to find several candidates for the maxima of the conditional TT model corresponding to the specified task parameters which correspond to the minima of the cost function. In cases where the underlying PDF is multimodal, the candidate solutions will be derived from multiple modes as TTGO can provide a diverse set of solutions. By evaluating the corresponding cost functions at the candidate solutions, we can then choose the best sample(s) and select the sample(s) with the lowest cost if multiple solutions are needed. In the second stage, we refine these proposed optima using an appropriate optimization technique, such as Newton-type solvers if the objective function is differentiable.

Next we provide the mathematical formulation of the approach described in Figure 4.2.

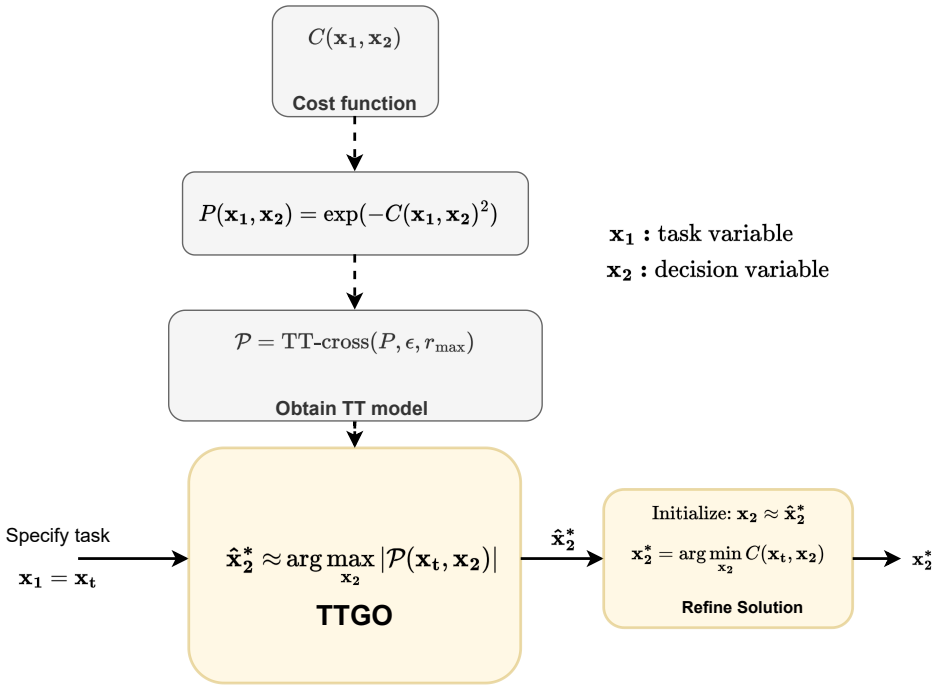


Figure 4.2: This figure shows how TTGO can be used for function optimization purposes in robotics applications. The cost function C, a function of task variable $\boldsymbol{x}_1$ and decision variable $\boldsymbol{x}_2$, is first transformed into an unnormalized probability function P. The probability function is modeled in TT format using TT-cross in the offline phase. In the online phase, given a task variable $\boldsymbol{x}_t$, we use TTGO on the conditioned TT model to obtain an approximate solution $\hat{\boldsymbol{x}}_2^*$ to the decision variable. To compensate for modeling error, this solution can be further fine-tuned using a local search method such as Newton-type optimization to obtain the optimal solution $\boldsymbol{x}_2^*$.

Methodology

Let $\boldsymbol{x}_1 \in \Omega_{\boldsymbol{x}_1}$ be the task parameter, $\boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}$ be the decision variables and $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$. Let $C(\boldsymbol{x}_1, \boldsymbol{x}_2)$ be a nonegative cost function. Given the task parameter $\boldsymbol{x}_1 = \boldsymbol{x}_t$, we consider the continuous optimization problem in which we want to minimize $C(\boldsymbol{x}_t, \boldsymbol{x}_2)$ w.r.t $\boldsymbol{x}_2$:

$$\begin{aligned} \boldsymbol{x}_2^* = &\arg\min_{\boldsymbol{x}_2} C(\boldsymbol{x}_1,\ \boldsymbol{x}_2) \\ \text{s.t. } &\boldsymbol{x}_1 = \boldsymbol{x}_t, \\ &\boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \end{aligned} \tag{4.1}$$

We assume that $\Omega_{\boldsymbol{x}_1} \in \mathbb{R}^{d_1}$, $\Omega_{\boldsymbol{x}_2} \in \mathbb{R}^{d_2}$ are both rectangular domain and let $\Omega_{\boldsymbol{x}} = \Omega_{\boldsymbol{x}_1} \times \Omega_{\boldsymbol{x}_2} \subset \mathbb{R}^d$ with $d = d_1 + d_2$. We decompose the procedure to solve such an

optimization problem into two steps:

1. Predict an approximate solution $\hat{\boldsymbol{x}}_2^*$ that corresponds to the given $\boldsymbol{x}_1 = \boldsymbol{x}_t$ using TTGO, then
2. Improve the solution $\hat{\boldsymbol{x}}_2^*$ using a local search (e.g., Newton type optimization) to obtain the optimal solution $\boldsymbol{x}_2^*$.

To find the approximate solution(s) $\hat{\boldsymbol{x}}_2^*$, we first convert the above optimization problem of minimizing a cost function into maximizing an unnormalized probability density function $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$ using a monotonically non-increasing transformation,

$$\begin{aligned} \boldsymbol{x}_2^* = &\arg\max_{\boldsymbol{x}_2} P(\boldsymbol{x}_1,\ \boldsymbol{x}_2) \\ \text{s.t. } &\boldsymbol{x}_1 = \boldsymbol{x}_t, \\ &\boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \end{aligned} \tag{4.2}$$

For example, we can define $P(\boldsymbol{x}) = e^{-\beta C(\boldsymbol{x})^2}$ with $\beta > 0$. Without loss of generality, in the remainder of the chapter we consider optimization problems to be of type (4.2) with the objective function being the density function.

In this probabilistic view, the solution $\boldsymbol{x}_2^*$ corresponds to the mode, i.e., the point with the highest density, of the conditional distribution $P(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$. In general, however, we do not have an analytical formula of $P(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$, and finding the model is as difficult as solving the optimization problem in (4.1). We overcome this issue by first approximating the unnormalized PDF $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$ using a TT model as the surrogate model to obtain the joint distribution $\Pr(\boldsymbol{x}_1, \boldsymbol{x}_2)$ (see (2.14)). Given the task $\boldsymbol{x}_1 = \boldsymbol{x}_t$, we condition the TT model to obtain the conditional distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$, i.e., the TT distribution corresponding to the conditional TT model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}$ (see Section 2.9.3). Finally, the TTGO allows us to produce the approximate solution(s) $\hat{\boldsymbol{x}}_2^*$ from the conditional TT model.

Approximating the PDF using TT model:

Given the unnormalized PDF $P(\boldsymbol{x}_1,\ \boldsymbol{x}_2)$, we use the TT-Cross algorithm (see Section 2.8) to compute its discrete analogue approximation, i.e., $\boldsymbol{\mathcal{P}}$, in the TT format. The construction of $\boldsymbol{\mathcal{P}}$ only requires the computation of $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$ at selected points $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ in the rectangular domain. Instead of computing every single possible value of P in the discretized domain ($\mathcal{O}(n^d)$), the TT-Cross algorithm only requires $\mathcal{O}(ndr^2)$ cost function evaluations, where n is the maximum number of discretization and r is the maximum rank

of the approximate tensor. The tensor model $\boldsymbol{\mathcal{P}}$ is an approximation of the unnormalized PDF corresponding TT distribution $\Pr(\boldsymbol{x})$ defined by (2.14).

Refinement of TT Model

Note that the optima found by TTGO belongs to the discretization set $\mathcal{X}$ of the domain $\Omega_{\boldsymbol{x}}$ used for TT modeling of the PDF P. For a better approximation of the solution from TTGO, we require the discretization set $\mathcal{X}$ of the domain $\Omega_{\boldsymbol{x}}$ to be very fine. In contrast, it is more efficient in practice to have coarser discretization to find the TT model of the density function $P(\boldsymbol{x})$ using TT-cross. We overcome this issue by using the refinement technique described in Section 2.5.2. We first use a coarse discretization $\mathcal{X}$ of the domain $\Omega_{\boldsymbol{x}}$ and find the TT model $\boldsymbol{\mathcal{P}}$ of the PDF P using TT-cross. Then, we consider a finer discretization $\hat{\mathcal{X}}$ of the domain $\Omega_{\boldsymbol{x}}$. The corresponding TT model $\hat{\boldsymbol{\mathcal{P}}}$ can be obtained by interpolation of the cores of $\boldsymbol{\mathcal{P}}$. Finally, we use $\hat{\mathcal{X}}$ and $\hat{\boldsymbol{\mathcal{P}}}$ for TTGO.

Re-Normalizing the TT Model

Due to some approximation error in modeling, the TT model $\boldsymbol{\mathcal{P}}$ could be negative. We pre-process it to be nonnegative by using the procedure described in Section 4.3.4 before using TTGO.

Conditioning TT Model:

After approximating the joint distribution, we can condition it on the given task. Given the task parameter $\boldsymbol{x}_1 = \boldsymbol{x}_t \in \Omega_{\boldsymbol{x}_1}$, we first condition the TT model $\boldsymbol{\mathcal{P}}$ to obtain $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}$. We then use it to construct the conditional TT distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$ as described in Section 2.9.3. This is the desired surrogate probability model for $P(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$.

Fine-tuning the solution:

The solution candidates $\hat{\boldsymbol{x}}_2^*$ obtained from the TTGO (i.e., maxima of $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$) can be further refined using local search methods to obtain the optimal solution $\boldsymbol{x}_2^*$. If all the decision variables are continuous and the cost function is differentiable, we can use $\hat{\boldsymbol{x}}_2^*$ as a warm-start for gradient-based optimization techniques to find the nearest local optima. If some of the decision variables are discrete, we can fix the discrete variables and only optimize the continuous ones. In this chapter, we used SLSQP for the refinement and we only deal with continuous variables.

Hierarchically Finding the TT Model

When the objective function includes multiple objectives, we can approximate the corresponding PDF in the TT model using TT-cross in an efficient manner by exploiting the algebra associated with the TT. Suppose the cost function to be minimized is $C(\boldsymbol{x}) = C_a(\boldsymbol{x}_a) + C_b(\boldsymbol{x}_b)$, (or $C(\boldsymbol{x}) = C_a(\boldsymbol{x}_a)\, C_b(\boldsymbol{x}_b)$) where $\boldsymbol{x} = \boldsymbol{x}_a \cup \boldsymbol{x}_b$. For example, C_a could be the cost for obstacle avoidance which is only a function of joint angles and C_b could be the cost for target reaching which is a function of joint angles and position of the target. Minimizing this cost function corresponds to maximizing the PDF $P(\boldsymbol{x}) = P_a(\boldsymbol{x}_a) + P_b(\boldsymbol{x}_b)$. We can find the TT model $\mathcal{P}$ corresponding to P using $\mathcal{P}_a$ and $\mathcal{P}_b$ which are TT models corresponding to P_a and P_b respectively and they are often more smoother and easier to model using TT-cross. Then, we can quickly compute $\mathcal{P} = \mathcal{P}_a + \mathcal{P}_b$ as an addition operation over tensor in TT format (see Section 2.6). This offers an efficient way to model a target PDF in TT format by separately modeling multiple individual components which are often favorable to compute in terms of dimensionality and smoothness.

4.5 Experiments

In this section, we evaluate the performance of the proposed algorithm with several applications. A PyTorch-based implementation of the algorithms and the accompanying videos are available at https://sites.google.com/view/ttgo/home. Our software is based on the library [79] for tensor networks. For all the applications considered in this chapter, the results are provided with Algorithm 4.

We evaluated it on challenging benchmark functions such as the Rosenbrock function, Himmelblau function, and Gaussian Mixture Models, which are difficult for gradient-based optimization techniques. The experimental analysis provided in Appendix B.1 demonstrates the proposed method can consistently find global optima, and multiple solutions when they exist, and it can adapt to task parameters that influence the locations of global optima. Additionally, the prioritized sampling approach used in Algorithm 4 is evaluated, with small α values generating samples that cover a wide region around many local optima and α values close to one producing samples close to global optima.

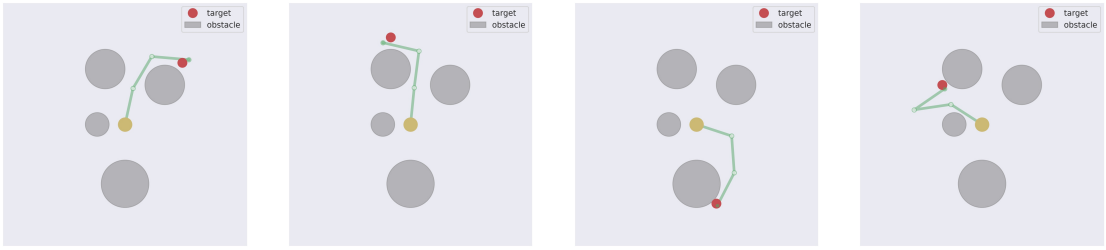
We then apply it to inverse kinematics with obstacles and motion planning problems. Besides qualitatively observing the solutions, we also perform quantitative analyses to evaluate the quality of the approximate solutions produced by our approach. We consider three different metrics:

TTGO for Numerical Optimization in Robotics

1. Training Phase (Offline):
   (a) Given:
      - Cost function $C(\boldsymbol{x}_1, \boldsymbol{x}_2)$,
      - Rectangular domain $\Omega_{\boldsymbol{x}} = \Omega_{\boldsymbol{x}_1} \times \Omega_{\boldsymbol{x}_2}$
   (b) Transform the cost function into an unnormalized PDF $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$.
   (c) Discretize the domain Ω_x into $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$.
   (d) Using TT-Cross, construct the TT-Model $\boldsymbol{\mathcal{P}}$ as the discrete analogue of $P(\boldsymbol{x})$ with discretization set $\mathcal{X}$. Refine and normalize the TT-model as described in Section 2.5.2 and 4.3.4.
2. Inference Phase (Online):
   (a) Given: The task-parameter $\boldsymbol{x}_1 = \boldsymbol{x}_t \in \Omega_{\boldsymbol{x}_1}$, the desired number of solutions K.
   (b) Construct the conditional TT Model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}$ from $\boldsymbol{\mathcal{P}}$ (see Section 2.9.3).
   (c) Generate N candidate solutions $\{\boldsymbol{x}_2^l\}_{l=1}^N$ from the TT distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t) = \frac{(\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}_{\boldsymbol{x}_2})^2}{Z}$ (Algorithm 4 or 5).
   (d) Evaluate the cost function at these samples and choose the best-K samples as approximation for the optima $\{\hat{\boldsymbol{x}}_2^{*^l}\}_{l=1}^K$.
   (e) Fine-tune the approximate solutions using gradient-based approaches on $C(\boldsymbol{x}_t, \boldsymbol{x}_2)$ to obtain the optima $\{\boldsymbol{x}_2^{*^l}\}_{l=1}^K$.

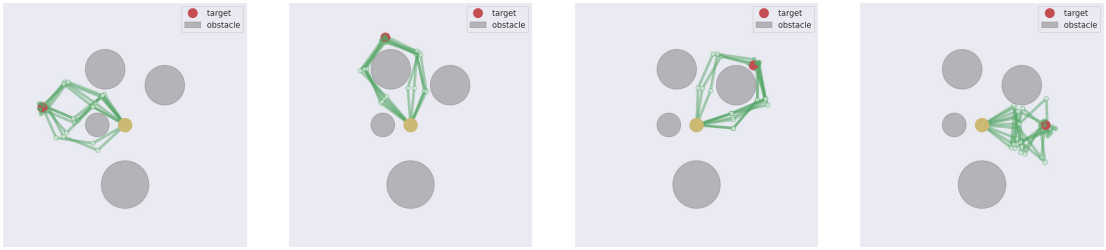
- c_i, the initial cost value of the approximate solutions from TTGO.
- c_f, the cost value after refinement.
- **Success**, the percentage of samples that converge to a good solution, i.e., with the cost value below a given threshold.

To compare the performance of the proposed approach with random initialization, we initialize the solver with random samples generated from the *uniform* distribution across the entire domain. To observe the effect of prioritized sampling, we also use TTGO with various values of α. The performance evaluation involves generating 100 random test cases within the task space. For each test case, we generate N samples using both the



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

Figure 4.3: A single sample taken from a conditional TT distribution with $\alpha = 1$ for inverse kinematics of a 3-link planar manipulator in the presence of obstacles (gray spheres). The yellow circle and the green segments depict the base and the links of the robot, respectively. The target end-effector positions are shown in red. The samples are very close to the targets and collision-free, even without fine-tuning the solutions.



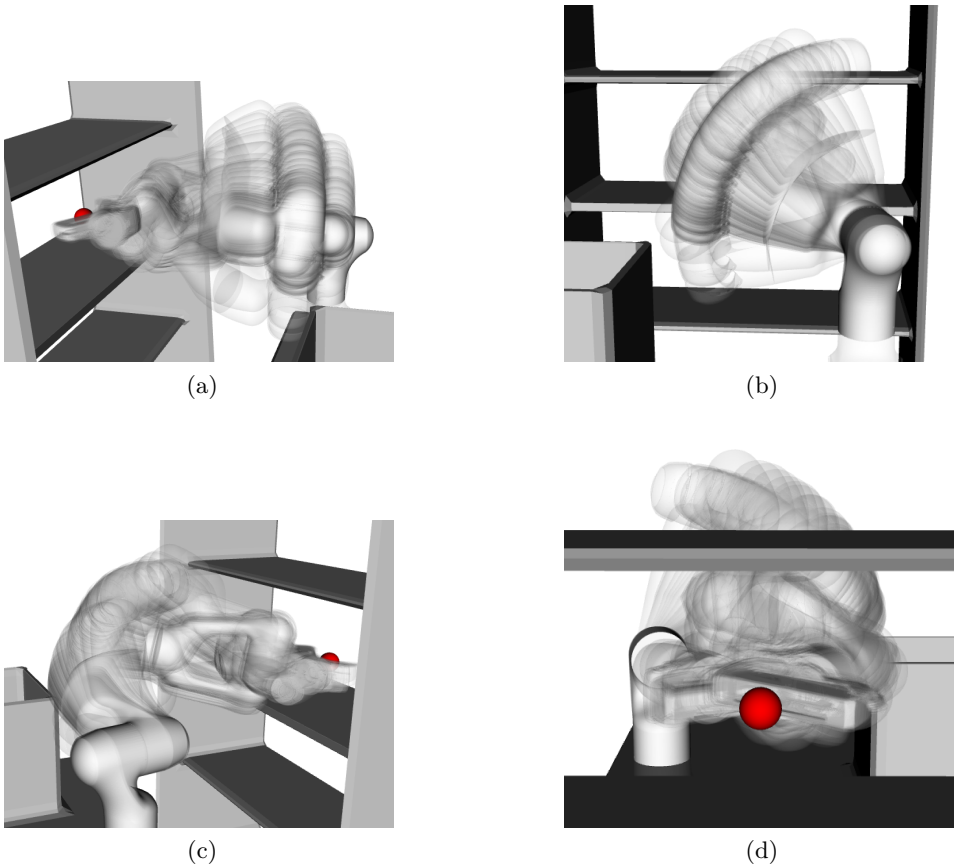
(a) (b) (c) (d)

Figure 4.4: Best 10 out of 50 samples taken from a conditional TT distribution with $\alpha = 0.8$ for inverse kinematics of a 3-link planar manipulator in the presence of obstacles. The samples are already close enough to the optima even without fine-tuning the solutions and the multimodality of the solutions is clearly visible.

TT and uniform distribution methods and select the best sample based on the initial cost value as the approximate solution. We vary the number of samples N from 1 to 1000. The SLSQP solver is used to optimize the sample with respect to the given cost function. We then evaluate the initial cost c_i, the final cost after refinement c_f, and the convergence status for each method. The average performance of both methods across all test cases is computed. The results for the robotics tasks are summarized in Table 4.1-4.3 and are discussed in the corresponding sections.

4.5.1 Inverse Kinematics Problems

We consider here the optimization formulation of numerical Inverse Kinematics (IK). The task parameters $\boldsymbol{x}_1$ then correspond to the desired end-effector pose, while the decision



(a) (b)

(c) (d)

Figure 4.5: The samples taken from a conditional TT distribution for the IK of a Franka Emika manipulator in the presence of obstacles, after refinement. We can see that there is a continuous set of solutions due to the additional degrees of freedom.

variables $\boldsymbol{x}_2$ are the joint angles. We use approximately $n_2 = 50$ discretization points for each of the joint angles ($\sim 5°$) and approximately $n_1 = 200$ discretization points (~ 0.5cm) for each task parameter. $\Omega_{\boldsymbol{x}_1} \subset \mathbb{R}^3$ is the rectangular space that includes the robot workspace. $\Omega_{\boldsymbol{x}_2} = \times_{k=1}^{d_2}[\theta_{\min_k}, \theta_{\max_k}]$, $\Omega_{\boldsymbol{x}_2} \subset \mathbb{R}^{d_2}$ where $[\theta_{\min_k}, \theta_{\max_k}]$ represents the joint angle limits for the k-th joint.

We consider two IK problems: 6-DoF IK to clearly demonstrate the multimodal solutions and 7-DoF IK with obstacle cost to consider the infinite solution space. In both cases, we transform the cost function into an unnormalized density function as $P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x})^2)$.

Inverse Kinematics for 6-DoF Robot:

A 6-DoF robot has a finite number of joint angle configurations that correspond to a given end-effector pose. In this section, we consider the 6-DoF Universal Robot that can have up to 8 IK solutions. While there is an analytical solution for such robots, it is a nice case study to illustrate the capability of TT to approximate multimodal distributions in a robotics problem where the modes are very distinct from one another. We constrain the end-effector orientation to a specific value (i.e., facing upward without any free axis of rotation), and set the end-effector position as the task parameter. Hence, $\boldsymbol{x}_1 \in \Omega_{\boldsymbol{x}_1} \subset \mathbb{R}^3$ while $\boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2} \subset \mathbb{R}^6$, so $d = 9$, where $\Omega_{\boldsymbol{x}_1}$ is the rectangular domain enclosing the workspace of the manipulator.

We observe that our approach is able to retrieve most of the 8 IK solutions for a given end-effector pose. Figure 4.6 shows the refined samples from TTGO by conditioning the TT distribution on a desired end-effector position. This validates our claim that TTGO is able to provide multimodal solutions even for a complex distribution.

Inverse Kinematics for 7-DoF Robot with Obstacle Cost:

A 7-DoF robot can have an infinite number of joint angle configurations for a given end-effector pose, unlike a 6-DoF robot. It can also have multiple solution modes similar to the 6-DoF robot. To ensure a collision-free solution, we introduce an obstacle cost to the optimization formulation, using the same collision cost as in CHOMP [63]. This collision cost uses a precomputed Signed-distance Function (SDF) to compute the distance between each point on the robot link and the nearest obstacle. When there are obstacles, numerical IK typically involves generating multiple solutions and checking for collision until a collision-free configuration is found. However, in cluttered environments, this approach may have a low success rate, requiring the user to generate many IK solutions before finding a collision-free one. By adding an obstacle cost to optimize for collision-free configurations directly, the non-convexity of the problem increases significantly, leading the solver to get stuck at poor local optima, especially with a high weight on the obstacle cost. Therefore, it is an interesting case study to demonstrate how TTGO can avoid poor local optima and find robust solutions in this challenging scenario.

We first test the IK with obstacle cost for a 3-DoF planar robot to provide some intuition on the effectiveness of our approach. Figure 4.3 and Figure 4.4 show some samples from TTGO conditioned on the target end-effector position (shown in red). By setting $\alpha = 1$, we focus the sampling around the mode of the distribution, enabling us to obtain a very good solution even with only 1 sample (Figure 4.3). As we decrease α to 0.8 and retrieve more samples, we can see that multiple solutions can be obtained easily (Figure 4.4).

Note that even without the refinement step, all samples reach the goal closely while being collision-free.

We then apply the formulation on the 7-DoF Franka Emika robot, where the collision environment is set to be a table, a box, and a shelf. The task parameters correspond to the end-effector position in the shelf, while the gripper is constrained to be oriented horizontally with one free DoF around the vertical axis. Hence, $\boldsymbol{x}_1 \in \mathbb{R}^3$ while $\boldsymbol{x}_2 \in \mathbb{R}^7$, so $d = 10$. The number of parameters of the TT cores is 1.4×10^7 whereas the original tensor $\boldsymbol{\mathcal{P}}$ has 1×10^{18} parameters. TT-cross found the tensor in TT-format using only 2×10^8 evaluations of the function P. For this application, a rank of 60 already produces satisfactory performance.

Figure 4.5 shows samples generated from a TT distribution on a given end-effector position after refinement. Note that unlike in the 6 DoF case, we can see here a continuous set of IK solutions due to the additional degrees of freedom. We also note that distinctly different modes of solutions can also be observed in this case, as can be seen in the accompanying video.

The results are reported in Table 4.1. We can see that our approach consistently outperforms uniform sampling by a wide margin across the three metrics. The initial cost values of TTGO samples are much lower than uniform samples, and after refinement, they converge to smaller cost values on average. The success rates of TTGO samples are also much higher. Furthermore, from qualitative analysis, the approximate solutions of TTGO are very close to the optimized solution. It is especially important to note that the best out of 1000 uniform samples (bottom right corner of the table) is still worse than a single sample from TTGO with $\alpha > 0.75$ (top left corner).

We can see the effect of prioritized sampling by comparing the performance of different values of α. In general, using higher values of α improves the performance, as we concentrate the samples around the high-density region. TTGO samples with $\alpha = 0.9$ have impressive performance with 94% success rates even by using only one sample per test case. However, higher α means less diversity of solutions, so a trade-off between solution quality and diversity needs to be considered when choosing the value of α. Note that even with $\alpha = 0$ we still obtain a very good performance by using as few as 10 samples.

Tables 3.1–3.3 The performance measures for three different applications with the Franka Emika manipulator. We compare the performance of our approach for initializing a given gradient-based solver (namely, SLSQP) against initialization from uniform distribution. I The three performance metrics are the cost at the initialization (c_i), the cost after optimization (c_f) using the solver and the success rate (**Success**). The criteria for success is that $c_f \leq 0.25$. We compute the average of each of these measures over 100 randomly chosen test cases. Each of the target points are chosen so that they are sufficiently away from the surface of the obstacle but they are not guaranteed to be feasible.

Table 4.1: Inverse kinematics of the Franka Emika robot

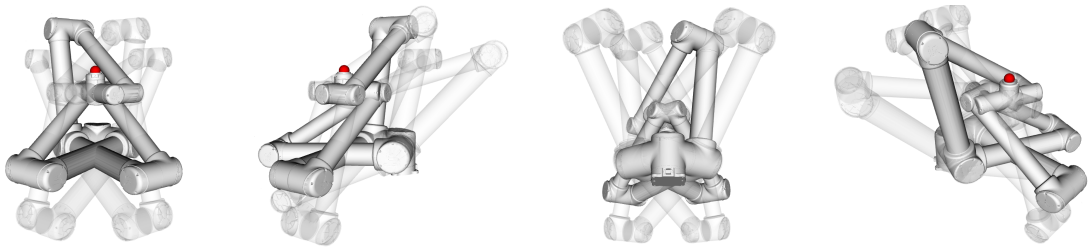
Method	α	# Samples											
		1			10			100			1000		
		c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**
TTGO	0.9	1.04	0.01	94.00%	0.55	0.02	98.00%	0.37	0.02	98.00%	0.26	0.02	99.00%
	0.75	1.52	0.07	84.00%	0.65	0.02	95.00%	0.37	0.02	95.00%	0.24	0.03	97.00%
	0.5	2.01	0.08	88.00%	0.85	0.04	93.00%	0.43	0.04	93.00%	0.28	0.01	98.00%
	0	2.88	0.17	71.00%	1.23	0.05	91.00%	0.68	0.05	91.00%	0.39	0.04	96.00%
Uniform	-	8.42	1.22	37.75%	4.47	0.91	45.50%	2.56	0.5	59.25%	1.59	0.24	75.00%

Table 4.2: Target Reaching

Method	α	# Samples											
		1			10			100			1000		
		c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**
TTGO	0.9	3.99	0.17	62.00%	1.1	0.09	86.00%	0.71	0.1	86.00%	0.58	0.09	88.00%
	0.75	5.63	0.21	53.00%	1.29	0.14	72.00%	0.78	0.1	86.00%	0.56	0.1	83.00%
	0.5	4.53	0.17	50.00%	1.54	0.14	64.00%	0.96	0.11	83.00%	0.62	0.1	84.00%
	0	6.7	0.31	46.00%	2.06	0.18	60.00%	1.3	0.12	82.0	0.84	0.12	86.00%
Uniform	-	13.85	1.34	19.25%	4.79	0.91	28.75%	3.02	0.68	41.00%	2.06	0.45	53.50%

Table 4.3: Pick-and-Place

Method	α	# Samples											
		1			10			100			1000		
		c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**
TTGO	0.9	2.41	0.16	70.00%	1.41	0.15	81.00%	1.05	0.15	79.00%	0.87	0.14	89.00%
	0.75	3.25	0.17	66.00%	1.71	0.17	66.00%	1.31	0.14	84.00%	1.01	0.15	78.00%
	0.5	4.31	0.26	54.00%	2.33	0.19	62.00%	1.66	0.17	77.00%	1.29	0.18	76.00%
	0	6.2	0.27	48.00%	2.98	0.23	48.00%	2.17	0.21	58.00%	1.61	0.18	71.00%
Uniform	-	9.64	0.78	23.75%	5.23	0.63	30.25%	3.95	0.49	39.5%	3.07	0.39	44.25%



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

Figure 4.6: 8 IK solutions of the UR10 robot for a given pose from TTGO samples after refinement, shown from four different views. 5 of the solutions are drawn transparently to provide better visualization. The desired end-effector position is shown in red.

4.5.2 Motion Planning of Manipulators

In this section, we explore the use of our framework in the motion planning of the Franka Emika robot to generate obstacle-free robot motions. This problem is high dimensional, as a robot with m degrees of freedom and T time intervals results in optimization variables $\boldsymbol{x}_2$ with mT dimensions. To address this issue, we adopt a trajectory representation using movement primitives with basis functions, as commonly done in learning from demonstration [80, 33]. The optimization variables in this representation consist of the superposition weights of the basis functions. Our formulation of movement primitives guarantees that the motion always starts from the initial configuration and ends at the given final configuration. In the case of a goal reaching in the task space, we need to first determine the corresponding final configuration, which can be done using inverse kinematics. In our motion planning formulation, we optimize both the final configuration and the weights of the basis functions jointly.

The cost function in our motion planning formulation includes the reaching cost, the joint limit cost, the smoothness cost, and the obstacle cost, which is the same cost used in inverse kinematics. It is important to note that if we want to ensure that the solution avoids small obstacles, the number of time discretizations must be large, which can result in more than 700 dimensions for motion planning with a Franka Emika robot. The use of the obstacle cost helps the solver directly optimize for a collision-free configuration, but it also significantly increases the non-convexity of the problem, making it susceptible to getting stuck in poor local optima, especially with a large weight on the obstacle cost. Details on the motion planning formulation can be found in Appendix B.3.

We consider two different motion planning tasks as follows:

1. **Target Reaching**: From the initial configuration $\boldsymbol{\theta}_0 \in \mathbb{R}^m$, reach a target location $\boldsymbol{p}_d \in \mathbb{R}^3$.
2. **Pick-and-Place**: From the initial configuration $\boldsymbol{\theta}_0 \in \mathbb{R}^m$, reach two target locations $\boldsymbol{p}_d^1$ (picking location) and $\boldsymbol{p}_d^2$ (placing location) in sequence before returning to the initial configuration $\boldsymbol{\theta}_0$.

For the target reaching problem, the task parameter is the target location $\boldsymbol{x}_1 = \boldsymbol{p}_d$ and the decision variables $\boldsymbol{x}_2 = (\boldsymbol{\theta}_1, \boldsymbol{w})$. Here, $\boldsymbol{\theta}_1 \in \Omega_{\boldsymbol{\theta}} \subset \mathbb{R}^m$ is the joint angle defining the final configuration and $\boldsymbol{w} = (\boldsymbol{w}^k)_{k=1}^m \in \mathbb{R}^{Jm}$, where $\boldsymbol{w}^k = (w_j^k)_{j=1}^J \in \boldsymbol{R}^J$ are the superposition weights of the basis functions representing the motion from $\boldsymbol{\theta}_0$ to $\boldsymbol{\theta}_1$. We use $J = 2$ and $m = 7$ for the 7-DoF Franka Emika manipulator, so the total number of dimensions for the reaching task is $d = 3 + 7 + 2 \times 7 = 24$.

For the pick-and-place problem, the task parameters are the two target locations (pick and place location): $\boldsymbol{x}_1 = (\boldsymbol{p}_d^1, \boldsymbol{p}_d^2)$. The decision variables are $\boldsymbol{x}_2 = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, {}^{01}\boldsymbol{w}, {}^{12}\boldsymbol{w}, {}^{20}\boldsymbol{w})$, where $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are the configurations corresponding to the two target points, $\boldsymbol{w} = ({}^{01}\boldsymbol{w}^k, {}^{12}\boldsymbol{w}^k, {}^{20}\boldsymbol{w}^k)_{k=1}^m$ where ${}^{uv}\boldsymbol{w} \in \mathbb{R}^{Jm}$ are the weights of the basis functions representing the movement from the configuration $\boldsymbol{\theta}_u$ to $\boldsymbol{\theta}_v$. Hence, the total number of dimensions for the pick-and-place task is $d = 2 \times 3 + 2 \times 7 + 3 \times 2 \times 7 = 62$.

We use the transformation $P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x})^2)$. The target location $\boldsymbol{p}_d$ for target reaching and $\boldsymbol{p}_d^1$ in the pick-and-place problem are inside the shelf as in the IK problems (picking location). For the pick-and-place task, the second target location $\boldsymbol{p}_d^2$ is on the top of the box (drop location). We discretize each of the task parameters using 100 points and the decision variables with 30 points. We use radial basis functions with $J = 2$, which we find sufficient for our applications. The bounds on the weights of basis function for a joint are the same as the joint limits i.e., $(w_{min}^k, w_{max}^k) = (\theta_{min_k}, \theta_{max_k})$.

Figure 4.7 shows some examples of a reaching task for a 3-DoF planar manipulator. We can see here that the TTGO samples lead to good solutions, i.e., they avoid collisions while reaching the target quite accurately. In comparison, random sampling initialization often results in poor local optima, where the final solutions still have collisions even after the refinement. Figure 4.8 shows the same reaching task for the Franka Emika robot, where the multimodality of the solutions is clearly visible. We also test the trajectory on the real robot setup as shown in Figure 4.10 and 4.11.

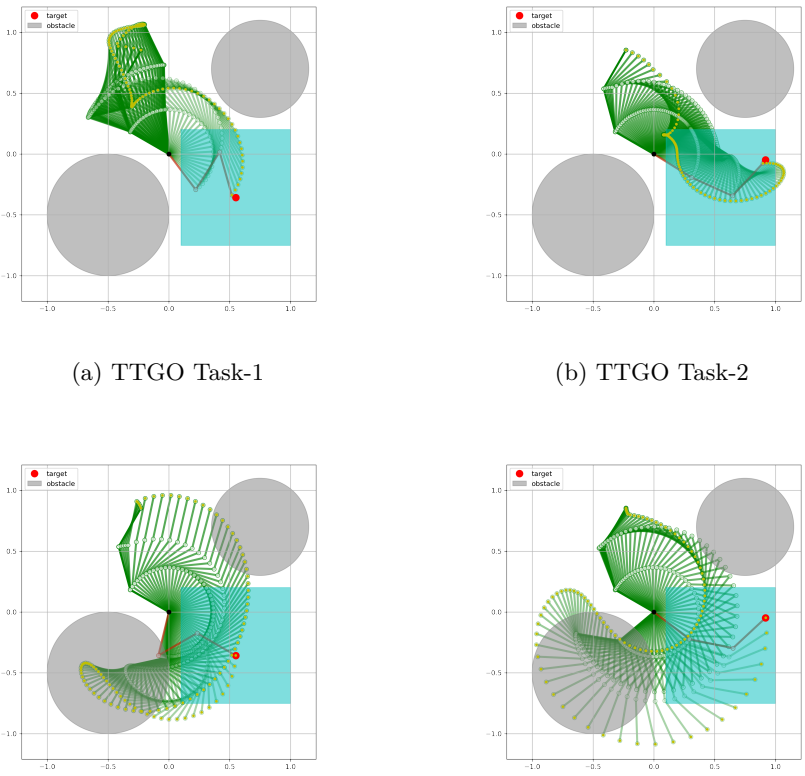
The results are presented in Table 4.2 and 4.3. Similarly to the IK results, our approach outperforms uniform sampling by a wide margin across all metrics. In reaching tasks and

especially in pick-and-place tasks, uniform sampling performs quite badly in terms of success rates, since the tasks are much more difficult than the IK problem. Taking only 1 TTGO sample also does not produce satisfying performance here (i.e., $\sim 60 - 70\%$) success rates, but using 10-100 samples already makes a good improvement. In pick-and-place tasks, since we consider the three different phases as a single optimization problem, it becomes quite complicated, and low values of α do not provide good success rates, but prioritized sampling with $\alpha = 0.9$ manages to achieve 89% success rates using 1000 TTGO samples. This is mainly because this task is of very high dimensions with complicated cost functions involved resulting in large modeling errors by TT-cross in representing the probability function in TT format.

4.5.3 Application to Single Task Optimization

Up to now, we have presented our approach using TTGO in its general form, where we take into account different task parameters during the training of the TT model. This approach allows us to rapidly generate approximate solutions for a specific task by conditioning the TT model. However, our approach is also applicable when we only need to solve a single task. In this case, the TT model represents the probability distribution of only the optimization variables, and the training time is substantially shorter compared to the general case. We found that a maximum TT-rank of less than 5 works well for the applications examined in this study. In terms of computational time and solution quality, our approach is comparable to evolutionary methods such as CMA-ES or GA, but with the added benefit of being able to provide multiple solutions.

For such applications involving a single task parameter, our work is closely related to TTOpt [59]. TTOpt is a gradient-free optimization technique based on TT-cross, which has been shown to perform comparably to evolutionary strategies. The goal of TTOpt is to maximize a reward function, which is similar to the probability density function considered in this work. TTOpt discretizes the reward function and assumes that the maximum element of the discrete approximation of the reward function closely approximates the true maximum. TT-cross is used in TTOpt to find the maximum of the discrete approximation. In this context, TT-cross is used not to build a TT approximation, but rather for its ability to identify the maximal elements of a tensor, which are likely to be in the maximum volume submatrix. This submatrix is found using the MAXVOL algorithm in TT-cross, and the maximal element of the submatrix is updated monotonically over iterations. During each iteration, the maximal element from the submatrix is stored in memory and updated until convergence. In contrast, we first model the density function using TT-cross and then use TTGO to approximate a solution, which is then refined using local search techniques, with the option of estimating



(a) TTGO Task-1

(b) TTGO Task-2

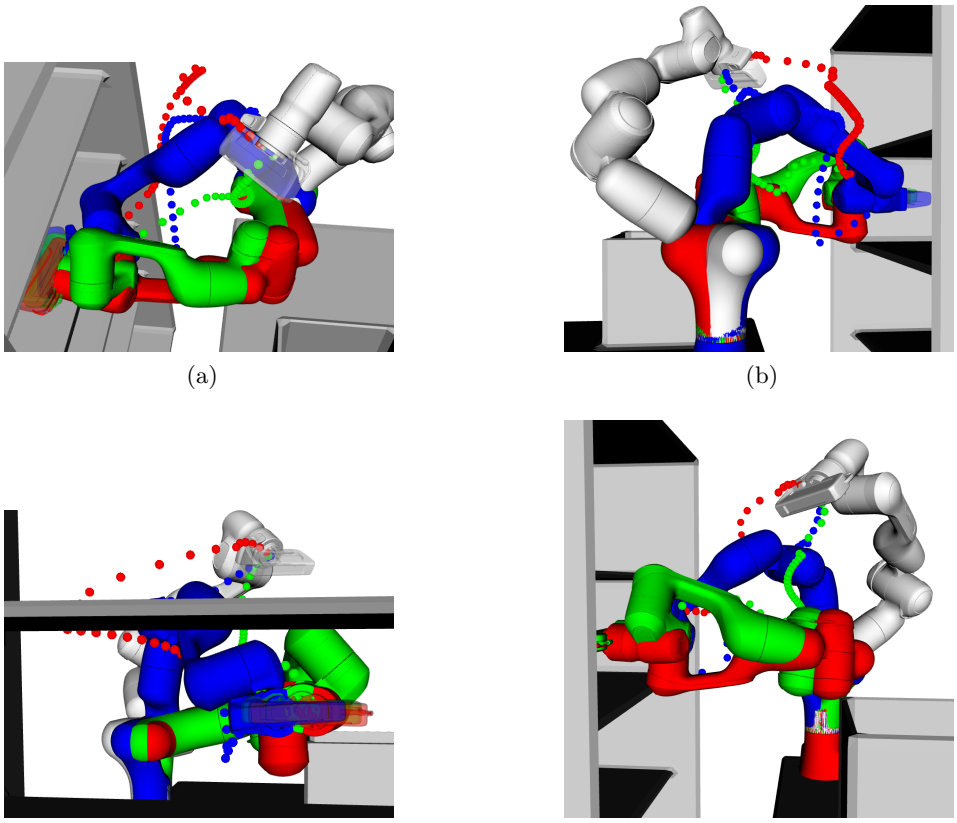
(c) Random Task-1

(d) Random Task-2

Figure 4.7: Motion Planning of Planar Manipulators: The task is to reach a given target point in the square region depicted in cyan (task space) from a fixed initial configuration (dark green configuration). The final configuration and the joint angle trajectory to reach the target point are the decision variables. The approximate solutions from TTGO for two different tasks are given in (a) and (b) (before refinement). The solution obtained by a gradient-based solver with random initialization can result in poor local optima, as can be seen in (c) and (d).

multiple solutions.

To test the performance of our approach for single-task optimization, we have applied it to motion planning of both the 2-D planar robot and Franka Emika manipulator. We set the initial and the desired final configurations, and TTGO finds the trajectory to move to the final configuration while avoiding the obstacles. The joint angle trajectory is represented using the motion primitives as described in Appendix B.4, thus the optimization variables are the weights of the basis functions. We used 2 radial basis functions for each joint.

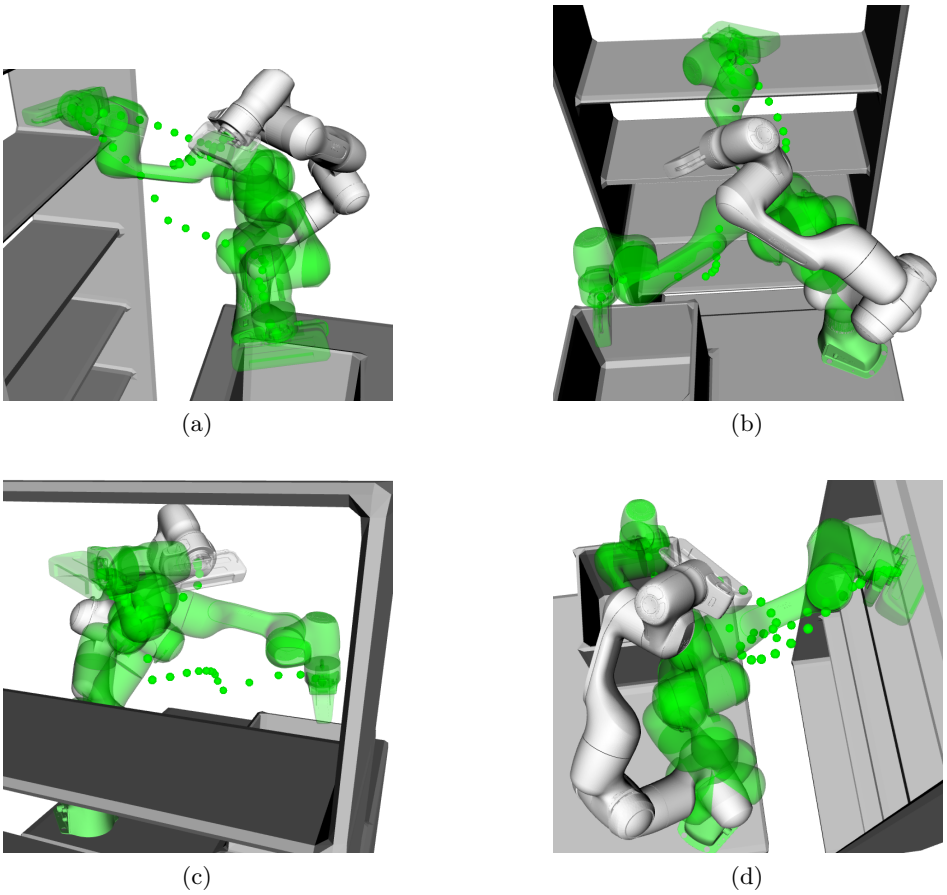


(a) (b) (c) (d)

Figure 4.8: Best 3 out of 1000 samples taken from a conditional TT distribution with $\alpha = 0.75$ for the reaching task of a manipulator in the presence of obstacles, after refinement. The initial configuration is shown in white, while the final configuration is shown in red, green, and blue, for each solution. The end-effector path is shown by the dotted curves. The multimodality is clearly visible from these three solutions.

For the 2-D planar robot, we replicate the setting in Figure 7 of [60], but we move the obstacle positions and add two more obstacles to increase the difficulty of the problem. With a fixed task parameter, the training of the TT model only takes less than 7 seconds, and we easily obtain multiple solutions. Figure 4.12 shows four solutions obtained by TTGO after the refinement step. We can clearly see the multimodality of the solutions.

For the Franka Emika manipulator, we use the same setting as in Section 4.5, i.e., with the shelf, table, and box as the collision objects. In addition, we add a cost to maintain the end-effector pose (horizontal) throughout the trajectory. The initial and final configurations are set such that both end-effector positions are located within the shelf, and they are computed using TTGO for IK, as explained in Section 4.5.1. With this



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

Figure 4.9: A sample taken from a conditional TT distribution for the pick-and-place task, after refinement. (a) to (d) represent the same motion in different perspectives. In green, we see the picking configuration (from the shelf) and placing configuration (on the box), while the initial configuration is shown in white. The end-effector positions in the shelf and the box are the task parameters.

setting, we are able to obtain multiple solutions consistently for all possible scenarios (we test with different end-effector positions within the shelf) with 10 iterations of TT-cross and a maximal TT-rank of 5. With the fixed task parameter, it only takes under 5 seconds to obtain the solutions (includes TT modeling, sampling and fine tuning). Some solutions for a given task are shown in Figure 4.1.

In comparison, SMTO [60] takes ~ 2 minutes to solve the 2-D planar robot problem and the 7-DoF manipulator example (using their MATLAB code), whereas LSMO [61] takes even longer, i.e., more than five minutes (according to their paper). For the 2-D example, SMTO fails to find any solution when we added more obstacles as in Figure 4.12, even

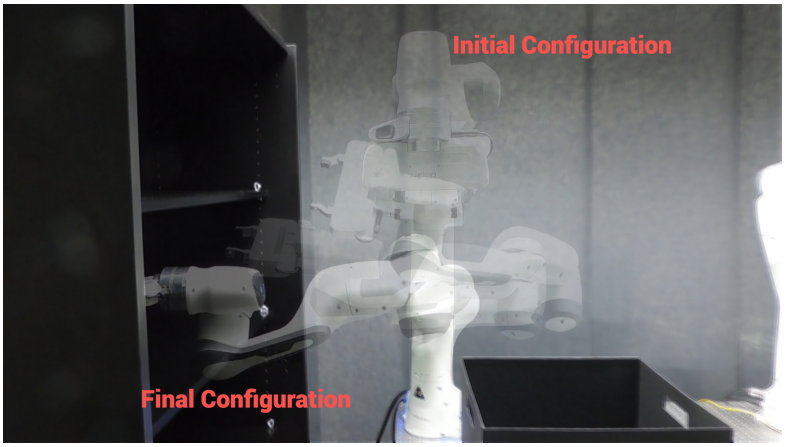


Figure 4.10: The motion from the initial configuration to the final configuration in the real robot implementation of one of the TTGO solutions for the reaching task.

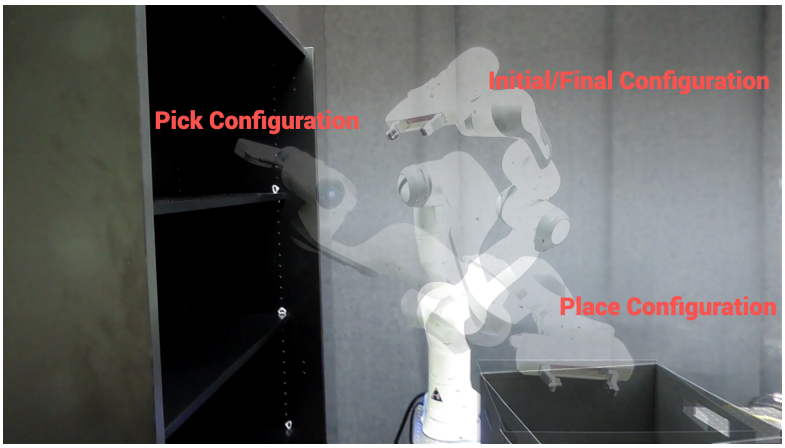
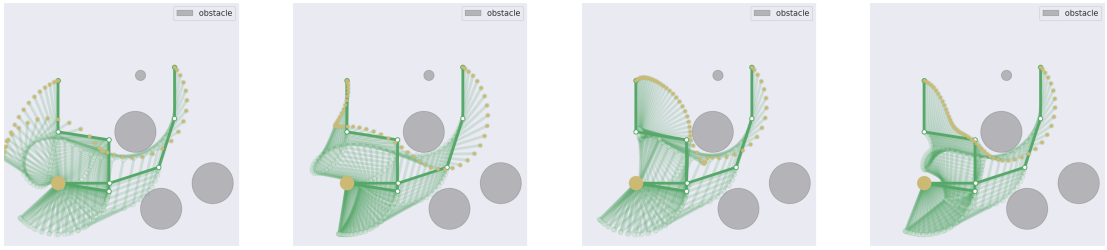


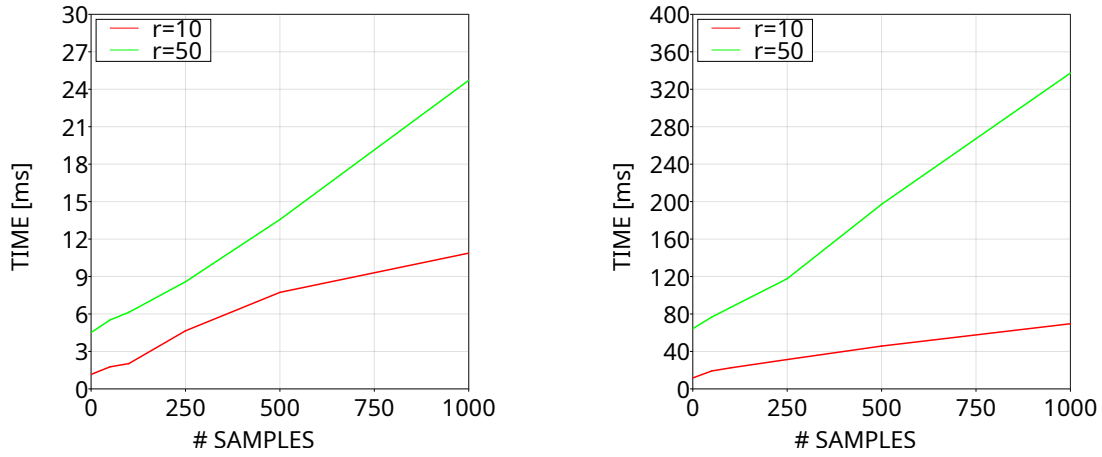
Figure 4.11: Real robot implementation of one of the TTGO solutions for the pick-and-place task. The motion from the initial configuration to the final configuration (same as the initial configuration in this case) via the picking configuration and placing configuration is depicted.

after increasing the covariance by 100 times. This is because none of the initial samples from the proposal distribution is close to the feasible region. We also tried increasing the number of samples from 600 (standard value) to 2000, but it still cannot find any solution. Furthermore, adding the number of samples by ~ 3 times increases the computation time of SMTO by ~ 3 times, i.e., from $\sim 150s$ to $\sim 500s$.



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

Figure 4.12: Four different solutions obtained by TTGO for a motion planning task with 4-link planar manipulator. The initial and final configuration are given (dark green) and the optimization variables are the weights of the basis functions (two basis functions per joint) that determine the joint angle trajectory.



(a) $d = 7$ (b) $d = 70$

Figure 4.13: Sampling Time: The sampling procedure has a computational complexity of $\mathcal{O}(ndr^2)$ and it is independent of the application. (a) and (b) show the computation time curves for two different values of d with the size of each mode being $n = 100$. For each figure, we show the sampling time for two different ranks as shown in red ($r = 10$) and green ($r = 50$).

4.6 Discussion

4.6.1 Quality of the Approximation

In this chapter, we used a TT model to approximate an unnormalized PDF. The quality of the approximation of the PDF using the TT model highly depends on the TT-rank. If the

approximation is good, the TTGO solutions are more accurate and the fine-tuning step is often not necessary. A nice property of our approach that is derived from the TT-Cross method is that the model capacity can be incrementally augmented (i.e., non-parametric modeling). By increasing the number of iterations of TT-Cross and allowing a higher rank of the TT model, the approximation accuracy can be improved continuously. For initialization purposes, though, we found that the discrete version is enough, as the initialization does not have to be precise.

When training the TT model, we can evaluate the quality of the approximation by picking a set of random indices, computing the value of the approximate function at those indices, and comparing it against the actual function value. This is an important evaluation for most applications that aim at finding an accurate low-rank TT decomposition of a given tensor across the whole domain. For our case, though, we are only interested in the maxima of the function, and we do not really care about the approximation accuracy in the low-density region, i.e., the region with high cost. Even if TT-Cross cannot find an accurate low-rank TT representation across the whole domain (e.g., due to non-smoothness), it can still capture the maximal elements robustly [59, 15] as the interpolation in the TT-Cross algorithm is done using the high magnitude elements. In practice, we found that even when the approximation errors do not converge during the training, the resulting samples from the TT model are still very good as initialization.

4.6.2 Comparison with Previous Work using Variational Inference

As described in Section 4.2.3, the work closest to our approach is SMTO [60] that also transforms the cost function into an unnormalized PDF. SMTO uses Variational Inference to find the approximate model as a Gaussian Mixture Model (GMM) by minimizing the forward KL divergence. Its main limitation, however, is that it requires a good proposal distribution to generate the initial samples for training the model. These samples are used to find the initial GMM parameters, and subsequent iterations sample directly from the GMM. Hence, the initial samples have a large effect on the final solutions. When the initial samples do not cover some of the modes, subsequent iterations will have a very small chance of reaching those modes. We verified this by running the open-source codes provided by the author. Even for the 4-DoF manipulator example (Figure 7 in their paper), with the standard parameters given by the author, SMTO cannot find a single solution when the position of the obstacles are changed to increase the difficulty of the motion planning problem (e.g., by moving the large obstacle closer to the final configuration). It starts to find a solution only after we increased the covariance of the proposal distribution by 10-100 times the standard values, because the initial samples can then cover the region near the feasible solutions. Furthermore, when we added one

more obstacle, SMTO failed to find any solution, even with the higher covariance and a larger number of samples. In comparison, we have shown in this work that our approach can solve difficult optimization problems reliably while also providing multiple solutions. Their 4-DoF setup is in fact very similar to our planar manipulator example in Figure 4.7, and we have shown in Section 4.5.3 that our approach can consistently produce good solutions for different target locations. Since our approach does not use any gradient information to find the TT model, it does not get stuck in poor local optima easily.

In [61], another method called LSMO was proposed to handle functions with an infinite set of solutions by learning the latent representation. As we showed in Appendix B.1 for sinusoidal and Rosenbrock functions, our approach is naturally able to handle these kinds of distributions, even without any special consideration or change on the method. Unlike our approach, SMTO and LSMO need to solve every single optimization problem from scratch. With the terminology used in our work, this corresponds to the task parameters being constant—a special case of the problem formulation considered so far in this chapter. For such problems, since we only have a single task, the training phase in our approach can be much faster by using a very low TT rank ($r < 10$ almost always works for most optimization problems without task parameters) and fewer iterations of TT-Cross. The advantage of our approach in such applications as compared to other global optimization approaches such as CMA-ES is that TTGO can provide multiple solutions. For example, we could find the optima of a 50D mixture of Gaussians with 5 components and 30D Rosenbrock considered in Section B.1 in less than 2 seconds. In this way, TTGO can be considered as a tool for global optimization that can offer multiple solutions.

In this work, we proposed a more generic approach based on TTGO. By anticipating and parameterizing the possible optimization problems using the task parameters, our approach allows the distribution of the computational effort into an offline and an online phase. In practice, this means that most of the computation time takes place during offline computation, while the online computation (conditioning on the TT model and sampling from it using TTGO) only takes a few milliseconds. SMTO and LSMO, in comparison, take several minutes to solve a single motion planning problem for the 7-DoF manipulator case. Similarly, most trajectory optimization solvers (e.g., CHOMP, TrajOpt) and global optimization solvers (e.g., CMA-ES) can only solve a given optimization problem at each run.

4.6.3 Multimodality

As we have shown in this chapter, TTGO is able to generate samples from multiple modes consistently. Furthermore, continuing the iteration of TT-Cross will result in covering more modes as the rank of TT-model can be dynamically increased in the TT-Cross algorithm. However, unlike GMM, it is not easy to sample from only a specific mode, or to identify how many modes there are in a given problem. If we need to cluster the samples, standard clustering algorithms such as k-means clustering can be used.

4.6.4 Computation Time

The computation time of our approach can be divided into *offline computation*, i.e., the time to construct the TT model $\boldsymbol{\mathcal{P}}$, and *online computation*, i.e., the time to condition the TT model on the given task parameters and to sample (TTGO).

The offline training uses an NVIDIA GEFORCE RTX 3090 GPU with 24GB memory, while the sampling time evaluation is performed on an AMD Ryzen 7 4800U laptop.

The offline computation time depends on the number of TT-Cross iterations, the maximum rank r, and the discretization (i.e., how many elements along each dimension of the tensor). The number of function evaluations has $\mathcal{O}(ndr^2)$ complexity hence linear in terms of the number of dimensions and the number of discretization points. The computation time of a single cost function also has a significant influence on the computation time. However, we used parallel implementation with GPU that allows us to construct all of the models in our applications in an unsupervised manner (using TT-Cross) in less than one hour.

The rank r and the number of iterations of TT-Cross also determine the variety in the solutions proposed by TTGO. If the application does not demand multiple solutions, we can keep the maximum allowable rank of the TT model and the number of iterations of TT-Cross to be very low, which results in a significant saving in offline computation time and the sampling time in the online phase. However, for the experiments in this work, we kept the rank r to be reasonably large (about $r = 60$ for IK and motion planning problems with manipulators) so that we could obtain a variety of solutions from TTGO.

For most of the 2D benchmark functions, it takes less than 0.01s to obtain the TT model. For the high-dimensional mixture of Gaussians and Rosenbrock functions with $d < 30$, we could obtain good enough TT-models in less than 60s. It takes about 30s for the inverse kinematics problem with the Franka Emika robot, which corresponds to 30 iterations of TT-Cross. Finally, the target reaching task takes around 10 minutes

while the pick-and-place task takes around 1 hour. The motion planning computation time is relatively slower due to the time for computing a single cost function since we compute the obstacle cost at small time intervals. It can be made faster by considering the continuous collision cost as done in TrajOpt [65], since it allows us to use coarser time discretization for evaluating the collision cost, resulting in a faster evaluation of the cost function.

For the online computation time, the conditioning time is insignificant as it is very fast, so we focus on the sampling time. Unlike the TT model construction, the sampling time does not depend on the cost function and only depends on the size of the tensor. The computation complexity is $\mathcal{O}(ndr^2)$. Results of sampling time evaluation with the different number of samples averaged over 100 tests are given in Figure 4.13. We show the results for $d = 7$ and $d = 70$, roughly corresponding to the IK and the pick-and-place task, respectively. We can see that due to the parallel implementation, generating 1000 samples is not much different compared to generating 1 sample. For the IK problem, generating 1 sample takes around 1-3ms, which is comparable to the solving time of a standard IK solver. For the pick-and-place task, generating 1 sample takes around 15ms, much faster than a typical computation time for motion planning (typically in the order of 1s).

4.7 Limitations

One of the major limitations of our approach is to scale it to very high-dimensional problems. Although it has been tested up to 100 dimensions, many robotics problems involve an even greater number of dimensions. To address this issue, we utilized here motion primitive representations, which are effective for some trajectory planning applications. However, for other purposes, we may need to use nonlinear dimensionality reduction techniques such as autoencoders as a preprocessing step to determine task parameters and decision variables for TTGO. Another potential solution to this challenge is to explore the product-of-experts strategy, as presented in [81], which we plan to investigate in future work.

Although constraints like joint limits can be handled naturally in TTGO, other constraints in the optimization problem need to be handled by imposing a penalty on the constraint violation in the cost function itself (i.e., formulated as soft constraints, similar to the problem formulation in evolutionary strategies and reinforcement learning). This may not be ideal for some applications in robotics that require hard constraints. However, the existing techniques for constrained optimization are mostly gradient-based, hence sensitive to initialization. Thus, we could still use TTGO for initializing such solvers.

It should be noted that for our approach to achieve fast offline computation time, it is necessary to process a batch of cost function evaluations in parallel. Without access to this parallelization capability, the time required to find the TT model using TT-Cross could become unacceptably long.

4.8 Further Possible Extensions

Our approach can be applied to other robotics problems as long as they can be expressed as optimization problems. For example, optimal control formulates the task of determining control commands as an optimization problem. Recent research has employed a database approach to warm start an optimal control solver (as explained in Section 4.2.2), which could potentially benefit from the use of TTGO. It should be noted that control problems can be more demanding than planning problems since the cost function is more sensitive (i.e., slight changes in the control commands can result in significantly different state trajectories and cost values). Therefore, additional research is necessary to adapt the approach to such problems. Moreover, certain applications, like task and motion planning [82] or footstep planning for legged robots [83], can be formulated as Mixed Integer Programming (MIP). Given that our approach does not necessitate gradient information, combining discrete and continuous optimization can provide another compelling area for further research.

The choice of transformation used to obtain the probability function from the cost function plays an important role in our approach. In the chapter, we used an exponential function as the transformation function, however, a study on other possible transformation functions should be investigated in future work. Moreover, in many robotics applications, the user has the flexibility to design the cost function. This will also play a role in our approach, as smoother functions can be captured as a low rank TT model using TT-Cross with significantly lower computational cost. In the robotic applications considered in this chapter, we used the standard cost functions and it was non-smooth due to the cost on collision avoidance. However, a smoother cost function could still potentially be designed for such applications. This could improve the performance and the computation time reported in this chapter.

We used here an unsupervised approach for obtaining the TT model (and consequently the TT distribution, which captures low-cost solutions) using TT-Cross, which only requires the definition of the cost function. This approach is motivated by the fact that in various applications, it may not be feasible to access the samples (or solutions) that correspond to low cost for different task parameters. Nevertheless, if we have a repository of good solutions (i.e., optimal solutions for different possible task parameters), we can

use an alternative approach. Instead of using TT-Cross to obtain the TT model, we can use other modeling techniques like supervised learning or density estimation techniques, as described in [26, 84, 25]. These techniques can still capture multiple solutions, given the expressive power and generalization abilities of TT models, while also enabling quick retrieval of solutions, as explained in this work using TTGO.

Morover, TTGO has the potential to be utilized for Learning-from-Demonstration in robotics. One way to do this is by employing density modeling approaches [26, 25] to create models of the demonstrations in TT format for different tasks. The techniques proposed in this chapter can then be used in the inference phase to generate a new solution for a new task. We describe this in more detail in the subsequent chapter.

4.9 Conclusion

In this chapter, we introduced TTGO and a novel framework for providing approximate solutions to optimization problems. Our evaluation on challenging benchmark optimization functions and robotics applications (including inverse kinematics and motion planning) shows that our approach can produce high-quality solutions for difficult optimization problems that are often unsolvable with standard random initialization of solvers. Additionally, TTGO can provide multiple solutions from different modes, where applicable, and allows for adjustment of the sampling priority to either focus on obtaining the best solution or generating a diverse set of solutions. These features are highly beneficial in initializing optimization solvers for challenging robotics problems. Moreover, TTGO has the potential to be applied to other robotics tasks that can be formulated as optimization problems, such as task and motion planning or optimal control, and learning from demonstration. It could also serve as an alternative to mixed integer programming, which is commonly used in legged robotics and contact-rich manipulation. Future work will investigate these possibilities.

5 Learning to Control using Tensor Train

Optimal control of dynamic systems with nonlinear dynamics poses a significant challenge in robotics. To tackle this challenge, we introduce a novel algorithm, Generalized Policy Iteration using Tensor Train (TTPI), rooted in Approximate Dynamic Programming (ADP). We employ Tensor Train (TT) to approximate the state-value and advantage functions. Leveraging the optimization technique Tensor Train for Global Optimization (TTGO) from Chapter 4 for policy retrieval from the advantage function, allows us to effectively address complex nonlinear systems beyond the capabilities of existing ADP algorithms. Importantly, our algorithm excels in handling robotic systems with hybrid action spaces, a formidable challenge for current methodologies. Unlike existing ADP algorithms, the proposed approach does not make any assumption on the structure of the dynamics model and only requires access to a simulator. We first test the approach for various classical control problems. We demonstrate the superiority of our approach over previous baselines for some benchmark problems with hybrid action spaces. Additionally, the robustness and generalization of the policy for hybrid systems are showcased through a real-world robotics experiment involving a non-prehensile manipulation task.

Publication Note

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

- S. Shetty, T. Xue, and S. Calinon, "Generalized policy iteration using tensor approximation for hybrid control," in *International Conference on Learning Representations (ICLR)*, 2024, (spotlight paper, 5% acceptance rate)

Supplementary material including videos and source codes related to this chapter are available at: https://sites.google.com/view/ttpi4control/home

5.1 Introduction

Robotic systems often exhibit complex nonlinear dynamics that may involve hybrid actions. The need for real-time control, high precision, and adequate robustness to cope with disturbances or changes in the environment can result in demanding computational requirements that are challenging to meet with classical control methods. Optimal Control (OC) based on the principles of Dynamic Programming (DP) is a popular tool in robotics but they are still limited to systems with continuous actions and differentiable dynamics.

Approximate DP (ADP) and Reinforcement Learning (RL) overcomes the curse of dimensionality faced by classical DP algorithms by using function approximation techniques [85, 86]. ADP is synonymous with OC and uses the system's model to obtain an optimal policy, while RL focuses on learning a policy through trial-and-error interactions with the environment. Both methods aim to find a compact representation of the value functions to obtain a control policy. ADP faces difficulty in approximating the value function throughout the entire state space. Conversely, RL restricts its approximation to a smaller region where data is collected, resulting in limited generalizability but greater scalability. However, the existing approaches for both ADP and RL face challenges in handling hybrid action space. Furthermore, existing ADP approaches make assumptions on the dynamic model of the system (e.g., control affine), hence can not work with modern simulation tools, and also find it challenging to cope with large action spaces and hybrid states.

In this chapter, we present a novel ADP algorithm, called Generalized Policy Iteration using Tensor Train (TTPI) which overcomes the challenges faced by existing ADP methods for hybrid system control, and unlike existing methods does not require any assumption on the system dynamics and the reward function. TTPI is an approximate version of the Generalized Policy Iteration (GPI) algorithm—a DP algorithm that encompasses both Value Iteration (VI) and Policy Iteration (PI) algorithms. We use Tensor Train (TT) (see Chapter 2), to model the state-value and the advantage function.

TT acts as a versatile function approximator that allows us to simultaneously handle continuous and discrete state and action variables as described in Chapter 2. It approximates a given function as a sum of products of univariate functions, allowing for fast algebraic operations and interpretation. The use of TT-Cross (see Section 2.8) provides us with a powerful gradient-free method to approximate functions in a nonparametric manner, allowing us to achieve TT approximation of state-value and advantage function with a desired accuracy in a fast manner, by exploiting the knowledge of the system model (e.g., a simulator) and the reward function. Moreover, the TT representation of

the advantage function enables us to use optimization techniques such as Tensor Train for Global Optimization (TTGO) as described in Chapter 4 to retrieve policies for hybrid action spaces.

The TT representation is particularly effective when the function being approximated is smooth, resulting in a low-rank representation in the TT format. Our experiments demonstrate that such property is frequently observed in ADP while dealing with hybrid systems. Indeed, even though the system dynamics and reward functions may be non-smooth and discontinuous, the optimal value functions typically exhibit low-rank structures.

Contributions: We introduce TTPI, a novel ADP algorithm for optimal control that leverages TT as a function approximator to address the challenges of hybrid system control in robotics. Our approach is interpretable and eliminates the need for differentiability of the system dynamics and reward function, which is a common assumption in the existing ADP algorithms. Our experiments demonstrate that TTPI outperforms state-of-the-art algorithms in terms of both training time and performance on various benchmark control tasks for hybrid control. To showcase the practicality and generalization of our approach, we conducted a real-world robotic experiment where we successfully tackled a non-prehensile planar manipulation task that is notoriously difficult for existing control methods. Our results demonstrate the robustness of the policy and highlight the potential of our approach to addressing complex control problems in robotics.

5.2 Background

5.2.1 The Optimal Control Problem

We consider a discrete-time dynamic system with d_1-dimensional state space and d_2-dimensional action space. For ease of presentation, we assume the dynamic system to be deterministic, however, our approach can also handle a stochastic model (see Section 5.3.2).

We denote the state at time t by $\boldsymbol{s}_t = (s_t^1, \ldots, s_t^{d_1})$, and action by $\boldsymbol{a}_t = (a_t^1, \ldots, a_t^{d_2})$. The dynamics of the system is given by:

$$\begin{aligned} \boldsymbol{s}_{t+1} &= f(\boldsymbol{s}_t, \boldsymbol{a}_t), \\ \text{s.t.} \quad s_t^i &\in \Omega_{s^i},\ \forall i \in \{1, \ldots, d_1\}, \\ a_t^j &\in \Omega_{a^j},\ \forall j \in \{1, \ldots, d_2\}, \end{aligned} \tag{5.1}$$

where the domain of each state Ω_{s^i} and action Ω_{a^j} can be a bounded real interval or a

discrete set. Let $\Omega_{\boldsymbol{s}}$ denote the state space and $\Omega_{\boldsymbol{a}}$ denote the action space.

Let $r(\boldsymbol{s}, \boldsymbol{a})$ represent the reward function and Δt be the time step for the discrete time control. We define $R(\boldsymbol{s}, \boldsymbol{a}) = r(\boldsymbol{s}, \boldsymbol{a})\Delta t$. Our goal is to obtain an optimal policy π^* for the following infinite horizon optimal control problem for any given initial state in the state space $\boldsymbol{s}_0 \in \Omega_{\boldsymbol{s}}$:

$$\begin{aligned} \pi^* = &\arg\max_{\pi} \sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)),\ \forall \boldsymbol{s}_0 \\ \text{where} \quad &\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)) \end{aligned} \tag{5.2}$$

where γ is the discount factor $0 \leq \gamma < 1$.

We do not make any assumption on the structure or differentiability of the dynamics f and the reward function r. For example, a black box deterministic simulator that returns the next state and the reward for the state-action pair satisfies our requirement. However, for a fast implementation of our algorithm described in Section 5.3.1, the simulator should ideally process a batch of state-action pairs for parallel implementation.

5.2.2 Dynamic Programming

The state-value function V^π corresponding to a policy π, with discount factor γ, is defined as follows:

$$\begin{aligned} V^\pi(\boldsymbol{s}_0) = &\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)),\ \forall \boldsymbol{s}_0, \\ \text{where} \quad &\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)),\ \forall t. \end{aligned} \tag{5.3}$$

Given a state-value function $V : \Omega_{\boldsymbol{s}} \to \mathbb{R}$, a policy π and the discount factor γ, the Bellman operator $\mathcal{B}^\pi$ is a functional that is defined as $\mathcal{B}^\pi V(\boldsymbol{s}) = R(\boldsymbol{s}, \pi(\boldsymbol{s})) + \gamma V(f(\boldsymbol{s}, \pi(\boldsymbol{s}))), \forall \boldsymbol{s} \in \Omega_{\boldsymbol{s}}$ where $\mathcal{B}^\pi : V \to V$.

We define the advantage function A_V corresponding to the value function V as follows:

$$A_V(\boldsymbol{s}, \boldsymbol{a}) = R(\boldsymbol{s}, \boldsymbol{a}) + \gamma(V(f(\boldsymbol{s}, \boldsymbol{a})) - V(\boldsymbol{s})),\ \forall (\boldsymbol{s}, \boldsymbol{a}) \in \Omega_{\boldsymbol{s}} \times \Omega_{\boldsymbol{a}}. \tag{5.4}$$

5.2.3 Challenges in Approximate Dynamic Programming

Algorithm 6 describes the value iteration (VI) algorithm [85], a popular DP algorithm.
One of the challenges in implementing the VI algorithm and other similar DP algorithms including the Policy Iteration (PI) algorithm [85] in practice is the curse of dimensionality in representing the value function when the involved state space is either high-dimensional or includes continuous states. ADP addresses this challenge by using function approximation techniques.
In addition, retrieving the policy π^k from the advantage function is difficult if it is nonconvex, if there are bounds on the actions, if the action space is large, or if the action space is hybrid. An inefficient optimization technique for policy retrieval increases the overall time of the algorithm, as it must be repeated for each state in every iteration, and it results in a sub-optimal policy. The lack of such policy retrieval techniques is a bottleneck in the development of ADP algorithms for a general nonlinear system, including hybrid systems.

Algorithm 6 VI Algorithm

Input: Initial value function V^0, convergence threshold ϵ
Output: Optimal policy π^*
Set $k = 0$
repeat
  $\pi^k(\boldsymbol{s}) := \arg\max_{\boldsymbol{a}} A_{V^k}(\boldsymbol{s}, \boldsymbol{a})$
  $V^{k+1} = \mathcal{B}^{\pi_k} V^k$
  if $\|V^{k+1} - V^k\|_\infty < \epsilon$ **then**
    break
  end if
  $k \leftarrow k + 1$
until convergence
$V^* = V^k$
$\pi^*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} A_{V^*}(\boldsymbol{s}, \boldsymbol{a})$

5.3 Generalized Policy Iteration using Tensor Train

We overcome the challenges mentioned in the ADP algorithms using TT as a function approximator. First, we propose to model the advantage function explicitly in TT format. As described in Chapter 2, the spline-based interpolation techniques (see Section 2.5.1) allow TT representation to handle a mix of continuous and discrete variables (in this case states and actions).

In addition to the availability of algorithms like TT-Cross for finding function approximation and the accompanying algebraic tools, an advantage of using TT decomposition for approximating functions in ADP is its ability to efficiently find optima over a mix of continuous and discrete variables. This was introduced as Tensor Train for Global

Optimization (TTGO) in Chapter 4. In this chapter, we exploit this framework and use the deterministic version of TTGO described in Section 4.3 for policy retrieval from the advantage function modeled in TT format.

Recall that TTGO provides approximate optima of a function in TT format. The solution obtained from such a procedure can be refined further using local optimization techniques such as Newton-type optimization for continuous variables. But, in practice, the refinement procedure is often not required. In this chapter, we identify and exploit TTGO's ability to handle a mix of continuous and discrete variables. In addition, we perform optimization in the batch form: we propose to model the advantage function $A(s, a)$ in ADP in TT format, and adapt TTGO to obtain the optimal actions a corresponding to a batch of states s (i.e. parallel computation of $\arg\max_a A(s, a)$) in an efficient manner.

5.3.1 Description of the Algorithm

By combining the conceptual ideas proposed so far, Algorithm 7 presents the TTPI algorithm, which addresses the previously mentioned challenges in ADP using TT as the function approximator for state-value and advantage functions and TTGO for policy retrieval. A pictorial description of the algorithm is presented in Figure 5.1.

In the TTPI algorithm, the value update step involves computation of the policy $\pi^k(s)$ (i.e., $\arg\max_a A_{V^k}(\boldsymbol{s}, \boldsymbol{a})$) numerous times across several iterations. To compute V_j^k in TT-format, the function $\mathcal{B}^{\pi_k} V_{j-1}^k$ is queried iteratively using TT-Cross($\mathcal{B}^{\pi_k} V_{j-1}^k,\ r_{\max},\ \epsilon$), with batches of states (usually ranging from 10,000 to 100,000 in practice). This requires computing the policy π^k for each of these states in batch form. We use TT-round to compress the value functions in TT format at the end of every policy evaluation (i.e., after updating the value function for the current policy). We use cubic spline-based interpolation for continuous variables which reduces the number of discretization points required by TT-cross to construct the TT model.

To resolve the bottleneck in policy retrieval, we propose to compute the advantage function A_{V^k} in TT format using TT-Cross. This is efficient as the calculation only requires evaluating V^k and $R(\boldsymbol{s}, \boldsymbol{a})$, which are cheap to compute. As a result, $\pi^k(\boldsymbol{s})$ over batches of states can be obtained quickly. Most importantly, this allows us to handle hybrid action space. The computational cost involved in retrieving a solution is $\mathcal{O}(nNd_2\, r_{\max}^2)$ which is linear in the number of discretizations (n) of an action variable, the number of candidates used in TTGO (N) and the dimension of action space (d_2).

A PyTorch-based GPU-accelerated implementation of these algorithm can be found

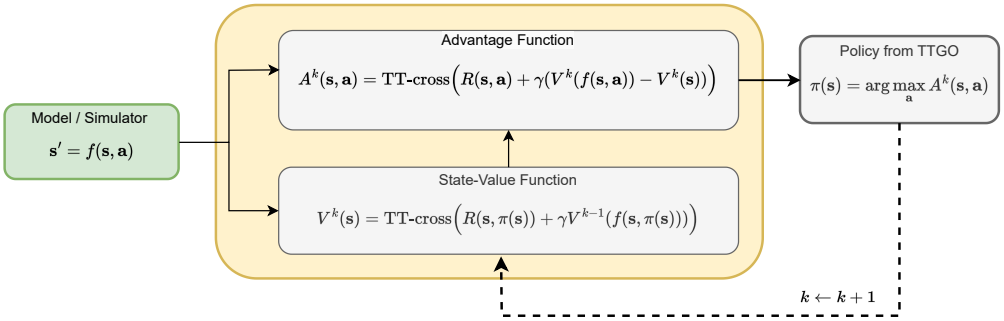


Figure 5.1: This figure shows an iteration of the TTPI algorithm described in Algorithm 7. At each iteration, we compute state-value function in TT format using TT-cross using the previous estimate of the state-value function and the policy, where the policy is retrieved from TTGO with the advantage function. The advantage function is also computed in TT format using TT-cross using the state-value function.

at https://sites.google.com/view/ttpi4control/home. The software is based on [79], a library for working with tensor networks. The computational cost of the algorithm increases linearly with the number of dimensions in both state and action spaces and grows quadratically with the rank of the functions represented in TT format, thanks to the properties of TT-Cross and TT-representation.

5.3.2 Adaptation to Stochastic Systems

In this section, we show how our approach can be extended to consider stochastic system dynamics. Instead of relying on deterministic system dynamics of the form $\boldsymbol{s}' = f(\boldsymbol{s}, \boldsymbol{a})$, we consider the transition probability $P(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a})$ and the reward function $R(\boldsymbol{s}, \boldsymbol{a})$ in TT format. The transition probability $P(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a})$ can be obtained by fitting a density model to data collected from the robot. To achieve this, we can employ the TT format for density modeling as suggested by [25] and [26]. Alternatively, if the function P is available in a different format such as NN, we can utilize TT-Cross. By leveraging the algebraic tools provided in TT format, we can normalize P such that $\sum_{\boldsymbol{s}'} P(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a}) = 1$ (or integrate if $\boldsymbol{s}'$ is continuous). The following outlines the procedure to update the

value function and policy under this approach:

$$\begin{aligned} V^k =& \text{TT-Cross}(U^k,\ \hat{\Omega}_{\boldsymbol{s}},\ r_{\max},\ \epsilon), \\ U^k(\boldsymbol{s}) =& R(\boldsymbol{s}, \pi^k(\boldsymbol{s})) + \gamma W^k(\boldsymbol{s}, \pi^k(\boldsymbol{s})), \\ W^k(\boldsymbol{s}, \boldsymbol{a}) =& \sum_{\boldsymbol{s}'} P(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a}) V^k(\boldsymbol{s}'), \\ A_{V^k}(\boldsymbol{s}, \boldsymbol{a}) =& R(\boldsymbol{s}, \boldsymbol{a}) + \gamma(W^k(\boldsymbol{s}, \boldsymbol{a}) - V^k(\boldsymbol{s})), \\ \pi^k(\boldsymbol{s}) =& \arg\max_{\boldsymbol{a}} A_{V^k}(\boldsymbol{s}, \boldsymbol{a}). \end{aligned} \tag{5.5}$$

In the above algorithm, as P and V^k are both in TT format, we can obtain W^k efficiently by using algebraic operation over TT format (namely, element-wise product and contraction operations over s'). Then A_{V^k} can be readily computed in TT-format using addition operations over the TT tensors as R, W^k, and V^k are also in TT format. We only need TT-cross to find V^k. Hence the algorithm would be very efficient if P is known in TT format. But, in this chapter, we will only consider systems with deterministic dynamics.

5.4 Experiments

In our experiments, we utilized an NVIDIA GeForce RTX 3090 GPU with 24GB of memory. For the applications considered, we discretized each continuous variable with 100 points using uniform discretization. To approximate the value and advantage functions in TT format using TT-Cross, an accuracy of $\epsilon = 10^{-3}$ proved sufficient. We set $r_{\max}$ to a large value of 100. The discount factor was chosen in the range of 0.99 to 0.9999, depending on the time step Δt which ranged from 0.01 to 0.001. The rank of the value function in the applications considered ranged between 5 to 50, and the rank of the advantage function was roughly twice that of the value function.

5.4.1 Simulation Experiments

Baseline: To the best of our knowledge, there are no established approaches for OC based on ADP algorithms that can handle hybrid actions. To evaluate our algorithm performance, we compared it against Deep RL techniques for hybrid action spaces such as HyAR, HPPO and PDQN [87, 88, 89]. The HyAR algorithm has shown superiority over other Deep RL techniques for high-dimensional hybrid action spaces. It is important to acknowledge that TTPI assumes access to the system dynamics (e.g., a simulator) and the reward function, whereas Deep RL techniques, in theory, are agnostic to the system

Algorithm 7 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
  $r(\boldsymbol{s}, \boldsymbol{a})$: Reward function
  Δt: Time Discretization
  $f(\boldsymbol{s}, \boldsymbol{a})$: Forward simulation
  $\hat{\Omega}_{\boldsymbol{s}}$: Discretization of state space
  $\hat{\Omega}$: Discretization of state-action space $(\hat{\Omega}_{\boldsymbol{s}} \times \hat{\Omega}_{\boldsymbol{a}})$
  N: Number of candidate samples for optima used in TTGO.
Output: Policy π^*
Initialize:
  Initialize in TT-format: $V^0 = 0$
  Initialize Advantage model: $A_{V^0} = \text{TT-Cross}(R(\boldsymbol{s}, \boldsymbol{a}),\ \hat{\Omega},\ r_{\max},\ \epsilon)$
  (alternatively, initialize arbitrarily)
  Set $k = 0$
while $\delta \leq \delta_{\max}$ **do**
  $k \leftarrow k + 1$
  $\pi^k(\boldsymbol{s}) := \underset{a}{\text{argmax}}\, A_{V^{k-1}}(\boldsymbol{s}, \boldsymbol{a})$ (Use TTGO)
  $V_0^k = V^{k-1}$
  for $j \leftarrow 1$ to n_v **do**
    $V_j^k(\boldsymbol{s}) = \text{TT-Cross}(\mathcal{B}^{\pi^k} V_{j-1}^k,\ \hat{\Omega}_{\boldsymbol{s}},\ r_{\max},\ \epsilon)$
  end for
  $V^k = \text{TT-round}(V_{n_v}^k,\ \epsilon)$
  $A^k(\boldsymbol{s}, \boldsymbol{a}) = R(\boldsymbol{s}, \boldsymbol{a}) + \gamma(V^k(f(\boldsymbol{s}, \boldsymbol{a})) - V^k(\boldsymbol{s}))$
  $A_{V^k} = \text{TT-Cross}(A^k,\ \hat{\Omega},\ r_{\max},\ \epsilon)$
  $\delta = \frac{\|V^k - V^{k-1}\|_2}{\|V^{k-1}\|2}$
end while
Set $V^* = V^k$
$\pi^*(\boldsymbol{s}) = \underset{\boldsymbol{a}}{\text{argmax}} A_{V^*}(\boldsymbol{s}, \boldsymbol{a})$

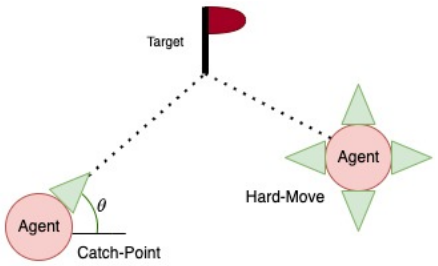


Figure 5.2: The tasks considered in this study involve controlling an agent to reach a target point in a 2D space. In the first task, called "Catch-Point", the agent has control over its heading direction (continuous) and the option to either stop or move toward the target (binary variable). In the second task, known as "Hard-Move", the agent is equipped with m actuators, and it can decide to activate or deactivate each actuator (m binary actions) and specify acceleration along each actuator (m continuous variables).

model and implicitly address a more challenging problem than TTPI. However, many of these methods are data-inefficient and, like TTPI, assume access to a simulator.

Evaluation: We evaluated our algorithm on two benchmark problems involving systems with hybrid action spaces: the Catch-Point (CP) Problem and the Hard-Move (HM) problem, as proposed by [87]. The Catch-Point Problem has four states and an action space with one discrete and one continuous action. The Hard-Move problem has m actuators, resulting in a total of $2m$ action variables (i.e., $d_2 = 2m$), with m binary and m continuous variables. Thus, this problem allows testing the scalability for high-dimensional action spaces by increasing m.

The results, as presented in the table, provide strong evidence of TTPI's superior performance compared to the baseline method. TTPI demonstrates faster training times and generates highly performant policies. In contrast, the baseline method struggles with generalization and produces lower-quality solutions, particularly for the Hard-Move problem with a number of actuation $d_2 > 24$. This is attributed to TT-Cross accurately modeling the value functions by leveraging the system model and reward function, in a fast manner and efficient policy retrieval using TTGO.

5.4.2 Additional Simulation Experiments

In addition to the benchmark problems on hybrid actions provided in the main section, we performed further experiments to evaluate the performance of our approach on some benchmark optimal control problems involving continuous states including Point-mass control (double integrator) with obstacles, Cart-Pole Swing-up, and Box-pivoting. The video and the supplementary material provided on the website associated with the chapter show the performance of the policy obtained by TTPI on these tasks.

	d_1	d_2	HPPO			PDQN			HyAR			TTPI		
			T	μ	S	T	μ	S	T	μ	S	T	μ	S
CP	4	2	0.5h	0.13 ±0.01	86% ±6%	1.9h	0.16 ±0.05	84% ±6%	4h	0.15 ±0.02	92% ±4%	30s	1	100%
HM(8)	4	16	1.4h	0.15 ±0.01	8% ±2%	2.1h	0.19 ±0.03	8% ±3%	8h	0.92 ±0.01	100%	850s	0.93 ±0.01	100%
HM(12)	4	24	NA	NA	NA	NA	NA	NA	10h	0.92 ±0.05	12% ±5%	946s	0.92 ±0.01	100%
HM(16)	4	32	NA	NA	NA	NA	NA	NA	10h	NA	0%	1743s	0.92 ±0.02	100%

Table 5.1: We used the success rate (S) for reaching the target position as one of the metrics. We note that the primary objective of both approaches, in the problems considered here, is to reach the goal in the shortest possible time or path. So as a second metric (μ), we calculate the square of the ratio between the length of the trajectory generated by each policy and the length of the shortest path for HM task. For CP task, μ is the inverse of the number of catch motions till reaching the goal. The table includes the training time (T) required to obtain the policy used for evaluation. The number of states is d_1 and the number of actions is d_2.

5.4.3 Real Robot Experiments

We demonstrate the effectiveness of our proposed method for hybrid system control on a planar pushing task with a face-switching mechanism [90] and involves discrete states and actions. The objective is to push a block with freedom in switching both the contact modes and faces. It is modeled using 6 states and 4 actions. The action includes a discrete variable representing the index of next contact face. Its underactuated and hybrid nature, coupled with multiple discrete contact modes, makes it difficult to design effective control strategies, and it has been a test-bed problem for the control of hybrid systems. Previous approaches, such as mixed integer programming [91] and hybrid Differential Dynamic Programming [92], have struggled with the high computational cost required for solving the problem, which requires robust algorithms that can handle the complexity of hybrid systems with both continuous and discrete variables. Note that typically such a non-prehensile manipulation problem is formulated differently as continuous control [93] due to a lack of methodologies to handle hybrid actions and is not representative of hybrid control in robotics applications.

The objective of the task is to push a block with the option of switching the face of the block to be pushed, as well as the contact mode used for pushing. We demonstrate that the proposed algorithms can achieve this task robustly in both simulation and the real world. A video of the experiments is provided in the supplementary material.

The state of the system is denoted as $[\boldsymbol{q}_s^\top \ \boldsymbol{q}_p^\top \ c_c]^\top$, where $\boldsymbol{q}_s = [s_x \ s_y \ s_\theta]^\top$ is the position and orientation of the block, $\boldsymbol{q}_p = [p_x \ p_y]^\top$ is the position of end-effector, and

$c_c \in \{0, 1, 2, 3\}$ is the current contact face. The action is expressed as $[\boldsymbol{v}^\top \; c_n]^\top$, where $\boldsymbol{v} = [v_n \; v_t]^\top$ is the velocity of the end-effector, and $c_n \in \{0, 1, 2, 3\}$ is the next contact face. The system, therefore, has $d_1 = 6$ states and $d_2 = 3$ control variables in total, including both continuous and discrete variables.

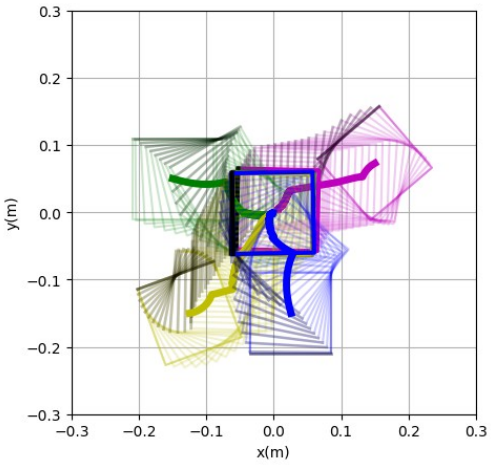


Figure 5.3: Simulation of the motion of the block under a policy from four different initial states. The colored trajectories represent the motion of the block to the target ($\boldsymbol{q}_s = [0\ 0\ 0]^\top$), by means of contact mode and face switching.

We first trained the control policy in simulation based on the predefined motion equation. The continuous variables in state and action spaces are discretized into 100 bins. 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 capability of the policy, we randomly selected 1000 initialization points in the domain. A success rate of 100% was obtained in under 10 minutes of training. Fig. 5.3 shows the simulation results. The reward function is defined as:

$$R(\boldsymbol{s}, \boldsymbol{a}) = -2\|\boldsymbol{q}_s\| - (1 - \delta(c_c - c_n)), \tag{5.6}$$

where $\boldsymbol{q_s}$ represent the block pose, $\delta(c_c - c_n)$ will return 1 if $c_c = c_n$ (no face switching), otherwise, 0. 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. 5.4), 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 updated at 100 HZ, with a low-level Cartesian velocity controller (1000 HZ) actuating the robot.

Table 5.2: Performance of three real-world experiments

Experiments	$\boldsymbol{x}_{\text{err}}$/cm	$\boldsymbol{y}_{\text{err}}$/cm	$\boldsymbol{\theta}_{\text{err}}$/rad
Reaching	-0.83	1.07	-0.06
Reaching with additional weight	2.89	-1.04	-0.01
Reaching with external disturbance	-4.78	-4.10	-0.04

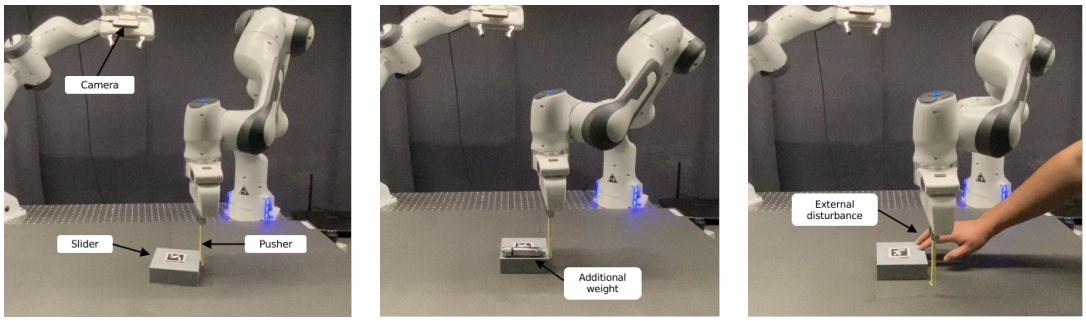
Three experiments were conducted to assess the robustness of our policy: **a) Reaching task:** The robot pushes the slider from $\boldsymbol{q}_{s_0} = [0.05\text{m } 0.16\text{m } 0]^\top$ to the origin (Fig. 5.4a); **b) Reaching with additional weight:** The robot pushes the block from the same initialization as before, but with an additional weight, 3 times heavier than the block (Fig. 5.4b); **c) Reaching with external disturbance:** The same initialization like before, but with a significant external disturbance of $\boldsymbol{q}_{\text{dist}} = [0.1\text{m } 0.03\text{m } 90°]^\top$ exerted by a human (Fig. 5.4c).

The results of these experiments are shown in Table 5.2. The results show 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 remain less than 4° and position errors remain less than 5cm even under significant disturbance. Experiment 3 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 ability of our method to handle both continuous and discrete variables in hybrid systems.

Our algorithm achieves robust performance 100% success rate (reaching the goal) in both simulation and real-world experiments for this task. The experiments demonstrate successful reaching of the target position and orientation, even in the presence of additional weight and external disturbances, as shown in Fig. 5.4. This indicates the potential of TTPI for solving complex hybrid system control.

5.5 Related Work

In recent years, research has surged in the domain of optimal control for hybrid systems which involve a mix of discrete and continuous state and action variables. Classical techniques, like Mixed-Integer Programming (MIP) [94], unify continuous and discrete variables in a single optimization problem. Abstraction and reachability analysis methods [95] help adapt hybrid systems for traditional solvers. However, they often involve high computational complexity and are not suitable for real-time decision-making. This motivates the development of Approximate Dynamic Programming (ADP) techniques, which involve approximating value functions to alleviate computational burdens and



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

Figure 5.4: Pusher-slider system where the robot pushes an object by contact switching. Three experiments were performed: The block being pushed towards the target as modeled (a), with additional weight on the block leading to nonuniform friction distribution (b), and with external disturbance (c). The policy obtained from TTPI was robust to handle these scenarios.

handle high-dimensional settings.

The use of low-rank tensor approximation techniques for solving ADP was previously proposed in [6], [7], and [96]. In [7] and [96], they proposed a TT-based value iteration algorithm, where the TT was used to approximate the value function, and the policy was retrieved using Newton-type optimization technique based on the value function. This limits the application and speed of the algorithm, as the policy retrieval procedure demands the system dynamics and the reward function to be differentiable and the action space to be continuous.

Some of the NN-based ADP for continuous state and action space have been proposed in fitted-Q iteration [97] and fitted-value iteration [98]. However, these methods have demonstrated their applicability only to systems with low dimensional systems and they have not been successful in handling hybrid action space. The NN-based ADP methods have been overshadowed by the rise of Deep RL as they have demonstrated scalability to problems with high dimensional state and action space. To overcome the issues in Deep RL for handling hybrid actions several improvements were proposed by [99, 100, 88] and [87].

5.6 Limitation and Future Work

TTPI approximates the state-value and advantage function over the entire state-action space, resulting in a highly generalizable policy. However, computational complexity and storage issues may arise when these functions are not low-rank in the TT representation. For instance, systems involving highly agile movements like the acrobat (double pendulum swing-up) can lead to high-rank in the TT representation. Nonetheless, decreasing the time step Δt has been observed to reduce the rank of these functions which may enable the approach to handle such systems at the expense of longer training time.

TTPI may be well-suited for commonly encountered systems with discontinuities and hybrid characteristics, such as manipulation and legged robotics. However, a drawback is its reliance on highly parallelized simulators. Hand-coding the system's dynamics and reward function, as demonstrated in this work, may not be practical for more complex dynamics involving contact. The availability of recently introduced GPU-based simulators like NVIDIA Isaac Gym presents an opportunity to test the algorithm on more intricate applications.

Concerning scalability, although existing Deep RL techniques struggle to produce optimal policies and handle hybrid action space, they can cope well with high-dimensional state space (e.g., images as states). On the other hand, TTPI can handle high-dimensional hybrid actions and perform better compared to existing ADP methods, it may not be suitable for very high-dimensional state spaces. However, we could potentially enable our method to handle such high-dimensional problems by formulating our approach as an RL problem instead of ADP or OC. In such cases, instead of TT-Cross, gradient-based methodologies (see Section 2.10) could be used. This is described in more detail in Section 5.6.1. Alternatively, we could use TTPI for model-based RL. In this case, a dynamic model and a reward model of the system could be learned in a latent space (lower dimension than the original observation space such as images) and then we can apply TTPI on the latent space dynamic model.

5.6.1 Neural Tensor Train for Policy Learning

In this section, we describe how techniques proposed in this chapter along with the TTGO proposed in Chapter 4 can enable policy learning in a data-driven setting. In the RL setting, more specifically Q-learning, we have the data $(\boldsymbol{s}, \boldsymbol{a}, r)$ (i.e., state, action, reward) in addition to the access to a simulator or real robot and the objective is to learn a Q-model. For imitation learning, we have only access to $(\boldsymbol{s}, \boldsymbol{a})$ (in some applications $(\boldsymbol{s}, \boldsymbol{a}, r)$) without a simulator.

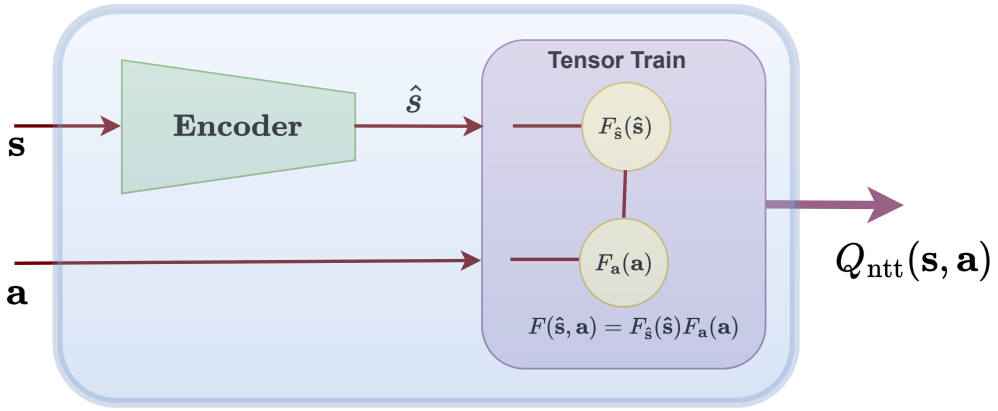


Figure 5.5: This figure shows the architecture of the Neural Tensor Train defined in (5.7). The state $\mathbf{s}$ is passed through an encoder (e.g., a neural network) to obtain a lower dimensional feature $\hat{\boldsymbol{s}}$. Then, the feature concatenated with action $\boldsymbol{a}$, i.e., $(\hat{\boldsymbol{s}}, \boldsymbol{a})$, is passed to the TT basis model $F(\hat{\boldsymbol{s}}, \boldsymbol{a})$. This architecture facilitates retrieval policy $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} Q_{\text{ntt}}(\boldsymbol{s}, \boldsymbol{a})$ through TTGO. For density modeling applications a decoder needs to be included as described in (5.10).

We propose to use the methodology described in Section 2.10 for function approximation (i.e., a linear combination of basis functions with weights represented in TT format as the model), called TT-basis model. Although it has been demonstrated to work for handling high-dimensional problems such as the classification of images in [23] and [24] and density estimation in [25], this framework does not still perform well for very high-dimensional problems when compared to Neural Networks and needs adaptation. We aim to overcome this issue for policy learning by using the observation that the action space is typically low-dimensional in robotics (approximately equal to the number of joints in a robot) and it is the state space (observation space) which is typically high-dimensional (e.g., images) in these frameworks. To overcome the dimensionality issues with state space, we propose to rely on a Neural Network (NN) to encode the state into a latent variable $\hat{\boldsymbol{s}} \in \mathbb{R}^{\hat{d}_1}$ with $\hat{d}_1 \leq d_1$. We concatenate the corresponding latent variable with the action $(\hat{\boldsymbol{s}}, \boldsymbol{a})$ and treat this as input to the TT-basis model. This framework still allows us to use TTGO for policy retrieval as described below for applications such as Q-learning and imitation learning. Thus, we combine the benefits of NN and TT, by proposing a new architecture called Neural Tensor Train (NeuralTT) for policy learning for data-driven settings in robotics.

Supervised Learning and Deep Q-Learning

Suppose we have access to data $\mathcal{D} = \{(\boldsymbol{s}, \boldsymbol{a}, q)^i\}_{i=1}^{i=M}$ where $q \in \mathbb{R}$ is a score defining how good is the state-action pair. For supervised learning, the input to the function approximation model is $(\boldsymbol{s}, \boldsymbol{a})$ and the target output is the score q. We define below the Neural Tensor Train model for such a setup:

$$\begin{aligned} \hat{\boldsymbol{s}} &= \text{Encoder}(\boldsymbol{s};\ \boldsymbol{\theta}),\ \hat{\boldsymbol{s}} \in \mathbb{R}^{\hat{d}_1}, \\ F((\hat{\boldsymbol{s}}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}) &= \Big\langle \boldsymbol{\mathcal{W}},\ \boldsymbol{\Phi}((\hat{\boldsymbol{s}}, \boldsymbol{a})) \Big\rangle, \\ Q_{\text{ntt}}((\boldsymbol{s}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}, \boldsymbol{\theta}) &= F((\hat{\boldsymbol{s}}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}), \end{aligned} \tag{5.7}$$

where $\boldsymbol{\Phi}((\hat{\boldsymbol{s}}, \boldsymbol{a}))$ is basis function and $\boldsymbol{\mathcal{W}}$ is the weight tensor in TT format as described in (2.20) in Section 2.10, and $\boldsymbol{\theta}$ is the parameters of encoder function which is a Neural Network.

Given this architecture and the dataset, we can learn the parameters $\boldsymbol{\mathcal{W}}$ and $\boldsymbol{\theta}$ of the parameterized function Q_{ntt} through stochastic batch gradient-descent and the familiar back-propagation algorithm. The choice of loss function includes the mean squared error with appropriate regularization on the learning parameters:

$$\text{Loss}(\boldsymbol{\mathcal{W}}, \boldsymbol{\theta}) = \sum_{(\boldsymbol{s}, \boldsymbol{a}, q) \in \mathcal{D}} (q - Q_{\text{ntt}}((\boldsymbol{s}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}, \boldsymbol{\theta}))^2 + \lambda_1 \|\boldsymbol{\mathcal{W}}\| + \lambda_2 \|\boldsymbol{\theta}\|. \tag{5.8}$$

Given the learned model Q_{ntt}, we can retrieve a policy using TTGO with:

$$\begin{aligned} \pi(\boldsymbol{s}) &= \arg\max_{\boldsymbol{a}} Q_{\text{ntt}}(\boldsymbol{s}, \boldsymbol{a}), \\ \text{i.e., } \pi(\boldsymbol{s}) &= \arg\max_{\boldsymbol{a}} F((\hat{\boldsymbol{s}}, \boldsymbol{a})),\ \text{where } \hat{\boldsymbol{s}} = \text{Encoder}(\boldsymbol{s};\ \boldsymbol{\theta}), \\ \text{i.e., } \pi(\boldsymbol{s}) &= \arg\max_{\boldsymbol{a}} \boldsymbol{\mathcal{F}}((\hat{\boldsymbol{s}}, \boldsymbol{a})), \end{aligned} \tag{5.9}$$

where we obtain the TT model $\boldsymbol{\mathcal{F}}$ of the TT-basis model $F((\hat{\boldsymbol{s}}, \boldsymbol{a}))$ for employing TTGO directly by discretizing and using the techniques described in Section 2.5.2.

We can also apply the above framework to Deep Q-learning (a.k.a. DQN) with Q_{ntt} as the approximation of the Q-function. In this case, the score q is the target value for the Q-function approximation in the Q-iteration algorithm. The advantage is that the policy can be retrieved from the TTGO algorithm which is otherwise a well-known bottleneck in DQN.

Imitation Learning

For this application, we assume the data to be a collection of state-action pair $\mathcal{D} = \{(\boldsymbol{s}, \boldsymbol{a})^i\}_{i=1}^{i=M}$ obtained from an expert. We frame this as a density modeling problem with a density model defined as:

$$\begin{aligned} \hat{\boldsymbol{s}} &= \text{Encoder}(\boldsymbol{s};\ \boldsymbol{\theta}),\ \hat{\boldsymbol{s}} \in \mathbb{R}^{\hat{d}_1}, \\ \tilde{\boldsymbol{s}} &= \text{Decoder}(\hat{\boldsymbol{s}};\ \hat{\boldsymbol{\theta}}),\ \tilde{\boldsymbol{s}} \in \mathbb{R}^{d_1}, \\ F((\hat{\boldsymbol{s}}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}) &= \Big\langle \boldsymbol{\mathcal{W}},\ \boldsymbol{\Phi}((\hat{\boldsymbol{s}}, \boldsymbol{a})) \Big\rangle, \\ Q_{\text{ntt}}((\boldsymbol{s}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}, \boldsymbol{\theta}) &= \frac{F((\hat{\boldsymbol{s}}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}})^2}{Z}, \end{aligned} \tag{5.10}$$

where $\boldsymbol{\Phi}((\hat{\boldsymbol{s}}, \boldsymbol{a}))$ is some basis function and $\boldsymbol{\mathcal{W}}$ is the weight tensor in TT format and Z is the normalization constant as described in Section 2.10 and (2.21). $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ are the parameters of the encoder and decoder function which are Neural Networks.

Given this architecture and the dataset $\mathcal{D}$, we can learn the parameters $\boldsymbol{\mathcal{W}}$, $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ of the parameterized function Q_{ntt} through stochastic batch gradient-descent and the familiar back-propagation algorithm. The choice of loss function for density estimation is described below, which includes a loss term for encoder-decoder architecture along with a negative log-likelihood for density estimation:

$$\text{Loss}(\boldsymbol{\mathcal{W}}, \boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \sum_{(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}} \Big(\|\boldsymbol{s} - \tilde{\boldsymbol{s}}\|^2 - \log(Q_{\text{ntt}}((\boldsymbol{s}, \boldsymbol{a});\ \boldsymbol{\mathcal{W}}, \boldsymbol{\theta})) \Big) + \lambda_1 \|\boldsymbol{\mathcal{W}}\| + \lambda_2 \|\boldsymbol{\theta}\| + \lambda_3 \|\hat{\boldsymbol{\theta}}\|. \tag{5.11}$$

Once the model is trained, we can retrieve the policy as in (5.9).

5.7 Conclusion

In this chapter, we presented TTPI, an ADP algorithm that can handle nonlinear systems without any assumption on the structure of dynamics. Through simulation experiments, we showed that the algorithm is superior to state-of-the-art algorithms for dealing with hybrid action spaces in terms of training time, generalization, and the quality of the policy. We demonstrated the robustness of the policy of TTPI through real-world experiments. The results demonstrate that our approach is promising for policy learning.

6 Conclusion

In many applications in robotics, the choice of function approximation plays a crucial role in determining the feasibility and the computation efficiency of the algorithms. This thesis demonstrates the significance of Tensor Networks (TNs) for function approximation in robotics. In particular, we have demonstrated how the Tensor Train (TT), a type of TN, can greatly benefit robotics by solving several well-known problems in robotics that were previously considered intractable, including inverse kinematics with obstacle avoidance, motion planning, ergodic exploration, and optimal control.

While classical function approximation techniques such as Gaussian Mixture Models (GMMs) and Gaussian Processes (GPs) possess desirable features for applications in robotics such as interpretability and facilitating calculus, probabilistic modeling, and optimization, they are limited in expressibility and scalability. On the other hand, modern deep learning frameworks based on Neural Networks (NNs) are expressive and scalable, however, they do not facilitate calculus, probabilistic modeling, and optimization, which limits their use in many applications in robotics. The TNs form a nice bridge by being expressive (low to medium scale), allowing calculus, probabilistic modeling, and optimization. This thesis identifies and exploits these properties of TT, a particular TN, and introduces a novel framework called TTGO for the optimization of functions in TT format, for solving a variety of problems in robotics that were not possible with other function approximation techniques.

As described in Chapter 2, TT is equipped with powerful techniques, including gradient-based and gradient-free approaches, to approximate target functions or the data in TT representation. Unlike other function approximation techniques such as NNs, they facilitate algebra and calculus, and hence operations such as multivariate integration, differentiations, and solving PDEs can be done efficiently. Furthermore, they are especially suited for probabilistic modeling and optimization (introduced in Chapter 4) which are

of key importance in robotics.

In Chapter 3, we proposed a solution to improve the scalability and deployability of a well-known ergodic exploration algorithm called Spectral Multiscale Coverage (SMC). The SMC algorithm suffers from a curse of dimensionality mainly due to the computation of Fourier-series coefficients and algebraic operations over several high-dimensional tensors required to compute the control commands. We overcome these issues by using TT representation. The use of TT allowed us to find Fourier series coefficients of high-dimensional functions efficiently. Furthermore, the TT representation of various tensors involved allowed us to perform algebraic operations over these tensors in an efficient manner. Thus, it enabled us to compute the control commands efficiently and implement ergodic exploration in a closed-loop manner, which is crucial for successfully handling tasks like peg-in-hole insertions.

The solution proposed in Chapter 3 applies to other robotics applications dealing with challenges involving calculus (specifically integration, differentiation, spectral analysis, and solving PDEs) and algebraic operations over tensors. For example, another popular exploration algorithm called HEDAC [54] suffers from a similar issue in scalability and deployability as it involves convolution over high-dimensional tensors and solving diffusion equations (a parabolic PDE). By using the TT representation we could potentially improve the computational efficiency of such algorithms.

In Chapter 4, we introduced a new technique called TTGO to optimize functions in TT format. This method is highly parallelizable and can handle a mix of continuous and discrete variables. This amplifies the importance of TT for function approximation, particularly in robotics where many problems are formulated as optimization problems. We demonstrated its effectiveness in solving inverse kinematics with obstacles and motion planning problems.

With TTGO in hand, in Chapter 5, we proposed an algorithm called TTPI for optimal control synthesis using Approximate Dynamic Programming (ADP). This algorithm overcomes the common issue of dealing with high-dimensional state and action space in ADP algorithms and can handle more complex systems including hybrid state and action space. Furthermore, it enables ADP algorithms to work with modern simulators without assumptions about the structure of the dynamic model. This suggests that TT representation is a promising tool for policy learning, impacting various algorithms used for learning policies, including Reinforcement Learning and Imitation Learning.

Although TT is applicable for a wide range of problems in robotics, it is not suitable for applications involving very high dimensionality and hence unsuitable for applications such as policy learning for systems with high-dimensional state space such as images. A novel

architecture called Neural Tensor Train (NeuralTT) as proposed in Section 5.6.1 has the potential to overcome this issue for policy learning by combining the benefits of modern deep learning frameworks and TT. The modern deep learning frameworks such as Neural Networks do not facilitate calculus or optimization and hence pose challenges in retrieving policies when the function being approximated (or learned) is a joint function of states and actions (e.g., Q-function or energy function). NeuralTT combines the scalability of deep learning frameworks with the benefits of TT for calculus and optimization. This makes it a promising tool for policy learning in various applications such as imitation learning (density modeling) and reinforcement learning.

The techniques proposed in this thesis using TT could be improved in terms of performance by using better discretization strategies (e.g., Chebyshev grid instead of uniform discretization), quantization procedures (e.g., use of quantized tensor train [16]) and improvement in cross-approximation (TT-Cross) algorithms. We have used spline-based interpolation for approximating functions of continuous variables, however, other methodologies such [101, 102, 48] which directly consider the interpolation scheme during the cross approximation could potentially improve the efficiency of approximating target functions in TT format.

In this thesis, we have targeted well-known problems such as ergodic exploration, inverse kinematics, motion planning, and policy learning. In addition to the several potential extensions of the techniques introduced in each chapter, it would be interesting to explore other areas in robotics with similar challenges including state estimation, system identification, Simultaneous Localization and Mapping (SLAM), and control using Koopman operators [103]. While this thesis has primarily utilized only TT, a specific TN architecture, it would be interesting to explore other more expressive and complex tensor networks, including tree tensor networks, MERA, and PEPS [1].

A Appendix to Chapter 3

A.1 Proof of Fourier Coefficients Decomposition

A one-dimensional integral

$$s = \int_{x=0}^{l} f(x)dx,$$

can be computed numerically with the Gaussian-Quadrature (GQ) rule as

$$s = \sum_{j=1}^{N} \alpha_j f(x_j),$$

where N represents the degree of approximation (specified by the user), x_j represents the discretization points, and α_j are the corresponding weights. For any polynomial function of degree less than $2N-1$, the above summation gives exact result without any error in the integration. For a given N, x_j and α_j can be computed and are readily available using software packages for scientific computing.

We need to evaluate the multidimensional integral (3.1) to find $\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}}$. Let x_j be the discretization points and α_j the corresponding weights, with $j \in \{1, \ldots, N\}$, obtained from the GQ rule for a given N. Let $\mathcal{J} = \{\boldsymbol{j} = (j_1, \ldots, j_d) : j_i \in \{1, \ldots, N\}\}$ be the index set. We can discretize the domain Ω at $\boldsymbol{x_j} = (x_{j_1}, \ldots, x_{j_d})$, with $\boldsymbol{j} \in \mathcal{J}$. Let $\boldsymbol{\mathcal{P}}$ be the tensor formed by evaluating the reference distribution $P(\boldsymbol{x})$ at the discretization points, i.e., $\boldsymbol{\mathcal{P}}_{\boldsymbol{j}} = P(\boldsymbol{x_j})$.

Then, we can evaluate (3.1) using GQ as

$$\begin{aligned}\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}} &= \sum_{\boldsymbol{j}\in\mathcal{J}} \alpha_{j_1}\cdots\alpha_{j_d} P(\boldsymbol{x_j})\boldsymbol{\Phi}_{\boldsymbol{k}}(\boldsymbol{x_j}),\\ &= \sum_{\boldsymbol{j}\in\mathcal{J}} \alpha_{j_1}\cdots\alpha_{j_d} P(\boldsymbol{x_j})\phi_{i_1}(x_{j_1})\cdots\phi_{i_d}(x_{j_d}),\ \forall \boldsymbol{k}\in\mathcal{K}. \end{aligned} \tag{A.1}$$

Discretizing $P(\boldsymbol{x})$ at the GQ points $\boldsymbol{x_j} = (x_{j_1},\ldots,x_{j_d})$ with $\boldsymbol{j}\in\mathcal{J}$, we can get the tensor $\boldsymbol{\mathcal{P}}$, with $\boldsymbol{\mathcal{P}}_{\boldsymbol{j}} = P(\boldsymbol{x_j})$.

Consider a TT-representation of $\boldsymbol{\mathcal{P}}$ given by the TT-cores $(\boldsymbol{\mathcal{P}}^1,\boldsymbol{\mathcal{P}}^2,\ldots,\boldsymbol{\mathcal{P}}^d)$, so that for $\boldsymbol{j} = (j_1,\ldots,j_d)\in\mathcal{J}$ we have

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{j}} = \boldsymbol{\mathcal{P}}^1_{:,j_1,:}\boldsymbol{\mathcal{P}}^2_{:,j_2,:}\cdots\boldsymbol{\mathcal{P}}^d_{:,j_d,:}.$$

Substituting the above expression in (A.1) yields

$$\begin{aligned}\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}} &= \sum_{\boldsymbol{j}\in\mathcal{J}} \alpha_{j_1}\cdots\alpha_{j_d}\boldsymbol{\mathcal{P}}^1_{:,j_1,:}\cdots\boldsymbol{\mathcal{P}}^d_{:,j_d,:}\phi_{k_1}(x_{j_1})\cdots\phi_{k_d}(x_{j_d})\\ &= \Big(\sum_{j_1=1}^{N}\alpha_{j_1}\boldsymbol{\mathcal{P}}^1_{:,j_1,:}\phi_{k_1}(x_{j_1})\Big)\cdots\Big(\sum_{j_d=1}^{N}\alpha_{j_d}\boldsymbol{\mathcal{P}}^d_{:,j_d,:}\phi_{k_d}(x_{j_d})\Big). \end{aligned} \tag{A.2}$$

Also, we know that the TT-decomposition of $\hat{\boldsymbol{\mathcal{W}}}$ takes the form

$$\hat{\boldsymbol{\mathcal{W}}}_{\boldsymbol{k}} = \hat{\boldsymbol{\mathcal{W}}}^1_{:,k_1,:}\cdots\hat{\boldsymbol{\mathcal{W}}}^d_{:,k_d,:},\ \forall\boldsymbol{k}\in\mathcal{K}. \tag{A.3}$$

Comparing (A.2) with (A.3), we obtain an expression for the TT-cores of the TT-decomposition of $\hat{\boldsymbol{\mathcal{W}}}$ as

$$\hat{\boldsymbol{\mathcal{W}}}^i_{:,k,:} = \sum_{j=1}^{N}\alpha_j\boldsymbol{\mathcal{P}}^i_{:,j,:}\phi_k(x_j),\ \begin{matrix}\forall k\in(1,\ldots,K),\\ \forall i\in(1,\ldots,d).\end{matrix} \tag{A.4}$$

B Appendices to Chapter 4

B.1 Evaluations on Benchmark Functions

We apply our framework to extended versions of some benchmark functions for numerical optimization techniques, i.e., Rosenbrock and Himmelblau functions. They are known to be notoriously difficult for gradient-based optimization techniques to find the global optima, which could be more than one. Some of the functions also have some parameters that can change the shape of the functions. We consider these parameters as the task parameters, hence making the problem even more challenging. The benchmark functions are considered as the cost functions and we transform them to obtain a suitable probability density function. In addition, we also include a sinusoidal function to show that TTGO can handle a cost function with an infinite number of global optima, and a mixture of Gaussians to test the performance of TTGO on a high-dimensional multimodal function.

Furthermore, we also evaluate the prioritized sampling approach proposed in this work. We show how the sampling parameter α influences the obtained solutions. When α is small, the generated samples cover a wide region around many different local optima. When α is close to one, the obtained samples are observed to be very close to the global optima. All the results can be observed in Figure B.1- B.6, where the samples from the TT distribution (without any refinement by another solver) are shown as blue dots. The contour plot corresponds to the cost function in Figure B.1-B.5 and the density function in Figure B.6, where the dark region is the region with low cost (i.e., high density).

In all of the test cases, we observe that the solutions proposed by TTGO are close to the actual optima and that the refinement using SLSQP quickly leads to global optima consistently. When there exist multiple solutions, we are also able to find them. Note that the task parameters influence the locations of the global optima, and TTGO can

adapt accordingly by conditioning the model on the given task parameters. In all of the following cases, we choose a uniform discretization of the domain with the number of discretization points $n_k = 500$ set for each variable to construct the TT model.

Except for the sinusoidal function, uniform sampling requires a large number of samples to reach the global optima. For the mixture of Gaussians case, it fails most of the time to get the global optima even after the refinement step. In contrast, we could consistently get the optima using TTGO with few samples. In fact, by using α close to 1, we could find the global optima with just one sample from the TT distribution.

Sinusoidal Function:

$$C(\boldsymbol{x}) = 1 - 0.5(1 + \sin\Big(4\pi||\boldsymbol{x}||/\sqrt{d}\Big))$$
$$P(\boldsymbol{x}) = 1 - C(\boldsymbol{x}),$$

where $\boldsymbol{x} = \boldsymbol{x}_2 = (y_1, y_2)$, , $\Omega_{\boldsymbol{x}_2} = [-2, 2]^2$ with no task parameters. For this function, finding the optima is not a difficult problem. However, as the cost function has uncountably many global optima (on the circles separated by one period of the sinusoidal function), we use it to test the approximation power of TT-model and check the multimodality in the TTGO samples. As we can see in Figure B.1 for $d = 2$, the samples from the TT model mainly come from the modes corresponding to the optima and the nearby region with cost values comparable to the optimal cost. At $\alpha = 0$, we can still observe a few samples in the white area (low density region), and as we increase α, the samples become more concentrated in the dark area, i.e., high-density region.

Rosenbrock Function:

$$C(a, b, y_1, \ldots, y_{d_2}) = \sum_{k=1}^{d_2/2} (y_{2k-1} - a)^2 + b(y_{2k-1} - y_{2k}^2)^2$$
$$P(\boldsymbol{x}) = \exp\Big(-C(\boldsymbol{x})^2\Big),$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$, $\boldsymbol{x}_1 = (a, b)$, $\boldsymbol{x}_2 = (y_1, \ldots, y_{d_2})$, $\Omega_{\boldsymbol{x}_1} = [-1.5, 1.5] \times [50, 150]$, $\Omega_{\boldsymbol{x}_2} = [-2, 2]^{d_2}$. The function is similar to a banana distribution which is quite difficult to approximate. The cost function $C(\boldsymbol{x})$ for a specified (a, b) has a unique global minima at $(a, a^2, \ldots, a, a^2)$. However, if we do not initialize the solution from the parabolic valley area (see Figure B.2), a gradient-based solver will have difficulty in converging to the

global optima quickly. We can see from Figure B.2 that TTGO samples are concentrated around this region, allowing most of them to reach the global optima after refinement. In fact, by increasing the α, the TTGO samples are already very close to the global optima (as shown in red).

Figure B.3 shows how the task parameters $\boldsymbol{x}_1 = (a, b)$ change the shape of the function with respect to $\boldsymbol{x}_2$ and consequently the location of the global optima. After the offline training, we condition our TT model on these task parameters and sample from the conditional distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = (a, b))$. We can see in this figure that TTGO can adapt to the new task parameters easily, as the samples are concentrated around the new global optima.

We also test TTGO performance on Rosenbrock functions for d_2 up to 30 and find that it can find the global optima consistently. We show in the figures the results for the 2D case, which are easier to visualize.

Himmelblau's function:

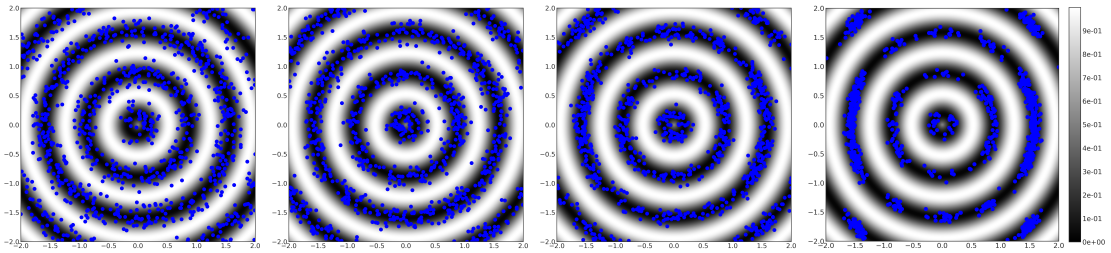
$$C(a, b, y_1, y_2) = (y_1^2 + y_2 - a)^2 + (y_1 + y_2^2 - b)^2$$
$$P(\boldsymbol{x}) = \exp\Big(-C(\boldsymbol{x})^2\Big),$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$, $\boldsymbol{x}_1 = (a, b)$, $\boldsymbol{x}_2 = (y_1, y_2)$, $P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x}))$, $\Omega_{\boldsymbol{x}_1} = [0, 15]^2$, $\Omega_{\boldsymbol{x}_2} = [-5, 5]^2$. The cost function $C(a, b, y_1, y_2)$ for a given (a, b) has multiple distinct global optima and many local optima. The samples from the TT distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = (a, b))$ are shown in Figure B.4–B.5 for different choice of task parameters and the prioritized sampling parameters α. We can see that TTGO can generate samples from all of the modes consistently according to the task parameters.

Mixture of Gaussians:

$$P(\boldsymbol{x}) = \sum_{j=1}^{J} \alpha_j \exp\Big(-\beta_j ||\boldsymbol{x} - \boldsymbol{a}_j||^2\Big),$$

We use an unnormalized mixture of Gaussian functions to define the probability function $P(\boldsymbol{x})$ to test our framework for high-dimensional multimodal functions. For verification, we design the mixture components so that we know the global optima *a priori* by carefully choosing the centers, mixture coefficients and variances. We test it for various values for the number of mixtures J, $\beta \in [1, 1000]$ and the dimension $d \in (2, \ldots, 50)$ of $\boldsymbol{x}$. We



(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.75$ (d) $\alpha = 0.9$

Figure B.1: 1000 samples (shown as blue dots) from the TT distribution of a 2D sinusoidal function for different values of α. The function has an infinite number of global optima (on the dark circles) and we see that TTGO is able to sample from these regions. As we increase α, the samples become more concentrated on the circles.

choose $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ with $\Omega_{\boldsymbol{x}} = [-2, 2]^d$ for various choices of values and dimension of $\boldsymbol{x}$. As TTGO does not differentiate between task parameters and optimization variables internally, we could consider various possibilities to segment $\boldsymbol{x}$ into $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ as task parameters and decision variables. We tested this problem for $d < 100$, and our approach could consistently find the optima with less than 100 samples from the TT-model, for arbitrary choice of variables being conditioned as task parameters. In contrast, finding the optima using Newton-type optimization with random initialization is highly unlikely for $\beta_j > 1$ and $d > 10$, even after considering millions of samples from uniform distribution for initialization.

Figure B.6 shows one particular example with $J = 10$, $\beta_j = 175$ and $d = 50$. To visualize, we choose $\boldsymbol{x}_1 \in \mathbb{R}^{d-2}$ and $\boldsymbol{x}_2 \in \mathbb{R}^2$, and we generate 1000 samples from the conditional TT distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1)$. With low values of α, the samples are generated around all the different modes, but as α is increased, the samples become more concentrated around the mode with the highest probability.

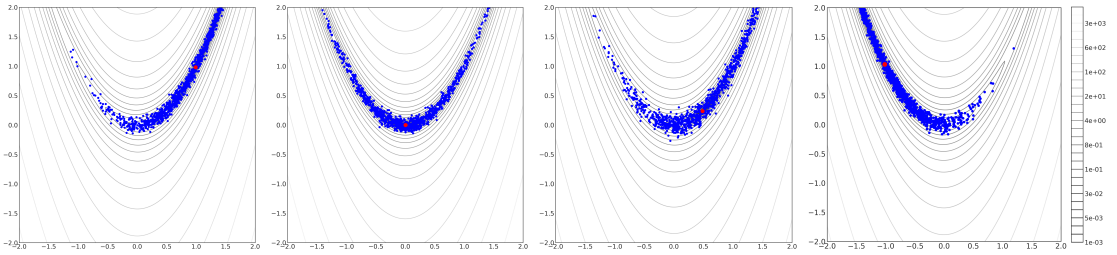
B.2 Inverse Kinematics Formulation

The cost function for the inverse kinematics problem in Section 4.5.1 is given by

$$C(\boldsymbol{x}) = \frac{1}{3}\left(\frac{C_p(\boldsymbol{\theta}, \boldsymbol{p}_d)}{\beta_p} + \frac{C_{\text{obst}}(\boldsymbol{\theta})}{\beta_{\text{obst}}} + \frac{C_{\text{orient}}(\boldsymbol{\theta})}{\beta_{\text{orient}}}\right), \tag{B.1}$$

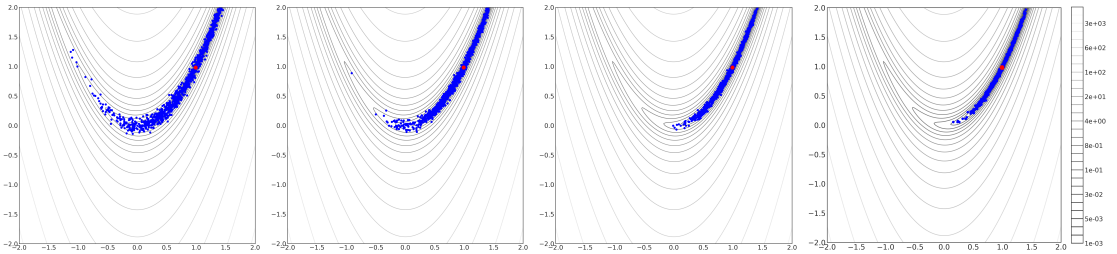
where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ and:

- $C_p(\boldsymbol{\theta}, \boldsymbol{p}_d) = \|\boldsymbol{p}_d - \boldsymbol{p}(\boldsymbol{\theta})\|$, Euclidean distance of the end effector position from the



(a) $(a, b) = (1, 100)$ (b) $(a, b) = (0, 140)$ (c) $(a, b) = (0.5, 60)$ (d) $(a, b) = (-1, 100)$

Figure B.2: 1000 samples from the conditional TT distribution of a Rosenbrock function for various choices of the task parameters (a, b) and $\alpha = 0$. The function has a unique global optimum at (a, a^2) as shown in red. As the task parameters change, the global optimum moves accordingly, but TTGO is still able to sample from the high-density regions.



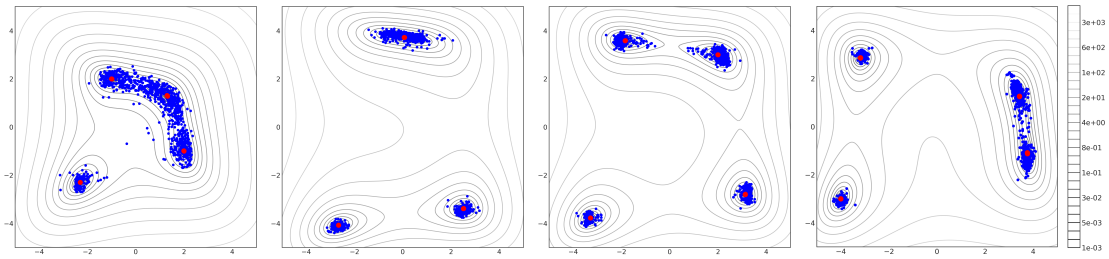
(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.75$ (d) $\alpha = 0.9$

Figure B.3: 1000 samples from the conditional TT distribution of a Rosenbrock function with the task parameters $a = 1, b = 100$ and various values of α. As α increases, the samples become more concentrated around the global optimum (as shown in red).

desired position.

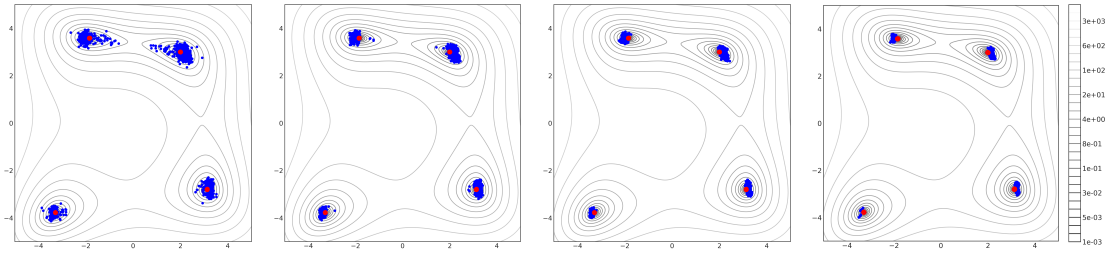
- $C_{\text{obst}}(\boldsymbol{\theta})$ represents the obstacle cost based on the Signed Distance Function (SDF). The links are approximated as a set of spheres (as done in CHOMP), and we use the SDF to compute the distance from each sphere to the nearest obstacle.

- $C_{\text{orient}}(\boldsymbol{\theta})$ represents the cost on the orientation of the end-effector. In our application, we specify a desired orientation of the end-effector, given by quaternion $\boldsymbol{q}_d$, while allowing a rotation around the axis of rotation $\boldsymbol{v}_d$ which corresponds to the z-axis of the world frame. This constraints the gripper orientation to be horizontal while allowing rotation around the z-axis. This is suitable for picking cylindrical objects from a shelf. The cost is then $C_{\text{orient}}(\boldsymbol{\theta}) = 1 - < \boldsymbol{v}(\boldsymbol{\theta}), \boldsymbol{v}_d >^2$ where $\boldsymbol{v}(\boldsymbol{\theta})$ represents the screw axis (computed from the quaternion) of the actual end-effector frame w.r.t. the desired frame. Alternatively, if the application demands a variation



(a) $(a, b) = (3, 3)$ (b) $(a, b) = (3, 14)$ (c) $(a, b) = (7, 11)$ (d) $(a, b) = (13, 5)$

Figure B.4: 1000 samples from the conditional TT distribution of a 2D Himmelblau function for various choices of the task parameters (a, b) and $\alpha = 0$. The location of the multiple global optima (in red) depend on the task parameters, but TTGO is able to generate the samples from the high-density regions.



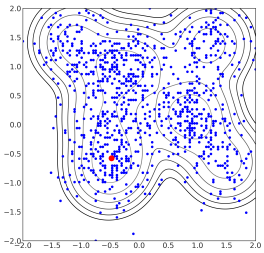
(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.9$ (d) $\alpha = 1$

Figure B.5: 1000 samples from the conditional TT distribution of a 2D Himmelblau function with task parameters $a = 7, b = 11$ for various values of α. As α increases, the samples become more concentrated around the global optima.

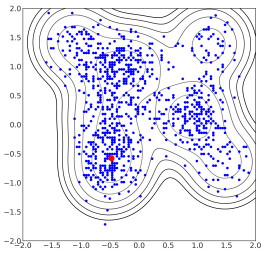
in the desired orientation, one could use the pose $(\boldsymbol{p}_d, \boldsymbol{q}_d)$ directly as the task parameter.

- $\beta_p, \beta_{\text{obst}}, \beta_{\text{orient}}$ are scaling factors for each cost. Intuitively, they represent the acceptable value for each cost. We use $\beta_p = 0.05$, $\beta_{\text{obst}} = 0.01$, and $\beta_{\text{orient}} = 0.2$ for the orientation.

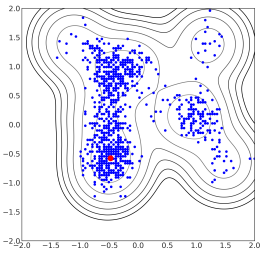
For the IK problem of the 6-DoF UR10 robot, there is no obstacle cost, and the orientation is specified to be identity (corresponding to upward-facing end-effector orientation) without any free axis of rotation.



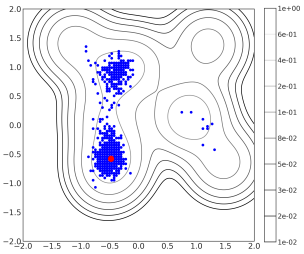
(a) $\alpha = 0$



(b) $\alpha = 0.5$



(c) $\alpha = 0.75$



(d) $\alpha = 0.9$

Figure B.6: 1000 samples from the conditional TT distribution of a mixture of Gaussians with $J = 10$, $d = 50$, $\beta_j = 175$, and various values of α. For visualization, we choose the the first $d - 2$ coordinates of μ_j to be the same for all j and choose the task-parameters to be the first $d - 2$ coordinate of the centers. This density function has one global optimum (in red) and some other modes that are comparable to the global optimum. As α increases, the samples become more concentrated around the mode with the highest density.

B.3 Motion Planning Formulation

For both the reaching and the pick-and-place tasks, the cost function, $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$, is given by

$$C(\boldsymbol{x}) = \frac{1}{4}\Big(\frac{C_p(\boldsymbol{x})}{\beta_p} + \frac{C_{\text{obst}}(\boldsymbol{x})}{\beta_{\text{obst}}} + \frac{C_{\text{orient}}(\boldsymbol{x})}{\beta_{\text{orient}}} + \frac{C_{\text{control}}(\boldsymbol{x})}{\beta_{\text{control}}}\Big), \tag{B.2}$$

with the following objectives:

- $C_p(\boldsymbol{x})$ represents the cost on the end effector position(s) from the target location(s).
- $C_{\text{obst}}(\boldsymbol{x})$ represents the cost incurred from the obstacles computed using SDF as in Section 4.5.1 but accumulated for the whole motion.
- $C_{\text{orient}}(\boldsymbol{x})$ represents the cost on the orientation of the end effector at the target location(s).
- $C_{\text{control}}(\boldsymbol{x})$ represents the cost of the length of the joint angle trajectory and the length of the end effector trajectory.

- $\beta_p, \beta_{\text{obst}}, \beta_{\text{orient}}, \beta_{\text{control}}$ are scaling factors for each cost. Intuitively, they represent the acceptable nominal cost value for each cost. We use $\beta_p = 0.05$, $\beta_{\text{obst}} = 0.1$, $\beta_{\text{orient}} = 0.2$, $\beta_{\text{control}} = 2$.

We consider the initial configuration of the manipulator to be fixed (we can relax this condition by considering the initial configuration as a task parameter). In the reaching task, the objective is to reach an end effector target location on the shelf. In the pick-and-place task, the objective is to reach a target on the shelf to pick an object, then move to another target above the box to place the object, and finally move back to the initial configuration.

Instead of considering the target in configuration space, we focus on Cartesian space, which presents unique challenges for optimization-based motion planning solvers. Reaching a target in configuration space typically yields a clear gradient to the solver, whereas reaching a Cartesian target poses a more difficult optimization problem due to the larger solution space, as the target may correspond to multiple configurations. A gradient-based solver will attempt to reach the target with the configuration that is closest to the initial configuration, particularly if initialized with a stationary trajectory at the initial configuration. However, if this solution is infeasible, it is challenging for the solver to find a different solution with a final configuration significantly different from the initial one, unless initialized well. One alternative approach is to use inverse kinematics (IK) to identify several possible final configurations, then use motion planning solvers to reach those configurations. However, it is not easy to select good configurations as the target, as it is difficult to determine whether a specific configuration is reachable from the initial configuration. Moreover, even when a solution is found, it may be highly suboptimal.

Our approach involves tackling both the IK problem and motion planning problem concurrently, with decision variables comprising the robot configuration(s) for the Cartesian target(s) and the joint angle trajectory needed to achieve those configurations. While optimizing both simultaneously can be challenging, our TTGO formulation enables us to obtain multiple solutions. To simplify the problem's complexity, we use motion primitives to represent the joint angle trajectory, as outlined in Appendix B.4. By utilizing our motion primitives formulation and given the initial and final configurations, we ensure that the movement always starts from the initial configuration and ends at the final configuration while complying with joint limitations.

Consider an m-DoF manipulator. The configuration of the manipulator can be represented using the joint angles $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m) \in \mathbb{R}^m$. We can assume that the domain of the joint angles is bounded by a rectangular domain $\Omega_{\boldsymbol{\theta}} = \times_{i=1}^{m} [\theta_{\min_i}, \theta_{\max_i}]$. We represent the trajectory evolution in terms of the phase of the motion, i.e., $t \in (0, 1)$ with $t = 1$

representing the end of the motion.

B.4 Motion Primitives

In our motion planning formulation, we generate motions using a basis function representation that satisfies the boundary conditions (with respect to phase/time) and the limits of the trajectory (the magnitude) while maintaining zero velocity at the boundary. Suppose we are given a choice of basis functions $\boldsymbol{\phi} = (\phi_k)_{j=1}^{J}, \phi_j(t) \in \mathbb{R}, \forall t \in [0,1]$. For example, we could use radial basis functions $\phi_j(t) = \exp(-\gamma(t-\mu_j)^2)$ with $\mu_j \in [0,1], \gamma \in \mathbb{R}^+$. We define a trajectory using a weighted combination of these basis functions as $\hat{\tau}(t) = \sum_{j=1}^{J} w_j \phi_j(t)$. We transform this trajectory so that the boundary conditions and joint limits are satisfied.

Given the trajectory $\hat{\tau}(t), t \in [0,1]$, and the boundary conditions $\tau(0) = \tau_0, \tau(1) = \tau_1$ and the limits $\tau_{\min} \leq \tau(t) \leq \tau_{\max}$, we can transform $\hat{\tau}(t)$ to obtain a trajectory $\tau(t) = \Psi(\hat{\tau}(t), \tau_0, \tau_1, \tau_{\min}, \tau_{\max})$ such that $\tau(0) = \tau_0$, $\tau(1) = \tau_1$ and $\tau_{\min} \leq \tau(t) \leq \tau_{\max}$. We define the transformation Ψ as follows:

1. Input: $\hat{\tau}, \tau_0, \tau_1, \tau_{\min}, \tau_{\max}$

2. Discretize the time interval $[0,1]$ uniformly to obtain $\{t_i\}_{i=0}^{N}$ so that $dt = t_{i+1} - t_i$, $t \in \{t_i\}_{i=0}^{N}$.

3. Define $\hat{z}(t) = \hat{\tau}(t) + \tau_0 - \hat{\tau}(0) + t(\tau_1 - \tau_0 + \hat{\tau}(0) - \hat{\tau}(1))$, which satisfies the specified boundary conditions.

4. Clip the trajectory within the joint limits to obtain $z(t) = \text{clip}(\hat{z}(t), \tau_{\min}, \tau_{\max})$. The clipping will result in non-smoothness.

5. Smoothen the trajectory $z(t)$ to obtain the desired trajectory $\tau(t)$: To do this, we append the trajectory $z(t)$ with the same values as initial value in the beginning and with the final value at the end. Then we can apply a moving average filter over the trajectory. This creates the desired smooth trajectory $\tau(t)$ that has zero velocity at the boundary.

This way we can generate smooth motion while satisfying the boundary conditions and the joint limits, and maintain zero velocity at the boundary.

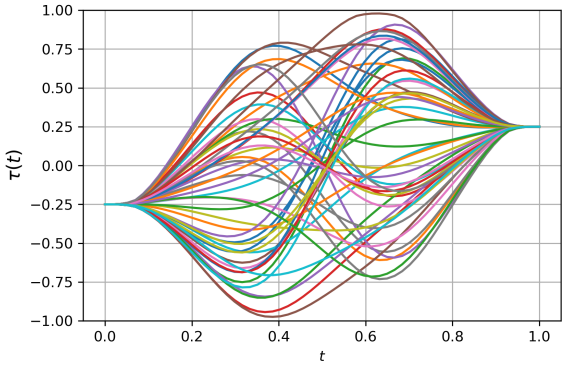


Figure B.7: A distribution of 50 smooth trajectories generated by transforming trajectories generated by using two radial basis functions with weights chosen uniformly in the range $[-1, 1]$. The transformations are done to maintain a boundary condition $\tau_0 = -0.25, \tau_1 = 0.25$ and the limits $\tau_{\min} = -1, \tau_{\max} = 1$.

B.5 Comparison of Various Function Approximation Techniques

In this section, we aim to illustrate the advantages of function approximation using the Tensor Train (TT) decomposition.

The standard procedure for approximating a Probability Density Function (PDF) with a GMM using Expectation Maximization (EM) involves the following steps: data collection (sampling from the PDF), determining the number of components for the GMM, and initializing the GMM parameters. However, it is essential to acknowledge that GMM-based approaches, particularly when using EM, can face challenges related to sensitivity to the number of components specified in the surrogate GMM. This sensitivity can impact the quality of the approximation.

To address some of these challenges, Variational Inference (VI) techniques have been developed as used in [10]. VI offers an alternative approach to approximating a PDF with a GMM and has the advantage of not requiring an exact number of mixture components as input. Instead, VI only necessitates an upper bound on the number of potential mixture components. In our experiments, we utilized the VI algorithm provided by the scikit-learn package to approximate a PDF using GMM which is also called Bayesian GMM (BGMM) approximation.

We aim to demonstrate that while GMM-based approaches have their merits, they may not be suitable for applications involving high-dimensional functions and the need for precise modeling of joint probability distributions. In such cases, as demonstrated in our

work, the TT decomposition offers distinct advantages in terms of accuracy and efficiency for function approximation, making it a preferred choice for the applications considered.

Experiment Setting: We created an ideal setting for function approximation using GMM. We use another GMM (target PDF/GMM) with a known mean and covariances as the target PDF and try to approximate it using BGMM. This allows us to quickly generate as many samples as possible from the target PDF (because sampling from the target GMM is straightforward). The amount of dataset generated for GMM modeling is scaled with the dimensionality. We specify the number of mixture components in the surrogate GMM to be twice that of the target GMM (an upper bound). This provides an ideal target PDF and hyperparameters for function approximation using GMM. The approximated model is finally evaluated with test data. We perform this evaluation for variations in dimensionality (d) number of mixture components (k) of the target GMM, and various choices of mean and covariances in the target GMM.

We created an ideal setup for GMM-based function approximation. Using another GMM as target PDF with known parameters, we aimed to precisely approximate it with GMM. This approach allowed us to efficiently generate a large dataset from the target GMM. The amount of dataset generated for GMM modeling is scaled with the dimensionality. The approximated model was tested across various dimensions, mixture components, and target GMM configurations.

Observations: The performance of the GMM model (see Figure B.8) was notably subpar and non-scalable in our evaluation. Notably, as the dimensionality (d) or the number of components (k) in the target GMM increased, the accuracy of the approximated GMM model deteriorated significantly, even within this idealized framework. This phenomenon is a well-documented challenge associated with GMM approximation, particularly when dealing with high-dimensional data [8].

In real-world scenarios, such as applications in robotics, the challenges further intensify:

- **Data Collection:** Gathering data from the target PDF for GMM approximation can be a formidable task and it is not possible and is often not feasible for real robotic applications. Approaches like importance sampling with variational inference attempt to address this challenge, but they inevitably introduce a drop in accuracy. For example, SMTO employs importance sampling with the proposal distribution centered around trajectories between two given paths.

- **Limitations in Expressiveness of GMM:** GMMs exhibit limited expressive power and are known to struggle when confronted with high-dimensional data [8].

In stark contrast, the TT format consistently outperforms GMM, even in ideal settings tailored for GMM modeling. The TT model demonstrates accuracy that is orders of magnitude higher than its GMM counterpart, and the computational time required to obtain the TT approximation is similarly orders of magnitude faster than GMM modeling. This is illustrated in Figure B.8, and the experimental details are provided in the accompanying software for reproduction.

Furthermore, we observed that GMM modeling experienced convergence challenges as dimensions and the number of mixture components increased. The impractical demands on data storage and computation time further hindered its scalability.

To demonstrate the scalability of TT, we conducted a similar comparison with Neural Networks (NN) under identical conditions (with the target function being a GMM). For NN regression, we generated a dataset by randomly sampling inputs from the function/pdf domain (NN inputs), with corresponding pdf values as the target outputs. This comparison is illustrated in Figure B.9. Our findings indicate that even for high-dimensional cases, the TT model outperforms the NN model in terms of accuracy and computation time.

Conclusion: This evaluation clearly highlights the formidable challenges associated with modeling a target Probability Density Function (PDF) using Gaussian Mixture Models (GMM). In practical applications, this approach frequently falls short in terms of accuracy and computational efficiency, rendering it impractical for tasks like modeling joint distributions for warm-starting, as demonstrated in our work.

The intrinsic advantages of the TT format for function approximation underscore our rationale for selecting this approach in our research. TT not only overcomes the limitations and challenges of existing methods employing GMM but also excels in terms of accuracy and computational speed. This underlines the effectiveness and suitability of TT as a preferred method for modeling complex joint distributions in scenarios like warm-starting, affirming the core contribution of our work.

In our experiment, TT-Cross was able to find the TT representation of the GMM accurately in under 20 seconds for each test case, while NN and GMM took several minutes and had a significantly higher error (often several orders of magnitude higher). Furthermore, the other methods required significant effort to tune the hyperparameters, whereas TT-Cross, as it is a non-parametric and unsupervised approach, was much easier to use. This is because TT-Cross finds the approximation by querying data (the function values at various points) intelligently [20] and exploits the structure in the function (i.e., low-rank or separability). It can do so as TT-Cross directly takes the function to be approximated as the input. On the other hand, NN takes a fixed set of samples

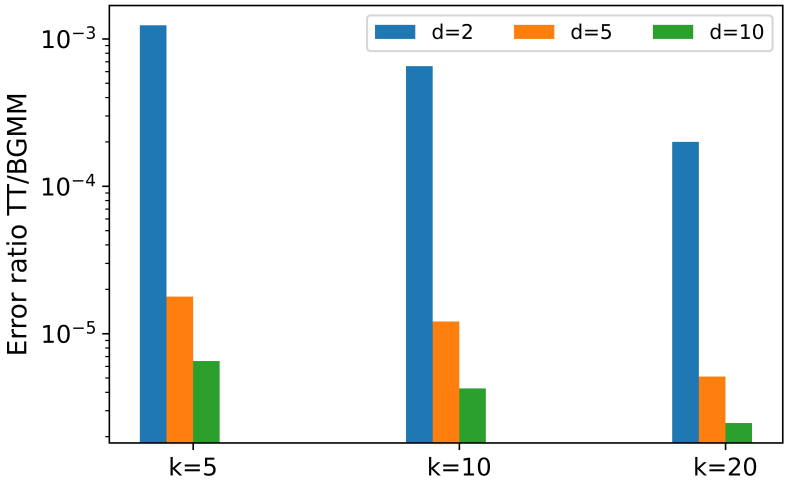


Figure B.8: The ratio of error in the approximation of the TT model found using TT-cross over BGMM model found using VI. The TT model is orders of magnitude more accurate than the GMM approximation. The experiment is conducted for the target PDF being GMM with variations in dimensions (d), and number of mixture components (k). For each case, the results are averaged over various choices of covariances (but constant volume) and mean in the target GMM.

from the function and does supervised learning to find the function approximation. We acknowledge that the approximation error in NN in our experiments could potentially be reduced by using more training data, and using a more exhaustive search for best hyperparameters. However, this would increase the training time and manual effort.

Although NN is an established tool for supervised learning over datasets, it is inefficient, compared to TT-Cross, when we need to approximate a known low-rank function accurately. Unlike TT-Cross, NN works with data collected from the function for the approximation and does not have a feedback mechanism to query points from the function during the approximation procedure. Thus, choosing NN as a function approximation technique in Approximate Dynamic Programming as described in Chapter 5, where we need to repeatedly approximate value functions from the previous estimations, comes with a drawback. The software code for this comparison is provided in the supplementary material at https://sites.google.com/view/ttgo/home.

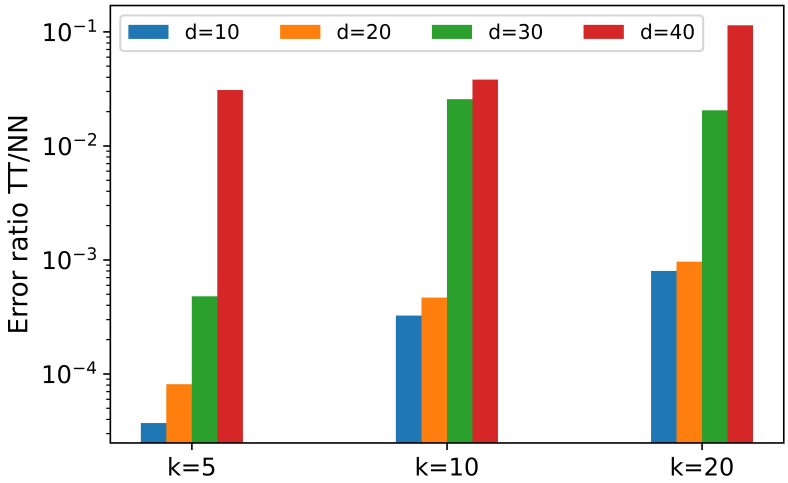


Figure B.9: The ratio of error in the approximation of the TT model was found using TT-cross over the Neural Network (NN) model. For high-dimensional problems, the TT model is orders of magnitude more accurate. The experiment is conducted for the target PDF being GMM with variations in dimensions (d), and number of mixture components (k). For each case, the results are averaged over various choices of covariances (but constant volume) and mean in the target GMM.

Bibliography

[1] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Annals of Physics*, vol. 349, pp. 117–158, 2013.

[2] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36(1), pp. 53–78, 2013.

[3] A. Cichocki, "Era of big data processing: A new approach via tensor networks and tensor decompositions," *ArXiv*, vol. 1403.2048, 2014.

[4] S. Rabanser, O. Shchur, and S. Günnemann, "Introduction to tensor decompositions and their applications in machine learning," *ArXiv*, vol. 1711.10781, 2017.

[5] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola, "TTHRESH: Tensor compression for multidimensional visual data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, pp. 2891–2903, 2018.

[6] M. B. Horowitz, A. Damle, and J. W. Burdick, "Linear Hamilton Jacobi Bellman equations in high dimensions," *IEEE Conference on Decision and Control (CDC)*, pp. 5880–5887, 2014.

[7] A. Gorodetsky, S. Karaman, and Y. Marzouk, "Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition," in *Proc. Robotics: Science and Systems (R:SS)*, July 2015, pp. 1–8.

[8] S. Shetty, J. Silvério, and S. Calinon, "Ergodic exploration using tensor train: Applications in insertion tasks," *IEEE Trans. on Robotics*, vol. 38, no. 2, pp. 906–921, 2022.

[9] S. Shetty, T. Lembono, T. Löw, and S. Calinon, "Tensor trains for global optimization problems in robotics," *International Journal of Robotics Research (IJRR)*, 2023.

[10] S. Shetty, T. Xue, and S. Calinon, "Generalized policy iteration using tensor approximation for hybrid control," in *International Conference on Learning Representations (ICLR)*, 2024, (spotlight paper, 5% acceptance rate).

[11] L. Brudermüller, T. Lembono, S. Shetty, and S. Calinon, "Trajectory prediction with compressed 3d environment representation using tensor train decomposition," in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, 2021, pp. 633–639.

[12] B. Nemec, M. M. Hrovat, M. Simonič, S. Shetty, S. Calinon, and A. Ude, "Robust execution of assembly policies using a pose invariant task representation," in *2023 20th International Conference on Ubiquitous Robots (UR)*, 2023, pp. 779–786.

[13] A. Cichocki, A. H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives," *ArXiv*, vol. 1708.09165, 2017.

[14] N. Kishore Kumar and J. Schneider, "Literature survey on low rank approximation of matrices," *Linear and Multilinear Algebra*, vol. 65, no. 11, pp. 2212–2244, 2017.

[15] S. A. Goreinov, I. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. Zamarashkin, "How to find a good submatrix," in *Matrix Methods: Theory, Algorithms And Applications: Dedicated to the Memory of Gene Golub*. World Scientific, 2010, pp. 247–256.

[16] A. Cichocki, N. Lee, I. V. Oseledets, A. H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Foundations and Trends in Machine Learning*, vol. 9, pp. 249–429, 2016.

[17] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, pp. 2295–2317, 2011.

[18] N. Lee and A. Cichocki, "Fundamental tensor operations for large-scale data analysis using tensor network formats," *Multidimensional Systems and Signal Processing*, vol. 29, no. 3, pp. 921–960, 2018.

[19] S. Dolgov and D. Savostyanov, "Parallel cross interpolation for high-precision calculation of high-dimensional integrals," *Computer Physics Communications*, vol. 246, p. 106869, 2020.

[20] D. V. Savostyanov and I. Oseledets, "Fast adaptive interpolation of multi-dimensional arrays in tensor train format," *The 2011 International Workshop on Multidimensional (nD) Systems*, pp. 1–8, 2011.

[21] I. Oseledets and E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays," *Linear Algebra and its Applications*, vol. 432, no. 1, pp. 70–88, 2010.

[22] S. Dolgov, K. Anaya-Izquierdo, C. Fox, and R. Scheichl, "Approximation and sampling of multivariate probability distributions in the tensor train decomposition," *Statistics and Computing*, vol. 30, pp. 603–625, 2020.

[23] A. Novikov, M. Trofimov, and I. V. Oseledets, "Exponential machines," in *International Conference on Learning Representations, ICLR*, 2017.

[24] E. M. Stoudenmire and D. J. Schwab, "Supervised learning with tensor networks," in *NIPS*, 2016.

[25] G. S. Novikov, M. E. Panov, and I. Oseledets, "Tensor-train density estimation," in *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, vol. 161. PMLR, 2021, pp. 1321–1331.

[26] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, "Unsupervised generative modeling using matrix product states," *Phys. Rev. X*, vol. 8, p. 031012, Jul 2018.

[27] J. Stokes and J. Terilla, "Probabilistic modeling with matrix product states," *Entropy*, vol. 21, 2019.

[28] A. Hubenko, V. A. Fonoberov, G. Mathew, and I. Mezic, "Multiscale adaptive search," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 4, pp. 1076–1087, 2011.

[29] L. M. Miller, Y. Silverman, M. A. MacIver, and T. D. Murphey, "Ergodic exploration of distributed information," *IEEE Transactions on Robotics*, vol. 32, pp. 36–52, 2016.

[30] G. Mathew and I. Mezić, "Metrics for ergodicity and design of ergodic dynamics for multi-agent systems," *Physica D: Nonlinear Phenomena*, vol. 240, no. 4-5, pp. 432–442, 2011.

[31] G. Mathew, S. Kannan, A. Surana, S. Bajekal, and K. R. Chevva, "Experimental implementation of spectral multiscale coverage and search algorithms for autonomous UAVs," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013, p. 5182.

[32] A. Mavrommati, E. Tzorakoleftherakis, I. Abraham, and T. D. Murphey, "Real-time area coverage and target localization using receding-horizon ergodic exploration," *IEEE Transactions on Robotics*, vol. 34, pp. 62–80, 2018.

[33] S. Calinon, "Mixture models for the analysis, edition, and synthesis of continuous time series," in *Mixture Models and Applications*, N. Bouguila and W. Fan, Eds. Springer, Cham, 2019, pp. 39–57.

[34] L. M. Miller and T. D. Murphey, "Trajectory optimization for continuous ergodic exploration," in *American Control Conference (ACC)*, 2013, pp. 4196–4201.

[35] L. Dressel and M. J. Kochenderfer, "Tutorial on the generation of ergodic trajectories with projection-based gradient descent," *IET Cyber-Physical Systems: Theory & Applications*, vol. 4, pp. 89–100, 2019.

[36] ——, "Using neural networks to generate information maps for mobile sensors," in *IEEE Conference on Decision and Control (CDC)*, 2018, pp. 2555–2560.

[37] E. Ayvali, H. Salman, and H. Choset, "Ergodic coverage in constrained environments using stochastic trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5204–5210.

[38] I. Abraham, A. Prabhakar, and T. D. Murphey, "An ergodic measure for active learning from equilibrium," *IEEE Transactions on Automation Science and Engineering*, 2021.

[39] S.-k. Yun, "Compliant manipulation for peg-in-hole: Is passive compliance a key to learn contact motion?" in *IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 1647–1652.

[40] P. R. Giordano, A. Stemmer, K. Arbter, and A. Albu-Schaffer, "Robotic assembly of complex planar parts: An experimental evaluation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 3775–3782.

[41] H. Park, J. Park, D.-H. Lee, J.-H. Park, M.-H. Baeg, and J.-H. Bae, "Compliance-based robotic peg-in-hole assembly strategy without force feedback," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 8, pp. 6299–6309, 2017.

[42] M. P. Polverini, A. M. Zanchettin, S. Castello, and P. Rocco, "Sensorless and constraint based peg-in-hole task execution with a dual-arm robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 415–420.

[43] D. Ehlers, M. Suomalainen, J. Lundell, and V. Kyrki, "Imitating human search strategies for assembly," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 7821–7827.

[44] M. J. A. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on Riemannian manifolds," *IEEE Robotics and Automation Letters (RA-L)*, vol. 2, no. 3, pp. 1240–1247, June 2017.

[45] S. Calinon, "Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control," *IEEE Robotics and Automation Magazine (RAM)*, vol. 27, no. 2, pp. 33–45, June 2020.

[46] G. Mathew, A. Surana, and I. Mezic, "Uniform coverage control of mobile sensor networks for dynamic target detection," *IEEE Conference on Decision and Control (CDC)*, pp. 7292–7299, 2010.

[47] C. Lubich, I. V. Oseledets, and B. Vandereycken, "Time integration of tensor trains," *SIAM J. Numerical Analysis*, vol. 53, pp. 917–941, 2015.

[48] D. Bigoni, A. P. Engsig-Karup, and Y. M. Marzouk, "Spectral tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 38, no. 4, pp. A2405–A2439, 2016.

[49] Hyeonjun Park, Ji-Hun Bae, Jae-Han Park, Moon-Hong Baeg, and Jaeheung Park, "Intuitive peg-in-hole assembly strategy with a compliant manipulator," in *IEEE International Symposium on Robotics (ISR)*, 2013, pp. 1–5.

[50] J. C. Triyonoputro, W. Wan, and K. Harada, "Quickly inserting pegs into uncertain holes using multi-view images and deep network trained on synthetic data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5792–5799.

[51] B. O. Koopman, "The theory of search. I. kinematic bases," *Operations research*, vol. 4, no. 3, pp. 324–346, 1956.

[52] ——, "The theory of search. II. target detection," *Operations research*, vol. 4, no. 5, pp. 503–531, 1956.

[53] ——, "The theory of search: III. the optimum distribution of searching effort," *Operations research*, vol. 5, no. 5, pp. 613–626, 1957.

[54] S. Ivić, B. Crnković, and I. Mezić, "Ergodicity-based cooperative multiagent area coverage via a potential field," *IEEE Transactions on Cybernetics*, vol. 47, pp. 1983–1993, 2017.

[55] M. Rakhuba and I. Oseledets, "Fast multidimensional convolution in low-rank tensor formats via cross approximation," *SIAM J. Sci. Comput.*, vol. 37, 2015.

[56] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[57] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[58] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits and Devices magazine*, vol. 5, no. 1, pp. 19–26, 1989.

[59] K. Sozykin, A. Chertkov, R. Schutski, A.-H. Phan, A. Cichocki, and I. Oseledets, "TTOpt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 052–26 065, 2022.

[60] T. Osa, "Multimodal trajectory optimization for motion planning," *International Journal of Robotics Research (IJRR)*, vol. 39, no. 8, pp. 983–1001, 2020.

[61] ——, "Motion planning by learning the solution manifold in trajectory optimization," *International Journal of Robotics Research (IJRR)*, vol. 41, no. 3, pp. 281–311, 2022.

[62] E. Pignat, T. Lembono, and S. Calinon, "Variational inference with mixture model approximation for applications in robotics," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 3395–3401.

[63] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *International Journal of Robotics Research (IJRR)*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[64] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2011, pp. 4569–4574.

[65] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *International Journal of Robotics Research (IJRR)*, vol. 33, no. 9, pp. 1251–1270, 2014.

[66] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *International Journal of Robotics Research (IJRR)*, vol. 37, no. 11, pp. 1319–1340, 2018.

[67] T. Sugihara, "Solvability-unconcerned inverse kinematics by the Levenberg–Marquardt method," *IEEE Trans. on Robotics*, vol. 27, no. 5, pp. 984–991, 2011.

[68] A. Escande, N. Mansard, and P.-B. Wieber, "Fast resolution of hierarchized inverse kinematics with inequality constraints," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2010, pp. 3733–3738.

[69] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, pp. 2594–2601, April 2020.

[70] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries." in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2006, pp. 3344–3349.

[71] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer Handbook of Robotics, 2nd Ed.*, 2016.

[72] N. Mansard, A. Del Prete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 2986–2993.

[73] K. K. Hauser, "Learning the problem-optimum map: Analysis and application to global optimization in robotics," *IEEE Transactions on Robotics*, vol. 33, pp. 141–152, 2016.

[74] M. Stolle, H. Tappeiner, J. Chestnutt, and C. G. Atkeson, "Transfer of policies based on trajectory libraries," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2007, pp. 2981–2986.

[75] N. Jetchev and M. Toussaint, "Trajectory prediction: learning to map situations to robot trajectories," in *Proc. Intl Conf. on Machine Learning (ICML)*, 2009, pp. 449–456.

[76] E. Dantec, R. Budhiraja, A. Roig, T. Lembono, G. Saurel, O. Stasse, P. Fernbach, S. Tonneau, S. Vijayakumar, S. Calinon, *et al.*, "Whole body model predictive control with a memory of motion: Experiments on a torque-controlled talos," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2021, pp. 8202–8208.

[77] D. A. Zheltkov and A. Osinsky, "Global optimization algorithms using tensor trains," in *International Conference on Large-Scale Scientific Computing*. Springer, 2019, pp. 197–202.

[78] A. Chertkov, G. V. Ryzhakov, G. S. Novikov, and I. Oseledets, "Optimization of functions given in the tensor train format," *ArXiv*, vol. abs/2209.14808, 2022.

[79] M. Usvyatsov, R. Ballester-Ripoll, and K. Schindler, "tntorch: Tensor network learning with PyTorch," *Journal of Machine Learning Research*, vol. 23, no. 208, pp. 1–6, 2022.

[80] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," *Advances in Neural Information Processing Systems (NIPS)*, vol. 26, 2013.

[81] E. Pignat, J. Silvério, and S. Calinon, "Learning from demonstration using products of experts: Applications to manipulation and task prioritization," *International Journal of Robotics Research (IJRR)*, vol. 41, no. 2, pp. 163–188, 2022.

[82] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. Robotics: Science and Systems (R:SS)*, 2018, pp. 1–8.

[83] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2014, pp. 279–286.

[84] J. Miller, G. Rabusseau, and J. Terilla, "Tensor networks for probabilistic sequence modeling," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 3079–3087.

[85] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 16, pp. 285–286, 2005.

[86] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1.

[87] B. Li, H. Tang, Y. ZHENG, J. HAO, P. Li, Z. Wang, Z. Meng, and L. Wang, "HyAR: Addressing discrete-continuous action reinforcement learning via hybrid action representation," in *International Conference on Learning Representations*, 2022.

[88] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," in *International Joint Conference on Artificial Intelligence*, 2019.

[89] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, "Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space," *arXiv preprint arXiv:1810.06394*, 2018.

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

[91] F. R. Hogan and A. Rodriguez, "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*. Springer, 2020, pp. 800–815.

[92] N. Doshi, F. R. Hogan, and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitives," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6759–6765.

[93] J. D. A. Ferrandis, J. P. De Moura, and S. Vijayakumar, "Nonprehensile planar manipulation through reinforcement learning with multimodal categorical exploration," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2023*. IEEE, 2023.

[94] T. Marcucci and R. Tedrake, "Warm start of mixed-integer programs for model predictive control of hybrid systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 6, pp. 2433–2448, 2020.

[95] R. Alur, T. Dang, and F. Ivančić, "Predicate abstraction for reachability analysis of hybrid systems," *ACM transactions on embedded computing systems (TECS)*, vol. 5, no. 1, pp. 152–199, 2006.

[96] A. I. Boyko, I. Oseledets, and G. Ferrer, "TT-QI: Faster value iteration in tensor train format for stochastic optimal control," *Computational Mathematics and Mathematical Physics*, vol. 61, pp. 836–846, 2021.

[97] A. Antos, C. Szepesvári, and R. Munos, "Fitted q-iteration in continuous action-space mdps," *Advances in neural information processing systems*, vol. 20, 2007.

[98] M. Lutter, B. Belousov, S. Mannor, D. Fox, A. Garg, and J. Peters, "Continuous-time fitted value iteration for robust policies," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[99] M. J. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *International Conference on Learning Representations, ICLR*, Y. Bengio and Y. LeCun, Eds., 2016.

[100] H. Fu, H. Tang, J. Hao, Z. Lei, Y. Chen, and C. Fan, "Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, ser. IJCAI'19. AAAI Press, 2019, p. 2329–2335.

[101] A. Gorodetsky, S. Karaman, and Y. Marzouk, "A continuous analogue of the tensor-train decomposition," *Computer methods in applied mechanics and engineering*, vol. 347, pp. 59–84, 2019.

[102] S. Dolgov, D. Kalise, and L. Saluzzi, "Data-driven tensor train gradient cross approximation for hamilton–jacobi–bellman equations," *SIAM Journal on Scientific Computing*, vol. 45, no. 5, pp. A2153–A2184, 2023.

[103] I. Abraham and T. D. Murphey, "Active learning of dynamics for data-driven control using koopman operators," *IEEE Transactions on Robotics*, vol. 35, pp. 1071–1083, 2019.

Lausanne, Switzerland
+41 762931120
suhan.n.shetty@gmail.com
suhannshetty.github.io
SuhanShetty
SuhanNShetty

Suhan Shetty

Summary

My research in robotics lies at the intersection of data-efficient machine learning and control engineering. In particular, using tensor networks I develop algorithms for Robot Control, Reinforcement Learning, and Motion Planning that were previously considered to be intractable.

Education

June 2019 – Feb 2024 **Doctor of Philosophy**.
École Polytechnique Fédérale de Lausanne
Thesis: Robot Learning using Tensor Networks

June 2014– June 2016 **Master of Engineering**.
Indian Institute of Science, Bangalore, India
Thesis: Trajectory Tracking and Control of Car-like Robots

Work Experience

June 2019 – Feb 2024 **Idiap Research Institute**, *Martigny*, Switzerland.
Research Assistant at Robot Learning and Interaction Group
Developing fast and memory efficient algorithms for robot exploration as used in the project CoLLaboratE for industrial assembly tasks, reinforcement learning and fast optimization algorithms as used in projects Learn-Real and MEMMO for robot control and motion planning.

2023 June – September **Disney Research Studios**, *Zurich*, Switzerland.
Research Intern
Developed deep reinforcement learning algorithms for concurrently learning the policy and state estimator for robust locomotion of bipedal robots.

Oct 2018 – April 2019 **Robert Bosch Center for Cyber-Physical Systems**, *Bangalore*, India.
Research Associate
Applied reinforcement learning to generate walking gaits for an in-house manufactured quadruped robot called Stoch.

July 2016 – Mar 2018 **The MathWorks Inc.**, *Bangalore*, India.
Engineering Development Group
Developed MATLAB and Simulink based models for demonstrating the applicability of MATLAB products such as Control System Toolbox, Robotics System Toolbox and Automated Driving System Toolbox in robotics applications.

Publications

2023 **S Shetty**, T Xue, and S Calinon, "Generalized Policy Iteration using Tensor Approximation for Hybrid Control", International Conference on Learning Representations (ICLR-2024). **[Spotlight Paper, 5% acceptance rate]**

2023 T Xue*, **S Shetty***, and S Calinon, "Dynamic Programming using Tensor Approximation for Contact-rich Manipulation", Workshop on Embracing Contacts, IEEE ICRA.

2023 **S Shetty**, T Lemobono, T Loew, and S Calinon, "Tensor Train for Global Optimization Problems in Robotics", The International Journal of Robotics Research (IJRR).

2021 **S Shetty**, J Silverio, and S Calinon, "Ergodic Exploration Using Tensor Train: Applications in Insertion Tasks", in *IEEE Transactions on Robotics*. **[Awarded Idiap's Paper of the year 2021 by Idiap Research Institute, Switzerland]**

2021 L Brudermüller, T Lembono, **S Shetty**, S Calinon, "Trajectory Prediction with Compressed 3D Environment Representation using Tensor Train Decomposition", in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*.

2019 S Kolathaya, A Joglekar, **S Shetty**, D Dholakiya, A Sagi, S Bhattacharya, A Singla, S Bhatnagar, A Ghosal, B Amrutur, "Trajectory based deep policy search for quadrupedal walking", in *28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*.

2019 **S Shetty,** A Ghosal, "Trajectory Tracking and Control of Car-Like Robots", in *Machines, Mechanism and Robotics, Lecture Notes in Mechanical Engineering. Springer, Singapore*.

Presentations

2023 "Generalized Policy Iteration using Tensor Approximation for Hybrid Control", *3rd NAVER LABS Europe International Workshop on AI for Robotics*, Grenoble, France.

2022 "Ergodic Exploration Using Tensor Train: Applications in Insertion Tasks", *International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan, (Presented virtually).

2020 "Mixture of Tensor-Normal Distribution for Imitation Learning in Robotics", *Swiss Machine Learning Days (SMLD)*, Lausanne, Switzerland.

Supervision

2022 "Tensor-variate dictionary learning of movement primitives", Mickael Gindroz, MSc Thesis, EPFL.

2021 "Multilinear Models for the Manipulation of Objects by Robots", Valentin Honorez, Faculté Polytechnique de Mons (UMONS), Belgium.

Programming Skills

- Languages: Python, C++, MATLAB
- Frameworks: Pytorch, Tensorflow, JAX, ROS, Git, LaTeX, Unix
- Robotic Simulators: Mujoco, Pybullet, NVIDIA Isaac Gym, NVIDIA Orbit

Academic Service

- Reviewer for IEEE Transactions on Robotics (2023), IROS (2023)

Miscellaneous

- **Ranked in top 100** among 200k candidates in the Graduate Aptitude Test in Engineering (Mechanical Engineering Division) in the year 2014. This is a highly competitive national level mathematics and technical aptitude test held by India's top tier universities for graduate studies.

- **Ranked 12th** in the Engineering Sciences in the National Eligibility Test for Junior Research Fellow in the year 2013. The test is held by the Council of Scientific and Industrial Research, India for research scholarship.