Journal Title
XX(X):1–26

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Tensor Train for Global Optimization Problems in Robotics

Suhan Shetty[1,2], Teguh Lembono[1,2], Tobias Loew[1,2], and Sylvain Calinon[1,2]

Abstract

The convergence of many numerical optimization techniques is highly sensitive to the initial guess provided to the solver. We propose an approach based on tensor methods to initialize the existing optimization solvers close to global optima. The approach uses only the definition of the cost function and does not need access to any database of good solutions. We first transform the cost function, which is a function of task parameters and optimization variables, into a probability density function. Unlike existing approaches that set the task parameters as constant, we consider them as another set of random variables and approximate the joint probability distribution of the task parameters and the optimization variables using a surrogate probability model. For a given task, we then generate samples from the conditional distribution with respect to the given task parameter and use them as initialization for the optimization solver. As conditioning and sampling from an arbitrary density function are challenging, we use Tensor Train decomposition to obtain a surrogate probability model from which we can efficiently obtain the conditional model and the samples. The method can produce multiple solutions coming from different modes (when they exist) for a given task. We first evaluate the approach by applying it to various challenging benchmark functions for numerical optimization that are difficult to solve using gradient-based optimization solvers with a naive initialization, showing that the proposed method can produce samples close to the global optima and coming from multiple modes. We then demonstrate the generality of the framework and its relevance to robotics by applying the proposed method to inverse kinematics and motion planning problems with a 7-DoF manipulator.

Keywords

Global Optimization, Multimodal Optimization, Motion Planning, Inverse Kinematics, Tensor Train, Tensor Factorization

1 Introduction

Numerical optimization has been one of the major tools for solving a variety of robotics problems including inverse kinematics, motion planning, control, and system identification. 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 which has a cost comparable to the global optima is usually sufficient. It is more 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.

There are stochastic procedures, often called evolutionary strategies (e.g., CMA-ES (Hansen et al. 2003), Genetic Algorithm (Whitley 1994), Simulated Annealing (Rutenbar 1989)), 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 in nature and require a good initial guess that determines the solution quality and the time required to find a solution.

Finding good techniques to initialize a Newton-type optimizer is an ongoing area of research. A common approach is to first build a database of *optimal* solutions in the offline phase for all possible robotics tasks that are of interest in a given application (Stolle and Atkeson 2006; Mansard et al. 2018; Lembono et al. 2020b). Then, to solve a given task in the online phase, an approximate solution is retrieved from the database to initialize the optimization solver. While this approach is simple to implement and can be applied to a general problem, building a good database is often challenging, since the solver often cannot find even a feasible solution for difficult problems without good initialization. Furthermore, predicting an initial guess from the database is also challenging when the underlying optimization problem is multimodal, i.e., when a given task corresponds to multiple solutions. Standard function approximation tools such as Gaussian Process Regression (GPR) (Rasmussen and Williams 2006) and Multilayer Perceptron (MLP) will average the different modalities, resulting in a poor prediction.

While the multimodal issue can potentially be overcome by keeping only one solution mode in the database, it is not ideal. Firstly, having multimodal solutions available can

[1]Idiap Research Institute, Martigny, Switzerland
[2] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
Email: name.surname@idiap.ch

be useful. For example, in many applications, before the solution is executed on the robot, it needs the approval of an expert (or the user). In such scenarios, multiple solutions for the given task are desirable so that the user has enough options. Secondly, it is often difficult to separate the different modes, making it impossible to keep only one solution mode in the database.

In this paper, we propose a novel approach to produce approximate solutions to a given optimization problem that we call Tensor Train for Global Optimization (TTGO). This approach combines several different techniques, mainly: Tensor Train (TT) decomposition for function approximation (Oseledets 2011), sampling from TT model (Dolgov et al. 2020), and numerical optimization using cross approximation technique (Sozykin et al. 2022). In contrast to the database approach, we firstly transform the cost function to an unnormalized probability density function and then approximate the density function using TT decomposition (Oseledets 2011), a technique from multilinear algebra. TT models, as shown recently by Dolgov et al. (2020), allow fast procedures to generate exact samples from the density model. Furthermore, we extend this approach to generate samples from a conditioned TT model with controlled priority for high-density regions (which in turn correspond to the low-cost regions) that can then be used as approximate solutions. This approach allows us to obtain a richer set of solutions, especially for multimodal problems. As it does not use any gradient information, it is also less susceptible to getting stuck at local optima.

We consider cost function as a function of the *task parameters* (e.g., the desired end effector pose) and the *optimization variables* (e.g., the robot configuration for an inverse kinematics problem or the joint angle trajectory for motion planning). We can formulate the problem of minimizing a cost function as maximization of the corresponding probability density function. Previous work that attempt to approximate the probability density function using Variational Inference (e.g., Osa (2020, 2022); Pignat et al. (2020)) usually treat the task parameters as constant, so the cost function is only a function of the optimization variables. In contrast, TTGO allows us to handle varying task parameters by approximating the joint probability distribution of the task parameters and the optimization variables. It exploits the correlation between the task parameters and the optimization variables (for example, a slight change in the task parameter usually results in a small change in the solution) that give the problem a *low-rank structure*, enabling the TT model to approximate the density function compactly. Once the model is trained, we can condition the model on the specific task parameters, and then generate samples that are approximate solutions to the corresponding task. This allows us to generate approximate solutions quickly during online execution for various tasks.

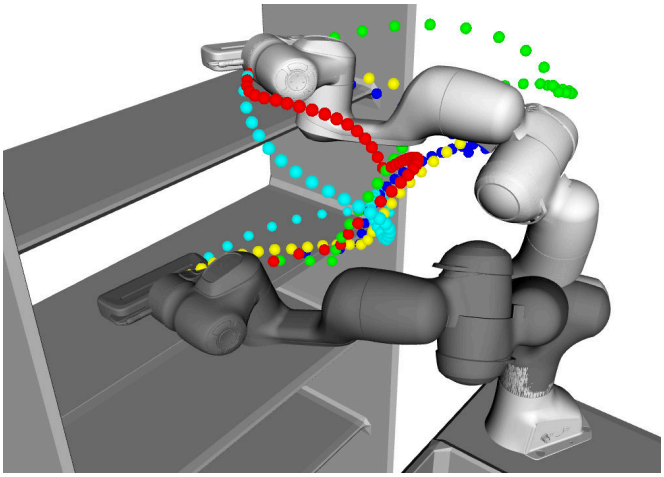
Our approach neither requires any external database of *good* solutions nor a separate regression model to retrieve approximate solutions from the database. With access only to the definition of the cost function, we build our database in the offline phase compactly in TT format in an unsupervised manner, i.e., without the need of another solver. Due to its structure, the TT representation provides efficient ways to retrieve approximate solutions in the online phase in the order of milliseconds, thus avoiding any need for a separate regression model for inferring the solution. When the underlying problem is multimodal, the retrieved solutions will also come from multiple modes.

In summary, our contributions are as follows:

- We propose a principled approach called TTGO (Tensor Train for Global Optimization) to obtain approximate solutions to a given optimization problem. The approximate solutions are close to the global optima (or the good local optima) and can then be used to initialize gradient-based solvers for further refinement.
- Our approach builds an implicit database in Tensor Train (TT) format by only using the definition of the cost function in an unsupervised manner, i.e., without requiring any gradient information or another solver.
- In the online phase, our approach can produce approximate solutions very quickly for a given task (i.e., in the order of milliseconds and linearly scaling with the dimensionality of the problem) by using samples from a conditioned TT model.
- We propose a prioritized sampling technique where we can adjust between sampling from only the high-density region (to obtain only the best solution) or from the whole distribution (to obtain a greater variety of solutions) via a continuous parameter.
- When the underlying optimization problem is multimodal, our approach can find multiple solutions that correspond to a given task.
- The approach is 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 and motion planning problems with a 7-DoF manipulator.

In this paper, we describe TTGO in its generic form where the TT model is constructed to model varying task parameters. However, TTGO can also be used when we only want to solve a single task. In this particular case, the TT model corresponds to the probability distribution of only the optimization variables, and the training will require much less time as compared to the generic form as the variation of optimization variables. In terms of computation time and quality of solution, it is similar to evolutionary strategies such as CMA-ES or GA, but TTGO can offer multiple solutions.

The paper layout is as follows. In Section 2, we provide a literature survey on initializing numerical optimization, multimodal optimization, and tensor methods. Section 3 explains the necessary background on Tensor Train modeling that is used in this paper. Then, in Section 4, we describe the TTGO method proposed in this paper. Section 5 presents the evaluation of our algorithm. We first test it on benchmark functions for numerical optimization and then apply it to inverse kinematics and motion planning problems with manipulators. In Section 6 and 7, we conclude the paper by discussing how our approach could lead to new ways of solving a variety of problems in robotics. We also discuss here the limitations and future work.



(a)

Figure 1. Solutions from TTGO for motion planning of a manipulator from a given initial configuration to a final configuration. The obtained joint angle trajectories result in different path for the end effector which are highlighted by dotted curves in different colors. The multimodality is clearly visible from these solutions.

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 methods. We discuss each topic briefly in this section.

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 (Zucker et al. 2013), STOMP (Kalakrishnan et al. 2011), TrajOpt (Schulman et al. 2014), GPMP (Mukadam et al. 2018)). Inverse kinematics for high dimensional robots is usually formulated as nonlinear least squares optimization (Sugihara 2011) or Quadratic Programming (QP) (Escande et al. 2010). In control, optimization-based controllers take the form of Task Space Inverse Dynamics (TSID) controller formulated as QP problem (Del Prete and Mansard 2016), or finite horizon optimal control (Mastalli et al. 2020; Kleff et al. 2021). 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 (Lembono et al. 2020b). 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.

2.2 Predicting good initialization

A majority of works that attempt to predict good initialization rely on a database approach, often called trajectory library (Stolle and Atkeson 2006) or memory of motion (Mansard et al. 2018). The idea is to first build a database of precomputed solutions offline. This database can be constructed from expert demonstrations (Stolle et al. 2007), using the optimization solver itself (Jetchev and Toussaint 2009), or using the combination of a global planner and the optimization solver (Dantec et al. 2021). 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. We can then use the database to train different function approximation techniques (e.g. k-Nearest Neighbors, Gaussian Process Regression, Gaussian Mixture Regression, Neural Network) to learn this map. During online execution, we query the function approximator to provide us with the initial guess of a given problem. 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. The database ideally covers the whole range of possible tasks, but many tasks are difficult to solve without a good initialization to the solver. Some work overcome this by relying on a global planner to provide the good initialization when building the database Dantec et al. (2021); Lembono et al. (2020a), but it remains difficult to cover the whole solution space efficiently.

Let us assume that we can obtain a good database, and we want to train the function approximators. Many problems in robotics are multimodal, i.e., there are multiple different solutions for a given task. Approximating this one-to-many mapping is difficult for most function approximators, which tend to average the different modalities resulting in poor predictions. Some work attempt to handle the multimodal prediction using mixture models, with Gaussian Mixture Regression (Lembono et al. 2020b) or Mixture Density Networks (Brudermüller et al. 2021), but while they perform better than the standard function approximators, we still observed some averaging behaviors that result in poor predictions. One of the reasons is related to the problem of constructing the database; for the mixture models to perform well, the database should contain enough data points from each mode, which is difficult to ensure in practice.

2.3 Multimodal Trajectory Optimization

Related to the problem of building the database above, most optimization solvers only produce one (locally) optimal solution. When more solutions are needed, heuristics approaches such as initialization from a uniform distribution

(random initialization) or manually-defined waypoints are usually used. A more principled way transforms the cost into an unnormalized probability density function (PDF) and uses probability density estimation techniques, most commonly using Variational Inference (Osa 2020, 2022; Pignat et al. 2020). This allows us to obtain multimodal solutions when multiple modes exist or to explore the whole solution space when the possible solutions are infinite.

The work closest to our approach is SMTO (Osa 2020). It approximates the unnormalized PDF with a GMM using Variational Inference by minimizing the forward KL divergence. Forward KL requires sampling from the target PDF, which is not feasible in practice. Osa (2020) tries to solve this issue by relying on importance sampling, where the samples are generated from a proposal distribution and their importance is weighted by the ratio between the proposal density and the target density. It allows them to obtain multiple solutions to a given optimization problem. However, the main limitation of the method is the requirement of a good proposal distribution and the locally optimal nature of the method. The approximate model can be optimized only around the generated samples from the proposal distribution. It means the resulting distribution will not deviate a lot from the proposal distribution. Especially, when some modes are not explored by the initial samples, the approximate model will not be able to cover these modes after the subsequent iterations. Furthermore, unlike our approach, it does not allow distribution of computation into offline and online phase, i.e., all the computation needed to solve a given task is done in the online phase.

While SMTO (Osa 2020) attempts to obtain multiple solutions from finitely many modes, Osa (2022) proposes LSMO that explores an infinite homotopic solution. It learns latent representations of solutions that can be used to generate an infinite set of solutions by modifying the continuous latent variables. Instead of using a GMM, it uses a neural network parameterized with the latent variables to approximate the PDF. Again, this approach solves one optimization problem at a time and the computation time is high for online operations. Our approach, by distributing the computation into offline and online phases, allows us to solve multiple optimization problems approximately in the offline phase and provide fast approximate solutions in the online phase. It can also handle the cases with either finite modes (as in Osa (2020)) or uncountably many solutions (as in Osa (2022)), as will be shown in Section 5.

2.4 Tensor Methods

Tensor factorization techniques (also called Tensor Networks) are extensions of matrix factorization techniques into multidimensional arrays (i.e., tensors). These techniques approximate a given tensor compactly using a set of lower-dimensional arrays (called factors). In addition to the compact representation, they allow efficient algebraic operations to be performed on them. Popular tensor factorization techniques include CP/PARAFAC decomposition, Tucker decomposition, Hierarchical Tucker decomposition, and Tensor Train (TT), see Sidiropoulos et al. (2017); Kolda and Bader (2009) for general surveys, and see Rabanser et al. (2017); Cichocki et al. (2015) for applications in machine learning and signal processing. Tensor factorization techniques have also been used in robotics to solve control problems that were previously considered to be intractable (Shetty et al. 2022; Horowitz et al. 2014; Gorodetsky et al. 2015).

Tensor Train (TT) decomposition, also known as Matrix Product States (MPS), provides a good balance between expressive power and efficiency of the representation, and it is equipped with robust algorithms to find the decomposition (Grasedyck et al. 2013). TT decomposition has been used to solve problems involving high-dimensional integration of multivariate functions in Dolgov and Savostyanov (2020); Shetty et al. (2022). Dolgov et al. (2020) used it to approximate probability density functions, with a fast procedure to sample from a probability distribution represented using the TT format. TT decomposition has also been used for data-driven density modeling (or generative modeling) (Han et al. 2018; Stokes and Terilla 2019; Miller et al. 2021; Novikov et al. 2021).

In Zheltkov and Osinsky (2019); Sozykin et al. (2022), it has been demonstrated that TT decomposition can be used for gradient-free optimization and that its performance is competitive with state-of-art global optimization algorithms such as CMA-ES and GA. These approaches using TT decomposition for global optimization are similar to evolutionary strategies as they solve one optimization problem at a time (too slow for the use-cases in robotics applications) and can provide only one solution. We take a different direction in this paper; we work with the probability density function (corresponding to the cost function) which is approximated using a TT decomposition and use efficient ways to sample from high-density regions of this surrogate model to approximate the solutions. This allows us to distribute the computationally intensive part to an offline phase and solve many optimization problems fast in an online phase. Moreover, our approach can be used for finding multiple solutions.

3 Background

In this section, we first describe what a tensor is (Section 3.1) and how a multivariate function can be approximated using a tensor (Section 3.2). The size of the tensor increases exponentially with the number of dimensions, rendering the naive approach of computing the whole tensor intractable for high-dimensional functions. We then describe how to overcome the curse of dimensionality by relying on tensor factorization techniques that allow efficient computation and storage of the tensor. We start with the matrix case for an easier example (Section 3.3 and 3.4) and proceed with the extension for the higher-order tensor (Section 3.5 and 3.6). When the target function is an unnormalized probability density function (PDF), we can construct a probability distribution from the tensor model (Section 3.7), allowing us to sample (Section 3.8-3.9) and condition (Section 3.10) on some of the dimensions.

3.1 Tensors

A tensor is a multidimensional array and as such, it is a higher-dimensional generalization of vectors and matrices. A vector can be considered as a first-order tensor and a

matrix as a second-order tensor. The order of a tensor, therefore, refers to the number of dimensions (or modes) of 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 define the index set $\mathcal{I}$ of the tensor $\boldsymbol{\mathcal{P}}$ to be a set $\mathcal{I} = \{\boldsymbol{i} = (i_1, \ldots, i_d), i_k \in \{1, \ldots, n_k\}, k \in \{1, \ldots, d\}\}$. This is used to uniquely identify the elements of the tensor. We denote the $\boldsymbol{i}$-th element of the tensor $\boldsymbol{\mathcal{P}}$ by $\boldsymbol{\mathcal{P}}_{\boldsymbol{i}}$.

A *fiber* is the higher-order analogue of matrix row and column which is a vector obtained by fixing every index but one. Similarly, a *slice* of a tensor is a matrix obtained by fixing every index but two.

3.2 Tensors as Discrete Analogue of a Function

In many applications, tensors arise naturally from the discretization of multivariate functions defined on a rectangular domain. Consider a function $P : \Omega_{\boldsymbol{x}} \subset \mathbb{R}^d \to \mathbb{R}$ with a rectangular domain $\Omega_{\boldsymbol{x}} = \times_{k=1}^{d} \Omega_{x_k}$, i.e., a Cartesian product of the intervals of each dimension. Unless stated otherwise, we discretize the intervals uniformly. We discretize each bounded interval $\Omega_{x_k} \subset \mathbb{R}$ into n_k number of elements. $\mathcal{X} = \{\boldsymbol{x} = (x_1^{i_1}, \ldots, x_d^{i_d}) : x_k^{i_k} \in \Omega_{x_k}, i_k \in \{1, \ldots, n_k\}\}$ represents the discretization set and the corresponding index set is defined as $\mathcal{I}_{\mathcal{X}} = \{\boldsymbol{i} = (i_1, \ldots, i_d) : i_k \in \{1, \ldots, n_k\}, k \in \{1, \ldots, d\}\}$. We have a canonical bijective discretization map that maps the indices to the tensor elements, i.e, $X : \mathcal{I}_{\mathcal{X}} \to \mathcal{X}$ 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 such a discretization, we can obtain a tensor $\boldsymbol{\mathcal{P}}$, a discrete analogue of the function P, 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 the notation, we overload the terminology and define $\boldsymbol{\mathcal{P}}_{\boldsymbol{x}} = \boldsymbol{\mathcal{P}}_{X^{-1}(\boldsymbol{x})}, \forall \boldsymbol{x} \in \mathcal{X}$. Note that 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 certain nodes of the tensor $\boldsymbol{\mathcal{P}}$.

For a high-dimensional function, naively approximating it using a tensor is intractable due to the complexity of both the computation and the storage of the tensor ($\mathcal{O}(n^d)$ where n is the maximum number of discretization and d is the dimension of the function and hence the order of the tensor). Tensor factorization solves the storage issue by representing a tensor with its factors that have a smaller number of elements. While many factorization techniques still require the computation of the whole tensor, one particular factorization technique called cross-approximation allows us to directly compute the factors by using only a function that can evaluate the value of the tensor given an index, hence solving the computation issue. The following sections start by discussing matrix factorization and cross-approximation for approximating 2D functions to provide some intuition and then extend it to higher-order tensors.

3.3 Separation of Variables using Matrix Factorization

Consider a continuous 2D function

$$P(x_1, x_2) \colon \Omega_{\boldsymbol{x}} \subset \mathbb{R}^2 \to \mathbb{R}. \tag{1}$$

Let $\Omega_{\boldsymbol{x}} = \Omega_{\boldsymbol{x}_1} \times \Omega_{\boldsymbol{x}_2}$ be the rectangular domain formed by the Cartesian product of intervals so that $x_1 \in \Omega_{\boldsymbol{x}_1}$ and $x_2 \in \Omega_{\boldsymbol{x}_2}$. We can find a discrete analogue $\boldsymbol{P}$ (which is a matrix in 2D case) of this function by evaluating the function on a grid-like discretization of the domain $\Omega_{\boldsymbol{x}}$. Let us discretize the interval $\Omega_{\boldsymbol{x}_1}$ and $\Omega_{\boldsymbol{x}_2}$ with n_1 and n_2 discretization points respectively. Let $(x_1^1, \ldots, x_1^{n_1})$ and $(x_2^1, \ldots, x_2^{n_2})$ be the corresponding discretization points of the two intervals. The discretization set is then given by $\mathcal{X} = \{\boldsymbol{x} = (x_1^{i_1}, x_2^{i_2}) \colon i_k \in \{1, \ldots, n_k\}, k \in \{1, 2\}\}$ and corresponding index set is $\mathcal{I}_{\mathcal{X}} = \{\boldsymbol{i} = (i_1, i_2) \colon i_k \in \{1, \ldots, n_k\}, k \in \{1, 2\}\}$. The corresponding discrete analogue is then given by the matrix defined as

$$\boldsymbol{P}_{i_1, i_2} = P(x_1^{i_1}, x_2^{i_2}), \quad \forall (i_1, i_2) \in \mathcal{I}_{\mathcal{X}}. \tag{2}$$

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

$$\boldsymbol{P}_{i_1, i_2} \approx \boldsymbol{P}^1_{i_1, :} \, \boldsymbol{P}^2_{:, i_2}. \tag{3}$$

The matrix factorization can be realized, for example, using QR, SVD, or LU decompositions. Such a factorization offers several advantages: firstly, it can be used to represent the original matrix $\boldsymbol{P}$ compactly if the rank r is low. Moreover, as we now show, it can be used to represent the function P in a separable form. First, note that (3) can only be used to evaluate the function P at the discretized points in $\mathcal{X}$. For a general $\boldsymbol{x} = (x_1, x_2) \in \Omega_{\boldsymbol{x}}$, we can use linear interpolation between the rows (or columns) and define the vector values functions

$$\begin{aligned} \boldsymbol{p}^1(x_1) &= \frac{x_1 - x_1^{i_1}}{x_1^{i_1+1} - x_1^{i_1}} \boldsymbol{P}^1_{i_1+1, :} + \frac{x_1^{i_1+1} - x_1}{x_1^{i_1+1} - x_1^{i_1}} \boldsymbol{P}^1_{i_1, :}, \\ \boldsymbol{p}^2(x_2) &= \frac{x_2 - x_2^{i_2}}{x_2^{i_2+1} - x_2^{i_2}} \boldsymbol{P}^1_{:, i_2+1} + \frac{x_2^{i_2+1} - x_2}{x_2^{i_2+1} - x_2^{i_2}} \boldsymbol{P}^2_{:, i_2}, \end{aligned} \tag{4}$$

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 higher-order polynomial interpolation here.

Then, we have the approximation for the function P in a separable form,

$$\begin{aligned} P(x_1, x_2) &\approx \boldsymbol{p}^1(x_1) \boldsymbol{p}^2(x_2), \quad \forall (x_1, x_2) \in \Omega_{\boldsymbol{x}} \\ &= \sum_{j=1}^{r} \boldsymbol{p}_j^1(x_1) \boldsymbol{p}_j^2(x_2). \end{aligned} \tag{5}$$

Such a factorization of multivariate functions as a sum of product of univariate functions is an extremely powerful representation. For example, the integration of the multivariate function can be computed using integration of the univariate functions (factors) (Dolgov and Savostyanov 2020; Shetty et al. 2022). If the multivariate function in hand is a probability density function, such separable representation also allows elegant sampling procedures (e.g., using conditional distribution sampling (Dolgov et al. 2020)) which will be discussed in Section 3.8.

In many engineering applications, we mostly deal with functions that have such separable forms. Moreover, we often have functions characterized by some smoothness

improving separability. The degree of separability of the function P determines a certain low-rank structure in the discrete analogue $\boldsymbol{P}$ of the function (often indicated by the number of sums in the sum of products of univariate functions representation). This implies that the rank r of the factors would be low and thus the number of parameters to represent the factors is low.

The approximation accuracy of (5) also depends on the number of discretization points and on the decomposition technique that we use to find the factors. For the case of 2D functions, a common approach is to use matrix decomposition techniques such as QR, SVD or LU decomposition to find the factors. However, a standard implementation of these algorithms require the whole matrix $\boldsymbol{P}$ to be computed and stored in memory, and incurs a computational cost of $\mathcal{O}(n_1 n_2)$. Although the resultant factors would require low memory for storage, if the discretization is very fine (i.e., n_1 and n_2 are very large numbers), computing and storing the matrix $\boldsymbol{P}$ becomes expensive and inefficient.

A particular factorization technique called the *cross approximation* method avoids the above problem. It can directly find the separable factors without having to compute and store the whole tensor in memory. In the next section, we briefly explain the matrix cross approximation technique and some of its interesting features that are exploited in TTGO.

3.4 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).

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, $\text{rank}(\tilde{\boldsymbol{P}}) = r$ so that the matrix $\boldsymbol{P}$ can be approximated with a lower rank r (i.e., $\text{rank}(\boldsymbol{P}) \approx r$). 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). 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 (Goreinov et al. 2010) 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 as follows:

1. *Input*: $\boldsymbol{P} \in \mathbb{R}^{n_1 \times n_2}$, the approximation rank r for the skeleton decomposition.
2. Find the columns index set $\boldsymbol{i}_2^*$ and the row index set $\boldsymbol{i}_1^*$ corresponding to the maximum volume submatrix.
   2.1 Randomly choose r columns $\boldsymbol{i}_2$ of the matrix $\boldsymbol{P}$ and repeat the following until convergence:
   - 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 $\boldsymbol{P}_{:,\boldsymbol{i}_2}$.
   - 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 $\boldsymbol{P}_{\boldsymbol{i}_1,:}$.
3. *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 non-singular. Thus, a more practical implementation uses the QR decomposition of these matrices and the MAXVOL algorithm is applied to the corresponding Q factor to find the columns (or rows) of the submatrix. Furthermore, instead of a random choice in step (2.1), one can choose the r columns from the multinomial distribution given by $p(i_2) = \frac{\|\boldsymbol{P}_{:,i_2}\|}{\|\boldsymbol{P}\|}, i_2 \in \{1, \ldots, n_1\}$ without sample replacement.

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.
- Since cross approximation algorithms follow the maximum volume principle, the factors are composed of *high magnitude* elements of the original matrix with high probability (Goreinov et al. 2010). This is very useful for TTGO as we are interested in finding the maxima from a tensor (discrete analogue of a probability density function) and the skeleton decomposition preserves this information.

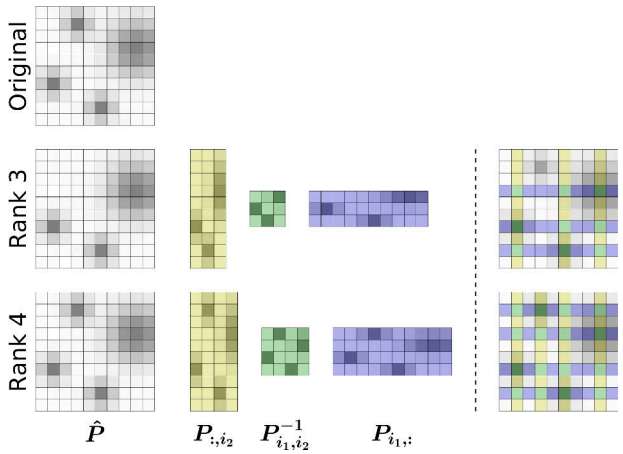


Figure 2. For a given matrix $\boldsymbol{P}$ (top-left), suppose we know r independent columns indexed by $\boldsymbol{i}_2 = (i_{2,1}, \ldots, i_{2,r})$, i.e., $\boldsymbol{P}_{:,\boldsymbol{i}_2} \in \mathbb{R}^{n_1 \times r}$ and r independent rows indexed by $\boldsymbol{i}_1 = (i_{1,1}, \ldots, i_{1,r})$, 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 $\text{rank}(\boldsymbol{P}) = r$, then $\hat{\boldsymbol{P}} = \boldsymbol{P}$ (bottom row). For $r < \text{rank}(\boldsymbol{P})$ 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.

- Cross approximation algorithms directly find the factors without computing and storing the whole matrix.

3.5 Tensor Train Decomposition

Similar to matrix factorization, tensor factorization allows us to represent a tensor by its factors. Among the popular factorization techniques, we concentrate in this work on the Tensor Train (TT) decomposition. 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 3, 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:

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{i}} = \boldsymbol{\mathcal{P}}^1_{:,i_1,:} \boldsymbol{\mathcal{P}}^2_{:,i_2,:} \cdots \boldsymbol{\mathcal{P}}^d_{:,i_d,:}, \tag{6}$$

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. 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 decomposition (6) (Oseledets 2011).

Similarly to (5), we can also obtain a continuous approximation of the function P as

$$P(x_1, \ldots, x_d) \approx \boldsymbol{P}^1(x_1) \cdots \boldsymbol{P}^d(x_d), \tag{7}$$

where $\boldsymbol{P}^k(x_k), k \in \{1, \ldots, d\}$ is obtained by interpolating each of the core, analogously to the matrix example in (4) (see Appendix A.1 for more detail). We overload the terminology again to define the continuous TT representation as

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{x}} = P(x_1, \ldots, x_d), \quad \forall \boldsymbol{x} \in \Omega_{\boldsymbol{x}}.$$

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 3.3, 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. 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). More formally, .

3.6 TT-Cross

The popular methods to find the TT decomposition of a tensor are TT-SVD (Oseledets 2011), TT-DMRG (Dolgov and Savostyanov 2020), and TT-cross (Savostyanov and Oseledets 2011). TT-SVD and TT-DMRG, like matrix SVD, require the full tensor in memory to find the decomposition, and hence they are infeasible for higher-order tensors. TT-cross approximation (TT-cross) is an extension of the cross approximation technique explained in Section 3.4 for obtaining the TT decomposition of a tensor. We refer the readers to Appendix A.2 for more detail on how the matrix cross-approximation algorithm described in Section 3.4 can be adapted to find the TT decomposition using TT-cross. It is appealing for many practical problems as it approximates the given tensor 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.

Suppose we have a function P and its discrete analogue $\boldsymbol{\mathcal{P}}$ (a tensor). Given a maximal TT-rank r for the approximation, TT-cross returns an approximate tensor in TT format $\hat{\boldsymbol{\mathcal{P}}} = \text{TT-cross}(P, r)$ to the tensor $\boldsymbol{\mathcal{P}}$ by querying only a portion of its elements ($\mathcal{O}(ndr^2)$ evaluations instead of $\mathcal{O}(n^d)$). This is very efficient if the TT-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

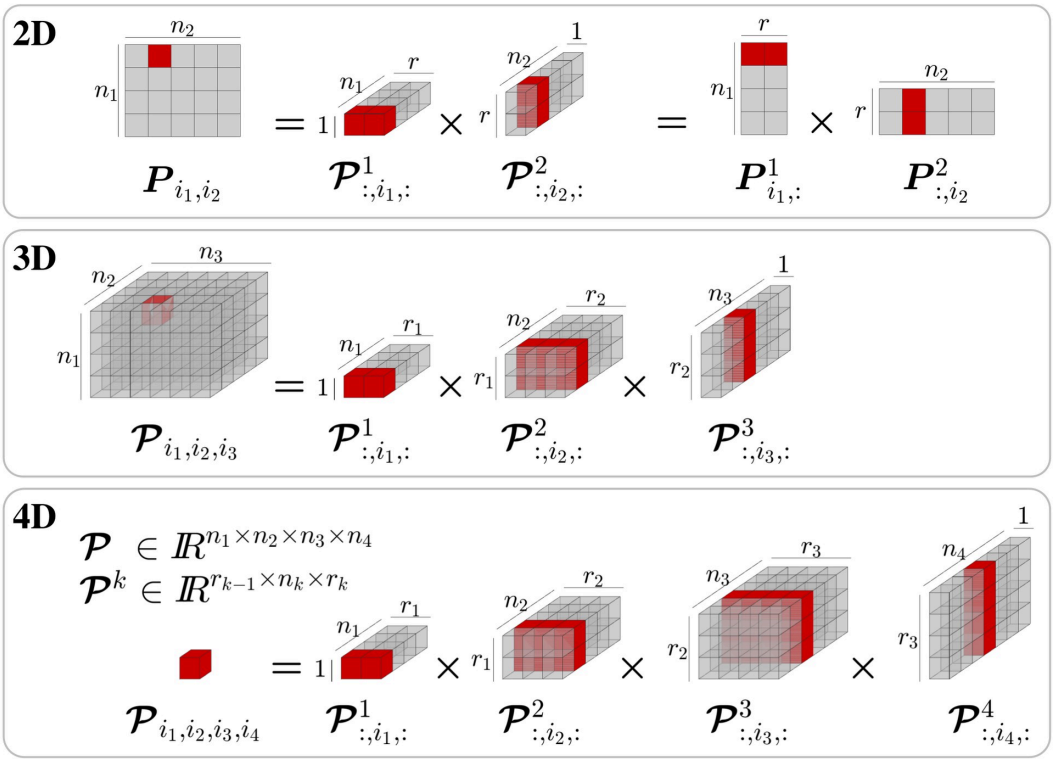


Figure 3. TT decomposition is an extension of matrix decomposition techniques to higher dimensional arrays. With a matrix decomposition, we can access an element of the original matrix by multiplying appropriate rows or columns of the factors. Similarly, an element of a tensor in TT format can be accessed 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.

tensor $\mathcal{P}$ at various query points, i.e., the fibers of the tensor $\mathcal{P}$.

3.7 TT Distribution

Suppose we use the tensor $\mathcal{P}$ in TT format to approximate an unnormalized probability density 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,

$$\Pr(\boldsymbol{x}) = \frac{|\mathcal{P}_{\boldsymbol{x}}|}{Z}, \quad \boldsymbol{x} \in \Omega_{\boldsymbol{x}}, \tag{8}$$

where Z is the corresponding normalization constant.* Due to the separable structure of the TT model, we can get the exact samples from the TT distribution in an efficient manner without requiring to compute the normalization factor Z. In the next section, we provide details about sampling from the above distribution for the discrete case $\boldsymbol{x} \in \mathcal{X}$ which is adapted from a continuous version introduced in Dolgov et al. (2020).

3.8 Sampling from TT distribution

Consider a probability distribution given by (8). 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. The distribution can be expressed as a product of conditional distributions

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

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

for $k = 1, \ldots, d$ **do**
  $x_k \approx \texttt{Pr}_k(x_k|x_1, \ldots, x_{k-1})$
end for

However, this is computationally intensive as sampling x_k requires the conditional distribution $\Pr_k$ which in turn requires the evaluation of the summation over several variables to find the marginal σ_k. It results in a computational cost that grows exponentially with the number of dimensions. This is where the TT format provides a nice solution by relying on the separability of the function. Let $\mathcal{P}$ be the discrete analogue of the function Pr (or P as $Z = 1$), a tensor in TT format, with the discretization set $\mathcal{X}$. Let the

*Alternatively, we could also define the TT distribution to be $\Pr(\boldsymbol{x}) = \frac{\mathcal{P}^2_{\boldsymbol{x}}}{Z}$. All the techniques, such as conditional sampling and prioritized sampling, used in this paper can also be adapted to this distribution. However, for simplicity of presentation, we do not consider it here.

Algorithm 1 TT-CD Sampling with Sample Prioritization

Input: TT Blocks $\boldsymbol{\mathcal{P}} = (\boldsymbol{\mathcal{P}}^1, \dots, \boldsymbol{\mathcal{P}}^d)$ corresponding to the distribution Pr, sample priority $\alpha \in (0, 1)$

Output: N $\alpha-$prioritized samples $\{(x_1^l, \dots, x_d^l)\}_{l=1}^N$ from the distribution Pr

$\hat{\boldsymbol{\pi}}_{d+1} \leftarrow 1$
for $k \leftarrow d$ to 2 **do**
  $\hat{\boldsymbol{\pi}}_k = (\sum_{x_k} \boldsymbol{\mathcal{P}}^k_{:,x_k,:})\hat{\boldsymbol{\pi}}_{k+1}$
end for
$\boldsymbol{\Phi_1} \leftarrow \mathbf{1} \in \mathbb{R}^{N\times 1}$
for $k \leftarrow 1$ to d **do**
  $\boldsymbol{\pi}_k(x_k) = \boldsymbol{\mathcal{P}}^k_{:,x_k,:}\hat{\boldsymbol{\pi}}_{k+1}, \forall x_k$
  for $l = 1, \dots, N$ **do**
    $\boldsymbol{p}_k(x_k) = |\Phi_k(l,:)\boldsymbol{\pi}_k(x_k)|, \forall x_k$
    $\boldsymbol{p}_k \leftarrow \frac{\boldsymbol{p}_k}{\max \boldsymbol{p}_k}$
    $\boldsymbol{p}_k \leftarrow \boldsymbol{p}_k^{\frac{1}{1-\alpha+\epsilon}}$, where ϵ is positive and $\epsilon \approx 0$
    $\boldsymbol{p}_k(x_k) \leftarrow \frac{\boldsymbol{p}_k(x_k)}{\sum_{x_k} \boldsymbol{p}_k(x_k)}, \forall x_k$ (normalization)
    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 model be given by the cores $(\boldsymbol{\mathcal{P}}^1, \dots, \boldsymbol{\mathcal{P}}^d)$, then we have

$$\begin{aligned}\sigma_k(x_1, \dots, x_k) &= \sum_{x_{k+1}} \cdots \sum_{x_d} \Pr(\boldsymbol{x}),\\ &\approx \sum_{x_{k+1}} \cdots \sum_{x_d} |\boldsymbol{\mathcal{P}}_{\boldsymbol{x}}|,\\ &= \sum_{x_{k+1}} \cdots \sum_{x_d} |\boldsymbol{\mathcal{P}}^1_{:,x_1,:}| \cdots |\boldsymbol{\mathcal{P}}^k_{:,x_k,:}||\boldsymbol{\mathcal{P}}^{k+1}_{:,x_{k+1},:}| \cdots |\boldsymbol{\mathcal{P}}^d_{:,x_d,:}|,\\ &= |\boldsymbol{\mathcal{P}}^1_{:,x_1,:}| \cdots |\boldsymbol{\mathcal{P}}^k_{:,x_k,:}| \Big(\sum_{x_{k+1}} |\boldsymbol{\mathcal{P}}^{k+1}_{:,x_{k+1},:}|\Big) \cdots \Big(\sum_{x_d} |\boldsymbol{\mathcal{P}}^d_{:,x_d,:}|\Big),\end{aligned} \tag{9}$$

where $\sum_{x_k} |\boldsymbol{\mathcal{P}}^k_{:,x_k,:}|$ is the summation of all the matrix slices (absolute values) of the third-order tensor (cores of TT). Thus, the TT-format reduces the complicated summation into one-dimensional summations. Noting that the same summation terms appear over several conditionals $\Pr_k$, we can have the an algorithm, i.e., Tensor Train Conditional Distribution (TT-CD) sampling (Dolgov et al. 2020), to efficiently get the samples from Pr.

3.9 Prioritized Sampling

The previous section explains how to sample from a TT distribution. In some applications, however, we do not necessarily want to sample from the whole distribution, but instead to focus on obtaining samples from the high-density region (e.g., when we only want to find the modes of the distribution). It is possible to adjust the previous sampling procedure to allow *prioritized sampling*. Namely, we can choose a hyperparameter $\alpha \in (0, 1)$ to prioritize samples from higher-density regions in the distribution $\Pr(\boldsymbol{x})$ given by (3.7). $\alpha = 0$ leads to generating exact samples from the true TT distribution whereas $\alpha = 1$ leads to sampling from regions closer to the mode of the distribution. Values of α higher than 0 reduce the likelihood of generating samples from low-density regions of the TT distribution. This algorithtm is described in Algorithm 1. The prioritized sampling can be relaxed by setting $\alpha = 0$ in the algorithm, resulting in the standard sampling procedure described in Section 3.8. Note that the algorithm allows parallel implementation to quickly generate a large number of samples.

3.10 Conditional TT Model and 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 (8).

Suppose $\boldsymbol{x}_1$ takes a particular value $\boldsymbol{x}_t$. 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

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

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

$$(\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t})^k = \begin{cases} \boldsymbol{\mathcal{P}}^k_{:,\boldsymbol{x}_{t_k},:}, & k \in \{1, \dots, d_1\} \\ \boldsymbol{\mathcal{P}}^k, & k \in \{d_1+1, d_1+d_2\} \end{cases} \tag{10}$$

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

$$\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t) = \frac{|\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}_{\boldsymbol{x}_2}|}{Z_1}, \forall \boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \tag{11}$$

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

4 Methodology

4.1 Problem Definition

Cost functions in a robotics application often depend on two kinds of variables: *task parameters* that are constant for a given optimization problem and *decision variables* that are the variables being optimized (i.e., optimization variables). The task parameters parameterize the possible tasks that could be encountered in a given application by the robot. For example, in an inverse kinematics (IK) problem with obstacles, the task parameters can be the desired end effector pose and the decision variables can be the robot configuration, i.e., the joint angles. In most applications, we can anticipate the possible range of the task parameters (e.g., the robot workspace for IK). This means that ideally, we can solve the optimization problem many times for the whole range of task parameters offline, and use this experience to speed up the online optimization for a new task.

We further note that the cost function in robotics is often a piecewise smooth function that imposes a certain structure (i.e., low-rank structure as explained in Section 3) among the variables of the cost function. For example, similarity in task parameters corresponds to a certain similarity in the solutions to the optimization problem. Furthermore, there can be strong correlations between the decision variables due to the cost function (e.g., the movements of the manipulator

joints are correlated when it needs to maintain the same orientation). Capturing this structure will enable us to model the relation among the variables compactly instead of relying on database approaches that store every single data point. To the best of our knowledge, such a structure has not been well exploited so far, despite the fact that it exists in many robotic applications.

In this article, we propose a framework that exploits such a structure to gather experience in the offline phase for faster optimization in the online phase. As discussed in Section 2, the common approach using database and function approximators does not work well when the optimization problems are highly non-convex with many poor local optima and the solutions are multimodal. Our framework provides a principled solution to these two problems. The following section presents the approach in detail.

4.2 Overview of the Proposed Approach

Given an optimization problem, our Tensor Train for Global Optimization (TTGO) framework predicts approximate solution(s) that can be refined using an optimization solver. The refinement can be done using standard Newton-type solvers such as SLSQP or L-BFGS-B, so we focus our discussion on the problem of predicting a good approximation of the solution.

Our approach first transforms the cost function into an unnormalized Probability Density Function (PDF) and approximates it using a surrogate probability model, namely a TT distribution. We view the cost function as a function of the optimization variables and the task parameters which parameterize the optimization problem. Hence, the surrogate model approximates the *joint distribution* of the task parameters and the optimization variables. During online execution, when the user specifies a task parameter, we condition this surrogate model on the corresponding task parameter. Then, we can sample from this conditional distribution to obtain approximate solutions corresponding to the specified task parameters. When the underlying PDF is multimodal, the samples will also come from the multiple modes. These samples are good candidates for the solutions. We can then select the best sample(s) by evaluating the corresponding cost functions and take the sample(s) with the lowest cost (when multiple solutions are needed, we can keep several best samples). In the second stage, these proposals for the optima are refined using a suitable optimization technique, e.g. Newton-type solvers if the objective function is differentiable.

The feasibility of such an approach depends on the properties of the surrogate probability model, namely:

- The surrogate probability model should be able to approximate a wide range of probability density functions we encounter in robotics by using only the cost function definition.
- Conditioning and sampling from the surrogate probability model determine the speed of the online execution and hence it should be fast.

The first requirement comes from the fact that we do not usually have access to the samples from the true probability distribution; we only have the definition of the density function (corresponding to the cost function) that can return the value of the function at a query point.

In our approach, we propose to use the TT distribution (Section 3.7) as the surrogate model that satisfies the above requirements. The TT model defining the TT distribution corresponds to the discrete analogue of the given unnormalized PDF, and it can be obtained efficiently using TT-Cross algorithm (Section 3.6). The efficiency is in terms of the number of evaluations of the target function to be modeled, the memory requirement, and the speed of computation. The resultant TT distribution allows fast sampling procedures (see Section 3.8). Since we use the samples from the TT distribution as the solution candidates, we are often mainly interested in samples from the high-density regions (i.e., the low-cost regions). This can be accomplished using the prioritized sampling procedures for TT distribution (see Section 3.9).

In the following section, we provide the mathematical formulation of the approach.

4.3 Mathematical Formulation

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 $\boldsymbol{x}_2$:

$$\begin{aligned} \boldsymbol{x}_2^* = &\arg\min_{\boldsymbol{x}_2} C(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &\text{s.t. } \boldsymbol{x}_1 = \boldsymbol{x}_t, \\ &\quad \boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \end{aligned} \tag{12}$$

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 $d = d_1 + d_2$. TTGO decomposes 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 $\boldsymbol{x}_1 = \boldsymbol{x}_t$, 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,

$$\begin{aligned} \boldsymbol{x}_2^* = &\arg\max_{\boldsymbol{x}_2} P(\boldsymbol{x}_t, \boldsymbol{x}_2) \\ &\text{s.t. } \boldsymbol{x}_1 = \boldsymbol{x}_t, \\ &\quad \boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \end{aligned} \tag{13}$$

For example, we can define $P(\boldsymbol{x}) = e^{-\beta C(\boldsymbol{x})}$ with $\beta > 0$. Without loss of generality, in the remainder of the paper we consider optimization problems to be of type (13) 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 mode is as difficult as solving the optimization problem in (12). TTGO overcomes 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 $\Pr(\boldsymbol{x}_1, \boldsymbol{x}_2)$. Given the task $\boldsymbol{x}_1 = \boldsymbol{x}_t$, we condition the TT model to obtain the conditional distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$. Finally, the TT model allows us to sample easily from its distribution to produce the approximate solution(s) $\hat{\boldsymbol{x}}_2^*$.

4.3.1 Approximating the unnormalized PDF using TT model: As described in Section 3, a TT model can approximate a multivariate function as its discrete analogue, i.e., by discretizing the function on a rectangular domain and storing the value in a tensor. For a high-dimensional function, however, it is intractable to construct and store the whole tensor. To avoid the curse of dimensionality, we rely on TT decomposition that allows us to store the tensor in a very compact form as TT cores. We use the TT-cross algorithm that allows us to compute the TT cores without having to construct the whole tensor, reducing the complexity of both the storage and the computation significantly.

Given the unnormalized PDF $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$, TTGO uses the TT-Cross algorithm (see Section 3.6) 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. More details on how to approximate the function using the TT decomposition are described in Section 3.

The tensor model $\boldsymbol{\mathcal{P}}$ is an approximation of the unnormalized PDF. We can construct the corresponding normalized TT distribution $\Pr(\boldsymbol{x})$ with (8), which requires the normalization constant as per the definition. However, as described in Section 3.8, we can sample from the TT distribution without calculating the normalization constant. Hence, in practice we can generate the samples by working directly with the unnormalized density $\boldsymbol{\mathcal{P}}$.

4.3.2 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 3.10. This is the desired surrogate probability model for $P(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t)$.

4.3.3 Sampling: As described in Section 3.8, it is possible to sample efficiently from a TT distribution. The sampling procedure consists of a repeated sampling of each dimension separately from a multinomial distribution, as described in Algorithm 1. Furthermore, a sampling parameter $\alpha \in (0, 1)$ can be chosen to the adjust the sampling priority (see Section 3.9). 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 many samples and select the best few samples according to their cost function values as the solution candidates $\hat{\boldsymbol{x}}_2^*$.

4.4 TTGO Algorithm

1. Training Phase (Offline):
   - 1.1 Given:
     - Cost function $C(\boldsymbol{x}_1, \boldsymbol{x}_2)$,
     - Rectangular domain $\Omega_{\boldsymbol{x}} = \Omega_{\boldsymbol{x}_1} \times \Omega_{\boldsymbol{x}_2}$
   - 1.2 Transform the cost function into an unnormalized PDF $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$.
   - 1.3 Discretize the domain Ω_x into $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$.
   - 1.4 Using TT-Cross, construct the TT-Model $\boldsymbol{\mathcal{P}}$ as the discrete analogue of $P(\boldsymbol{x})$ with discretization set $\mathcal{X}$.
2. Inference Phase (Online):
   - 2.1 Given: The task-parameter $\boldsymbol{x}_1 = \boldsymbol{x}_t \in \Omega_{\boldsymbol{x}_1}$, the desired number of solutions K.
   - 2.2 Construct the conditional TT Model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}$ from $\boldsymbol{\mathcal{P}}$ (see Section 3.10).
   - 2.3 Generate N samples $\{\boldsymbol{x}_2^l\}_{l=1}^N$ with the sampling parameter $\alpha \in (0, 1)$ from the TT distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t) = \frac{|\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}_{\boldsymbol{x}_2}|}{Z}$ (Algorithm 1).
   - 2.4 Evaluate the cost function at these samples and choose the best-K samples as approximation for the optima $\{\hat{\boldsymbol{x}}_2^{*^l}\}_{l=1}^K$.
   - 2.5 Fine-tune the approximate solutions using gradient-based approaches on $C(\boldsymbol{x}_t, \boldsymbol{x}_2)$ to obtain the optima $\{\boldsymbol{x}_2^{*^l}\}_{l=1}^K$.

5 Experiments

In this section, we evaluate the performance of the proposed algorithm with several applications.* We first apply it to some benchmark functions for numerical optimization solvers to show the capability of TTGO to find global optima and multiple solutions consistently. We then apply it to robotics applications, i.e., numerical inverse kinematics and motion planning of manipulators. Besides qualitatively observing the solutions, we also perform quantitative analyses to evaluate the quality of the approximate solutions produced by TTGO. We consider three different metrics:

- c_i, the initial cost value of the approximate solutions.
- 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.

For comparison, we use random samples from the *uniform* distribution across the whole domain to initialize the solver. We also use TTGO with different values of α to observe the effect of prioritized sampling. We then evaluate the

*A PyTorch-based implementation of TTGO and the accompanying videos are available at https://sites.google.com/view/ttgo/home

performance as follows. First, we generate 100 random test cases within the task space. For each test case, we generate N samples and take the best sample (in terms of the initial cost value) as the approximate solution. We use the number of samples N ranging from 1 to 1000 and generate the samples using both methods (TT and uniform distribution). We use the SLSQP solver to optimize this sample according to the given cost function. Finally, we evaluate the three metrics on this sample, i.e., the initial cost c_i, the final cost after refinement c_f, and the convergence status. We then compute the average performance of both methods across the whole test cases. The results for the robotics applications can be found in Table 1-3 and will be explained in the corresponding sections.

5.1 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 article. 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 4- 9, 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 4-8 and the density function in Figure 9, 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.

5.1.1 Sinusoidal Function:

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

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 4 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.

5.1.2 Rosenbrock Function:

$$C(a, b, y_1, \ldots, y_{d_1}) = \sum_{k=1}^{d_2/2} (y_{2k-1} - a)^2 + b(y_{2k-1} - y_{2k}^2)^2$$
$$P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x})^2),$$

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 5), a gradient-based solver will have difficulty in converging to the global optima quickly. We can see from Figure 5 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 6 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.

5.1.3 Himmelblau's function:

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

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 7–8 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.

5.1.4 *Mixture of Gaussians:*

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

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 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 9 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.

5.2 *Inverse Kinematics*

We consider here the optimization formulation of Inverse Kinematics (IK), i.e., numerical IK instead of analytical one. The task parameters $\boldsymbol{x}_1$ then correspond to the desired end effector pose, while the decision 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 a density function as $P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x})^2)$.

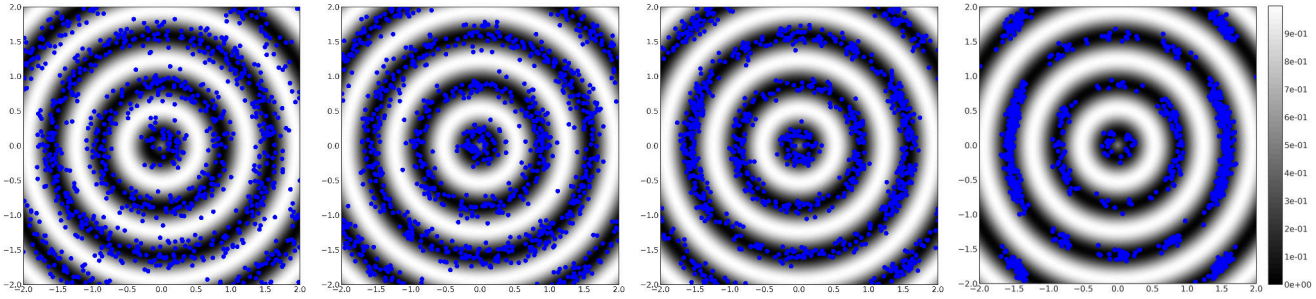
5.2.1 Inverse Kinematics for 6-DoF Robot: A 6-DoF robot has a finite number of joint angle configurations that correspond to a given end effector pose. In this section, we consider the 6-DoF Universal Robot that can have up to 8 IK solutions. While there is an analytical solution for such robots, it is a nice case study to illustrate the capability of TTGO 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 TTGO is able to retrieve most of the 8 IK solutions for a given end effector pose. Figure 10 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 approximate multimodal solutions even for a complex distribution.

5.2.2 Inverse Kinematics for 7-DoF Robot with Obstacle Cost: Unlike a 6-DoF robot, a 7-DoF robot can have an infinite number of joint angle configurations that correspond to a given end effector pose. It can also have several distinct solution modes as in the 6-DoF case. Furthermore, we add an obstacle cost to the optimization formulation such that the feasible solution is collision-free. We use the same collision cost that is used in CHOMP (Zucker et al. 2013), i.e., by using the precomputed Signed-distance Function (SDF) to compute the distance between each point on the robot link to the nearest obstacle. When there are obstacles, the standard way of doing numerical IK is to generate multiple solutions and check for collision until we obtain one that is collision-free. When the environment is cluttered with collision objects, the success rate of this approach can be low, meaning that the user needs to generate a lot of IK solutions before finding one that is collision-free. The addition of an obstacle cost helps the solver to directly optimize a collision-free configuration, but at the same time, it increases the non-convexity of the problem significantly. The solver can get stuck very easily at poor local optima, especially with a large weight on the obstacle cost. This makes it an interesting case study to showcase the TTGO capability of avoiding poor local optima. We demonstrate that TTGO could be used to find solutions robustly.

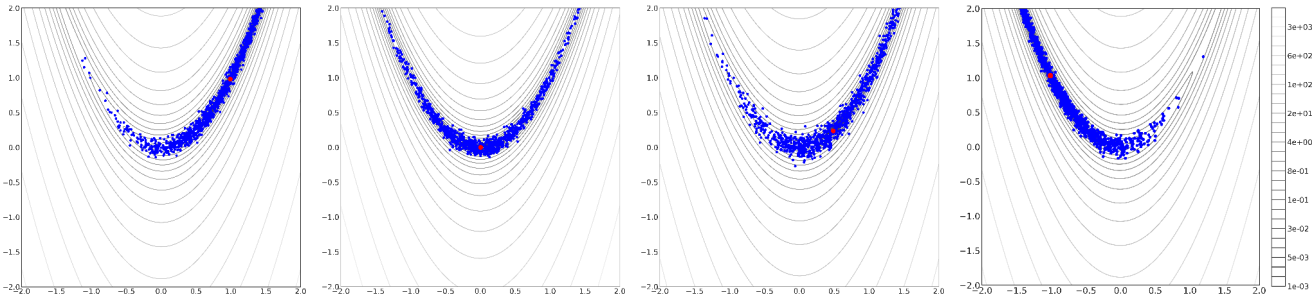
We first test the IK with obstacle cost for a 3-DoF planar robot to provide some intuition on the effectiveness of TTGO. Figure 11 and Figure 12 show some samples from TTGO conditioned on the target end effector position (shown in red). By setting $\alpha = 1$, we focus the sampling around the mode of the distribution, enabling us to obtain a very good solution even with only 1 sample (Figure 11). As we decrease α to 0.8 and retrieve more samples, we can see that multiple solutions can be obtained easily (Figure 12). 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 $d = 10$.



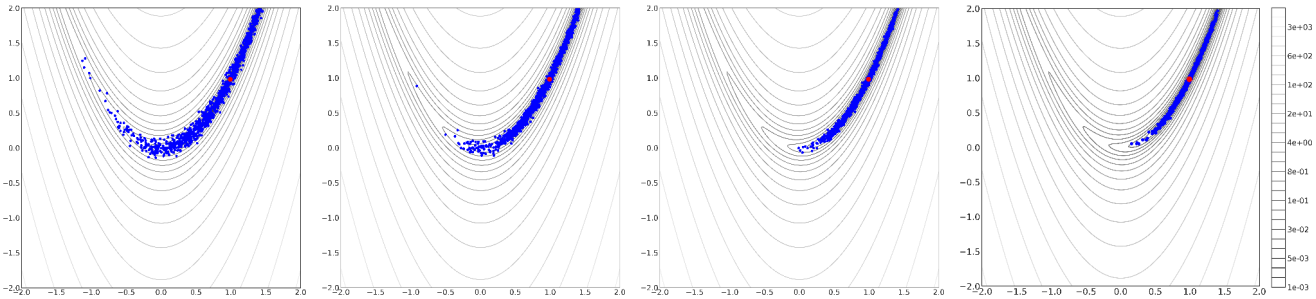
(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.75$ (d) $\alpha = 0.9$

Figure 4. 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.



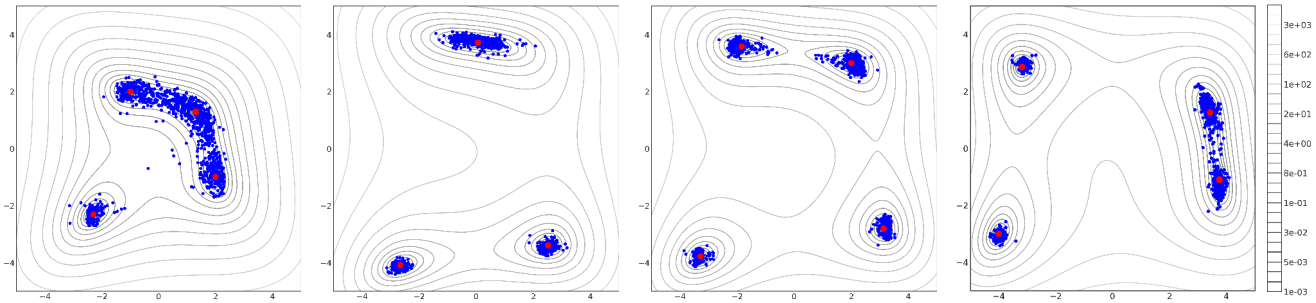
(a) $(a, b) = (1, 100)$ (b) $(a, b) = (0, 140)$ (c) $(a, b) = (0.5, 60)$ (d) $(a, b) = (-1, 100)$

Figure 5. 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.



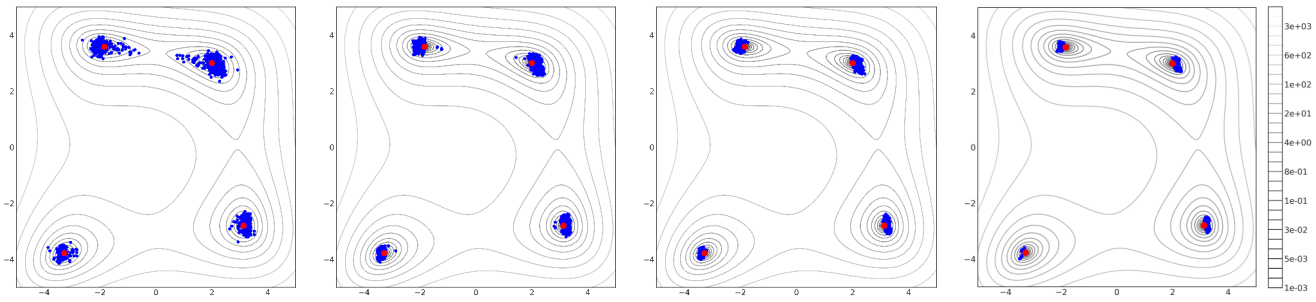
(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.75$ (d) $\alpha = 0.9$

Figure 6. 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).



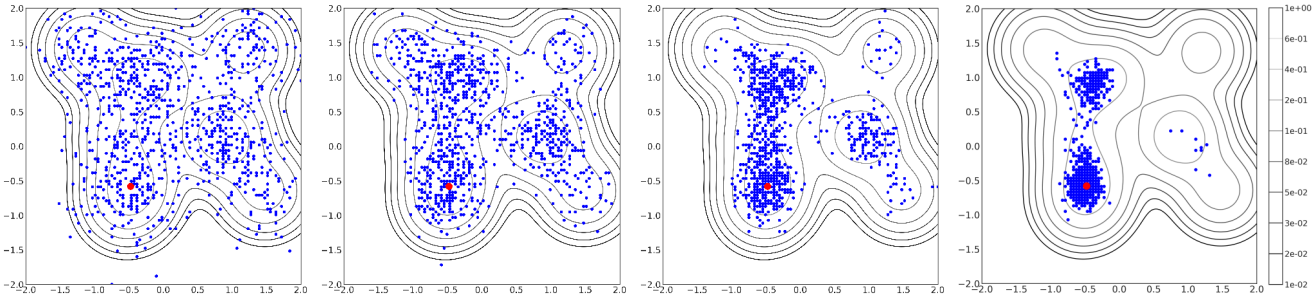
(a) $(a, b) = (3, 3)$ (b) $(a, b) = (3, 14)$ (c) $(a, b) = (7, 11)$ (d) $(a, b) = (13, 5)$

Figure 7. 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.



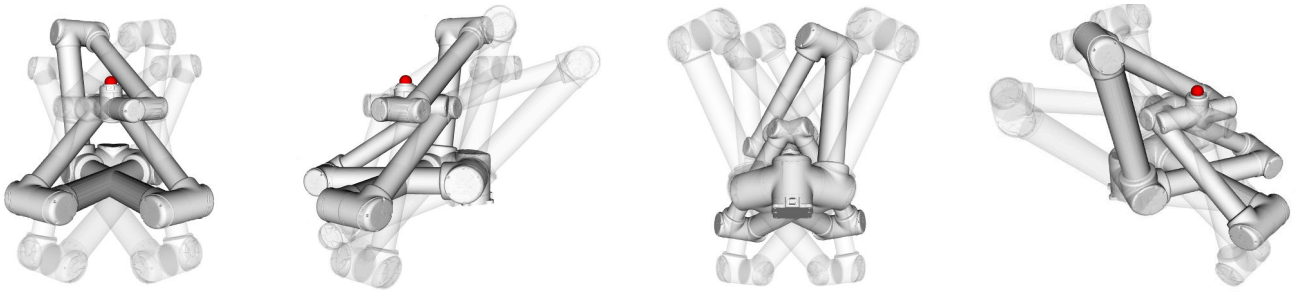
(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.9$ (d) $\alpha = 1$

Figure 8. 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.



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

Figure 9. 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.



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

Figure 10. 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.

The number of parameters of the TT cores is 1.4×10^7 whereas the original tensor $\mathcal{P}$ 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 13 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 result can be seen in Table 1. We can see that TTGO 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.

5.3 Motion Planning of Manipulators

In this section, we apply our framework to the motion planning of the Franka Emika robot to find robot motions that avoid obstacles. Note that it is generally a very high-dimensional problem. Given a robot with m DoF and considering T time intervals, the optimization variables $\boldsymbol{x}_1$ have mT dimensions. If we want to ensure that the solution avoids small obstacles, the number of time discretization should be large, i.e, bigger than 100 in the case of CHOMP. That gives us more than 700 dimensions for motion planning with a Franka Emika robot. To reduce the dimensionality, we use movement primitives with basis functions as the trajectory representation, as commonly done in learning from demonstration (Paraschos et al. 2013; Calinon 2019). With this representation, the optimization variables consist of the superposition weights of the basis functions, which is much smaller than the number of configurations. Furthermore, our specific formulation of movement primitives, as described in Appendix A.5, ensures that the motion always starts from the initial configuration and ends at the given final configuration. When the goal is given in the task space, this means that we need to first find the corresponding final configuration, e.g., using IK. In our motion planning formulation, we treat both the final configuration and the weights of the basis functions as optimization variables and solve them jointly. Finally, the cost function consists of the reaching cost, the joint limit cost, the smoothness cost, and the obstacle cost (the same cost as used in IK). More details on the motion planning formulation can be found in Appendix A.4

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 $\boldsymbol{p}_{\boldsymbol{d}}^1$ (picking location) and $\boldsymbol{p}_{\boldsymbol{d}}^2$ (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): $\boldsymbol{x}_1 = (\boldsymbol{p}_{\boldsymbol{d}}^1, \boldsymbol{p}_{\boldsymbol{d}}^2)$. 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 $\boldsymbol{p}_{\boldsymbol{d}}^1$ 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 $\boldsymbol{p}_{\boldsymbol{d}}^2$ 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 14 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 15 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 17 and 18.

The results are presented in Table 2 and 3. Similarly to the IK results, TTGO 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.

6 Discussion

6.1 Quality of the Approximation

In this paper, we used a TT model to approximate an unnormalized PDF. The quality of the approximation highly depends on the TT-rank. A nice property of TTGO 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. Furthermore, we can also use the continuous version of the TT model to allow continuous sampling. 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

The performance measures for three different applications with the Franka Emika manipulator. We compare the performance of TTGO for initializing a given gradient-based solver (namely, SLSQP) against initialization from uniform distribution. 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. 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 1. Inverse kinematics of the Franka Emika robot

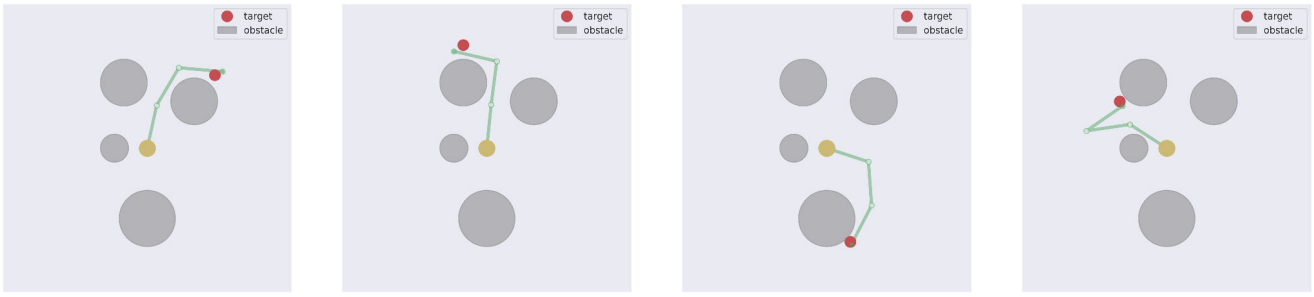
Method	α	# Samples											
		1			10			100			1000		
		c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**
TTGO	0.9	1.04	0.01	94.0	0.55	0.02	98.0	0.37	0.02	98.0	0.26	0.02	99.0
	0.75	1.52	0.07	84.0	0.65	0.02	95.0	0.37	0.02	95.0	0.24	0.03	97.0
	0.5	2.01	0.08	88.0	0.85	0.04	93.0	0.43	0.04	93.0	0.28	0.01	98.0
	0	2.88	0.17	71.0	1.23	0.05	91.0	0.68	0.05	91.0	0.39	0.04	96.0
Uniform	-	8.42	1.22	37.75	4.47	0.91	45.5	2.56	0.5	59.25	1.59	0.24	75.0

Table 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	62.0	1.1	0.09	86.0	0.71	0.1	86.0	0.58	0.09	88.0
	0.75	5.63	0.21	53.0	1.29	0.14	72.0	0.78	0.1	86.0	0.56	0.1	83.0
	0.5	4.53	0.17	50.0	1.54	0.14	64.0	0.96	0.11	83.0	0.62	0.1	84.0
	0	6.7	0.31	46.0	2.06	0.18	60.0	1.3	0.12	82.0	0.84	0.12	86.0
Uniform	-	13.85	1.34	19.25	4.79	0.91	28.75	3.02	0.68	41.0	2.06	0.45	53.5

Table 3. Pick-and-Place

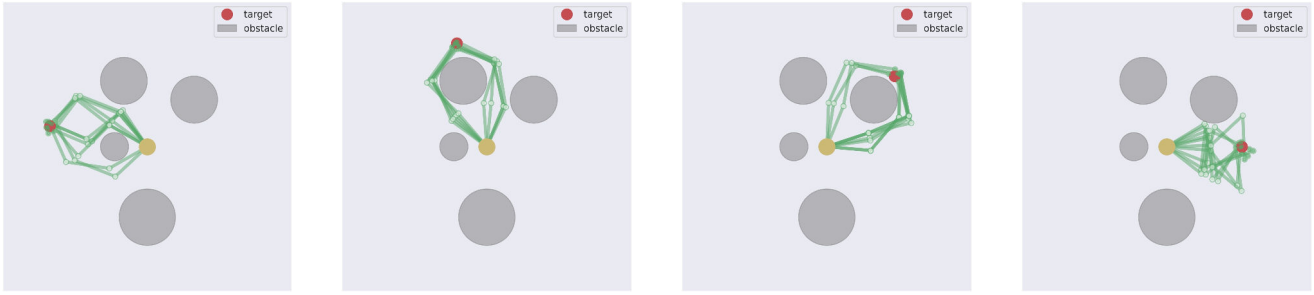
Method	α	# Samples											
		1			10			100			1000		
		c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**	c_i	c_f	**Success**
TTGO	0.9	2.41	0.16	70.0	1.41	0.15	81.0	1.05	0.15	79.0	0.87	0.14	89.0
	0.75	3.25	0.17	66.0	1.71	0.17	66.0	1.31	0.14	84.0	1.01	0.15	78.0
	0.5	4.31	0.26	54.0	2.33	0.19	62.0	1.66	0.17	77.0	1.29	0.18	76.0
	0	6.2	0.27	48.0	2.98	0.23	48.0	2.17	0.21	58.0	1.61	0.18	71.0
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)

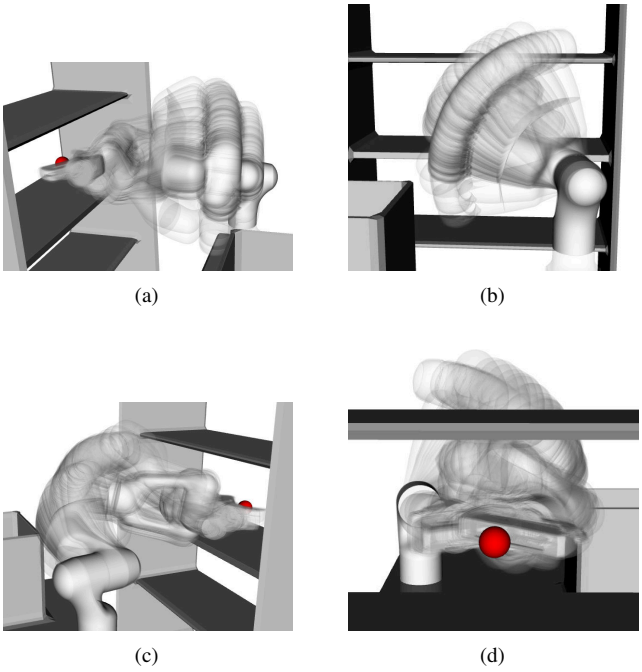
Figure 11. A single sample taken from a conditional TT distribution with $\alpha = 1$ for inverse kinematics of a 3-link planar manipulator in the presence of obstacles (gray spheres). The yellow circle and the green segments depict the base and the links of the robot, respectively. The target end effector positions are shown in red. The samples are very close to the targets and collision-free, even without refinement.

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 the high cost. Even if TT-cross cannot find an accurate low-rank



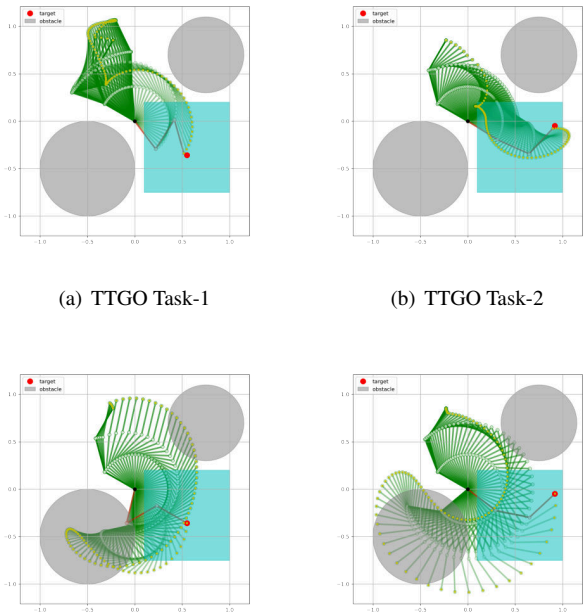
(a) (b) (c) (d)

Figure 12. 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 refinement and the multimodality of the solutions is clearly visible.



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

Figure 13. 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.



(a) TTGO Task-1 (b) TTGO Task-2

(c) Random Task-1 (d) Random Task-2

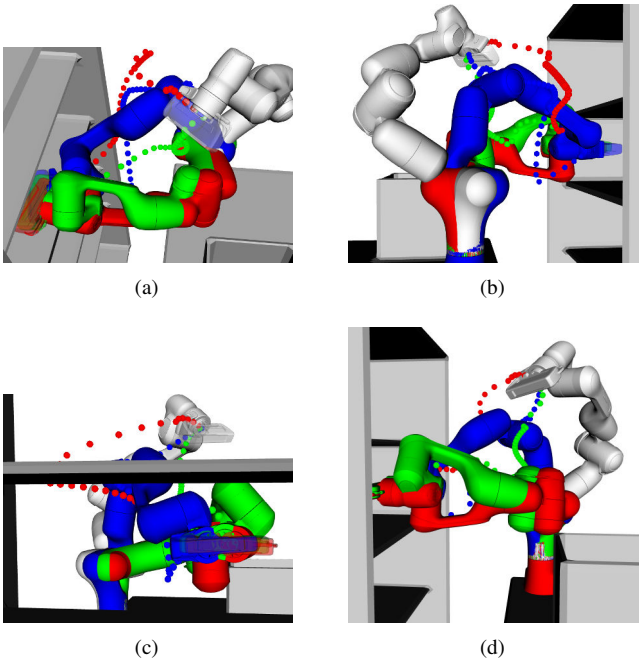
Figure 14. Motion Planning of Planar Manipulators: The task is to reach a given target point in the square region depicted in cyan (task space) from a fixed initial configuration (dark green configuration). The final configuration and the joint angle trajectory to reach the target point are the decision 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 could result in poor local optima as can be seen in (c) and (d).

TT representation across the whole domain (e.g., due to non-smoothness), it can still capture the maximal elements robustly (Sozykin et al. 2022; Goreinov et al. 2010) 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.

6.2 Computation Time

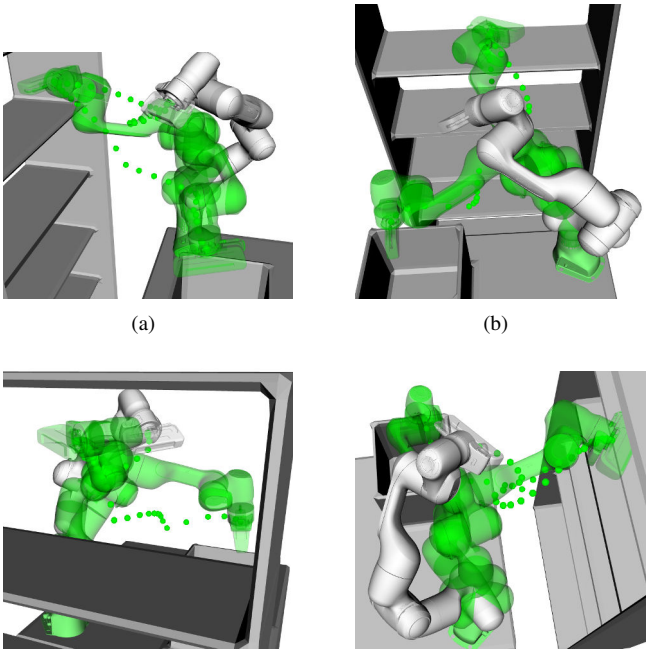
The computation time of TTGO can be divided into offline computation, i.e., the time to construct the TT model $\mathcal{P}$, and online computation, i.e., the time to condition the TT model on the given task parameters and to sample.

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 TTGO computation time. However, we used parallel implementation with GPU that allows us to construct all of the models in our applications in less than one hour.



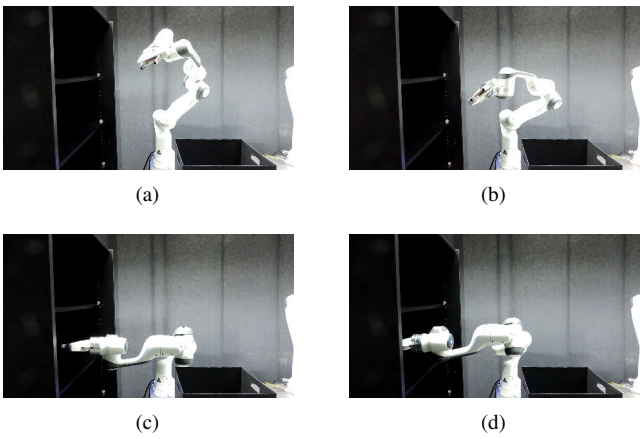
(a) (b) (c) (d)

Figure 15. 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.



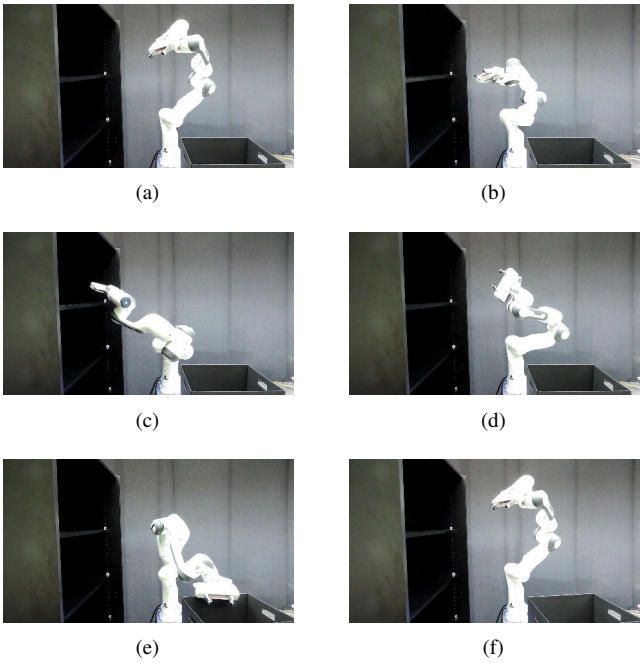
(a) (b) (c) (d)

Figure 16. 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.



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

Figure 17. Real robot implementation of one of the TTGO solutions for the reaching task. (a) to (d) shows the motion from the initial configuration to the final configuration.

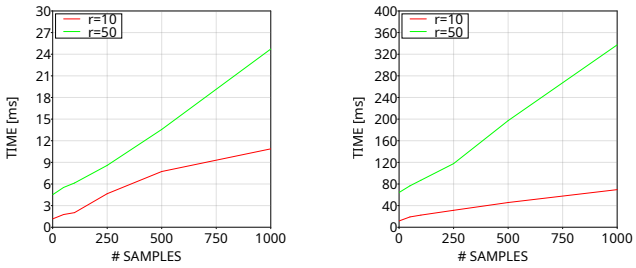


(a) (b) (c) (d) (e) (f)

Figure 18. Real robot implementation of one of the TTGO solutions for the pick-and-place task. (a) and (f) represent the initial and the final configuration of the robot (same in this case), (a) to (c) show the motion from the initial configuration to the picking configuration, (c) to (e) show the motion from the picking configuration to the placing configuration.

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 paper 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



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

Figure 19. 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 ($r = 50$).

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 (Schulman et al. 2014), 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 19. 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 1-3 ms, 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 $1s$).

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.

6.3 Comparison With Previous Work Using Variational Inference

As described in Section 2.3, the work closest to our approach is SMTO (Osa 2020) 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 article that TTGO 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 14, and we have shown that TTGO can consistently produce good solutions for different target locations. Since TTGO does not use any gradient information to find the TT model, it does not get stuck in poor local optima easily. We provide more detail on the comparison in Appendix A.6.

In Osa (2022), the author proposed another method called LSMO to handle functions with an infinite set of solutions by learning the latent representation. As we showed in Section 5.1 for sinusoidal and Rosenbrock functions, TTGO is naturally able to handle these kinds of distributions, even without any special consideration or change on the method.

Unlike TTGO, SMTO and LSMO need to solve every single optimization problem from scratch. In TTGO's terminology, this corresponds to the task parameters being constant—a special case of the problem formulation considered so far in this paper. For such problems, since we only have a single task, the training phase in TTGO 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 TTGO 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 5.1 in less than 2 seconds. In this way, TTGO can be considered as a tool for global optimization that can offer multiple solutions. Appendix A.6 discuss this in more detail.

In this paper, however, we proposed TTGO as a more generic tool. By anticipating and parameterizing the possible optimization problems using the task parameters, TTGO allows distribution of the computational effort into the offline and the 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) 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.

6.4 Multimodality

As we have shown in this paper, 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.

6.5 Further possible extensions

As test cases for TTGO, we considered in this article two robotics problems that are commonly formulated as optimization problems with specific formulations of the IK and the motion planning problems. However, the proposed method is more general and can be applied to a variety of applications in robotics. For example, we can consider other choices of task parameters, e.g., by including the initial configuration, the end effector orientation, or even the position of obstacle(s) as task parameters.

TTGO approximates the joint distribution of the task parameters and the decision variables. In the TT model, it does not differentiate between these two types of variables internally. While in this article we always conditioned the model on the given task parameters, we could actually choose any subset of variables from the joint distribution to condition on. For example, for the IK problem, it is possible to also condition on one of the joints, when we want to set a particular value for that joint. For the pick-and-place task, it is even possible to condition only on the first target point, and the second target point is treated as the decision variables. This means that we can obtain possible values of placing locations that are optimal with respect to the cost functions and the first target point. This flexibility does not exist in most other existing methods tackling similar problems, and it would be interesting to further extend this capability in other robotics applications.

Other robotics problems can be considered as long as they can be formulated as optimization problems. For example, optimal control formulates the problem of finding the control commands as an optimization problem. There are recent works that used a database approach to warm start an optimal control solver (as described in Section 2.2), which could potentially be improved by the use of TTGO. Note here that such control problems can be more challenging than the planning problems presented here as the cost function is sharper (i.e., a slight change in the control command can result in a very different state trajectory and hence the cost value). Further research would then be required to adapt TTGO to such problems. Furthermore, some applications such as task and motion planning (Toussaint et al. 2018) or footstep planner for legged robots (Deits and Tedrake 2014) can be formulated as Mixed Integer Programming. Since TTGO does not require gradient information, such a combination of discrete and continuous optimization provides another interesting application area to be explored.

In this paper, we obtained the TT model (correspondingly the TT distribution which captures the low cost solutions) in an unsupervised manner using TT-cross with access to only the definition of the cost function. This approach was motivated by the fact that in many applications we do not have the access to the samples (or solutions) that correspond to low cost for different task parameters. However, if we have a database of good solutions (i.e., optimal solutions corresponding to different possible task parameters), we can still use TTGO in an alternative way. In such cases, instead of using TT-cross to obtain the TT model, we can use other modeling techniques such as supervised learning or density estimation techniques as described in Han et al. (2018), Miller et al. (2021), Novikov et al. (2021). Such approaches, due to the expressive power and generalization abilities of TT models, can still capture multiple solutions while allowing fast ways to retrieve solutions as described in this paper. However, in robotics applications as described in Section 1, obtaining the database of good solutions is a challenging problem.

The choice of transformation used to obtain the probability function from the cost function plays an important role in TTGO. In the paper, 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 TTGO, 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 paper 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 given in this paper.

6.6 Limitations

One of the major limitations of TTGO is to scale it to very high-dimensional problems. While we have tested TTGO up to $d = 100$ dimensions, many robotics problems involve an even higher number of dimensions. For example, a standard motion planning formulation of a 7-DoF manipulator in CHOMP can easily exceed 100 dimensions. In this paper, we overcome this issue by relying on motion primitive representations, which works well for some trajectory planning applications. For other purposes, we may need to rely on other nonlinear dimensionality reduction techniques as preprocessing such as autoencoders to determine the choice of task parameters and the decision variables for TTGO.

Although constraints like joint limits can be handled naturally in TTGO, other constraints in the optimization problem needs 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.

Note that to achieve fast offline computation time, TTGO requires a batch of cost function evaluations to be processed in parallel. Without such parallelization available for computation, the time to find the TT model using TT-cross would be too long.

7 Conclusion

In this article, we have presented TTGO as a novel framework to provide approximate solutions of an optimization problem. By applying it on several challenging benchmark optimization functions and robotics applications (inverse kinematics and motion planning), we have shown that TTGO can provide diverse good quality solutions for challenging optimization problems in which the random initialization of solvers often fails. Furthermore, it can provide multiple solutions from different modes (when these different options exist). We have also shown that we can adjust the sampling priority, i.e., either to focus on obtaining the best solution or to produce more diverse solutions. All of these features can be very helpful for initializing optimization solvers on challenging robotics problems. The method could potentially be applied to other robotics tasks that can be formulated as optimization problems such as task and motion planning or optimal control, as we plan to investigate in future work.

References

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

Calinon S (2019) Mixture models for the analysis, edition, and synthesis of continuous time series. In: Bouguila N and Fan W (eds.) *Mixture Models and Applications*. Springer, Cham, pp. 39–57.

Cichocki A, Mandic DP, Lathauwer LD, Zhou G, Zhao Q, Caiafa CF and Phan AH (2015) Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine* 32: 145–163.

Dantec E, Budhiraja R, Roig A, Lembono T, Saurel G, Stasse O, Fernbach P, Tonneau S, Vijayakumar S, Calinon S et al. (2021) 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)*. pp. 8202–8208.

Deits R and Tedrake R (2014) Footstep planning on uneven terrain with mixed-integer convex optimization. In: *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*. pp. 279–286.

Del Prete A and Mansard N (2016) Robustness to joint-torque-tracking errors in task-space inverse dynamics. *IEEE Transactions on Robotics* 32(5): 1091–1105.

Dolgov S, Anaya-Izquierdo K, Fox C and Scheichl R (2020) Approximation and sampling of multivariate probability distributions in the tensor train decomposition. *Statistics and Computing* 30: 603–625.

Dolgov S and Savostyanov D (2020) Parallel cross interpolation for high-precision calculation of high-dimensional integrals. *Computer Physics Communications* 246: 106869.

Escande A, Mansard N and Wieber PB (2010) Fast resolution of hierarchized inverse kinematics with inequality constraints. In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. pp. 3733–3738.

Goreinov SA, Oseledets I, Savostyanov DV, Tyrtyshnikov EE and Zamarashkin N (2010) How to find a good submatrix. In: *Matrix Methods: Theory, Algorithms And Applications: Dedicated to the Memory of Gene Golub*. World Scientific, pp. 247–256.

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

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

Han ZY, Wang J, Fan H, Wang L and Zhang P (2018) Unsupervised generative modeling using matrix product states. *Phys. Rev. X* 8: 031012.

Hansen N, Müller SD and Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation* 11(1): 1–18.

Horowitz MB, Damle A and Burdick JW (2014) Linear Hamilton Jacobi Bellman equations in high dimensions. *IEEE Conference on Decision and Control (CDC)* : 5880–5887.

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

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

Kleff S, Meduri A, Budhiraja R, Mansard N and Righetti L (2021) High-frequency nonlinear model predictive control of a manipulator. In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. pp. 7330–7336.

Kolda TG and Bader BW (2009) Tensor decompositions and applications. *SIAM review* 51(3): 455–500.

Lembono TS, Mastalli C, Fernbach P, Mansard N and Calinon S (2020a) Learning how to walk: Warm-starting optimal control solver with memory of motion. In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. pp. 1357–1363.

Lembono TS, Paolillo A, Pignat E and Calinon S (2020b) Memory of motion for warm-starting trajectory optimization. *IEEE Robotics and Automation Letters (RA-L)* 5(2): 2594–2601.

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

Mastalli C, Budhiraja R, Merkt W, Saurel G, Hammoud B, Naveau M, Carpentier J, Righetti L, Vijayakumar S and Mansard N (2020) Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. pp. 2536–2542.

Miller J, Rabusseau G and Terilla J (2021) Tensor networks for probabilistic sequence modeling. In: *International Conference*

on Artificial Intelligence and Statistics. PMLR, pp. 3079–3087.

Mukadam M, Dong J, Yan X, Dellaert F and Boots B (2018) Continuous-time gaussian process motion planning via probabilistic inference. *Intl Journal of Robotics Research* 37(11): 1319–1340.

Novikov GS, Panov M and Oseledets I (2021) Tensor-train density estimation. In: *Uncertainty in Artificial Intelligence*. pp. 1321–1331.

Osa T (2020) Multimodal trajectory optimization for motion planning. *Intl Journal of Robotics Research* 39(8): 983–1001.

Osa T (2022) Motion planning by learning the solution manifold in trajectory optimization. *Intl Journal of Robotics Research* 41(3): 281–311.

Oseledets I and Tyrtyshnikov E (2010) TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications* 432(1): 70–88.

Oseledets IV (2011) Tensor-train decomposition. *SIAM Journal on Scientific Computing* 33: 2295–2317.

Paraschos A, Daniel C, Peters JR and Neumann G (2013) Probabilistic movement primitives. *Advances in Neural Information Processing Systems (NIPS)* 26.

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

Rabanser S, Shchur O and Günnemann S (2017) Introduction to tensor decompositions and their applications in machine learning. *ArXiv* 1711.10781.

Rasmussen C and Williams C (2006) *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press.

Rutenbar RA (1989) Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine* 5(1): 19–26.

Savostyanov DV and Oseledets IV (2011) Fast adaptive interpolation of multi-dimensional arrays in tensor train format. *The 2011 International Workshop on Multidimensional (nD) Systems* : 1–8.

Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K and Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. *Intl Journal of Robotics Research* 33(9): 1251–1270.

Shetty S, Silvério J and Calinon S (2022) Ergodic exploration using tensor train: Applications in insertion tasks. *IEEE Trans. on Robotics* 38(2): 906–921.

Sidiropoulos ND, Lathauwer LD, Fu X, Huang K, Papalexakis EE and Faloutsos C (2017) Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing* 65: 3551–3582.

Sozykin K, Chertkov A, Schutski R, Phan A, Cichocki A and Oseledets I (2022) TTOpt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning. *ArXiv* abs/2205.00293.

Stokes J and Terilla J (2019) Probabilistic modeling with matrix product states. *Entropy* 21.

Stolle M and Atkeson CG (2006) Policies based on trajectory libraries. In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. pp. 3344–3349.

Stolle M, Tappeiner H, Chestnutt J and Atkeson CG (2007) Transfer of policies based on trajectory libraries. In: *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*. pp. 2981–2986.

Sugihara T (2011) Solvability-unconcerned inverse kinematics by the Levenberg–Marquardt method. *IEEE Transactions on Robotics* 27(5): 984–991.

Toussaint MA, Allen KR, Smith KA and Tenenbaum JB (2018) Differentiable physics and stable modes for tool-use and manipulation planning. In: *Proc. Robotics: Science and Systems (R:SS)*. pp. 1–8.

Whitley D (1994) A genetic algorithm tutorial. *Statistics and computing* 4(2): 65–85.

Zheltkov DA and Osinsky A (2019) Global optimization algorithms using tensor trains. In: *International Conference on Large-Scale Scientific Computing*. Springer, pp. 197–202.

Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, Bagnell JA and Srinivasa SS (2013) CHOMP: Covariant hamiltonian optimization for motion planning. *Intl Journal of Robotics Research* 32(9-10): 1164–1193.

A Appendix

A.1 Interpolation of Tensor Cores

Given the discrete analogue tensor $\boldsymbol{\mathcal{P}}$ of a function P, we can obtain the continuous approximation by interpolating the TT cores, in a similar way as in the matrix case in Section 3.3. 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 $k \in \{1, \dots, d\}$,

$$\boldsymbol{P}^k(x_k) = \frac{x_k - x_k^{i_k}}{x_k^{i_k+1} - x_k^{i_k}} \boldsymbol{\mathcal{P}}^k_{:,i_k+1,:} + \frac{x_k^{i_k+1} - x_k}{x_k^{i_k+1} - x_k^{i_k}} \boldsymbol{\mathcal{P}}^k_{:,i_k,:}, \tag{14}$$

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

$$P(x_1, \dots, x_d) \approx \boldsymbol{P}^1(x_1) \cdots \boldsymbol{P}^d(x_d). \tag{15}$$

Note that a higher-order polynomial interpolation can also be used if needed.

A.2 TT-Cross Algorithm

In this section, we outline TT-cross algorithm for finding a TT decomposition. Here, we only sketch the algorithm for a fixed rank approximation and highlight how it can be adapted to adjust the rank of the approximation dynamically, as used in this paper. For more detail, we refer the readers to Oseledets and Tyrtyshnikov (2010); Savostyanov and Oseledets (2011); Dolgov and Savostyanov (2020).

1. *Input*: d-th order tensor $\boldsymbol{\mathcal{P}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, approximation rank $\boldsymbol{r} = (r_1, \dots, r_{d-1})$
2. *Set:* $r_0 = 1, r_d = 1, n_0 = 1, n_{d+1} = 1$
3. *Initialize:* randomly choose initial column index sets $\boldsymbol{j}^k = (j_1^k, \dots, j_{r_k}^k) \subset (1, \dots, n_{k+1} \cdots n_d)$, for $k = 1, \dots, d-1$
4. Unfold the tensor $\boldsymbol{\mathcal{P}}$ along mode 1 into matrix: $\boldsymbol{P}^1 = \mathbb{R}^{n_1 \times n_2 \cdots n_d}$
5. Repeat until convergence:

5.1 for $k = 1, \dots, d-1$ (forward sweep)
- using MAXVOL to find the row index set $\boldsymbol{i}^k = (i_1^k, \dots, i_{r_k}^k) \subset (1, \dots, n_k r_{k-1})$ corresponding to the maximum submatrix of $\boldsymbol{P}^k_{:,\boldsymbol{j}^k}$
- reshape the matrix $\boldsymbol{P}^k_{\boldsymbol{i}^k,:}$ into matrix $\boldsymbol{P}^{k+1} \in \mathbb{R}^{n_{k+1} r_k \times n_{k+2} \cdots n_d}$.

5.2 reshape matrix $\boldsymbol{P}^d \in \mathbb{R}^{n_d r_{d-1} \times 1}$ appropriately (mode-2 unfolding) to obtain the TT-core $\boldsymbol{\mathcal{P}}^d \in \mathbb{R}^{r_{d-1} \times n_d \times 1}$

5.3 for $k = d-1, \dots, 1$ (backward sweep)
- using MAXVOL to find the column index set $\boldsymbol{j}^k = (j_1^k, \dots, j_{r_k}^k) \subset (1, \dots, n_{k+1} \cdots n_d)$ corresponding to the maximum submatrix of $\boldsymbol{P}^k_{\boldsymbol{i}^k,:}$
- reshape the matrix $\boldsymbol{P}^k_{:,\boldsymbol{j}^k}(\boldsymbol{P}^k_{\boldsymbol{i}^k,\boldsymbol{j}^k})^{-1} \in \mathbb{R}^{n_k r_{k-1} \times r_k}$ appropriately (mode-2 unfolding) to obtain the TT-core $\boldsymbol{\mathcal{P}}^k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$

6. *Output:* The TT cores $\boldsymbol{\mathcal{P}}^k$ for $k = 1, \dots, d$

Note that in practice, the MAXVOL and matrix operations are performed using the QR decomposition of the matrices in the above algorithm to avoid numerical instabilities. Moreover, in a practical implementation of the above algorithm the matrices $\boldsymbol{P}^k$ need not be computed, we directly work with the function that returns its elements (or sub-matrices) given the indices. Commonly used convergence criteria include the maximum number of iterations in TT-cross (forward and backward sweeps in the above algorithm) and a lower bound on the change in the norm of the TT approximation over successive iterations of TT-cross. In the algorithm, the rank r can also be adapted dynamically after each iteration by either reducing the index set or augmenting it with a randomly chosen enrichment set for each mode during the iterations. We refer to Savostyanov and Oseledets (2011); Dolgov et al. (2020) for the details.

A.3 Inverse Kinematics Formulation

The cost function for the inverse kinematics problem in Section 5.2.2 is given by

$$C(\boldsymbol{x}) = \frac{1}{3}\Big(\frac{C_p(\boldsymbol{\theta}, \boldsymbol{p}_d)}{\beta_p} + \frac{C_{\text{obst}}(\boldsymbol{\theta})}{\beta_{\text{obst}}} + \frac{C_{\text{orient}}(\boldsymbol{\theta})}{\beta_{\text{orient}}}\Big), \quad (16)$$

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 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 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.4 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

$$C(\boldsymbol{x}) = \frac{1}{4}\Big(\frac{C_p(\boldsymbol{x})}{\beta_p} + \frac{C_{\text{obst}}(\boldsymbol{x})}{\beta_{\text{obst}}} + \frac{C_{\text{orient}}(\boldsymbol{x})}{\beta_{\text{orient}}} + \frac{C_{\text{control}}(\boldsymbol{x})}{\beta_{\text{control}}}\Big) \quad (17)$$

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 5.2.2 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.

We consider the target to be in Cartesian space instead of the configuration space. Note that an optimization-based motion planning solver can handle both types of targets by adjusting the reaching cost term. Reaching a target in the configuration space is usually an easier optimization problem as it provides a clear gradient to the solver, which is not the case for reaching a Cartesian target. On the one hand, a Cartesian target implies a larger solution space, as the target may correspond to more than one configuration. On the other hand, the locally optimal behavior of a gradient-based solver means that it will try to reach the target with the configuration that is closest to the initial configuration,

especially if it is initialized by a stationary trajectory at the initial configuration. When such a solution is not feasible, it is difficult for a gradient-based solver to find another solution with a final configuration significantly different from the initial one, except with a good initialization. An alternative is to first determine several possible final configurations using IK, and then use the motion planning solvers to reach those configurations. However, choosing the good configurations as the target is not trivial, as we cannot easily guess whether a particular configuration is reachable from the initial configuration. Even when it can find a solution, the solution may be highly suboptimal.

In our formulation, we consider the IK problem and the motion planning problem simultaneously. The decision variables consist of two parts: the robot configuration(s) that correspond to the Cartesian target(s), and the joint angle trajectory that reaches those configurations. While simultaneously optimizing them is quite difficult, our TTGO formulation allows us to obtain even multiple solutions. To reduce the dimensionality of the problem, we represent the joint angle trajectory using motion primitives, as described in Section A.5. Given the initial and final configuration, our motion primitives formulation ensures that the movement always starts from the initial configuration and ends at the final configuration while satisfying joint limits.

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.

A.5 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.

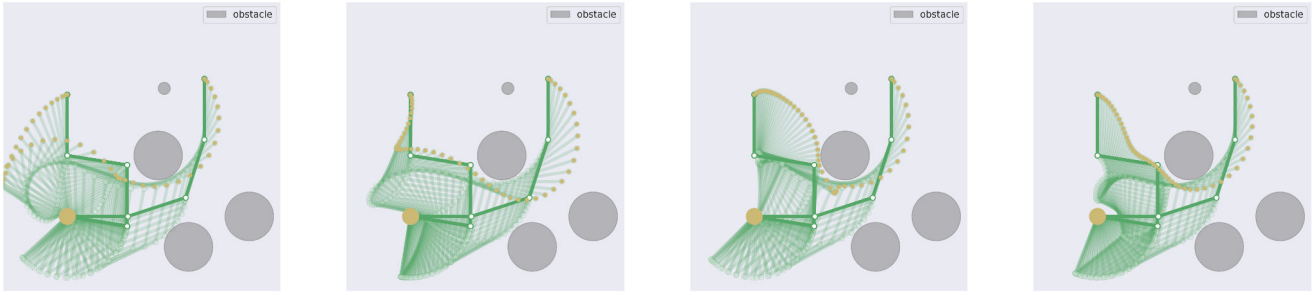
This way we can generate smooth motion while satisfying the boundary conditions and the joint limits, and maintain zero velocity at the boundary.

A.6 TTGO with Constant Task Parameters

In this paper, we described TTGO in its generic form, where we consider varying task parameters when training the TT model. This allowed us to produce approximate solutions to a given task quickly by conditioning the TT model. However, TTGO can also be used when we only want to solve a single task. In this particular case, the TT model corresponds to the probability distribution of only the optimization variables, and the training will require significantly less time as compared to the generic form. A maximum TT-rank < 5 works well for the applications considered in this paper. In terms of the computation time and the quality of solution, it is comparable to evolutionary strategies such as CMA-ES or GA, but TTGO can offer multiple solutions.

For such applications, our work is closely related to TTOpt (Sozykin et al. 2022) which is a gradient-free discrete optimization method based on TT-cross. The performance of the method has been shown to be competitive to evolutionary strategies. In this approach, the objective is to maximize a reward function (analogous to the probability density function in TTGO). TTOpt discretizes the reward function and it assumes that the maximal element of the discrete analogue of the reward function closely approximates the maximum of the reward function. The maximum of the discrete analogue is found using TT-cross. Here, the main interest to use TT-cross is not for building a TT approximation but the following feature of TT-cross: the maximal elements of the tensor is highly likely to be in the maximum volume submatrix which is found using MAXVOL in TT-cross and the maximal element of the submatrix increases monotonically over the iterations. During each iteration of TT-cross, the maximal element from the submatrix found using MAXVOL is stored in the memory and updated in the following iterations until convergence. Unlike TTOpt, TTGO uses TT-cross to model the density function first and uses the samples from the TT model to approximate a solution which is then refined using local search techniques, while providing the option of estimating multiple solutions.

To test the performance of TTGO as a single task optimizer, 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



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

Figure 20. 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.

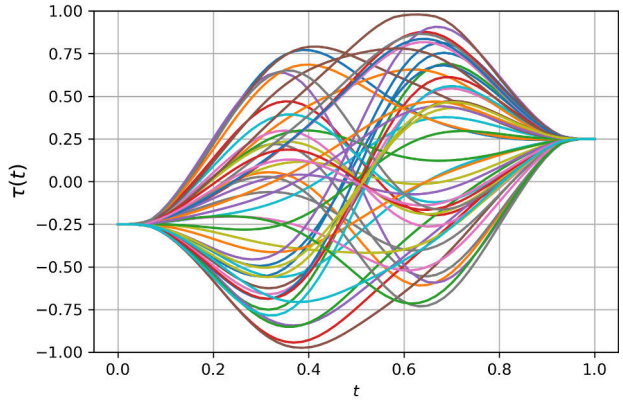


Figure 21. A distribution of 50 smooth trajectories generated by transforming trajectories generated by using two radial basis functions with weights chosen uniformly in the range $[-1, 1]$. The transformations are done to maintain a boundary condition $\tau_0 = -0.25, \tau_1 = 0.25$ and the limits $\tau_{\min} = -1, \tau_{\max} = 1$.

in Appendix A.5, thus the optimization variables are the weights of the basis functions. We used 2 radial basis functions for each joint.

For the 2-D planar robot, we replicate the setting in Figure 7 of Osa (2020), 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 20 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 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 5.2.2. With this 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 1.

In comparison, SMTO (Osa 2020) takes around 2 minutes to solve the 2-D planar robot problem and 1 minute to solve the 7-DoF manipulator example (using their matlab codes), whereas LSMO (Osa 2022) 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 20, even 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 $\sim 500s$.

Acknowledgement

This work was supported in part by the Swiss National Science Foundation through the LEARN-REAL project (https://learn-real.eu/, CHIST-ERA-17-ORMR-006) and by the European Commission's Horizon 2020 Programme through the MEMMO project (Memory of Motion, https://www.memmo-project.eu/, grant agreement 780684).